硬件篇

前言：拿到产品后我该怎么做？

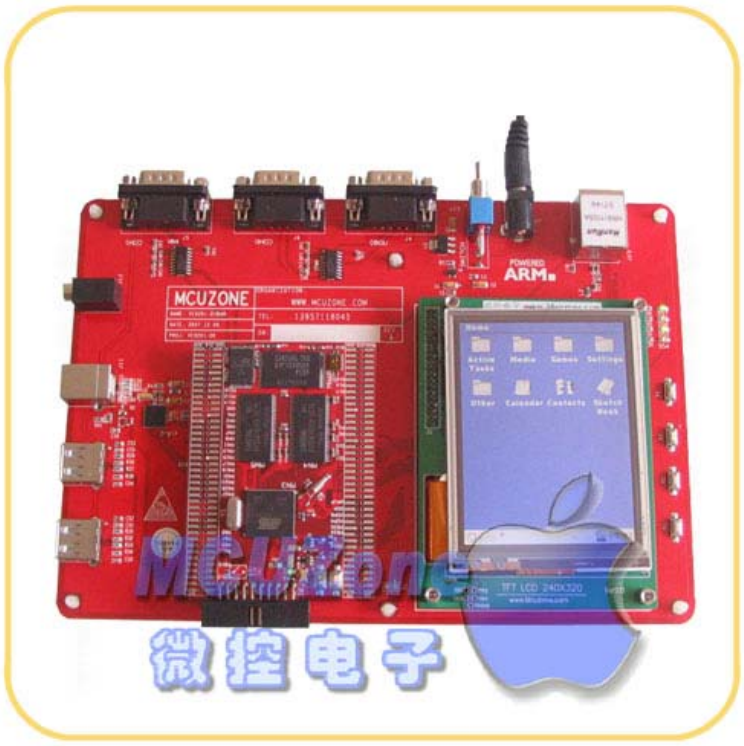
9261 核心板有好几款配套板卡，在和不同的板卡配套使用的时候，需要注意一些情况。

如果你拿到的是如下图所示的单纯核心板，可以通过核心板方面的 miniUSB 来给系统供电，然后就可以通过 SAM-BA 进行连接或者 JTAG 调试。

当然，你也可以找到核心板两侧排针上的 5V 和 GND 插针，直接加外部 5V 给系统供电。在自行设计底板的时候，也只需要通过该 5V 对核心板供电即可，核心板板载 3.3V 和 1.2V LDO。

注意：如果直接通过外部电源给核心板供电，请断开核心板反面 miniUSB 接口附近的标号为 NC 的 0 欧姆 0805 封装电阻。以避免外部电源和 PC 机的 USB 电源（仅当采用外部电源供电，且 miniUSB 口连接至 PC 机的情况下才会出现电源冲突风险）发生冲突。

如果你购买的是如下图的 VC9261-EK，即核心板加大底板并配备了 LCD，请注意供电情况，该套装通过标配的外部直流电源供电，连接好板子后，请将电源开关向上拨，即可通电。如果你要使用别的外部电源供电，请注意输入范围为 6-9V DC，外负内正。建议 6-7V，如果电压过高，1117-5 LDO 将明显发烫。



注意：该 VC9261-EK 的 DBGU（调试串口）使用的是 RS-232 串口，即 J4。如果要观察 DBGU 输出信息，请使用交叉串口线连接。

注意：出厂已经写入 Wince Demo，请耐心等待 15－20 秒钟即会出现触摸屏校准显示：

请在“＋”中心位置进行点击，首先是中间位置，然后是 4 个角落（并不是真正的物理角落，只是 4 个定位点而已），总共 5 个位置需进行位置校准，校准完毕点任意位置即可进入 Wince。

第一章：AT91SAM9261 介绍

一、ATMEL 英文原文介绍：

The AT91SAM9261 is an ultra low-power, deterministic microcontroller based on the ARM926EJ-S processor, with 16K byte instruction and 16K byte data cache memories. It operates at 210 MIPS with a 190 MHz clock. It features 160K bytes of SRAM and 32K bytes of ROM with single cycle access at maximum processor or bus speed, together with an external bus interface with controllers for SDRAM and static memories including NAND Flash and CompactFlash.

Its extensive peripheral set includes USB Full Speed Host and Device interfaces, an LCD Controller, Multimedia Card Interface (MCI), Synchronous Serial Controllers (SSC), USARTs, Master/Slave Serial Peripheral Interfaces (SPI), a three-channel 16-bit Timer Counter (TC) and a Two Wire Interface (TWI). Three 32-bit Parallel I/O Controllers multiplex the pins to/from these peripherals in order to reduce the device pin count, and peripheral DMA channels maximize the data throughput between these interfaces and the on- and off-chip memories.

The AT91SAM9261 has a fully featured system controller for efficient system management, including a reset controller, shutdown controller, clock management, advanced interrupt controller (AIC), debug unit (DBGU), periodic interval timer, watchdog timer and real-time timer.

The AT91SAM9261 throughput and the DSP extensions to its instruction set allow complex DSP functions, such as biometrics, voice recognition, software modems, or encryption/decryption algorithms like RSA, to be executed very quickly in burst mode, so the system can be shut down much of the time. It consumes only 2.5 uA in standby mode, and 400 uA at 500 Hz. In industrial temperature range, its current consumption at 200 MIPS with all peripherals turned on is just 65 mA. It is supplied in a 217-ball LFBGA RoHS-compliant package.

The AT91SAM9261 is supported by the AT91SAM9261 Evaluation Board and extensive third-party application development tools. It supports both Linux and Windows CE. It is targeted at low power, high throughput wireless handheld applications, such as wireless PoS devices.

二、中文介绍：

AT91SAM9261 是 ATMEL 新近主推的一款通用 ARM9 处理器，是一款以 ARM926EJ-S 处理器为核心的超低功耗"确定过程式"(deterministic) 微控制器。该型号乃专为功耗低、数据吞吐量大的无线手持式应用 (例如无线销售终端 (point-of-sale, PoS) 设备) 市场而开发的，其待机电流仅为 2.5 μA；工作频率 500Hz 时，电流消耗则为 400μA 。在工业级温度范围内，处理器性能达到 200MIPS 时，即使所有外设开启，其工作电流也仅仅是 65mA。

由于 AT91SAM9261 能提供庞大的数据吞吐量，并采用含数字信号处理器 (DSP) 扩展功能的指令集，因此能够在成组模式下快速地执行复杂的 DSP 功能，例如生物认证、语音识别、软件调制解调器或象 RSA 等加密 / 解密算法，从而令系统大多数时间都处于关闭状态。就一般的销售终端应用例如租车服务而言，电池使用寿命为 4 小时，如果使用 Atmel 的 MCU 便可以把电池寿命延长 4 倍，达到 16 小时。

——多层式高速总线内部架构及专用的外设 DMA

由于这款 MCU 采用多层总线矩阵的并行机理，因此可以显着提高数据流量。该总线矩阵能把五个高速总线 (AHB) 控制 (包括处理器指令和数据总线、外设直接存取内存 (DMA) 控制器 (peripheral DMA controller, PDC)，以及两个用于 USB 主机和 LCD 控制器的专用 DMA)，同时连接到片上外设，以及内置或外置的存储器上。此外，Atmel 已经扩展了基于 ARM7 MCU 的外设 DMA 控制器，在 AT91SAM9261 中增加了 19 条 PDC 通道。PDC 的作用是为外设及存储器之间传输数据，这样既可分担 CPU 的工作量，而且也能提高数据传输率，并让 ARM9 处理器专注于繁重的计算任务。

——大容量片上 SRAM 和灵活的确定过程式 TCM 支持机制。

AT91SAM9261 设有 16K 字节的数据高速缓存、16K 字节的指令 / 写操作缓存、160K 字节的 200 MHz 单循环存取 SRAM，以及 USB 器件专用的 2K 字节内置式 DPRAM。Atmel 公司充分发挥了 ARM926EJ-S 紧密式耦合内存 (Tightly Coupled Memory, TCM) 构架之优势，让传统 (非高速缓存) SRAM 直接连接到 ARM 处理器上、而不会出现滞后情况。AT91SAM9261 的独立指

令和数据缓存能支持 WinCE 和 Linux 操作系统;而其基于 TCM 构架的 SRAM 区块则是"确定过程式"实时操作系统的理想安排。另一方面，160K 字节的片上 SRAM 可划分成多个 16K 字节的区块，作为"指令 TCM"、"数据 TCM"或片上外设的缓冲，从而让编程人员灵活地优化系统的性能和功耗。

——内置式 LCD 控制器

大部份 PoS 或其它数据输入设备都配备了 LCD 图形接口，以小型 LCD 屏幕显示选项菜单。AT91SAM9261 的片上 LCD 控制器，能支持黑白显示和高达 16M 的彩色显示，令显示器可以达到 2048 X 2048 的高分辨率。此外，160K 字节的片上 SRAM 可配置为帧缓冲，而不必采用外置的帧缓冲，大大节省了材料和延长电池寿命。

——全速 USB 主控制器

AT91SAM9261 内置的 USB 主控制器能够无缝地与各种 USB 设备（包括鼠标、键盘、条码读取设备、无线 LAN、DECT 或蓝牙调制解调器）连接。其专用的 DMA 控制器和多层式 AHB 内部总线架构，能有效地避免处理器参与数据传输。另外，这个 USB 端口也可以连接一台 PC，以升级和维护系统。

——8 个级别的优先中断控制器

ARM9 的中断控制设计（interrupt control scheme）不足以满足由中断主导的实时系统的需求。有见及此，Atmel 公司在此设计基础上进行了改良，加入具有 8 个优先级别、并可单独屏蔽的向量中断控制器。该中断控制器能处理多达 32 个内部或外部的中断源，将中断的反应时间缩到最短。

——系统控制器支持实时应用

AT91SAM9261 设有一个配备了完整监管功能的系统控制器，包括一些时钟源（振荡器、PLL)、实时周期间隔及监视定时器、复位和关闭控制器、后备寄存器和实时定时器、一个功率管理控制器、一个调试单元、以及多个用于 I/O 多任务操作的 PIO 控制器。这个系统控制器能为处理器的内核和各外设提供可配置的时钟速率。此外，还提供四个可编程的输出时钟，并将内核和特定的外设设定于闲置模式，这样便可把功耗降至最低。

——片上外设

AT91SAM9261 的外设包括了 SD 及多媒体兼容的多媒体接口（MMC)、三

个同步串行控制器、三个通用同步 / 异步收发器 (USART)、一个除错用的通用异步收发器 (UART)、两个主 / 从串行外设接口 (SPI)、一个 3 通道的 16 位定时器 / 计数器、双线接口 (TWI)；而所有数字信号接脚上都设有 IEEE1149.1 JTAG 边界扫描测试接口。

三、中文数据手册摘要

特点

- 融合了 ARM926EJ-S™ ARM® Thumb® 处理器
  - 扩展 DSP 指令
  - ARM Jazelle 技术提供了 Java 加速功能
  - 16K 字节数据缓存，16K 字节指令缓存，写缓冲器
  - 工作于 190 MHz 时性能高达 210 MIPS
  - 存储器管理单元
  - 嵌入式 ICE ，支持调试信道
  - 中等规模的嵌入式宏单元结构
- 附加的嵌入式存储器
  - 32K 字节片内 ROM,最大总线速率下单周期访问
  - 160K 字节片内 SRAM，最大处理器或总线速率下单周期访问
- 外部总线接口（EBI）
  - 支持 SDRAM，静态存储器，NAND Flash 和 CompactFlash
- LCD 控制器
  - 支持被动或主动显示
  - 在 STN 彩色模式下达 16 位深每像素
  - 在 TFT 模式下达 16 M 色（24 位深每像素），分辨率高达 2048*2048
- USB
  - USB 2.0 全速（12M 位每秒）主机双端口
    双重片上收发器
    集成 FIFOs 和专用 DMA 通道
  - USB 2.0 全速（12M 位每秒）设备端口
    片上收发器，2K 字节可配置的集成 FIFOs
- 总线矩阵
  - 管理五个主控和五个从控
  - 启动模式选择选项
  - Remap 命令
- 全特征系统控制器(SYSC)提供了有效系统管理，包括
  - 复位控制器，掉电控制器，支持总共 16 字节的四个 32-bit 电池备份寄存器
  - 时钟发生器和功率管理控制器
  - 先进的中断控制器和调试部件
  - 周期间隔定时器，看门狗定时器和实时定时器
  - 三个 32 位 PIO 控制器
- 复位控制器(RSTC)
  - 基于上电复位的单元，复位源辨认和复位输出控制
- 掉电控制器(SHDWC)
  - 可编程掉电引脚控制和唤醒电路

- 时钟发生器(CKGR)
  - 电池备份电源上的 32.768KHz 低功率振荡器，提供一个永久的慢速时钟
  - 3 到 20MHz 的片上振荡器和两个 PLL
- 功率管理控制器(PMC)
  - 超慢速时钟操作模式，软件可编程功率优化能力
  - 四个可编程外部时钟信号
- 先进的终端控制器(AIC)
  - 可单独屏蔽的，8 级优先级，向量中断源
  - 三个外部中断源和一个快速中断源，伪中断保护
- 调试部件(DBGU)
  - 2 线 USART 兼容接口，可通过编程禁止通过 ICE 访问
- 周期间隔定时器(PIT)
  - 20 位间隔定时器加 12 位间隔计数器
- 看门狗定时器(WDT)
  - 受预设值保护的、一次性可编程的、运行在慢速时钟的 12 位窗口计数器
- 实时定时器(RTT)
  - 运行于慢速时钟的 32 位自由运行的(备份)计数器
- 三个 32 位并行输入/输出控制器(PIO)PIOA,PIOB 和 PIOC
  - 96 可编程 I/O 口线多路复用支持达两个外设 I/O 口
  - 在每个 I/O 口线上具有输入改变中断能力
  - 单独得可编程开漏，上拉电阻和同步输出
- 19 个外设 DMA 通道(PDC)
- 多媒体卡接口(MCI)
  - 支持 SD 卡和 MultiMediaCard（MMC 卡）
  - 自动协议控制,通过 PDC 与 MMC 和 SD 卡进行快速自动数据传输
- 三个同步串行控制器(SSC)
  - 每个接收器和发送器都具有独立的时钟和帧同步信号
  - 支持 IIS 模拟接口，支持时分多路复用
  - 支持 32 位数据传输的高速连续数据流功能
- 三个通用同步/异步收发器(USART)
  - 独立的波特率发生器，IrDA 红外调制/解调
  - 支持 ISO7816 T0/T1 智能卡，硬件和软件握手信号，支持 RS485
- 两个 主/从串行外设接口(SPI)
  - 8 到 16 位可编程数据长度，四个外部外设片选
- 一个三通道 16 位定时器/计数器(TC)
  - 三个外部时钟输入，每个通道有两个多用途 I/O 引脚
  - 倍速 PWM 发生功能，捕捉波形模式，递增/递减计数功能
- 一个两线接口(TWI)
  - 支持主控模式，支持所有两线 Atmel EEPROM
- IEEE 1149.1 JTAG 边界扫描,可以支持所有数字引脚
- 电源
  - 为 VDDCORE 和 VDDBU 提供 1.08V 到 1.32V 电压
  - 为 VDDOSC 和 VDDPLL 提供 3.0V 到 3.6V 电压
  - 为 VDDIOP（外设 I/O 口）提供 2.7V 到 3.6V 电压

— 为 VDDIOM（存储器 I/O 口）提供 1.65V 到 1.95V 和 3.0V 到 3.6V 电压
- 符合 RoHS 的 217 球的 LFBGA 封装

描述

AT91SAM9261 是以 ARM926EJ-S ARM Thumb 处理器为核心的完全的片上系统，它扩展了 DSP 指令集和 Jazelle Java 加速器。主时钟频率 190MHz 时性能高达 210 MIPS

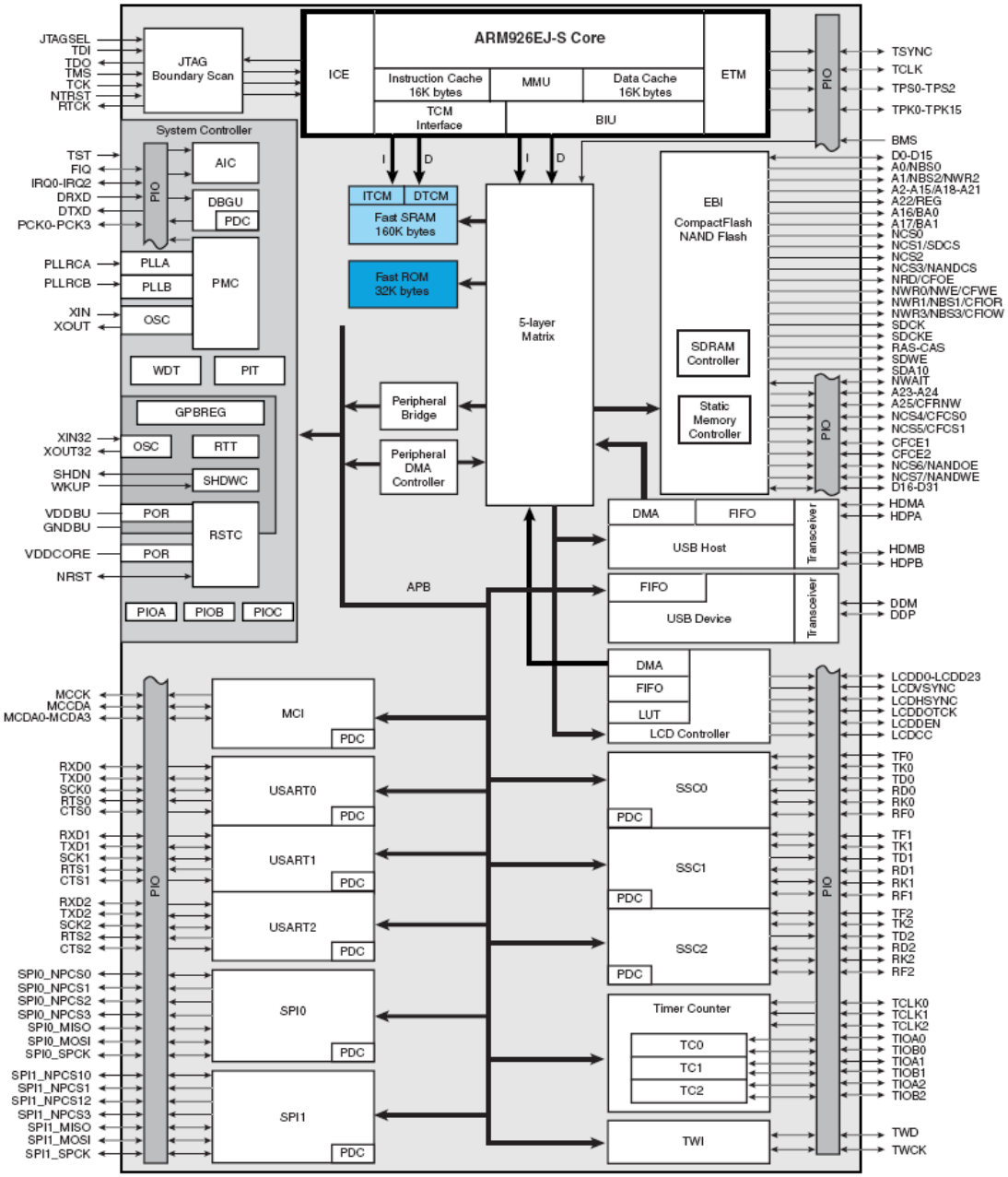
AT91SAM9261 是一个为带 LCD 显示应用而优化了的主机处理器。它的集成 LCD 控制器支持 BW 而且达到 16M 色彩，主动和被动 LCD 显示。160K 字节的片上 SRAM 可配置为帧缓冲，能将 LCD 刷新对整体处理器性能上的影响减到最小。外部总线接口包括支持同步 DRAM（SDRAM）和支持静态存储器的控制器，并有特殊接口电路以支持 CompactFLASH、NAND Flash。

AT91SAM9261 集成了一个支持影射的基于 ROM 的 Boot loader，例如，从外部 DataFlash 影射到外部 SDRAM。由软件控制的功率管理控制器(PMC)通过有选择的启用和关闭处理器、各种外设和工作频率的调解能使系统功率损耗保持最小。

AT91SAM9261 还集成了宽范围的调试特性。包括 JTAG-ICE,一个专用的 UART 调试通道(DBGU)和嵌入式实时追踪。这使所有应用的调试和开发，特别是严格要求实时性的应用得以实现。

方框图

图 2-1. AT91SAM9261 方框图



第二章：AT91SAM9261 核心板硬件介绍

本站的 AT91SAM9261 核心板参考 ATMEL 的原装 EK 原理图进行设计，并在此基础上进行了部分裁减和冗余设计。本章主要介绍本站 AT91SAM9261 核心板的资源和原理图。

一、AT91SAM9261 核心板资源：

1、CPU

- AT91SAM9261 或 AT91SAM9261S，工业级，无铅，217BGA

2、存储器

- 64MB SDRAM ，为 K4S561632 或 HY57V561620
- 4MB DataFash，AT45DB321，存储系统启动代码；可另配 8M DATAFLASH
- 64/128MB NandFlash,存储系统内核和应用程序 ；可升级为 256MB

3、接口

- JTAG 调试接口 ，标准 20 芯接口
- 一个 SD 卡接口,无容量限制,AT91SAM9261 自带 SD 控制器（位于反面）
- 一个 USB 2.0 F-S Device 端口,mini USB 封装 （位于反面）
- 全部 GPIO 引出
- 地址总线和数据总线引出

5、电源

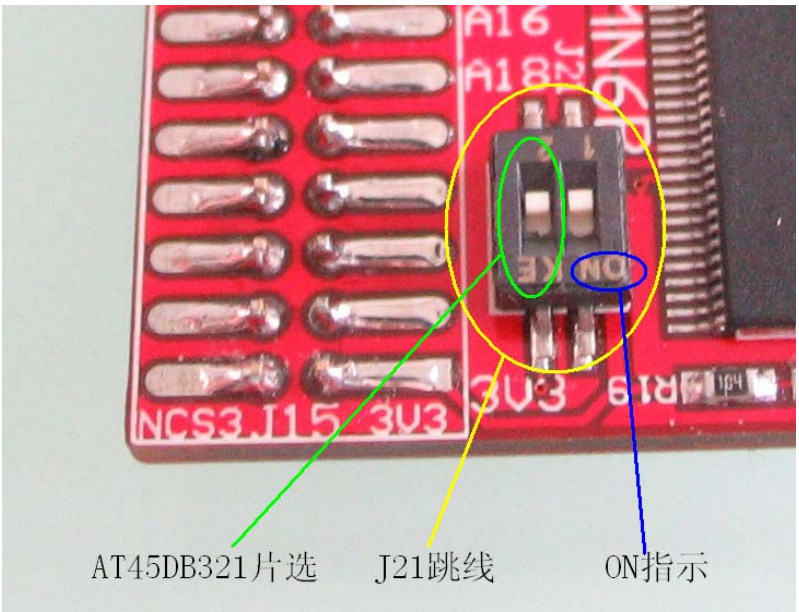
- +7.5V 电源输入，板载 1.2V 和 3.3V 高效率、低压降、极低静态电流 LDO
- 后备电源接口 J1 引出

二、AT91SAM9261 核心板参考图片

AT91SAM9261 核心板实物照片：

其中 NAND FLASH 和 DATAFLASH 可以根据用户要求进行定制。

核心板上最关键的一个跳线是 FLASH 片选，如下图：



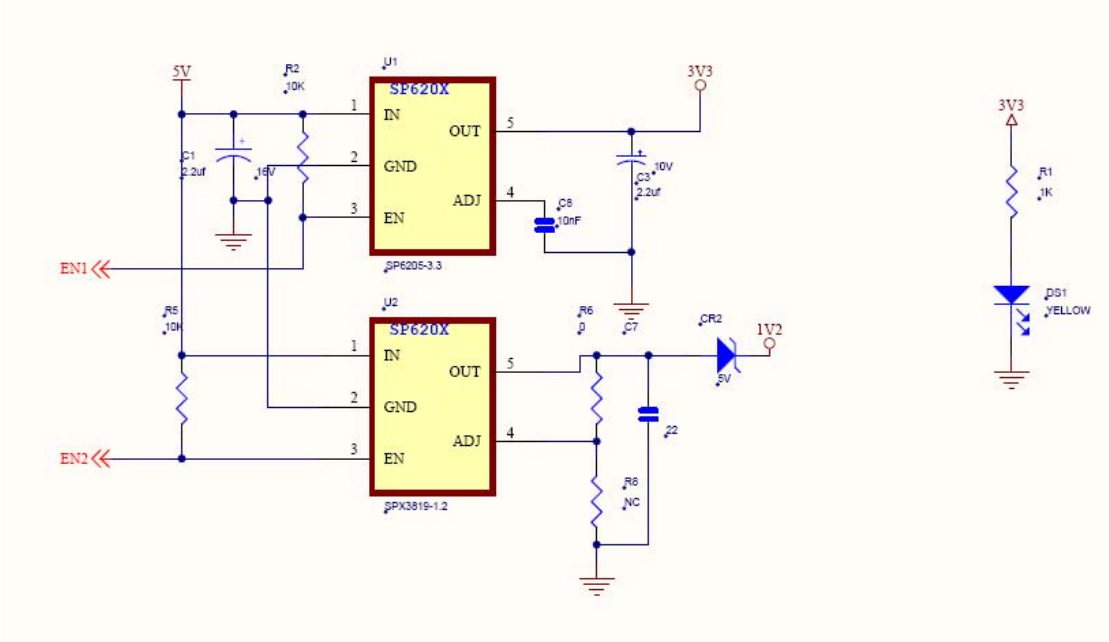
绿色框内为 AT45DB 的片选，打到 ON 这边选中，打到另外一边，未选中，在用 SAM-BA 下载的时候要使用到该跳线。

注意：不同批次的核心板上的拨位开关的焊接反向有可能不一样，您只要注意 ON 的那边是选中即可，不管如何焊接，只要认准 ON 的位置即可。

三、AT91SAM9261 核心板原理图分析

接下来我们来看一下各个模块的原理图。

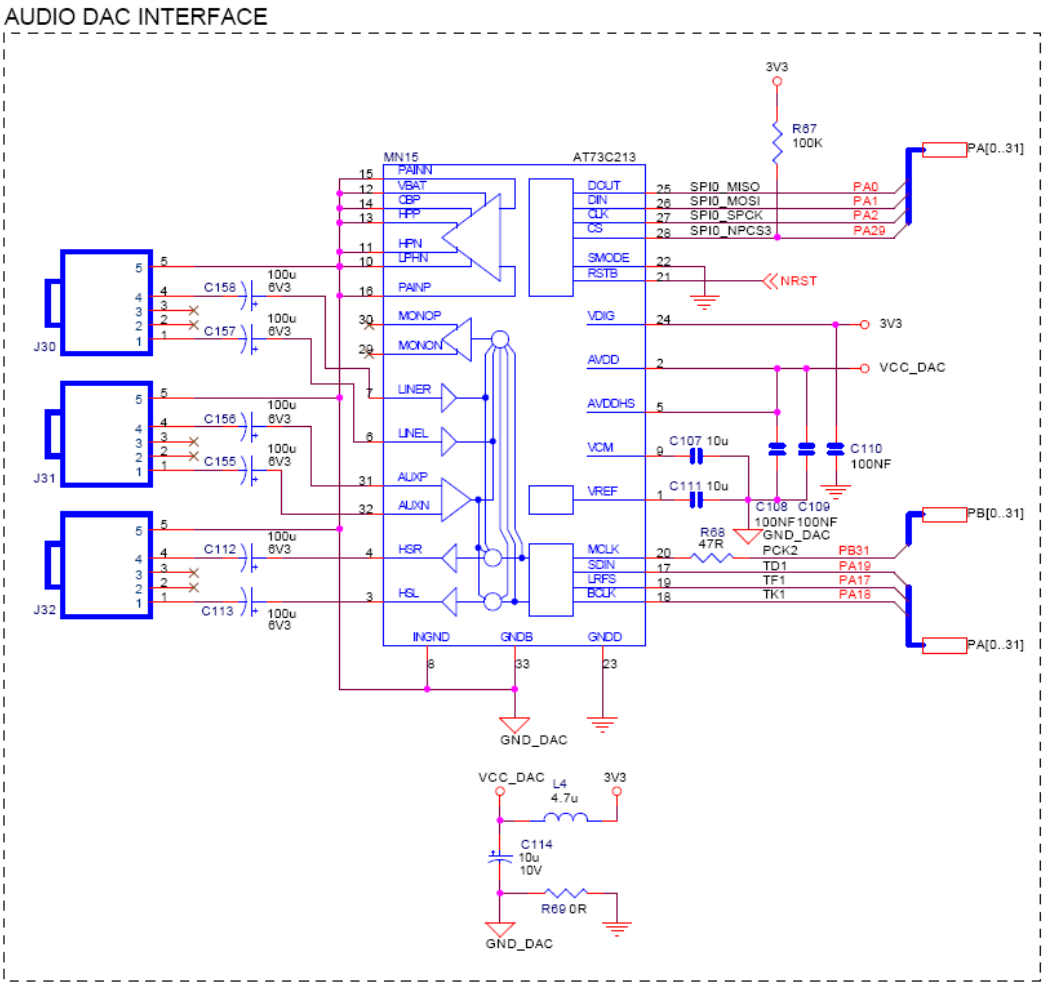
3.1 电源电路：



ATMEL 原装 EK 的电源电路采用的是 LT1963AEQ-3.3 来提供 3.3V 电源，TPS60500 来产生 1.2V 电源。考虑到兼容性和方便器件采购，同时 3.3V 和 1.2 只需要给核心板部分电路供电，所以我们采用了 SOT23-5 封装的 SP6205-3.3 和 SPX3819-1.2 分别用来产生 3.3V 和 1.2V，同时 CR2 直接用 0 欧姆电阻，由于使用了 SPX3819-1.2 固定电压版本，所以 R6,R8 也不必使用，这两片 LDO 都有使能脚，可以通过该使能脚来管理核心板电源控制，同时 9261 的 SHDN 引脚和 WKUP 引脚也都引出，用户可以根据实际需要组建一个完善的电源管理系统，更好的控制整个系统的功耗。

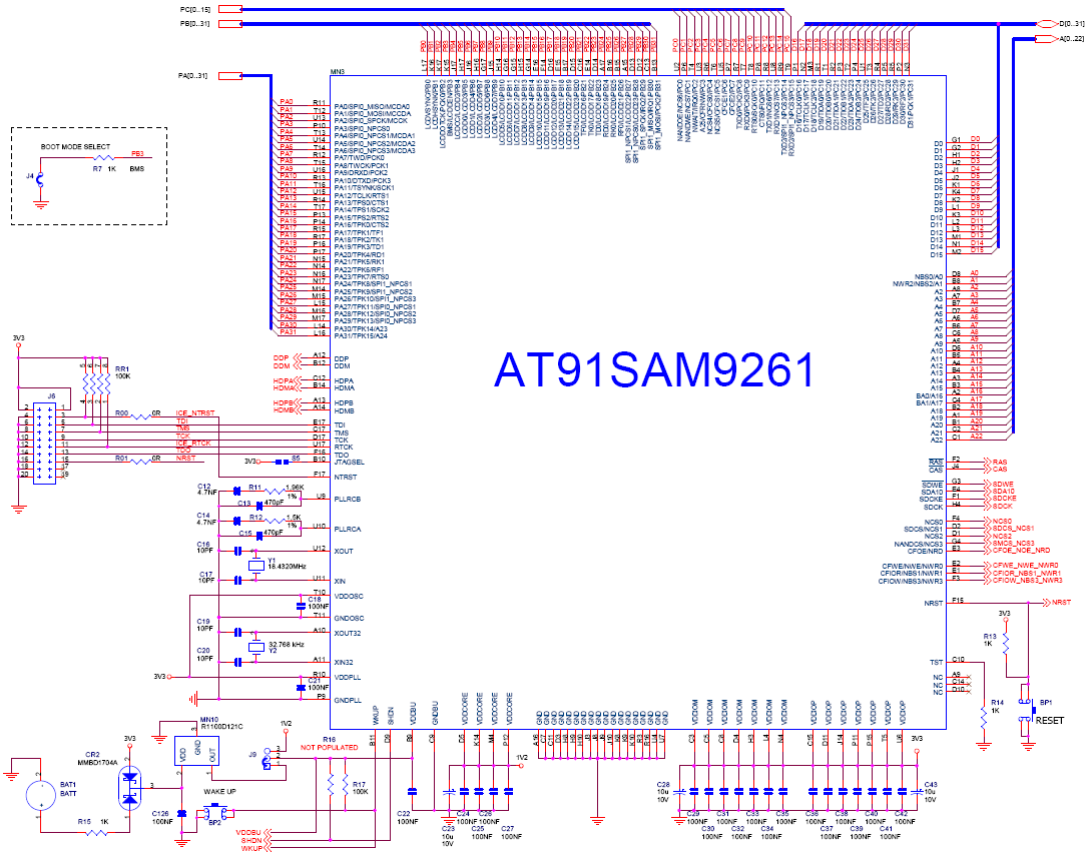
实际使用过程中，3.3V 电源指示灯是常亮的。

3.2 音频电路（位于配套盖板）：



使用的是 ATMEL 生产的高性能音频 DAC 芯片 AT73C213。2 路输入 1 路输出。

3.3 CPU 电路



我们主要要了解几个比较重要的引脚。

首先是 BMS 引脚，与 PB3 复用，该引脚用来选择启动方式，当复位的时候，如果检测到该引脚为低电平，则从外部外部总线接口片选 0 存储器启动；如果检测到该引脚为高电平，则从内部 ROM 启动。

BMS=1，从内嵌的 ROM 启动

系统用启动程序（Boot ROM）启动。

· DataFlash 启动

-从 SPI DataFlash 进入到内部 SRAM 下载并运行一个应用程序

-从 SPI DataFlash 调入的代码大小受 SRAM 大小限制

-自动监测有效的应用程序

-SPI DataFlash 连接到 SPI NPCSO

· 万一在外部 SPI DataFlash 检测到无有效的应用程序则启动上传

-SAM-BA 应用程序，一种小型监控程序(读/写/运行)接口

-自动检测通信连接

一个 DBGU（XModem 协议）上的串行通信

USB 设备端口(CDC 协议)

BMS=0,从外部存储器启动

· 慢时钟(32,768Hz)启动

· 带缺省配置的静态存储控制器启动，字节选择模式，16 位数据总线，

片选控制的读/写，允许在 16 位非易失存储器上启动。

客户编程的软件必须完成了一个完整的配置。

当在 32kHz EBI CS=0(BMS=0)时，为了加速启动次序，用户必须执行以下步骤：

1. 编程 PMC（主振荡器使能或旁路模式）。
2. 编程并启动 PLL。
3. 重编程 SMC 设置，周期，保持，CS0 的模式定时寄存器以适应新时钟。
4. 主时钟切换至新值

第二个需要注意的引脚是 JTAGSEL 引脚，当 JTAGSEL 引脚维持高电平(接到 VDDBU)时被用作 JTAG 边界扫描。此引脚集成了一个连接于 GNDBU 的 15K 欧姆的下拉电阻，所以在正常运行时可以悬空。注意，如果要使用仿真器调试，JTAGSEL 引脚必须为低电平或者悬空！

第三个需要注意的引脚是 TST 引脚，当测试引脚(TST)维持高电平(有效)时被用作生产测试目的。此引脚集成了一个连接于 GNDBU 的 15K 欧姆的永久上拉电阻，所以正常运行时可以悬空。当以高电平驱动此引脚时将导致难以预料的结果。

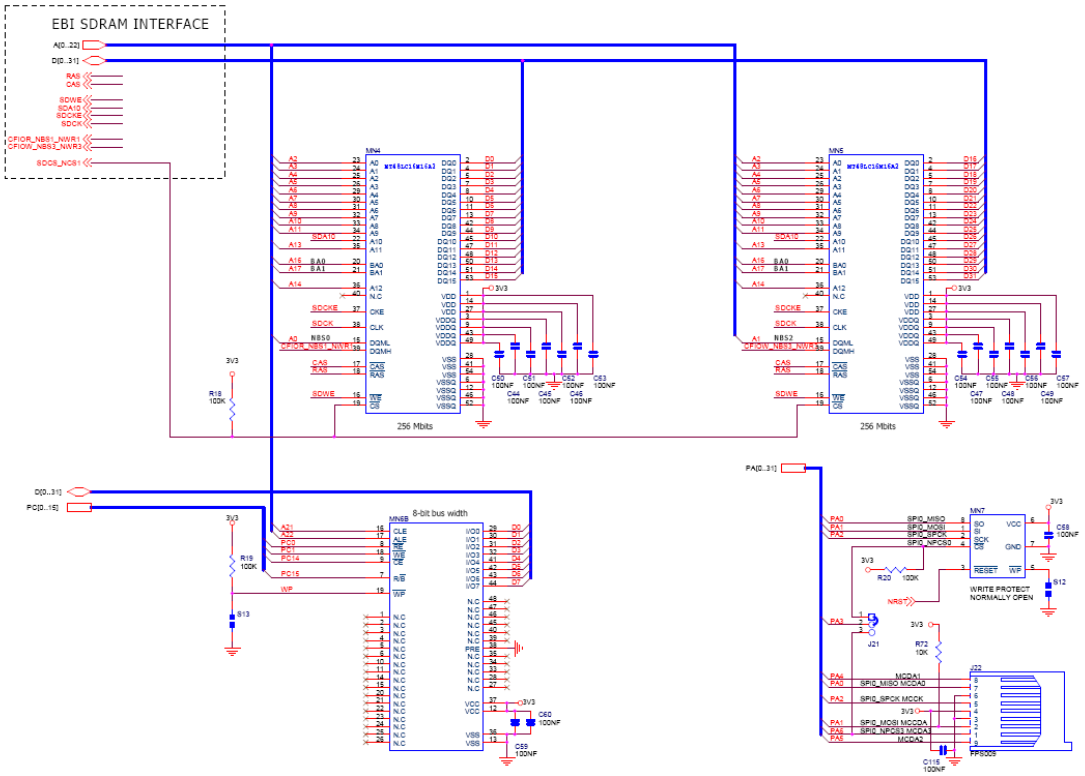
第四个需要注意的引脚是 NRST 引脚，NRST 是一个集成了一个不可编程的上拉电阻的开漏输出。输出电压可以达到 VDDIOP。由于 9261 集成了上电复位单元，当没有外部复位电路的情况下，NRST 引脚也可以直接悬空处理。NRST 引脚集成了一个至少为 100K 欧姆的永久上拉电阻并连接到 VDDIOP。NRST 信号嵌于边界扫描中。

第五个需要注意的引脚是 VDDBU 引脚，VDDBU 引脚是慢速时钟振荡器和部分系统控制器电源；电压范围 1.08V-1.32V, 额定值 1.2V。这里我们直接将 VDDBU 引脚和 VDDCORE 连接在一起，一同连接到 1.2V 电源。如果希望 CPU 能掉电保存 RAM 数据，VDDBU 可以由后备电池供电。

最后了解一下 AT91SAM9261 电源供应情况，AT91SAM9261 有 6 种类型的电源引脚：

- VDDCORE 引脚：核心电源，包括处理器，存储器和外设；电压范围从 1.08V 到 1.32V，额定值 1.2V
- VDDIOM 引脚：外部总线接口 I/O 口线电源；电压范围从 1.65V-1.95V 和 3.0V-3.6V, 额定值 1.8V-3.3V
- VDDIOP 引脚: 外设 I/O 口线和 USB 收发器电源;电压范围 2.7V-3.6V, 额定值 3.3V
- VDDBU 引脚：慢速时钟振荡器和部分系统控制器电源；电压范围 1.08V-1.32V, 额定值 1.2V
- VDDPLL 引脚：PLL 部件电源；电压范围 3.0V-3.6V, 额定值 3.3V
- VDDOSC 引脚：主振荡器部件电源；电压范围 3.0V-3.6V,额定值 3.3V

3.4 存储器电路



由两片 32MB 的 SDRAM 组成 32 位宽，64MB 的 SDRAM 空间。SDRAM 可以是 K4S561632 也可以是 HY57V561620。

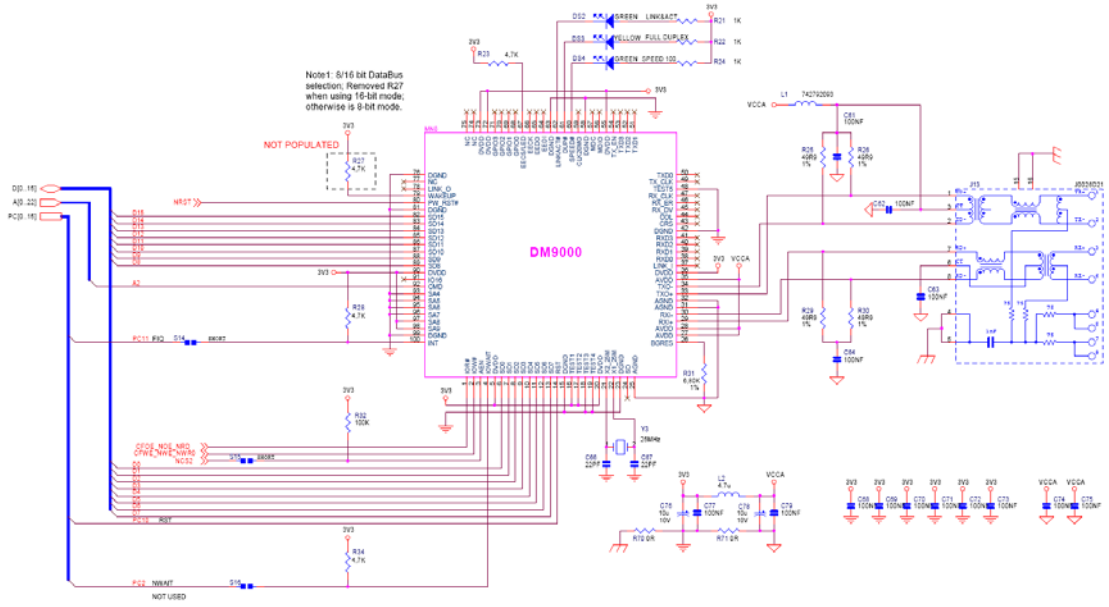
NAND FLASH 采用 8 位总线，使用的是 SAMSUNG 的 128MB NAND FLASH，型号 K9F1G08，用户也可以按照实际使用情况进行更换。比如更换成 64MB 的 K9F1208 以节约成本，或更换成 256MB 的 K9F2G08 以增加存储空间。

ATMEL 的 AT91SAM9 全系列都支持 DATAFLASH 启动，AT91SAM9261 也不例外，DATAFLASH 主要优势是引脚少，使用可靠，寿命较长，但是单位存储价格也相对较高。这里我们使用的是 4MB 的 AT45DB321，我们板子上面采用的是多封装设计，既可以支持 CANSON 8 封装，也可以支持 SOIC-8 封装，还可以只是 TSSOP 28 封装，用户也可以按照实际使用需要更换成 8MB 的 AT45DB642。

AT91SAM9261 具备多媒体卡接口 MCI，可以直接支持 SD 卡和 MMC 卡，所以 EK 上设计了一个 SD 卡接口。可以接受 SD 卡，MMC 卡，DATAFLASH 卡。

注意，NAND FLASH 和 DATAFLASH 都有写保护引脚，使用的时候请注意写保护状态，默认为允许读写。

3.5 以太网电路（位于配套底板上）



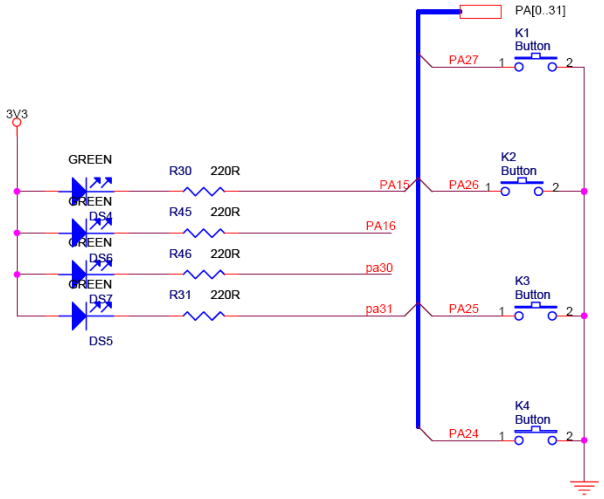
AT91SAM9261 并没有集成 MAC 电路，所以我们采用 DM9000 网络芯片，为了保证网络可靠运行，我们使用了集成网络变压器的 RJ45，HR911105A。

网络电路上有 3 个电阻跳线，客户在自行设计应用的时候请特别注意。

另外，请特别注意 R27，即 DM9000 的 WAKEUP 引脚，当使用 16 位数据总线时请移去 R27，即不要焊接 R27；如果使用 8 位数据总线，则请焊接 R27。

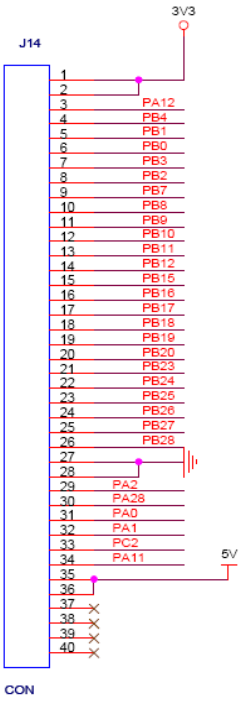
注：此手册原理上对所有的公板 9261 均适用，但是不同的板子上，可能标号未必均为 R27，在此主要是提醒用户注意 WAKEUP 引脚的电平。请参考相应的原理图。

3.6 用户接口（位于配套底板上）



4 个 LED，4 个按键。**与原装不同，使用时请注意。**

3.7 LCD 接口（位于配套底板上）



采用 18 位数字液晶接口。LCD 标配为 TFT 240×320 统宝 3.5 寸液晶屏。
注意：实际 wince 下配置的是 16 位接口

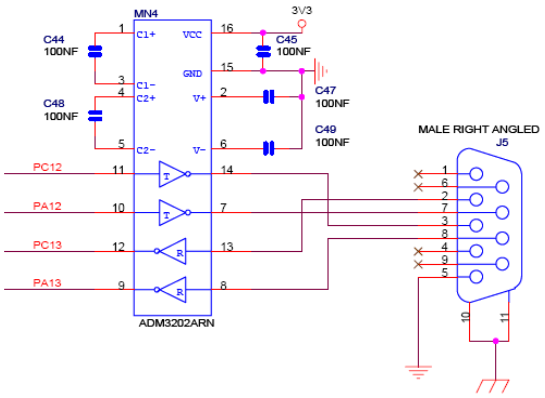
LCD 接口使用了 3.3V 和 5V 两种电源，请注意 AT91SAM9261 核心板的输入电源电压，为了保证板上器件的安全，请提供严格的 5V 稳压电源，不然对 USB HOST 上的 U 盘等设备、LCD 屏幕等使用 5V 的设备可能产生毁灭性影响。

3.8 串口电路

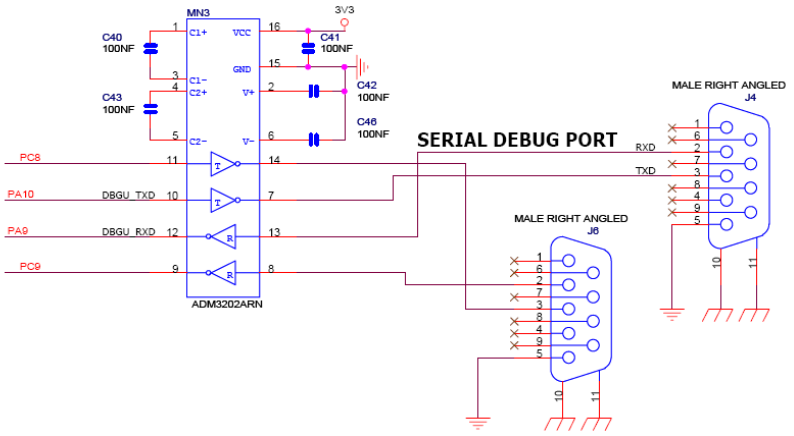
AT91SAM9261 核心板上引出了绝大部分的 GPIO，用户可以按照实际需要来使用串口和调试串口。

以下是 VC9261-EK 的串口电路，用户有需要可以参考。

4 线串口电路图：

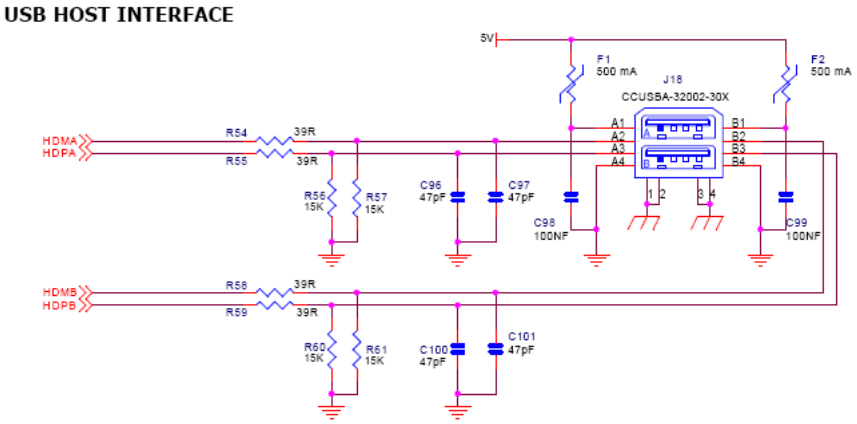


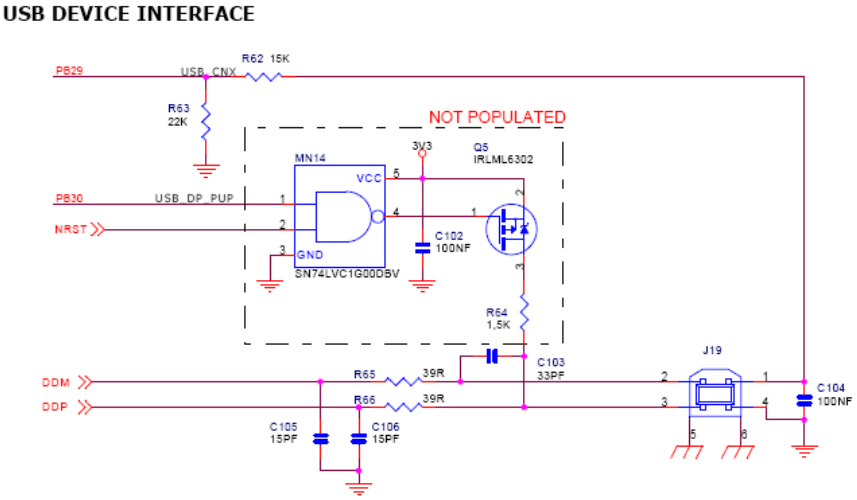
2 线 DBGU 调试串口和普通串口：



3.9 USB 电路（位于配套底板上）

AT91SAM9261 具备两个 USB HOST，1 个 USB DEVICE。





PB29 用来检测 USB 设备连接情况，PB30 用来控制 1.5K 上拉电阻。虚线框内的器件并没有焊接，1.5K 上拉电阻由 9261 芯片内部提供。

第三章：LCD 测试

AT91SAM9261 具备高级 LCD 控制器，这使得 9261 在图形应用中具备很高的性价比，特别是在手持设备应用，显示部件更是必不可少的。我们推荐大家在使用 AT91SAM9261-EK 的时候配备一个 3.5 寸的 240×320 的竖屏 TFT LCD。

一、AT91SAM9261 的 LCD 控制器介绍

- 支持被动或主动显示
- 在 STN 彩色模式下达 16 位深每像素
- 在 TFT 模式下达 16 M 色（24 位深每像素），分辨率高达 2048*2048

AT91SAM9261 是一个为带 LCD 显示应用而优化了的主机处理器。它的集成 LCD 控制器支持 BW 而且达到 16M 色彩，主动和被动 LCD 显示。160K 字节的片上 SRAM 可配置为帧缓冲，能将 LCD 刷新对整体处理器性能上的影响减到最小。

二、3.5 寸 240×320 竖屏 TFT LCD 介绍

本 3.5 寸（8.89cm）LCD 模块为主动矩阵彩色 TFT LCD 模块。具备低温显示技术，并且内建水平垂直驱动电路。该模块高度整合，包含了触摸屏，背光，并且仅需少量外设即可应用。

主要参数：

Item		Description	Unit
Display Size (Diagonal)		3.5 inch (8.9cm)	-
Display Type		Transflective	-
Active Area (HxV)		53.64 x 71.52	mm
Number of Dots (HxV)		240 x RGB x 320	dot
Dot Pitch (HxV)		0.0745 x 0.2235	mm
Color Arrangement		RGB Stripe	-
Color Numbers		262,144 (6 bits)	-
Outline Dimension (HxVxT)		64 x 85 x 4.05 (Max 4.9)*	mm
Weight		42	g
Power consumption	LCD Panel + T-CON + L/S	50 (Typ)	mW
	Backlight	432 (Typ, I_F= 20mA)	

* Exclude FPC and protrusions.

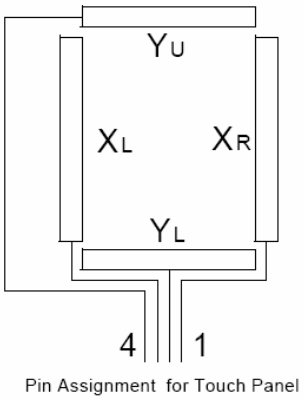
FPC 引脚定义：

Pin	Symbol	I/O	Description	Remark
1	VDD	-	Supply voltage for H/V driver (+ 12V)	
2	DE	I	Data enable	
3	NC	-	No connection (leave this pin open)	
4	NC	-	No connection (leave this pin open)	
5	GND	-	Ground	
6	VEE	-	Supply voltage for V driver (-6.5V)	
7	GND	-	Ground	
8	NC	-	No connection (leave this pin open)	
9	R0	I	Video data red 0 (LSB)	
10	R1	I	Video data red 1	
11	R2	I	Video data red 2	
12	R3	I	Video data red 3	
13	R4	I	Video data red 4	
14	R5	I	Video data red 5 (MSB)	
15	G0	I	Video data green 0 (LSB)	
16	G1	I	Video data green 1	
17	G2	I	Video data green 2	
18	G3	I	Video data green 3	
19	G4	I	Video data green 4	
20	G5	I	Video data green 5 (MSB)	
21	B0	I	Video data blue 0 (LSB)	
22	B1	I	Video data blue 1	
23	B2	I	Video data blue 2	
24	B3	I	Video data blue 3	
25	B4	I	Video data blue 4	
26	B5	I	Video data blue 5 (MSB)	
27	NC	-	No connection (leave this pin open)	
28	NC	-	No connection (leave this pin open)	
29	NC	-	No connection (leave this pin open)	
30	CLK	I	Video data clock	
31	NC	-	No connection (leave this pin open)	
32	NC	-	No connection (leave this pin open)	
33	VCC5	-	Supply voltage for 5V logic	

34	VCC5	-	Supply voltage for 5V logic	
35	NC	-	No connection (leave this pin open)	
36	NC	-	No connection (leave this pin open)	
37	VCC3	-	Supply voltage for 3.3V logic	
38	VCC3	-	Supply voltage for 3.3V logic	
39	NC	-	No connection (leave this pin open)	
40	NC	-	No connection (leave this pin open)	
41	GND	-	Ground	
42	Anode R	-	LED Power Supply (+)	
43	Cathode R	-	LED Power Supply (-)	
44	Anode L	-	LED Power Supply (+)	
45	Cathode L	-	LED Power Supply (-)	
46	GND	-	Ground	
47	XR	-	Touch Panel Right Side Pin	
48	YL	-	Touch Panel Lower Side Pin	
49	XL	-	Touch Panel Left Side Pin	
50	YU	-	Touch Panel Upper Side Pin	

4 线触摸屏引脚定义：

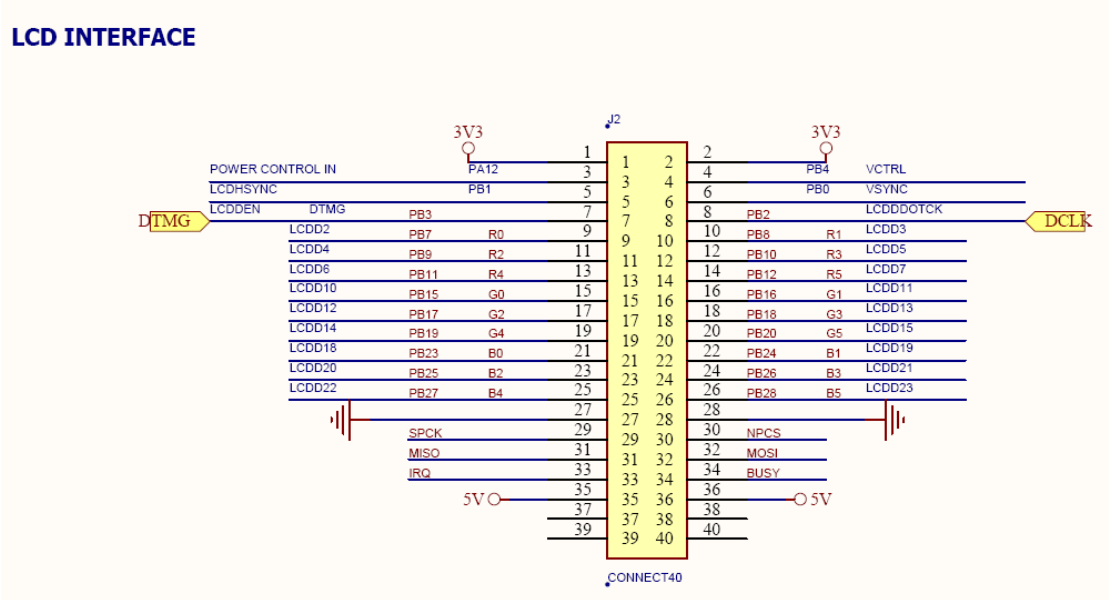
Touch Panel Pin	Module Pin	Symbol	Description	Remark
1	47	XR	Touch Panel Right Side	
2	48	YL	Touch Panel Lower Side	
3	49	XL	Touch Panel Left Side	
4	50	YU	Touch Panel Upper Side	



Pin Assignment for Touch Panel

LCD 屏为 54 芯 fpc 连接线，对于用户不必详细了解该 54 芯引脚定义。用户主要关注的应该是 LCD 显示模块和 9261-EK 的连接。

LCD 和 9261-EK 的连接引脚定义：

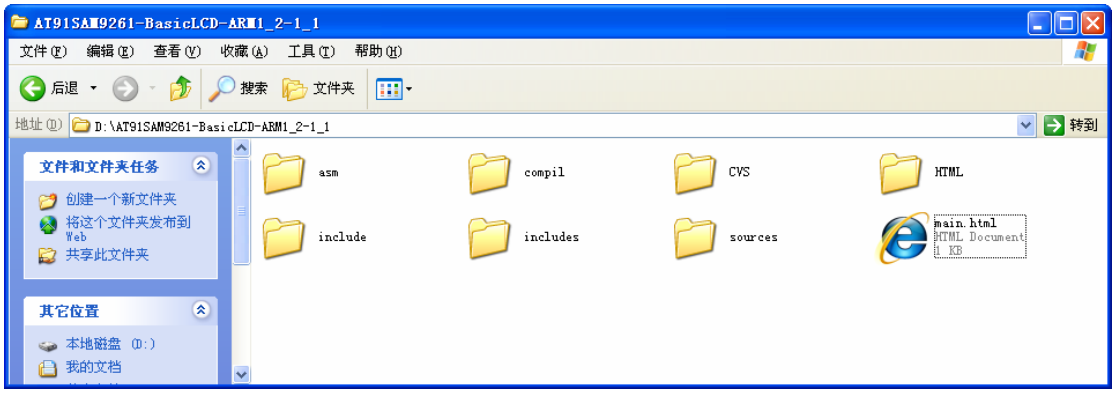


三、ADS 下测试 LCD

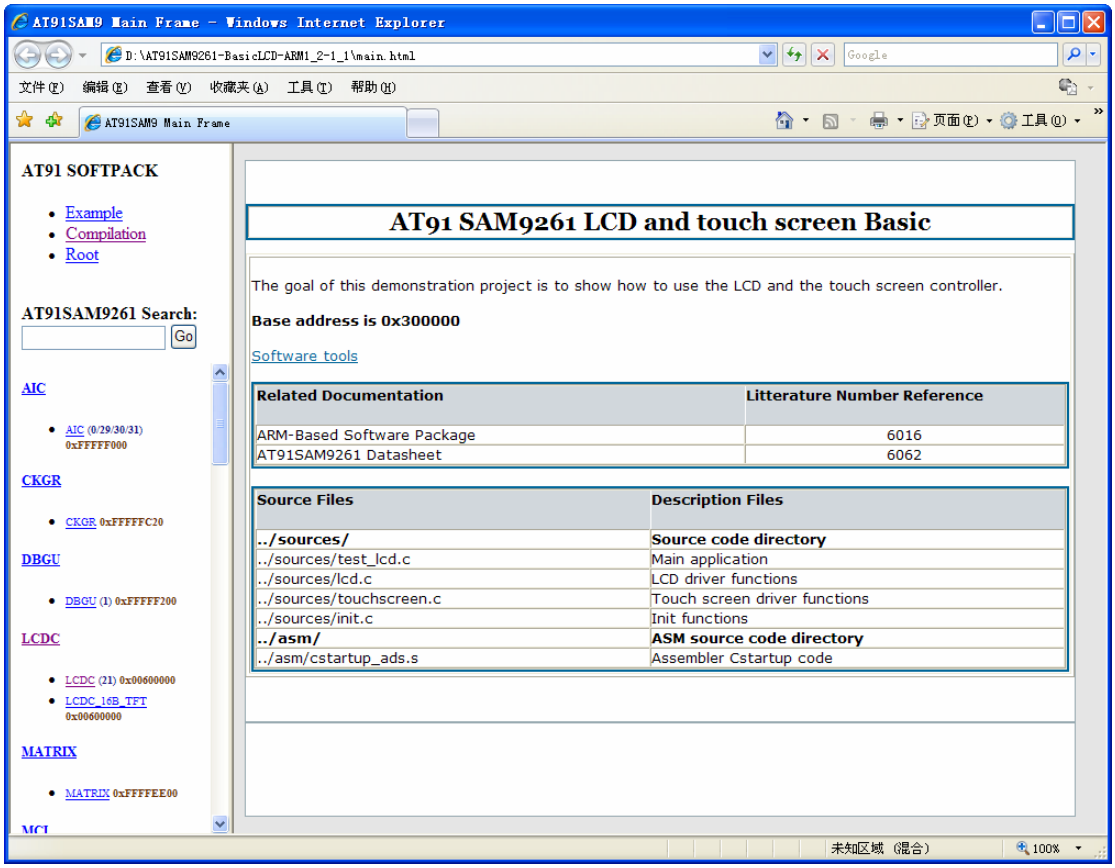
ATMEL 提供了一个单独的 LCD 测试程序，有 ADS 版本和 IAR 版本。这里我们以 ADS 为例。

请先安装 ADS（ADS 是 ARM 公司很早就推出的 ARM 调试环境，由于 ADS 支持的是针对内核，所以只要是 ARM7，ARM9 等内核的芯片，ADS 都可以支持），然后打开 LCD 测试程序：AT91SAM9261-BasicLCD-ARM1_2-1_1，该测试程序可以在 www.atmel.com 或者 www.ATARM.com，或者随板光盘里面找到。

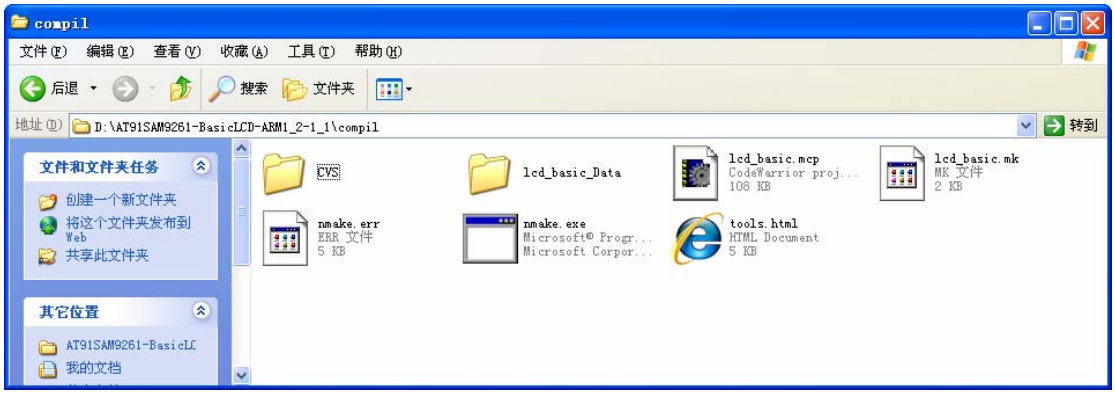
解压后可以在看到以下文件/文件夹：



Main.html 为资料文档，介绍了很多外设和 API，建议浏览。



双击进入 compile 文件夹：



Lcd_basic.mcp 为工程文件，可以直接用 ADS 打开；tools.html 为使用说明。
注意：Codewarrior 不支持中文路径，请注意。

以下为 tools.html 内容：

1 Cross compiling using ARM ADS 1.2

Software requirements:

ARM ADS v1.2

basic.zip file

Procedure :

Unzip the files to one of your hard disk drive.
Double-click on **basic.mcp** project file.
Once loaded with **CodeWarrior**, click on "Project" then "Make" or type "F7".
It should compile and create an **axf** and a **bin** file.

2 Cross compiling using ARM nmake

Software requirements:
ARM ADS v1.2
basic.zip file
nmake.exe

Procedure :
Unzip the files to one of your hard disk drive.
A default MCP project file is delivered with a makefile example **project_basic.mk** and with **nmake.exe**.
To compile with armcc and armasm type the command line :
nmake /f project_basic.mk
It should compile and create a **bin** file.

Note that this makefile can easily be ported for another compiler chain.

3 Debug under AXD

Software requirements:
ARM ADS v1.2
basic.axf file

Hardware requirements:
Multi-ICE Emulator

Procedure :
Connect the **Multi-ICE** to the JTAG connector.
Power-up the board.
Start **Multi-ICE Server**.
Start **ARM AXD Debugger**.
If "RDI Warning 00201: Cannot open target: the target is not responding" error message occurs, from the "Fatal AXD Error" window, Click on "CONFIGURE", then from the "Target Environments" field,
select "Multi-ICE". If this option is not available, click "Add" and browse folders up to the ARM Multi ICE installation folder and select "Multi-ICE.dll" file.
Then click on "Configure" to configure the Multi-ICE. Once done, click OK to return to AXD.
Once done double click on **basic.axf** file, AXD should load the code in memory.

4 Loading code using "U-boot"

Software requirements:
U-boot
basic.bin file

Procedure :
Power-up the board with J4 closed.
If programmed in Dataflash U-boot should run.
Get the **base_address** field in the html file.
Load the code at this address with **tftp** or **loadb** command. (Refer to U-boot manual for more details.)
Type : **go "base_address"** (in hex without "0x" prefix), the code starts.

5 Loading code using "SAM-BA"

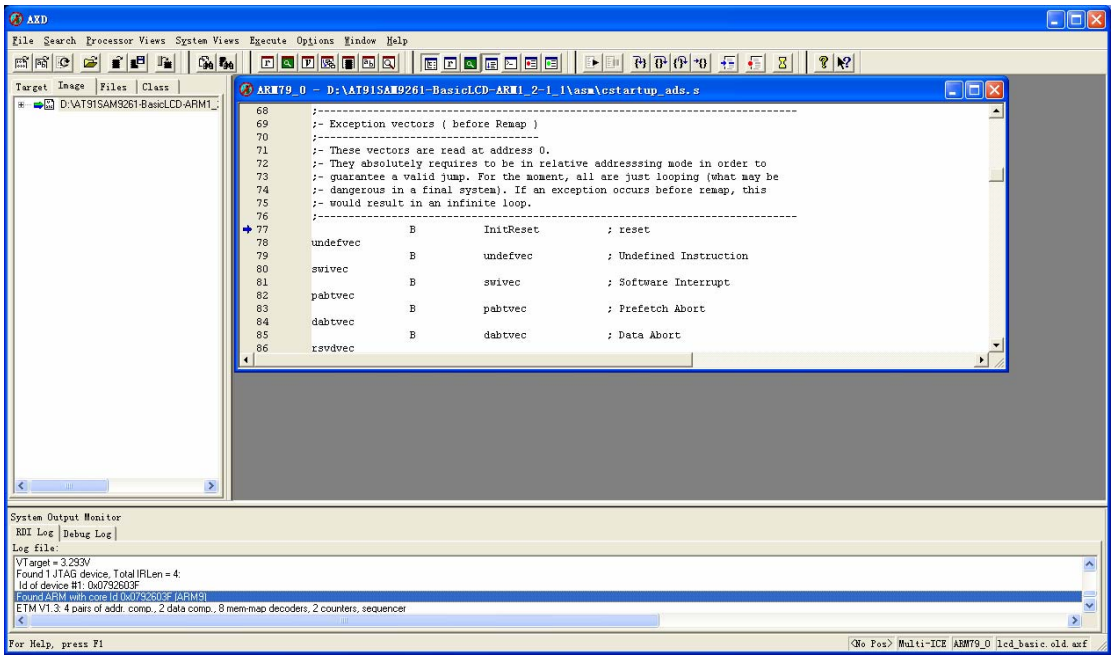
Software requirements:
SAM-BA software
basic.bin file

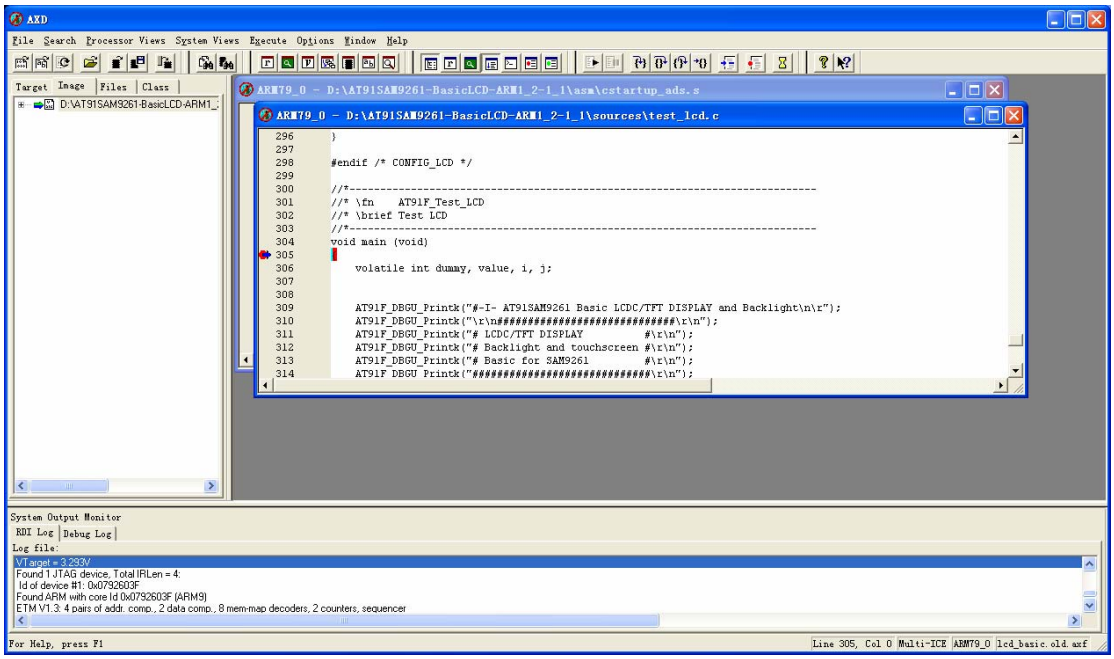
Procedure :
Reset the board with J4 open. This boots in ROM and screens **RomBOOT>** in the hyperterminal.
Launch **SAM-BA** and choose "USB connection". In **SAM-BA** window choose the memory corresponding to the **base_address**.
Especially for the SDRAM, it should be initialized executing the script "Enable SDRAM" in the Script list.
Browse the PC to find the correct **bin** file and enter the correct address below.
Once done, click on "Send File". This load the file into the memory.
In the window below type : **go "base_address"** (in hex with "0x" prefix)
Type "Yes" or "OK" each time it is required, the code starts.

该文档对几种调试方法说的比较详细。如果需要更详细的调试工具和编译环境的使用，可以查找相关使用手册和资料，这里不再详细说明。

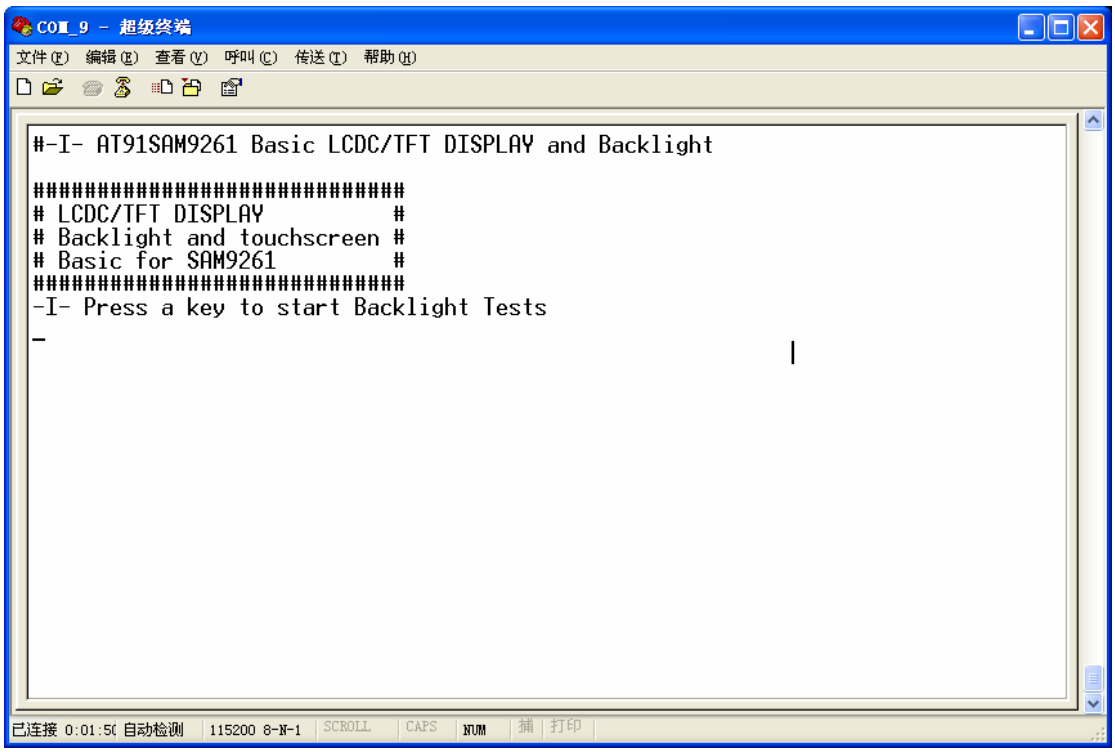
可以先用 Codewarrior 打开 mcp 文件，然后通过 project 菜单下的 debug 进入 axd 调试环境。或者直接打开 axd 调试环境，然后 load 之前生成的 axf 文件。

默认已经在 main 函数处设置了一个断点，进入 axd 可以直接点击一次 go 按钮，程序即停在 main 函数处。

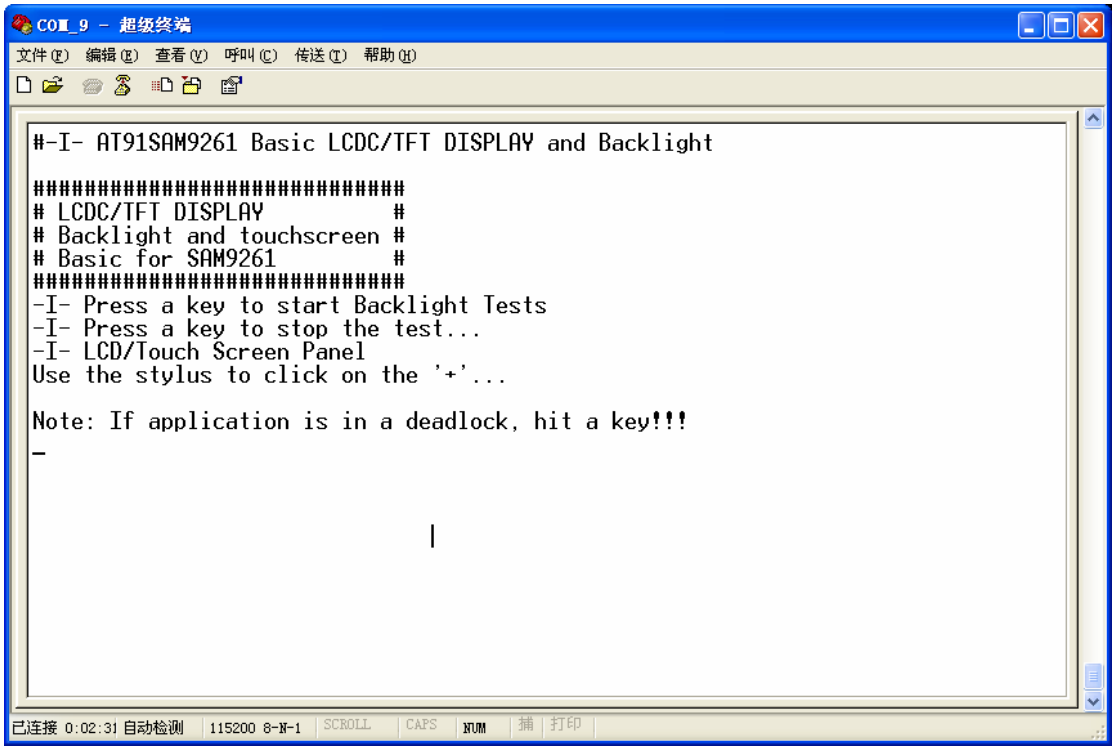




这个时候请连接好 DBGU 调试串口，并打开超级终端观察输出信息。
再次点击 go 后，超级终端输出以下信息：

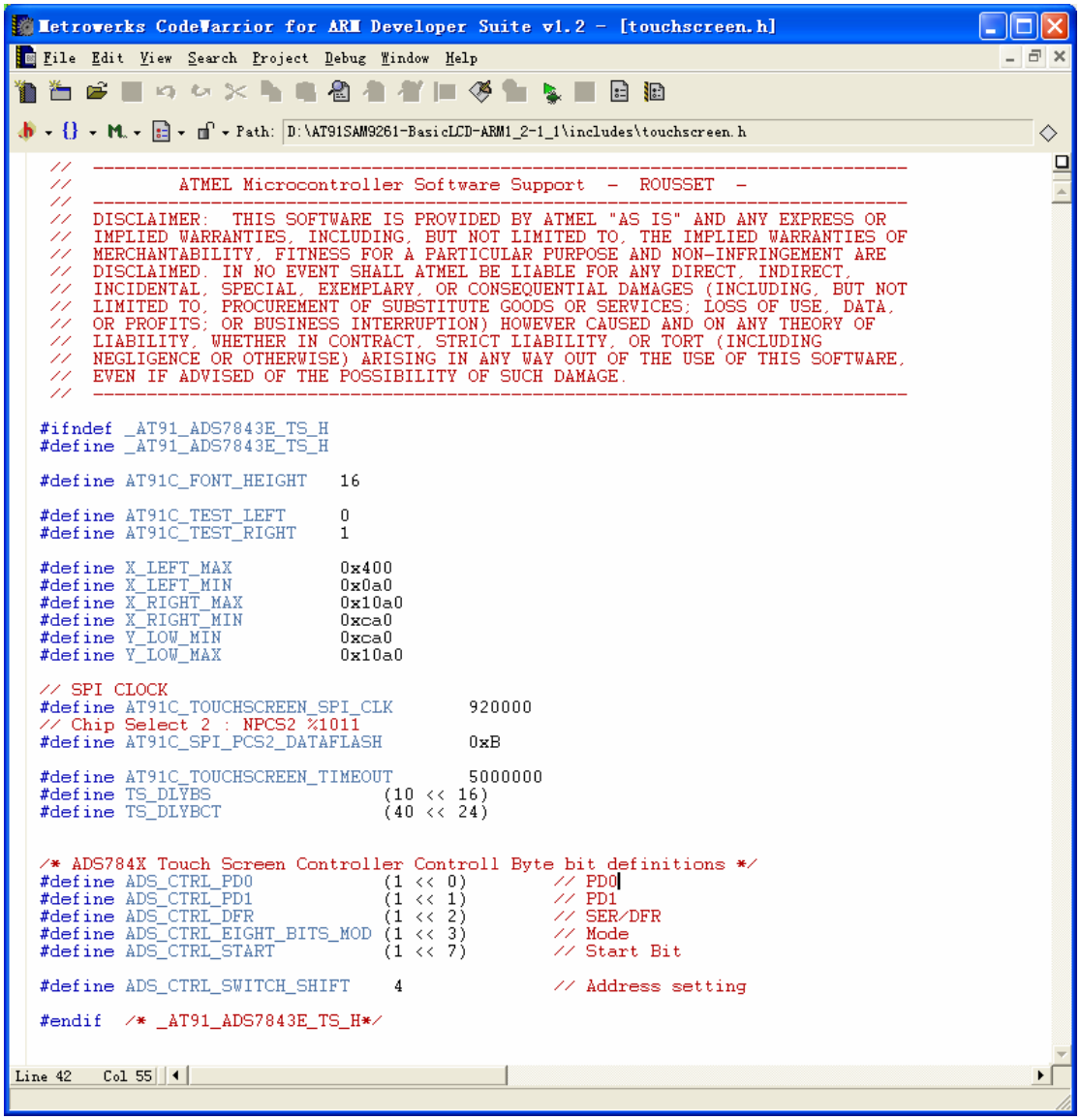


该 LCD 模块的背光强制点亮，背光测试可以通过按任意键直接跳过。



跳过背光测试后进入触摸屏校准，屏幕上会出现 TouchScreen Calibration。然后按照＋提示点选位置，如果位置出错，测试程序会提示还有 4 次重试机会，如果在这 4 次重试机会内正确点击指定位置，最终程序会提示 Test OK，不然会提示 Test Fail。

关于点击范围的精度可以在 touchscreen.h 里面修改。


Metrowerks CodeWarrior for ARM Developer Suite v1.2 - [touchscreen.h]
File Edit View Search Project Debug Window Help
Path: D:\AT91SAM9261-BasicLCD-ARM1_2-1_1\includes\touchscreen.h
// --
// ATMEL Microcontroller Software Support - ROUSSET -
// --
// DISCLAIMER: THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
// DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
// OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
// EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
// --
#ifndef _AT91_ADS7843E_TS_H
#define _AT91_ADS7843E_TS_H
#define AT91C_FONT_HEIGHT 16
#define AT91C_TEST_LEFT 0
#define AT91C_TEST_RIGHT 1
#define X_LEFT_MAX 0x400
#define X_LEFT_MIN 0x0a0
#define X_RIGHT_MAX 0x10a0
#define X_RIGHT_MIN 0xca0
#define Y_LOW_MIN 0xca0
#define Y_LOW_MAX 0x10a0
// SPI CLOCK
#define AT91C_TOUCHSCREEN_SPI_CLK 920000
// Chip Select 2 : NPCS2 %1011
#define AT91C_SPI_PCS2_DATAFLASH 0xB
#define AT91C_TOUCHSCREEN_TIMEOUT 5000000
#define TS_DLYBS (10 << 16)
#define TS_DLYBCT (40 << 24)
/* ADS784X Touch Screen Controller Controll Byte bit definitions */
#define ADS_CTRL_PD0 (1 << 0) // PD0
#define ADS_CTRL_PD1 (1 << 1) // PD1
#define ADS_CTRL_DFR (1 << 2) // SER/DFR
#define ADS_CTRL_EIGHT_BITS_MOD (1 << 3) // Mode
#define ADS_CTRL_START (1 << 7) // Start Bit
#define ADS_CTRL_SWITCH_SHIFT 4 // Address setting
#endif /* _AT91_ADS7843E_TS_H*/
Line 42 Col 55

第四章：网络测试

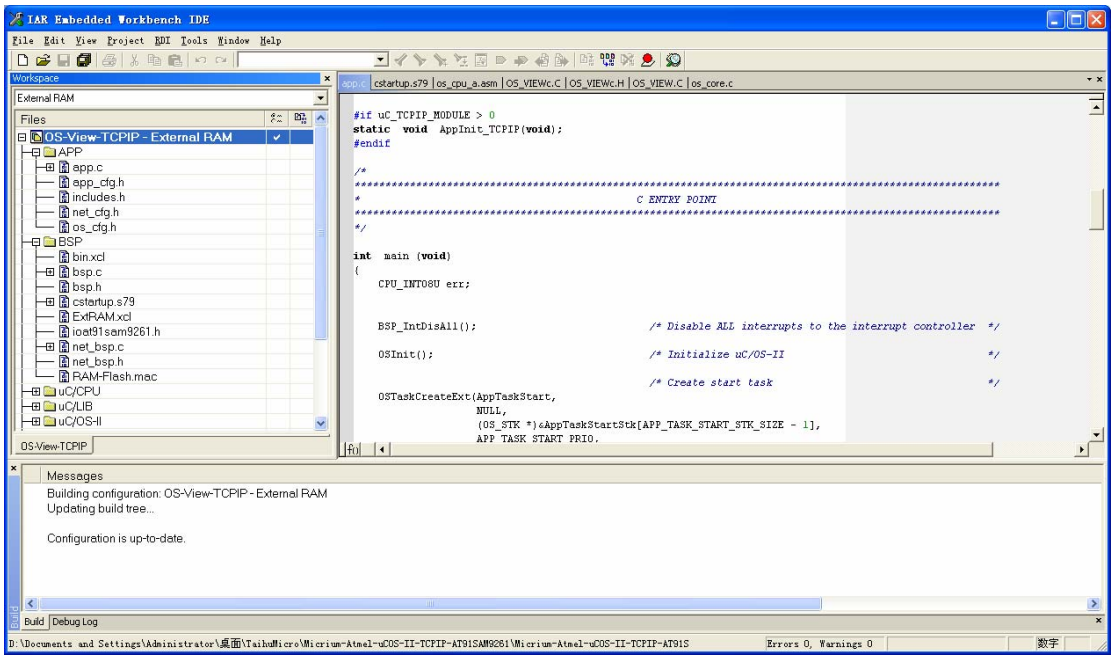
感谢 Micrium 公司为 ATMEL 的 AT91SAM9261-EK 提供 uC/TCP-IP 测试代码： Micrium-Atmel-uCOS-II-TCPIP-AT91SAM9261.exe， 该测试代码可以在 www.atarm.com 或者随板光盘里面找到。

下载后将该文件解压，在以下路径找到 IAR 的工程文件：

Micrium-Atmel-uCOS-II-TCPIP-AT91SAM9261\Micrium-Atmel-uCOS-II-TCPIP-AT91SAM9261\Micrium\Software\EvalBoards\Atmel\AT91SAM9261\IAR\OS-View-TCPIP，OS-View-TCPIP.ewp。

该测试代码是基于 IAR for ARM 4.41a 版本的，我们测试的时候使用的是 4.42a 版本，请按照要求安装相应版本的 IAR。

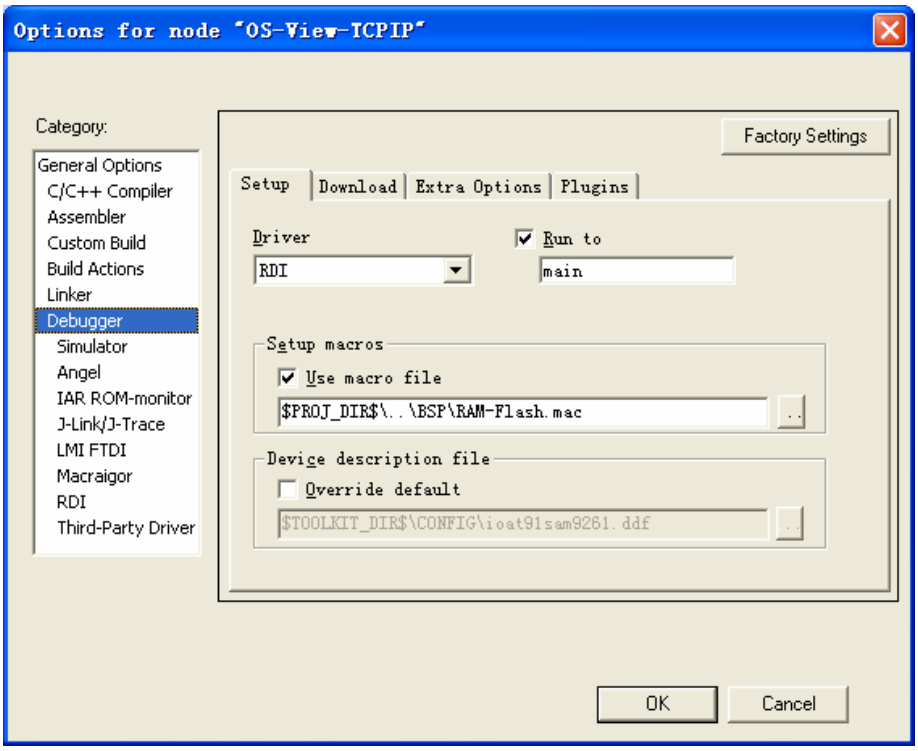
用 IAR 打开该工程，稍加设置即可：



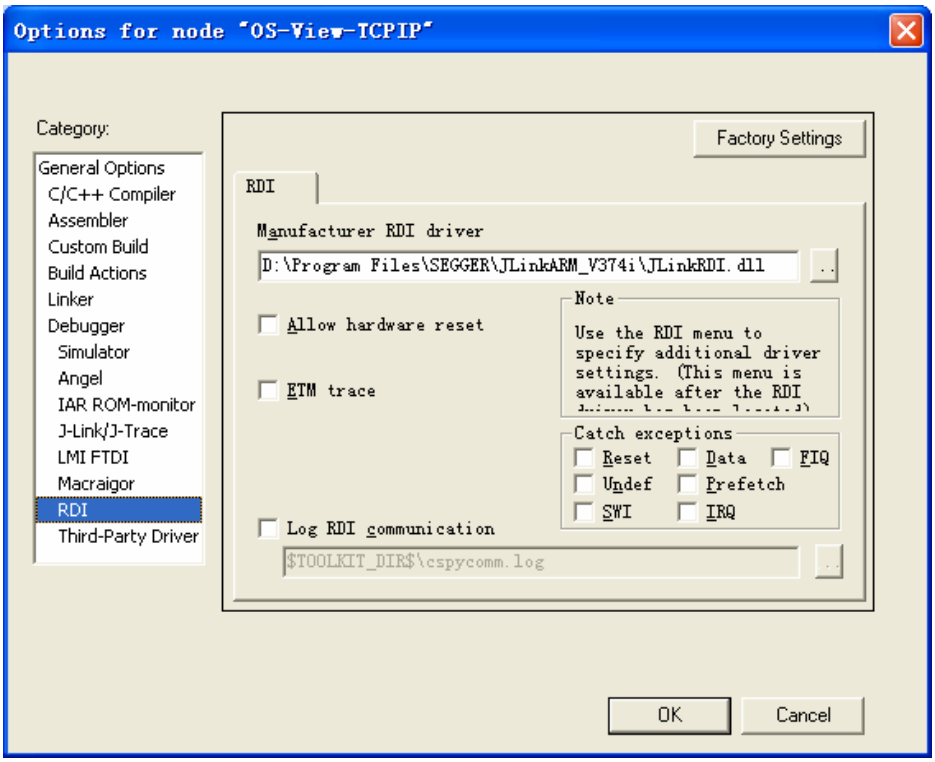
我们以 J-Link 为例简单说明以下设置情况，该范例编译后可以直接运行，默认的 IP 为 192.168.1.60，请注意设置网关、子网掩码和注意网线是否交叉。我们建议 PC 机和 AT91SAM9261-EK 均连接到一个路由器。

注意：在进入调试前，请将 AT91SAM9261-EK 上的 J21 跳线悬空并复位，以避免因 9261-EK 运行了 OS 而出现 IAR 控制不了 9261CPU 的情况！

打开工程设置页面：



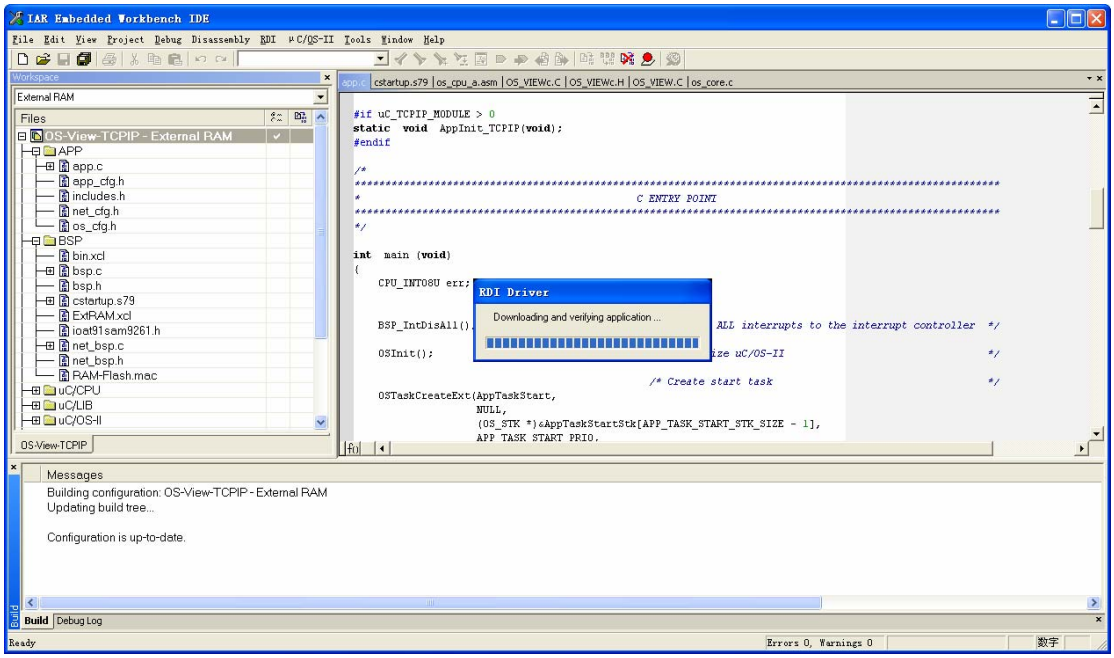
在 Driver 下拉菜单里面将 JLINK 改为 RDI，然后在 RDI 选项这边指定 DLL。

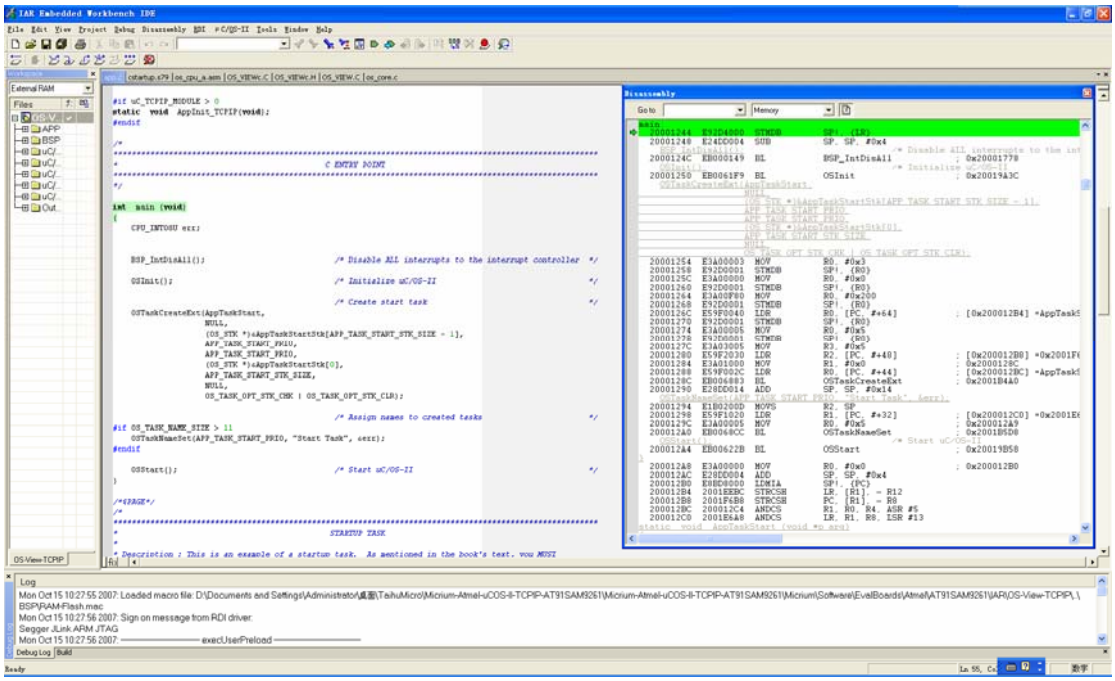


如果你使用别的仿真器，比如 Multi-ICE、wiggler 等，也需要使用 RDI 接口

设置。

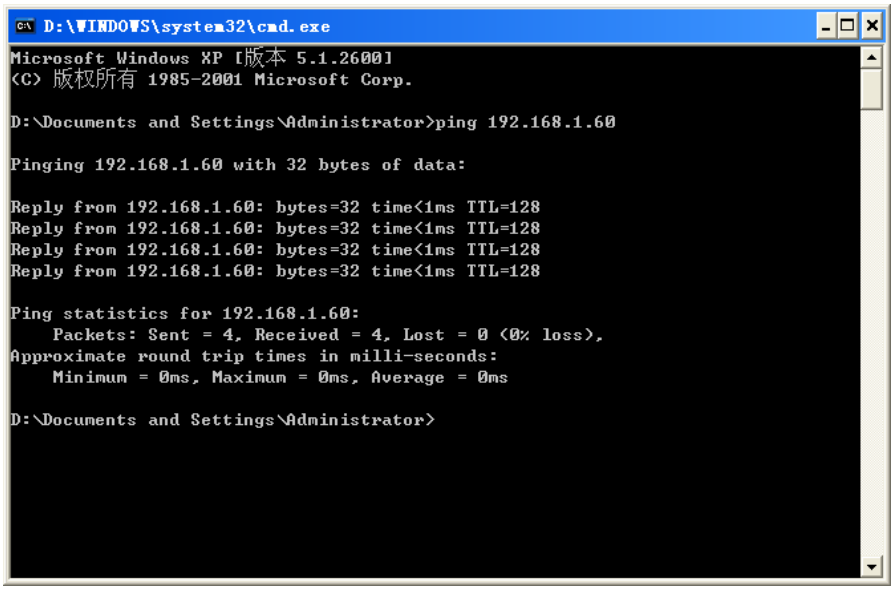
设置完成后，点击 Project 菜单下的 debug，IAR 开始编译，然后进入调试状态：





点击 GO，全速运行。

然后在 PC 端打开 CMD 窗口，运行 ping 命令，可以看到 AT91SAM9261-EK 的响应：



程序运行后 9261-EK 板上 DS7 和 DS8 两盏 LED 不停闪烁，同时，当运行 ping 命令时，RJ-45 边上的 DS2 会有闪烁。

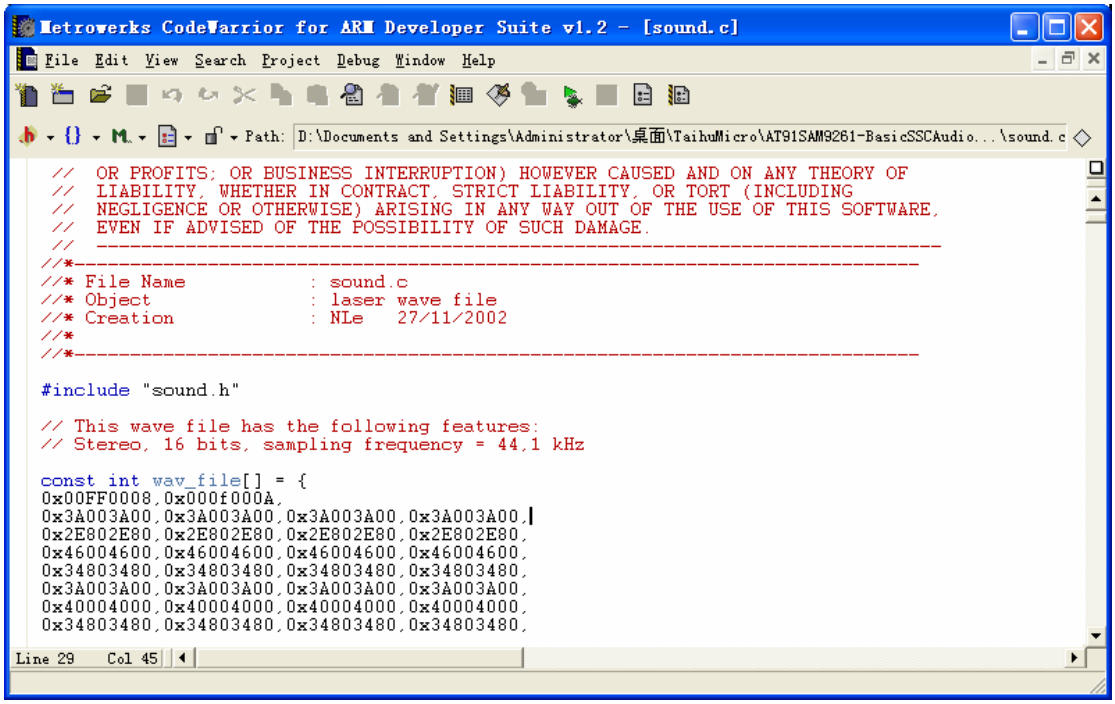
更详细的内容可以参考测试程序的源文件和测试程序自带的 AN-3261.pdf。

第五章：音频测试

AT91SAM9261-EK 带了一个基于 ATMEL 的 AT73C213 的音频 DAC，有了该 DAC，可以轻松组建一套音频，视频系统。这在 wince 上已经测试过了。这里我们跑一下 ATMEL 提供的音频测试程序（AT73C213 位于配套底板或者盖板）：

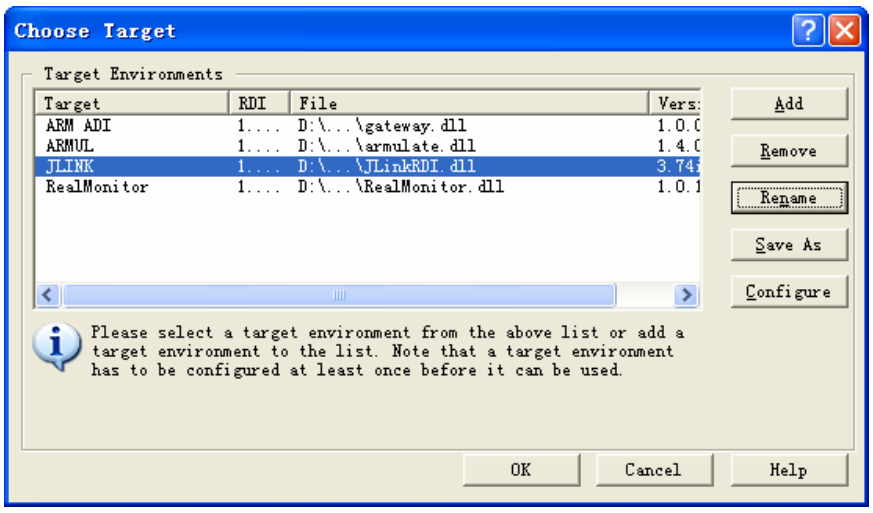
AT91SAM9261-BasicSSCAudio-ARM1_2-1_1

该测试程序是驱动 AT73C213 输出“laser wave”声音，感觉就是电影里面的激光枪的音调。测试的时候请注意保护耳朵！输出为 16 位立体声，44.1KHz 采样率。

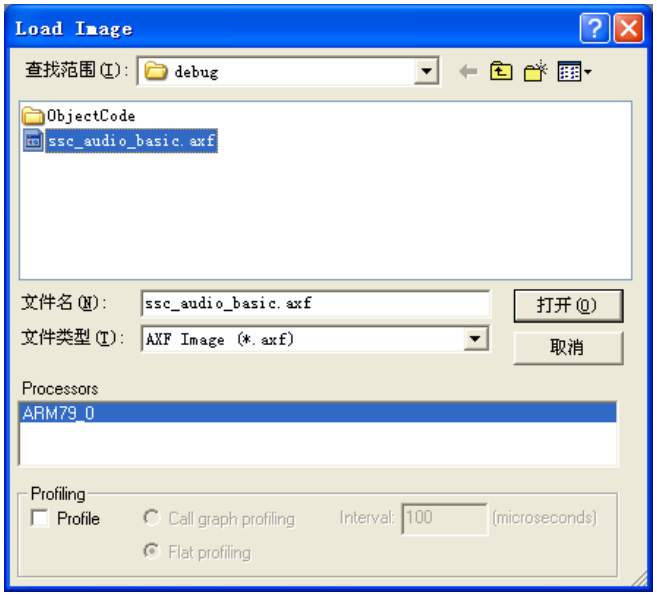


首先用 CodeWarrior 打开 MCP 工程文件，然后按 project 下的 debug 进入 AXD，也可以直接打开 AXD，然后调入 axf 文件。

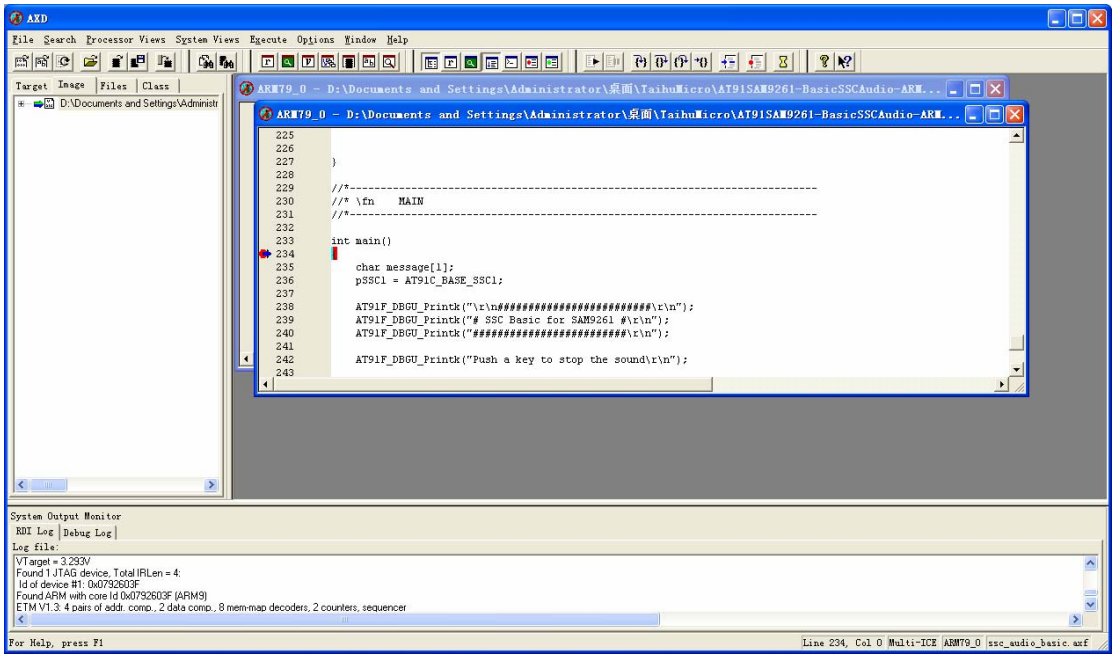
调试这个代码我们使用 JLINK，如果你使用的是别的仿真器，可以在下图所示设置页面通过 add 按钮添加：



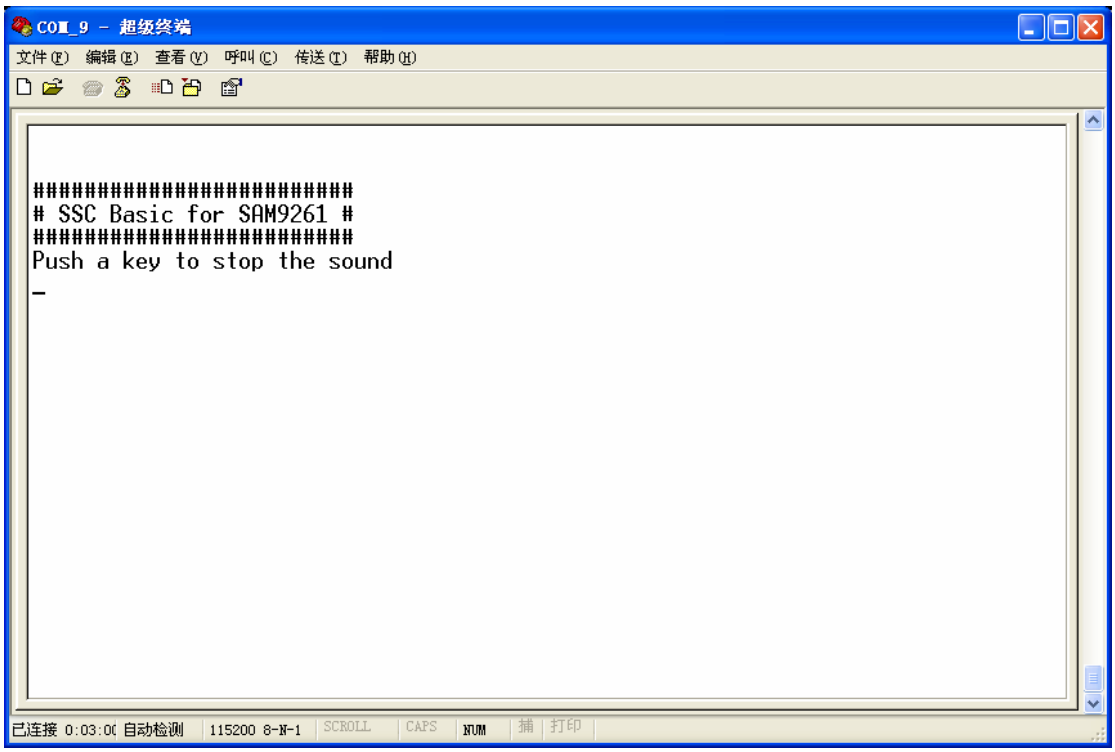
调入 axf 文件：



点击运行，程序会停在 main 函数，再点一下运行，程序即可全速运行：



打开超级终端观察输出信息：



在超级终端随意敲入一个键后中止测试。

第六章：ADS 下 9261 调试方法

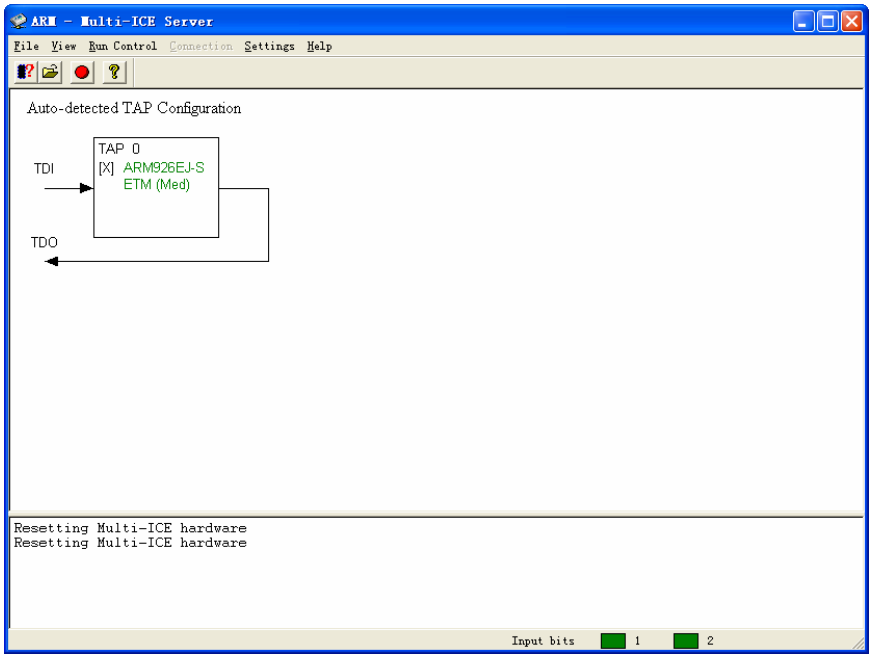
Multi-ICE 是 ARM 公司官方提供的调试工具，随着 ARM 的推广，国内很多公司推出了兼容型的 Multi-ICE 仿真器，本站就推出了并口 Multi-ICE 兼容型仿真器和 USB 接口 Multi-ICE 兼容型仿真器。下面简述一下用 Multi-ICE 调试 9200 的过程。

一、Multi-ICE 的设置：

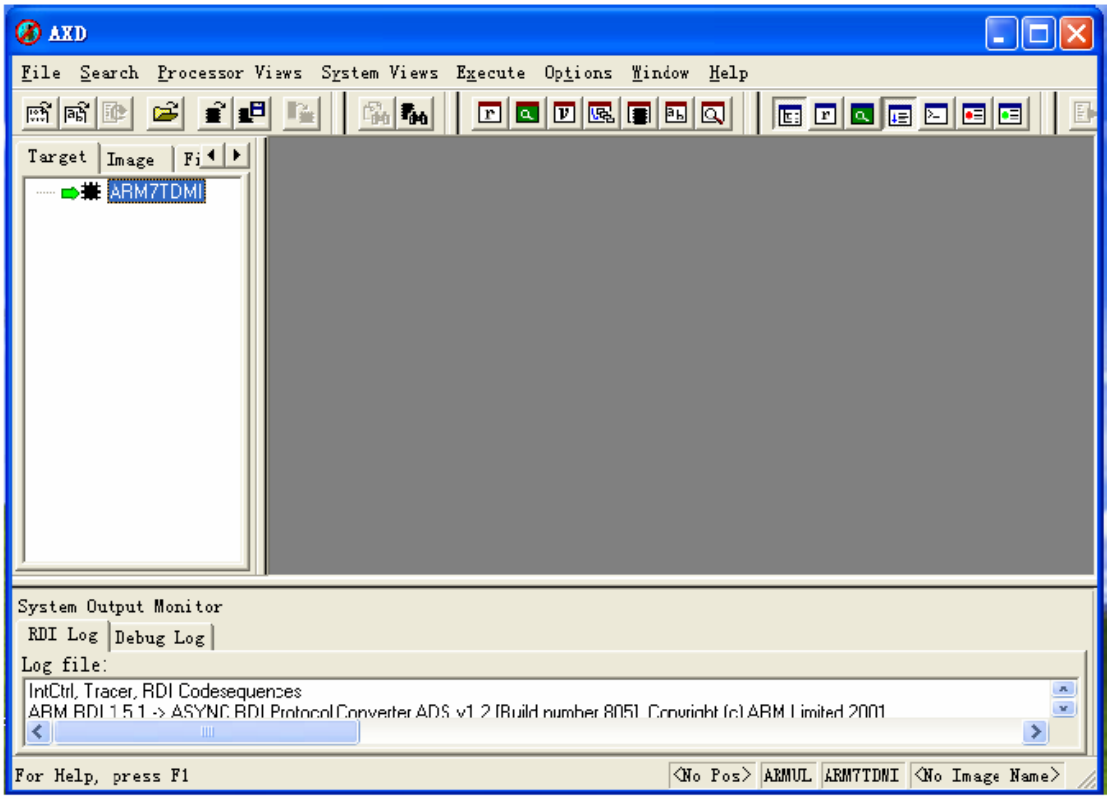
首先是 ADS 开发环境，关于 ADS 的介绍、安装、使用请自行查阅相关资料。

进行调试前请先把仿真器和 9261-EK 通过 20 芯线连接起来。如果使用并口版本的 Multi-ICE，请注意检查一下你的并口设置，建议设置成 EPP+ECP 模式。

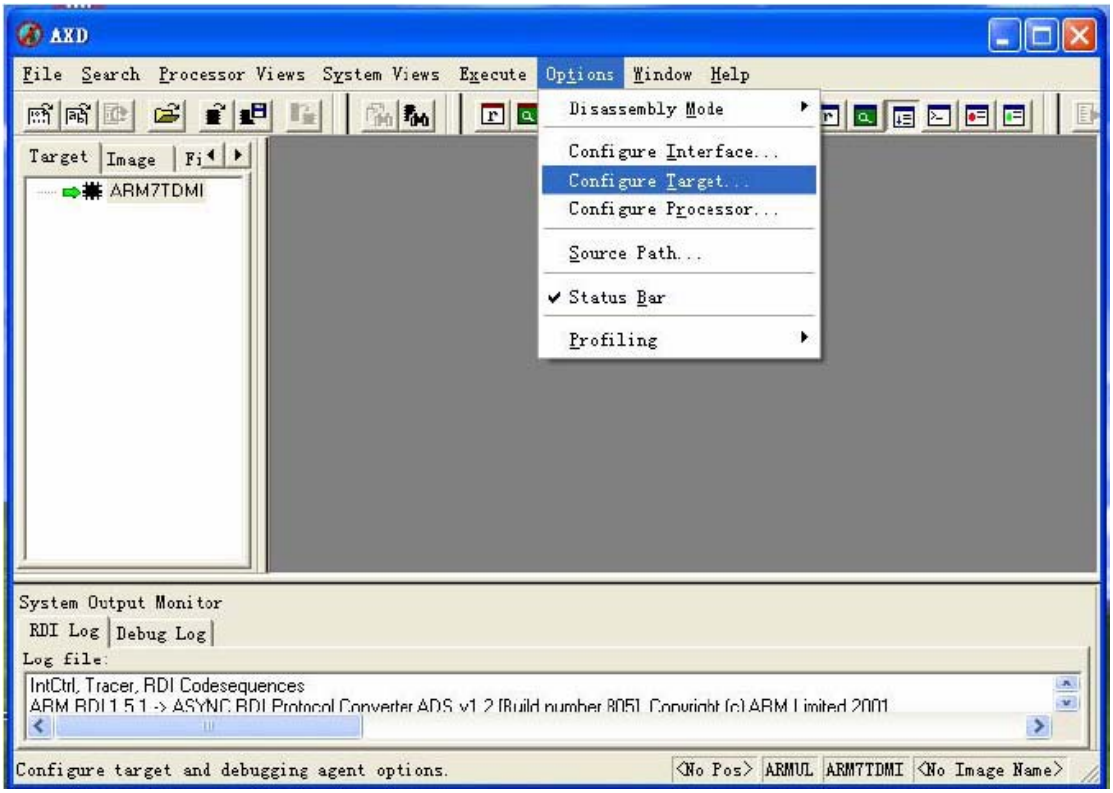
然后给 9261 目标板上电，注意，如果使用本站的并口或者 USB Multi-ICE，均不需要对仿真器进行额外供电，并口的 Multi-ICE 直接从 9261 目标板获取电源，USB Multi-ICE 则直接从 USB 取电工作。如果是使用别家的 Multi-ICE，有可能需要对 Multi-ICE 进行单独供电。完成以上步骤后请运行 Multi-ICE Server 软件，稍等片刻，将找到 926EJ-S 内核，如下图：



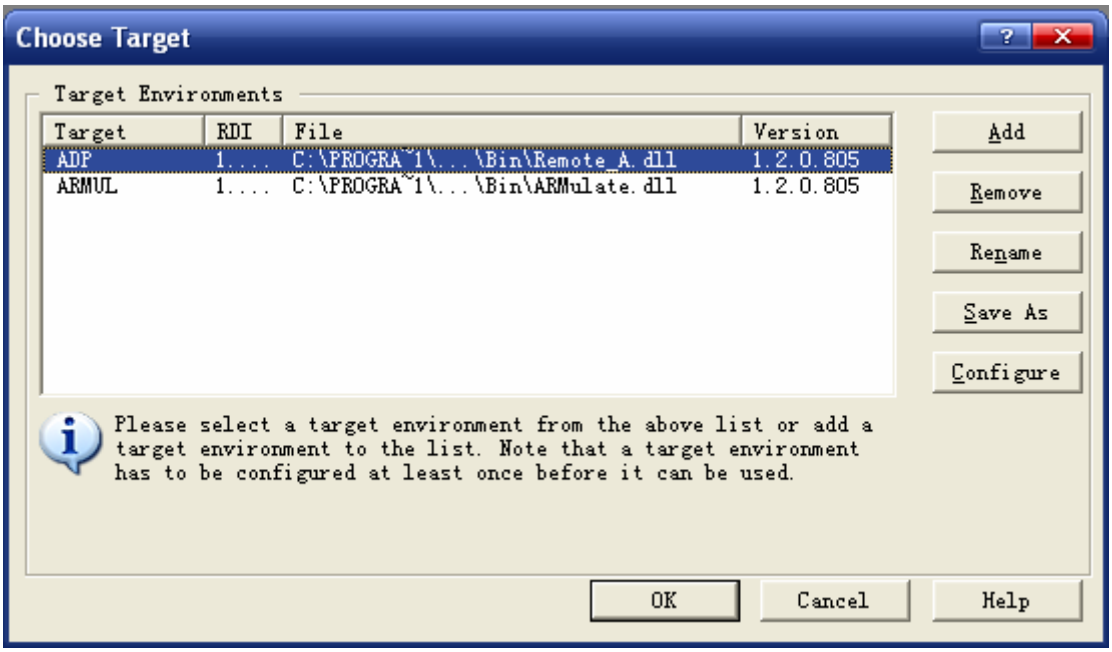
接下来就可以直接设置 AXD 了。首先打开 AXD，或者通过 ADS 进入 AXD：



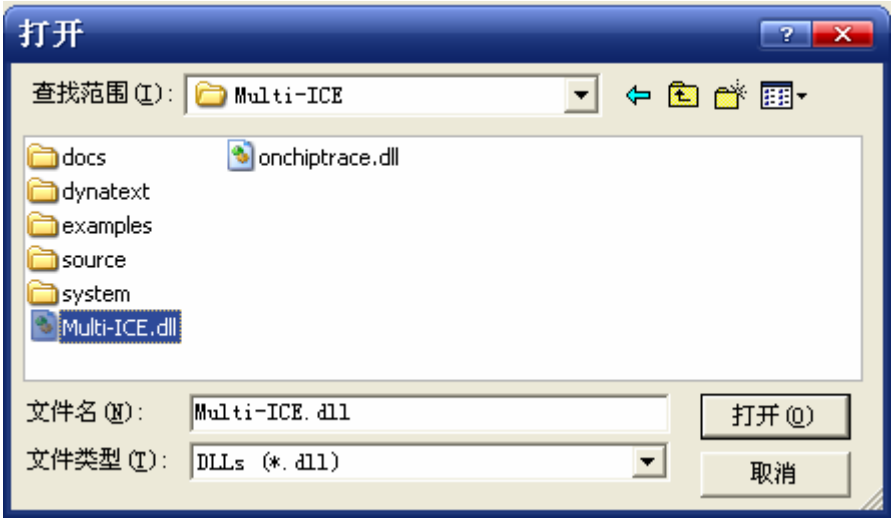
然后打开 Option 菜单，选择“Configure Target”：



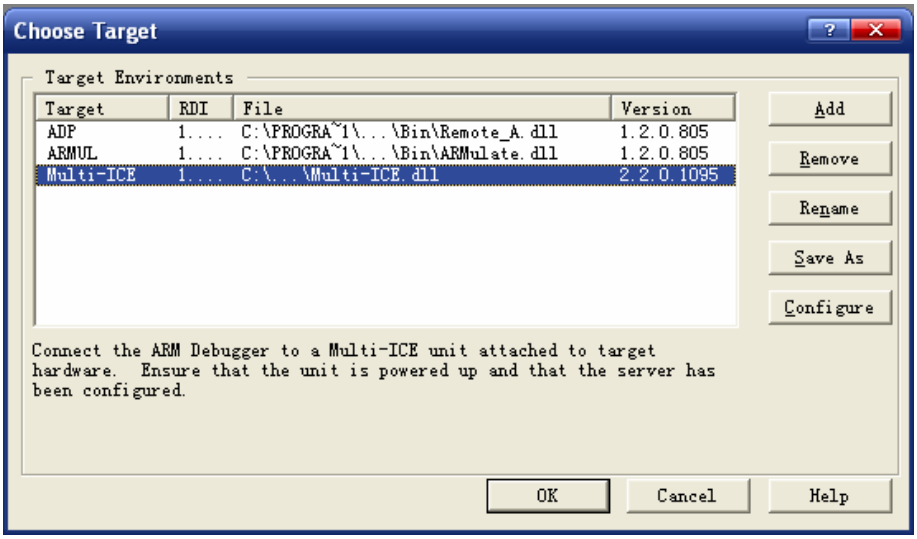
出现如下页面：



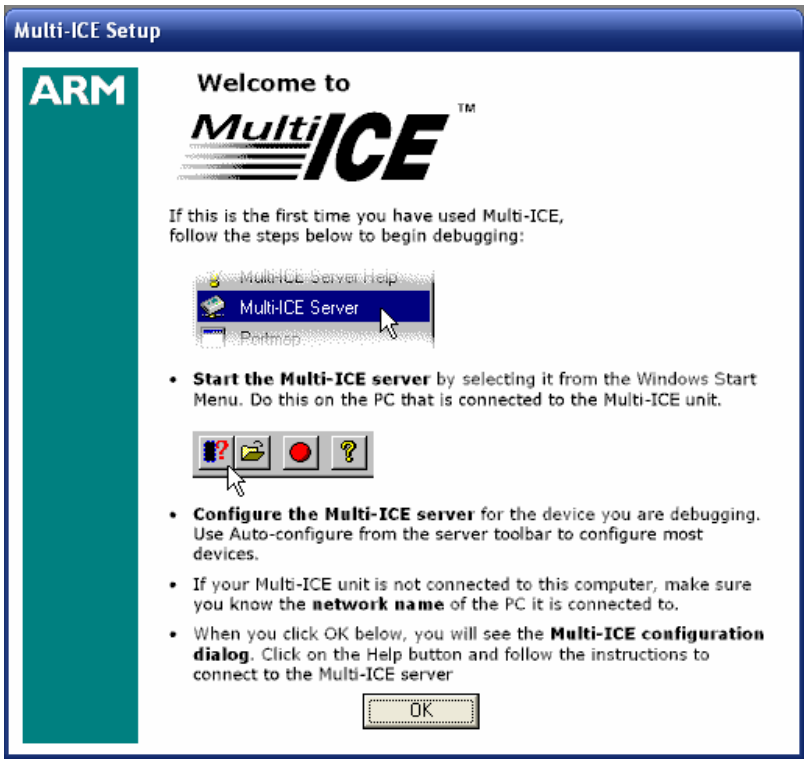
点击“Add”按钮，找到 Multi-ICE 的安装路径，找到 Multi-ICE.DLL 文件：



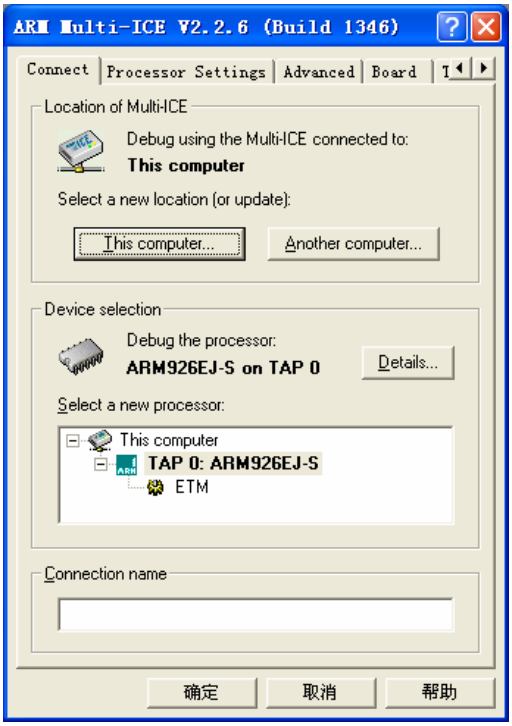
点击 OK 按钮进行确认。



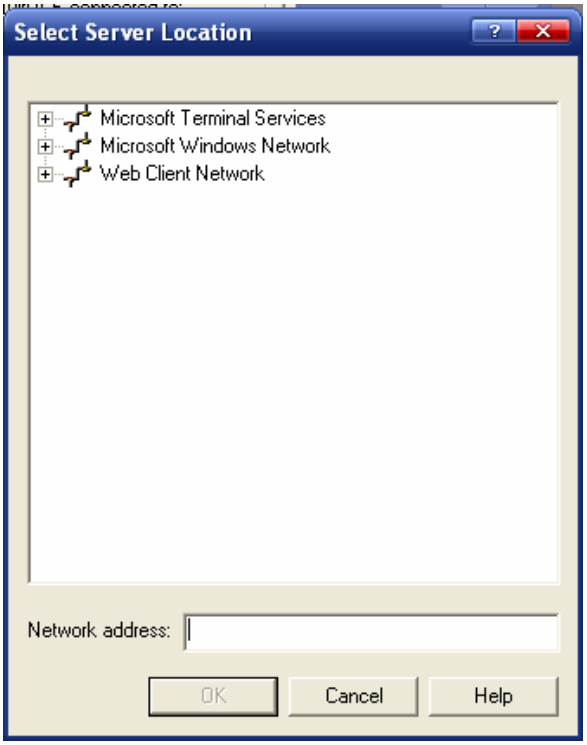
点击 Configure，出现 Multi-ICE Setup 的页面：



点击 OK 进入下一步：



Multi-ICE Server 支持远程调试，如果目标板在另外一台电脑上，请点击“Another Computer”按钮：

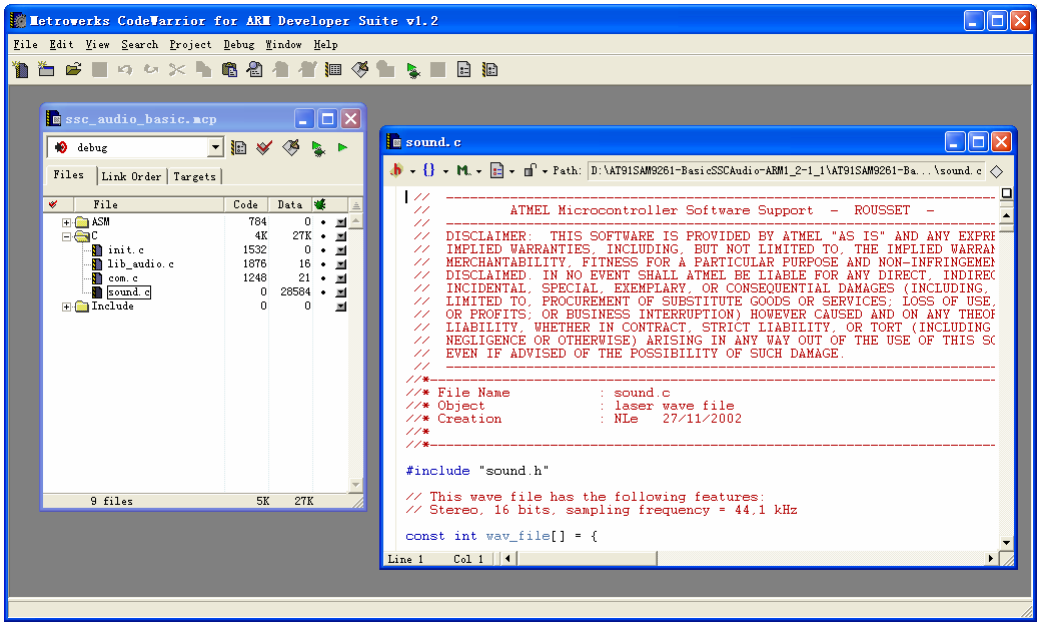


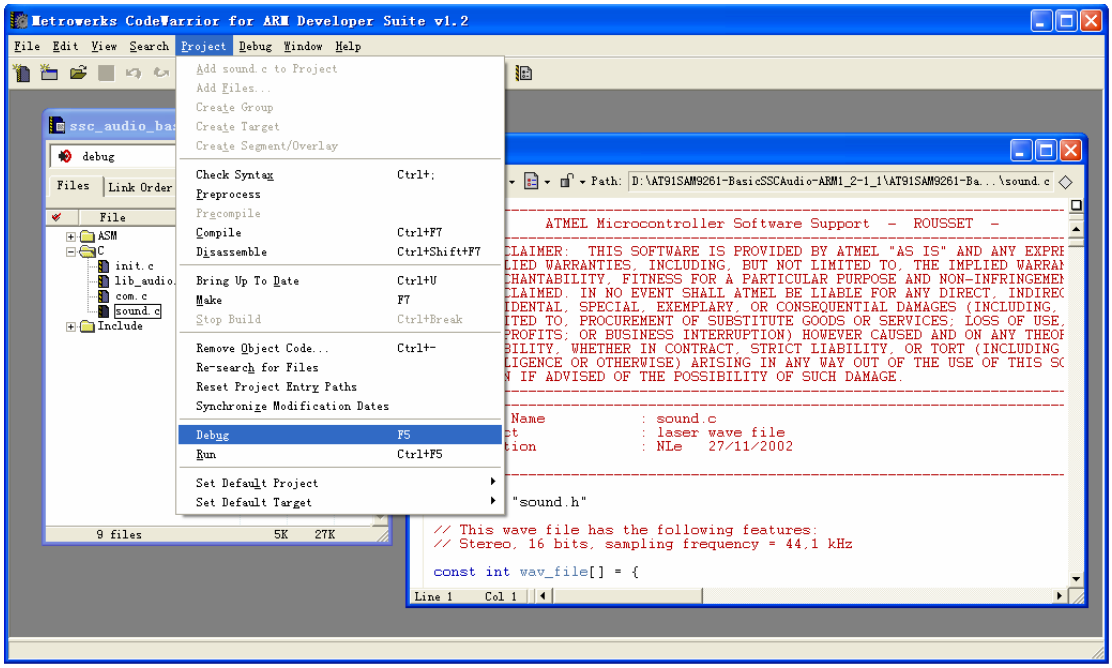
然后查到挂接了目标板，并开启了 Multi-ICE Server 的网络电脑。

二、利用 Multi-ICE 进行调试

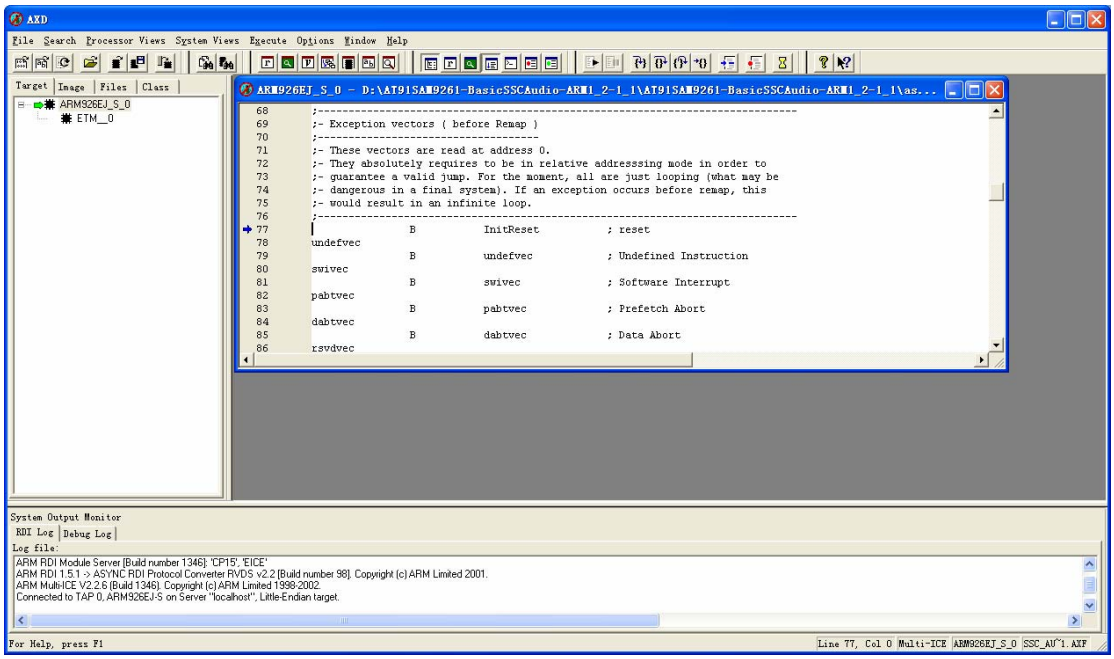
下面简单介绍一下在 ADS 下用 Multi-ICE 进行调试的流程。

首先打开一个工程，可以直接使用 ATMEL 提供的工程，可以到 www.atmel.com 网站，也可以到 www.atarm.com 网站进行下载，以下我们使用的是一个 AT91SAM9261-BasicSSCAudio-ARM1_2-1_1 范例，请下载该范例，进入 AT91SAM9261-BasicSSCAudio-ARM1_2-1_1\compil 目录，然后请双击打开 ssc_audio_basic.mcp 工程文件，如果是纯英文目录可以直接双击打开，如果是带中文路径，请先打开 CodeWarrior for ARM Developer Suite，然后调入该 basic.mcp 工程文件，不然 ADS 会提示出错。具体的 ADS 软件的使用可以找相关资料或者 ADS 自带的 HELP 文档。

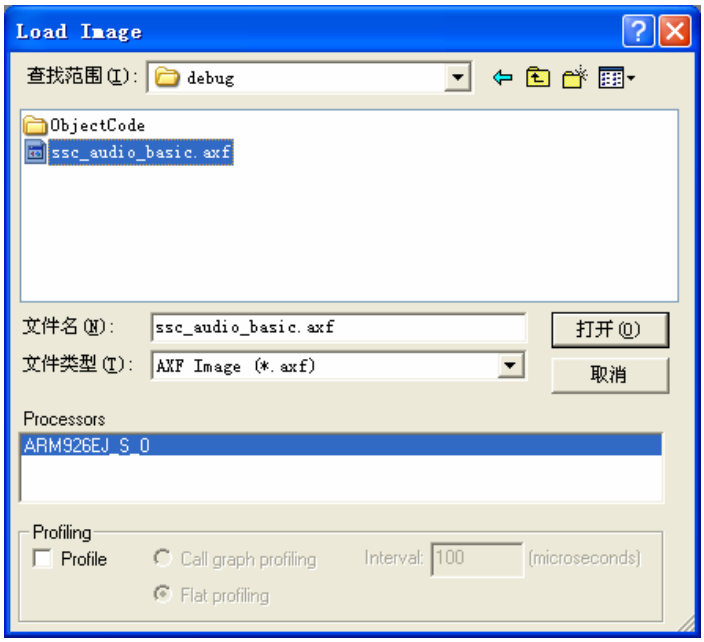




点击 Debug 进入 AXD 调试环境：



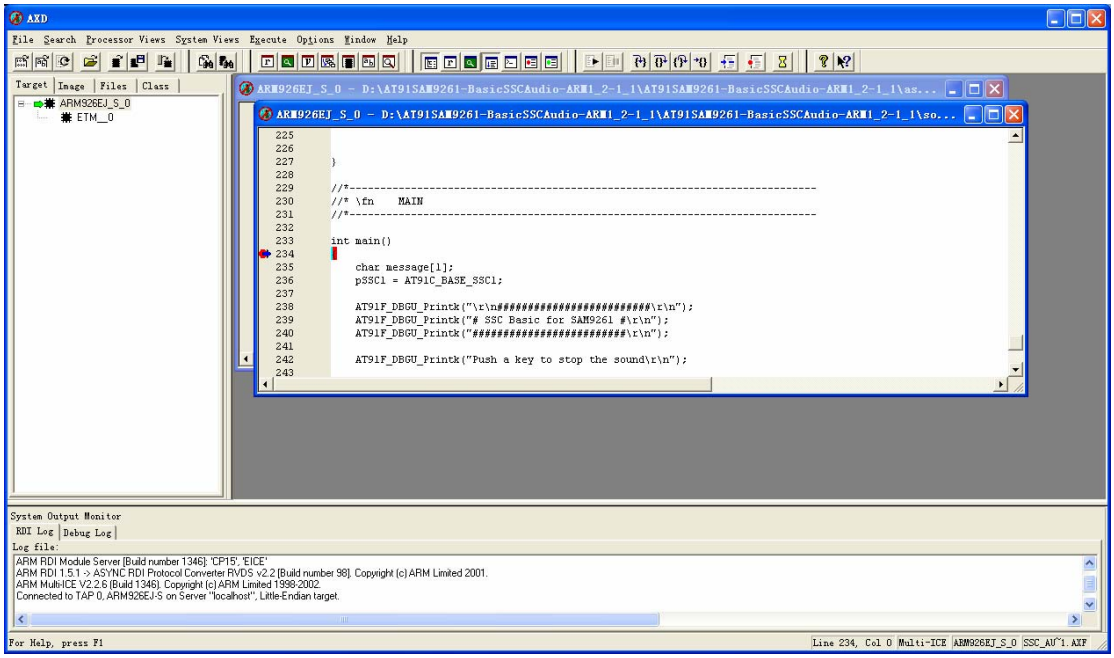
如果程序已经编译通过，axf 文件已经生成，也可以不从 Codewarrior 进入 AXD，即直接打开 AXD 调试环境，然后调入 axf 文件：



Axf 文件默认地址为：

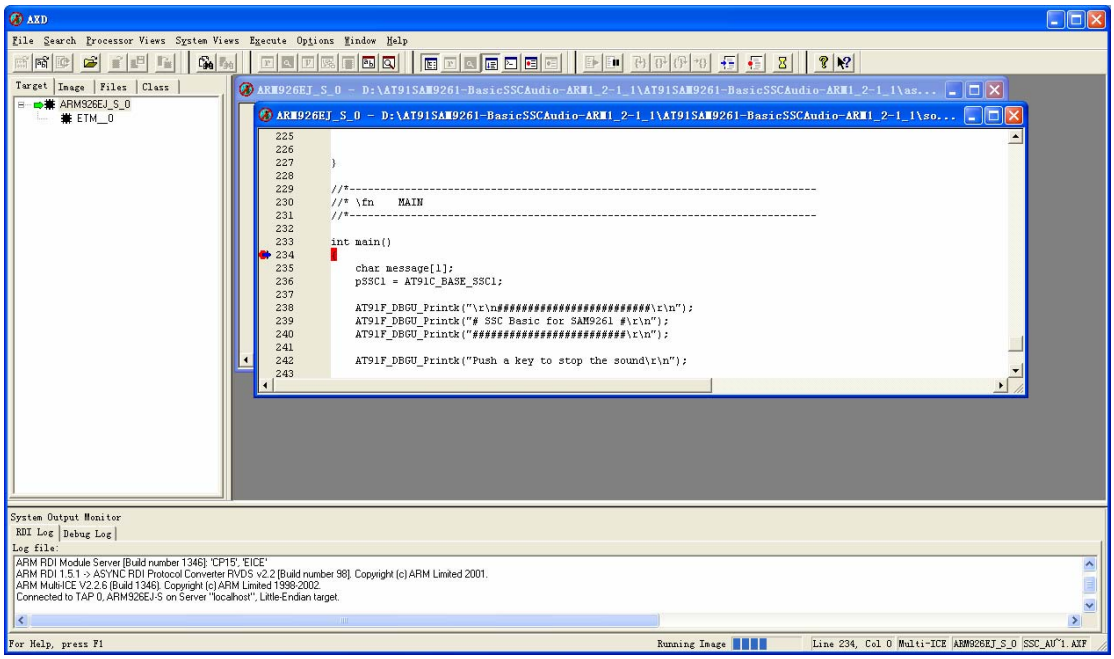
AT91SAM9261-BasicSSCAudio-ARM1_2-1_1\compil\ssc_audio_basic_Data\debug

然后点击 Go 运行：



程序将停在 main 函数处，因为在该处设置了一个断点。

点 Go 继续运行：

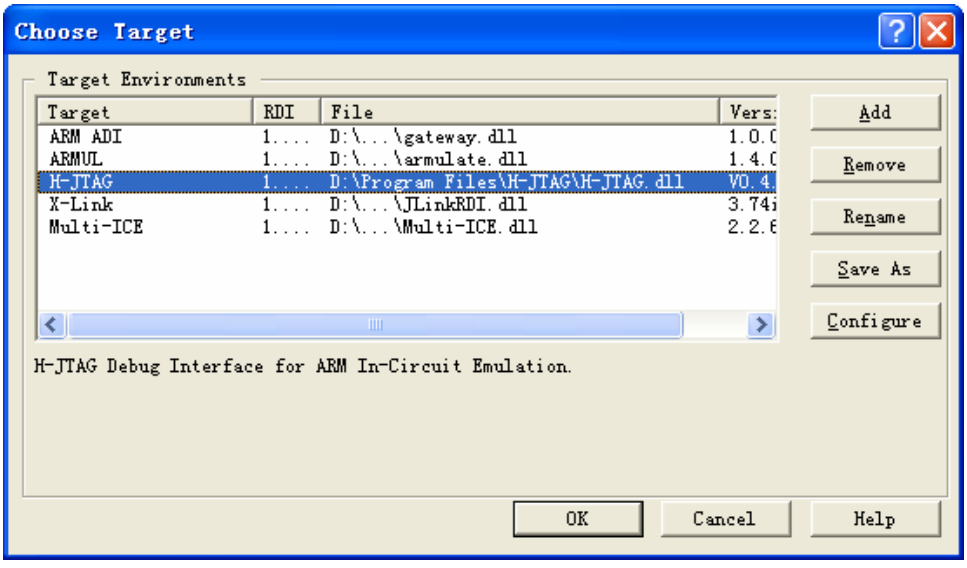


注意观察 DBGU 串口输出，输出内容可以参见之前声卡测试章节。

看一下 main 函数内容：

```
//*---------------------------------------------------------------------------
//* \fn       MAIN
//*---------------------------------------------------------------------------
int main()
{
        char message[1];
        pSSC1 = AT91C_BASE_SSC1;
        AT91F_DBGU_Printk("\r\n##########################\r\n");
        AT91F_DBGU_Printk("# SSC Basic for SAM9261 #\r\n");
        AT91F_DBGU_Printk("##########################\r\n");
        AT91F_DBGU_Printk("Push a key to stop the sound\r\n");
        // Init I2S
        AT91F_InitDrivers();
        message[0] = AT91F_DBGU_getc();
// while(!stop)
        AT91F_CloseDrivers();
        AT91F_DBGU_Printk("\r\n");
        return 0;
}       // End main
```

如果使用别的仿真器调试，只要添加相关的 DLL 文件即可，如下图所示：



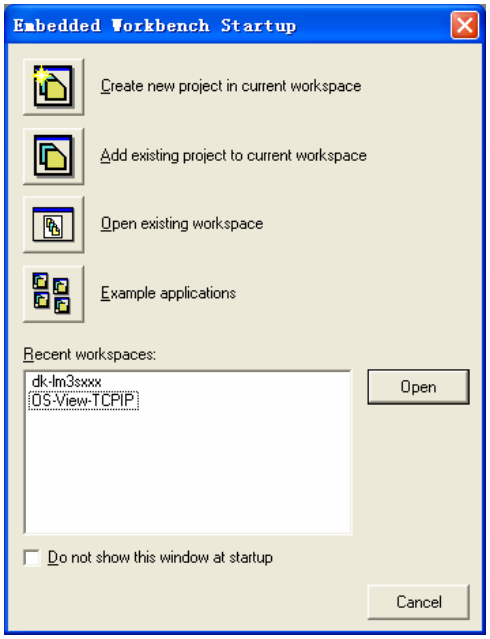
注意：调试程序前请确认没有 OS 正在运行，建议拔掉 J21 跳线，然后按一下 Reset 按钮或者重新上电后再开始调试！

第七章：IAR 下 9261 调试方法

IAR 作为业界知名的编译器提供商，其 ARM 编译器也有相当大的用户群。这里我们用 IAR 提供的范例简单演示一下 IAR 下的调试过程。

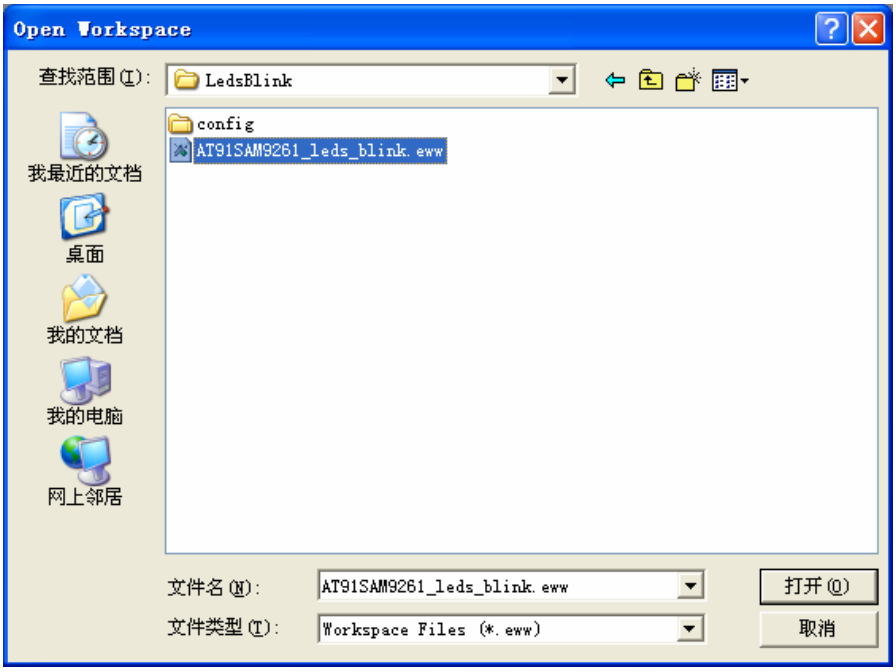
IAR for ARM 评估版本软件可以直接在 IAR 的网站下载到，也可以在 www.Mcuzone.com 或者 www.ATARM.com 下载到。

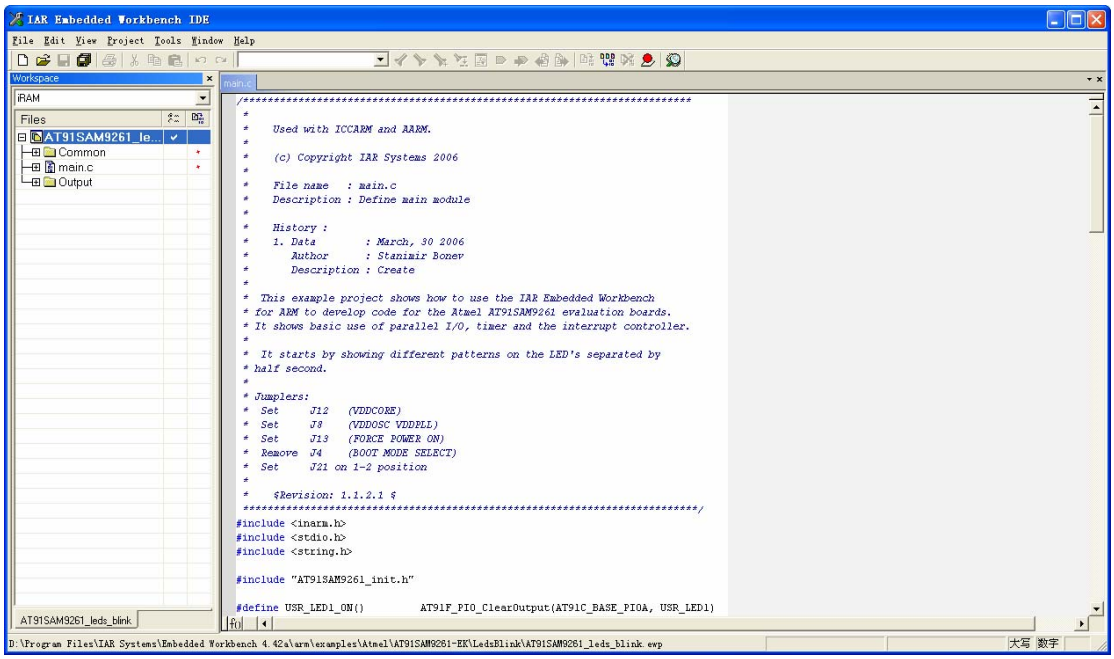
安装完成后打开，一开始 IAR 会有一个 workbench startup 界面：



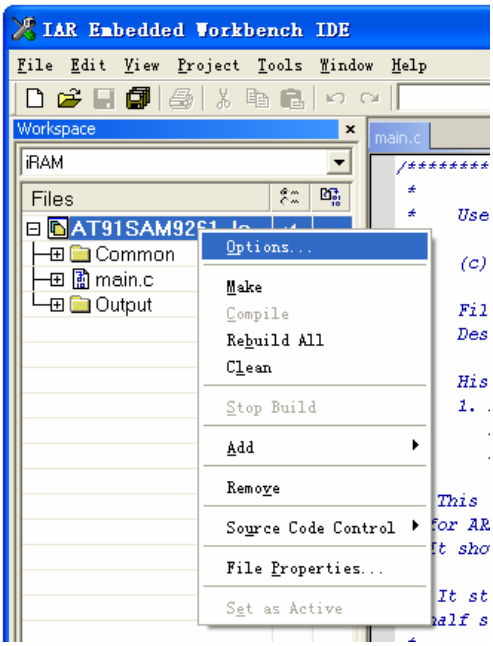
然后我们选择 Open existing workspace，在以下目录找到 AT91SAM9261-EK 范例：

Embedded Workbench 4.42a\arm\examples\Atmel\AT91SAM9261-EK



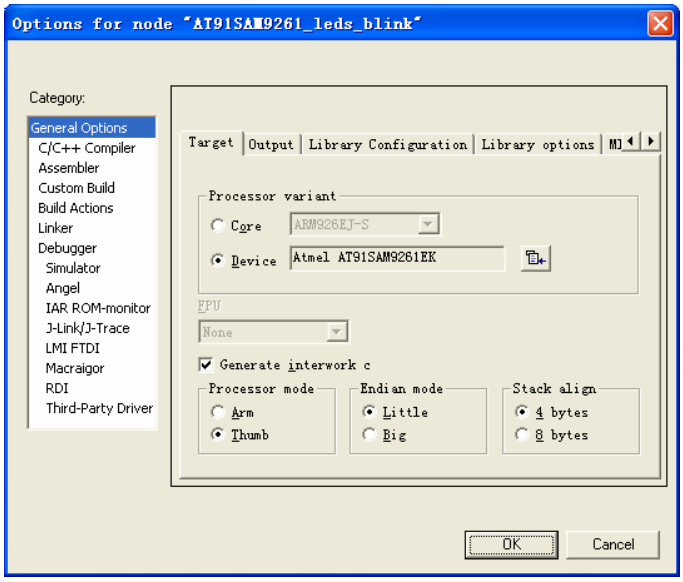


下面来设置一下：

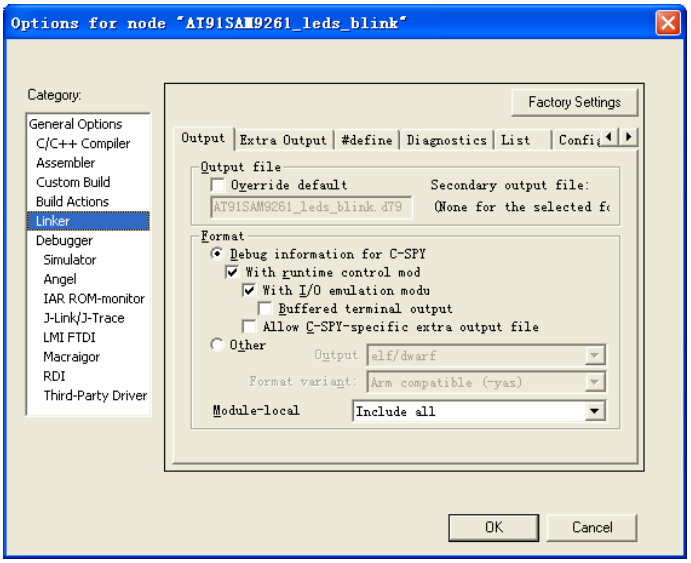


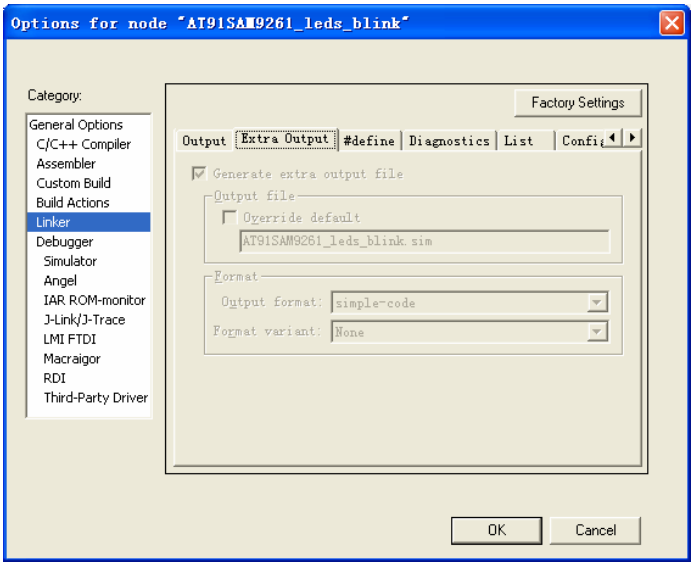
该 LED 跑马的例子可以在片内 RAM 运行，也可以下载到 DATAFLASH 运行，这里我们选择在 iRAM 运行。

点中左边工程根目录，然后右键选择 Options，出来以下界面：



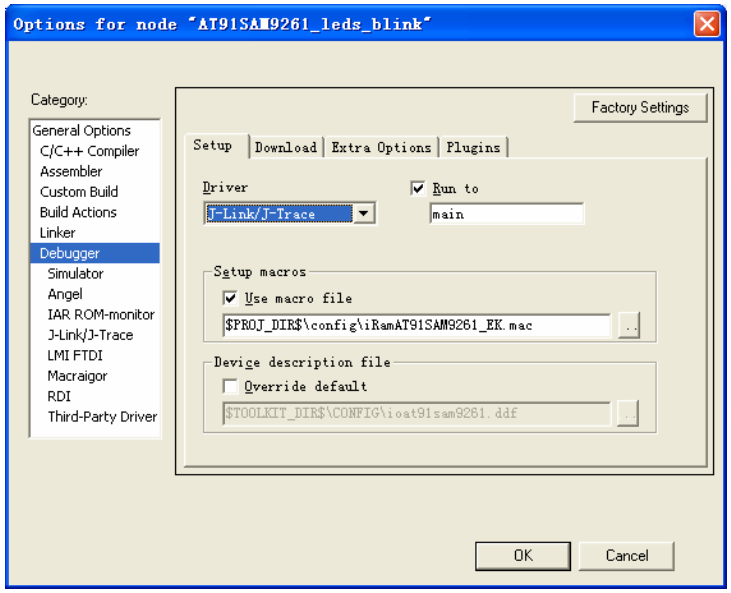
有几个地方需要确认并设置，请先点中 Linker：



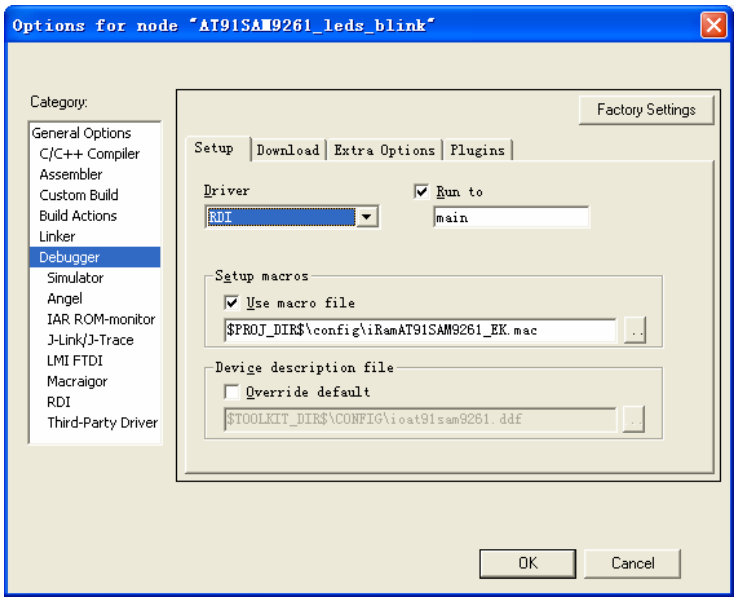


主要核对 Output 选项卡和 Extra Output 选项卡。

然后设置 Debugger：

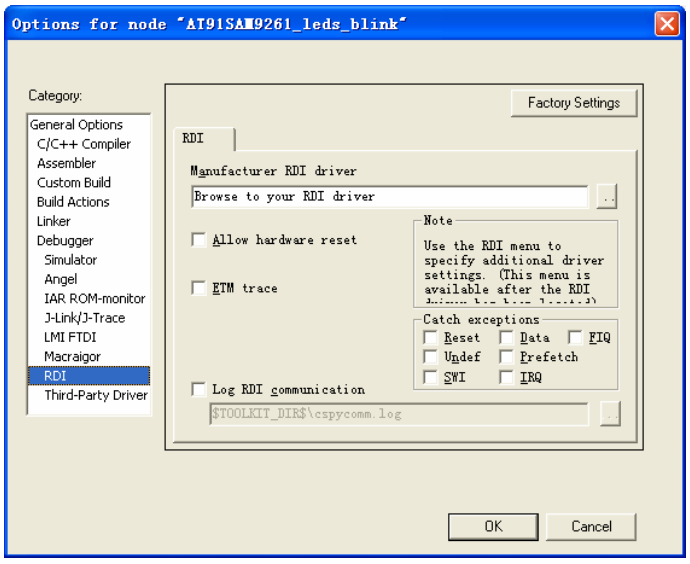


默认的 Debugger 是 SEGGER 公司的 J-Link，但是并不是每个人都有 J-Link 调试器，幸好 IAR 集成了很多 Debugger 的驱动，并且还提供了 RDI 接口。

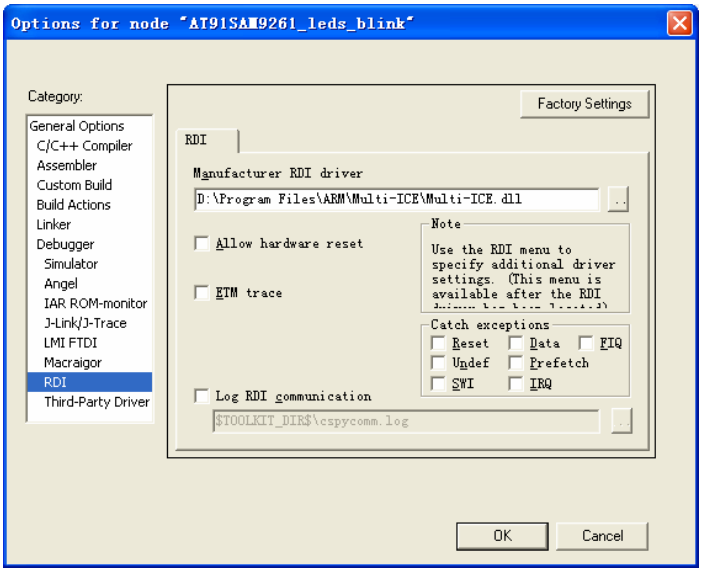


请注意不要修改 Setup macros 的设置。

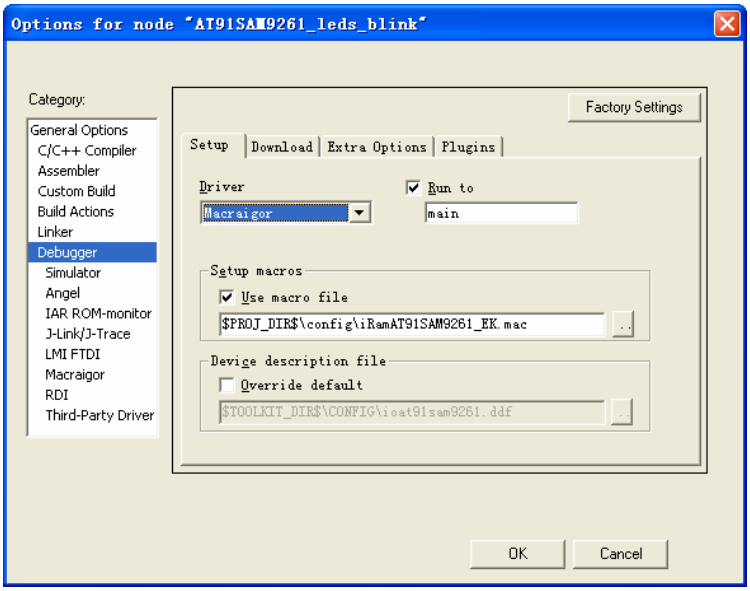
由于我们当前是在 iRAM 调试，所以 Download 菜单项不需要更改。

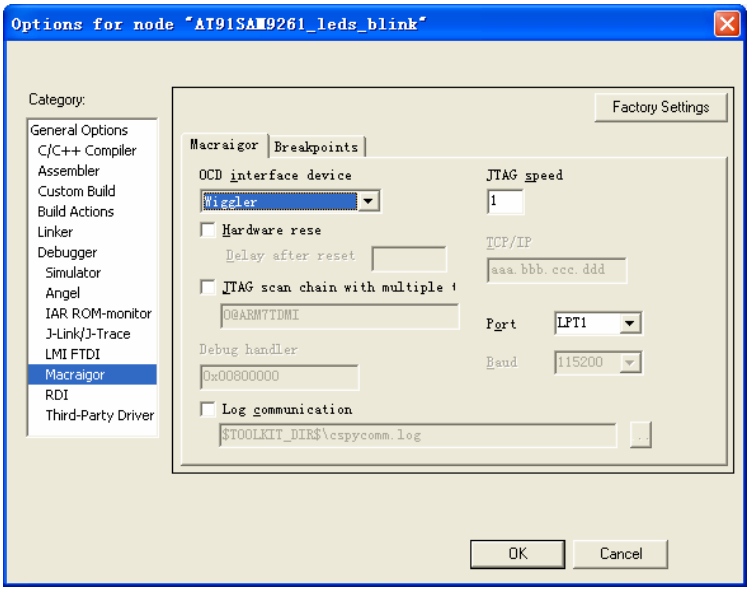


在 RDI 下，我们通过 Brouse 按钮来选择对应的 DLL 文件，如果你使用 Multi-ICE，请将 RDI Driver 指向到 Multi-ICE.dll；如果你使用的是全功能版本的 J-Link，请将 RDI Driver 指向到 JLinkARM.dll；如果你使用的是 H-JTAG，请将 RDI Driver 指向到 H-JTAG.dll；如果你使用别的 RDI 接口仿真器，请指向到相关的 dll 文件。

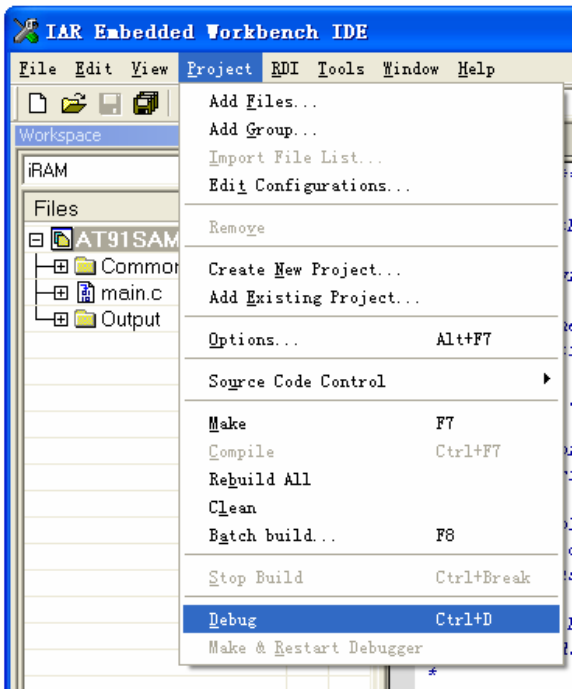


如果你使用的是 wiggler 简易调试头，也可以直接用 Macraigor 的 wiggler 作为驱动：

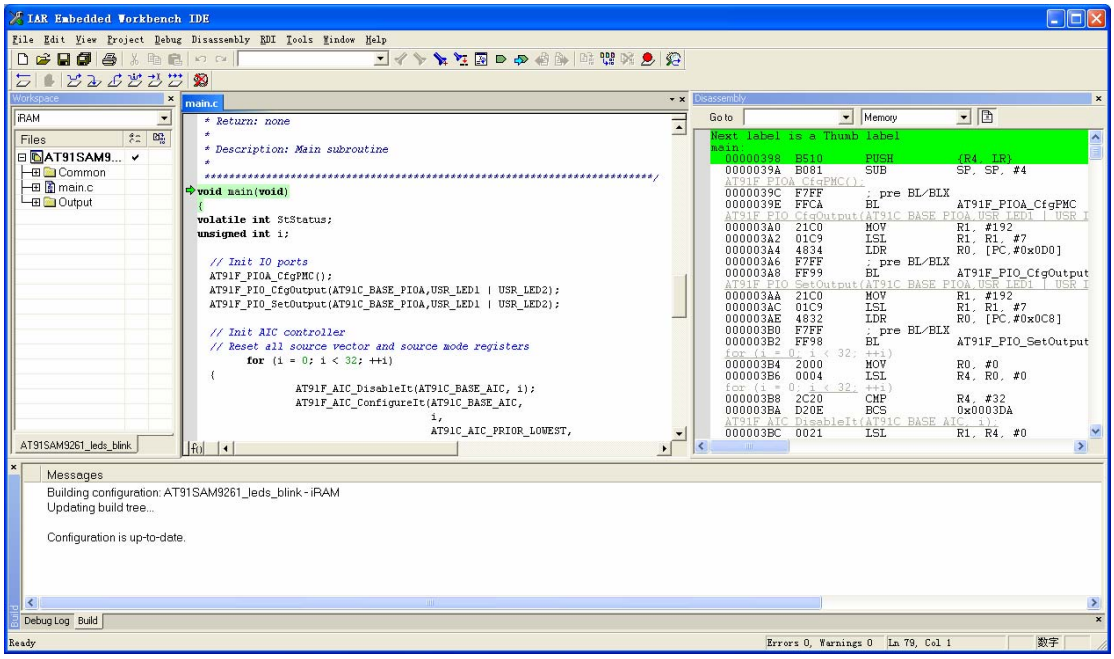

Options for node "AT91SAM9261_leds_blink"
Category:
General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
Simulator
Angel
IAR ROM-monitor
J-Link/J-Trace
LMI FTDI
Macraigor
RDI
Third-Party Driver
Factory Settings
Setup
Download
Extra Options
Plugins
Driver
Macraigor
Run to
main
Setup macros
Use macro file
$PROJ_DIR$\config\iRamAT91SAM9261_EK.mac
Device description file
Override default
$TOOLKIT_DIR$\CONFIG\ioat91sam9261.ddf
OK
Cancel


Options for node "AT91SAM9261_leds_blink"
Category:
General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
Simulator
Angel
IAR ROM-monitor
J-Link/J-Trace
LMI FTDI
Macraigor
RDI
Third-Party Driver
Factory Settings
Macraigor
Breakpoints
OCD interface device
Wiggler
JTAG speed
1
Hardware rese
Delay after reset
TCP/IP
aaa.bbb.ccc.ddd
JTAG scan chain with multiple
0@ARM7TDMI
Port
LPT1
Debug handler
0x00800000
Baud
115200
Log communication
$TOOLKIT_DIR$\cspycomm.log
OK
Cancel

设置完成，即可选择 Project 下的 Debug 按钮，进入调试状态，



由于文件很小，所以很快即可下载完成并进入 Debug 状态：



在 main 函数处有一个断点，程序停在 main 函数处。

可以按照需要进行单步、断点、全速等操作，全速运行后可以看到板上两个 LED 开始闪烁，说明程序运行正常！

第八章：Keil 下 9261 调试方法

Keil 对很多搞过 8051 的工程师而言都是很熟悉的，Keil 的 C51 编译器可谓是业内公认的标准，经过几年的发展，Keil 的 ARM 编译器也已经很成熟，特别是 Keil 被 ARM 公司收购后，做为 ARM 公司旗下公司，Keil 的认可度更被业界接受。一开始 Keil for ARM 主要支持 ARM7，但是现在 Keil for ARM 也支持了不少型号的 ARM9 了，比如 2410，9200，9261 等，下面我们就大致的讲述一下用 Keil 调试 AT91SAM9261-EK 的简要步骤。

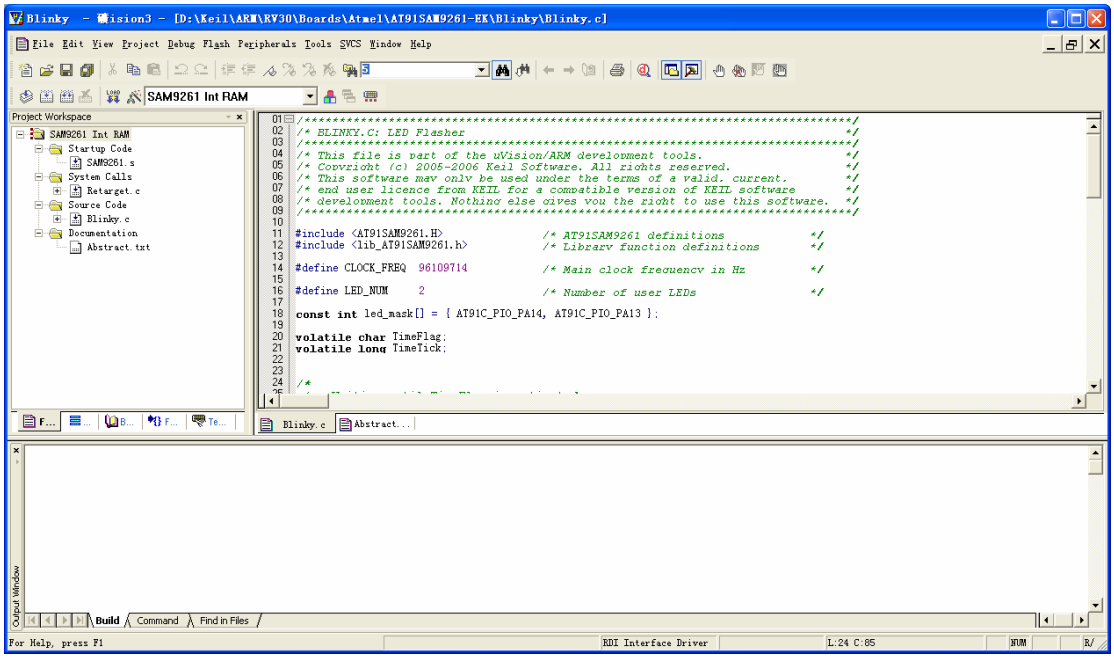
1，工程设置

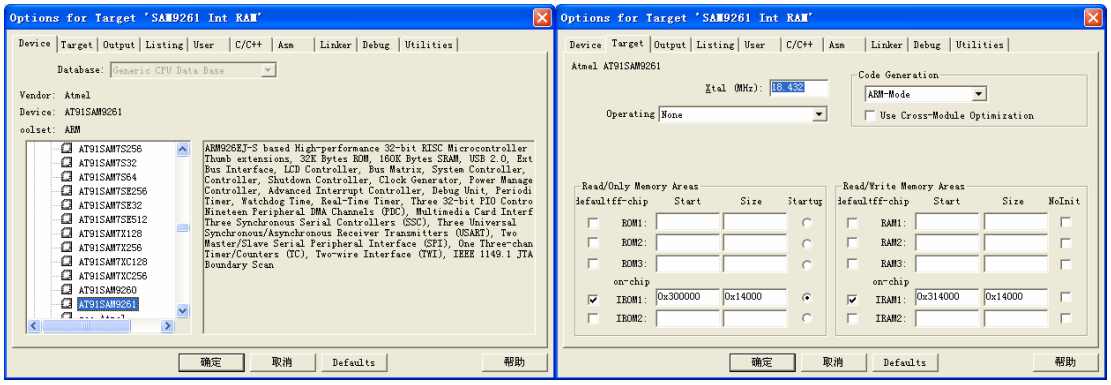
首先请安装好 Keil，评估版本的软件可以在 www.keil.com 或者 www.mcuzone.com 或者 www.atarm.com 下载到，并连接好 ULINK2 和 9261-EK 目标板。

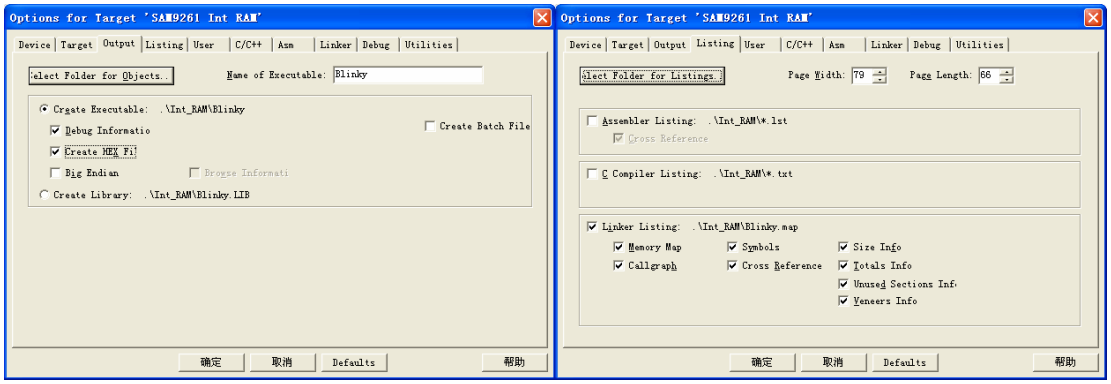
AT91SAM9261-EK 的范例位于：

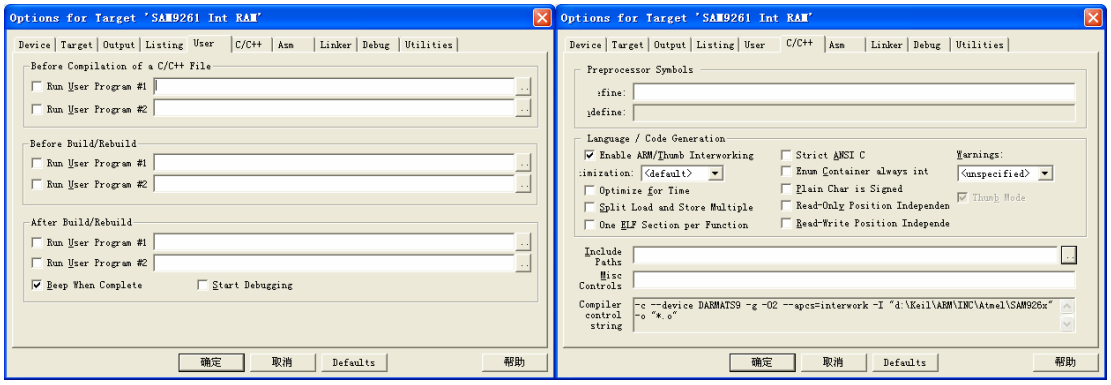
Keil\ARM\RV30\Boards\Atmel\AT91SAM9261-EK\Blinky

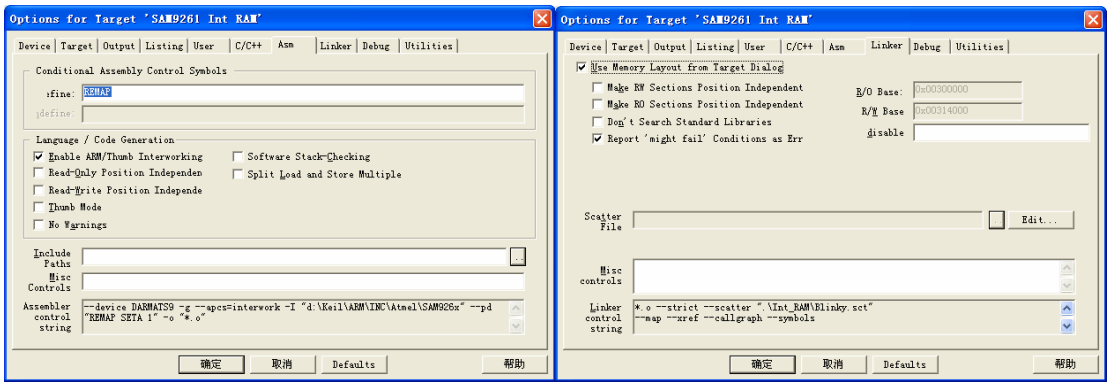
我们打开 Blinky.Uv2：

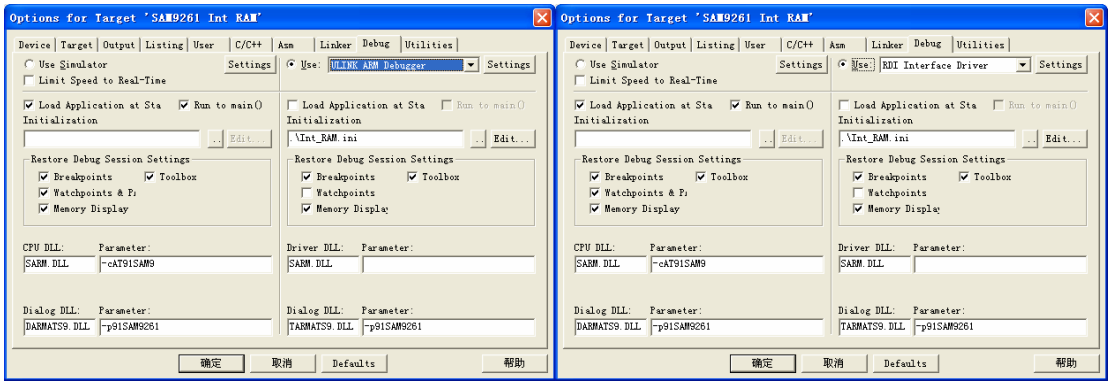



Options for Target 'SAM9261 Int RAM'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Database: Generic CPU Data Base
Vendor: Atmel
Device: AT91SAM9261
AT91SAM7S256
AT91SAM7S32
AT91SAM7S64
AT91SAM7SE256
AT91SAM7SE32
AT91SAM7SE512
AT91SAM7X128
AT91SAM7X256
AT91SAM7XC128
AT91SAM7XC256
AT91SAM9260
AT91SAM9261
ARM926EJ-S based High-performance 32-bit RISC Microcontroller
Atmel AT91SAM9261
Xtal (MHz): 18.432
Operating None
Code Generation
ARM-Mode
Use Cross-Module Optimization
Read/Only Memory Areas
Read/Write Memory Areas
ROM1:
ROM2:
ROM3:
IROM1: 0x300000 0x14000
IROM2:
RAM1: 0x314000 0x14000
RAM2:
RAM3:
确定 取消 Defaults 帮助


Options for Target 'SAM9261 Int RAM'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Name of Executable: Blinky
Create Executable: .\Int_RAM\Blinky
Debug Informatio
Create HEX Fi
Big Endian
Create Library: .\Int_RAM\Blinky.LIB
Create Batch File
Page Width: 79
Page Length: 66
Assembler Listing: .\Int_RAM*.lst
Cross Reference
C Compiler Listing: .\Int_RAM*.txt
Linker Listing: .\Int_RAM\Blinky.map
Memory Map
Symbols
Size Info
Callgraph
Cross Reference
Totals Info
Unused Sections Inf
Veneers Info
确定 取消 Defaults 帮助

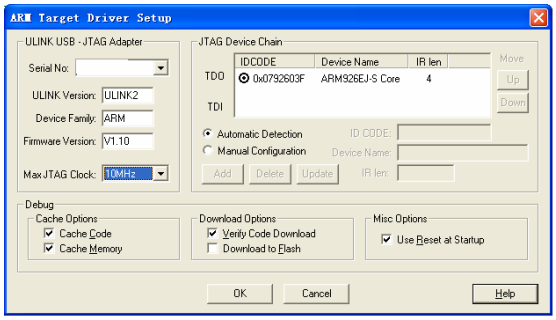

Options for Target 'SAM9261 Int RAM'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Before Compilation of a C/C++ File
Run User Program #1
Run User Program #2
Before Build/Rebuild
After Build/Rebuild
Beep When Complete
Start Debugging
Preprocessor Symbols
Language / Code Generation
Enable ARM/Thumb Interworking
Strict ANSI C
Optimize for Time
Plain Char is Signed
Split Load and Store Multiple
Read-Only Position Independen
One ELF Section per Function
Read-Write Position Independe
Thumb Mode
Include Paths
Misc Controls
Compiler control string
-c --device DARMATS9 -g -O2 --apcs=interwork -I "d:\Keil\ARM\INC\Atmel\SAM926x"
-o "*.o"
确定 取消 Defaults 帮助

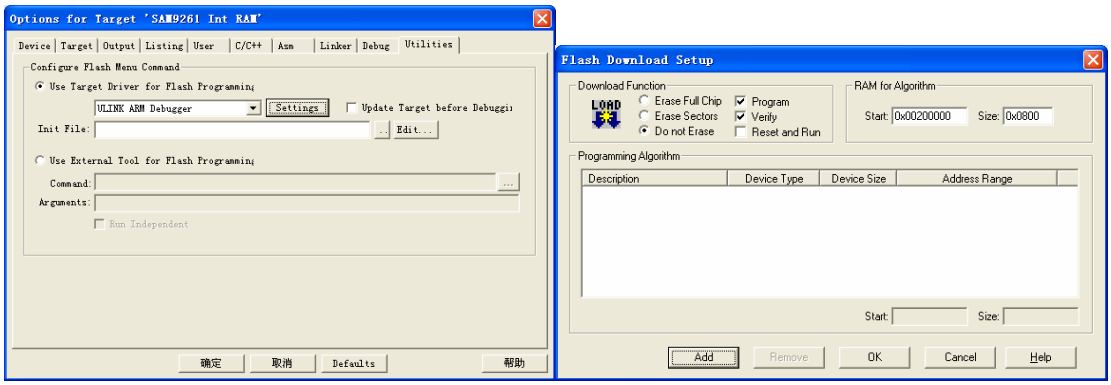

Options for Target 'SAM9261 Int RAM'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Conditional Assembly Control Symbols
REMAP
Language / Code Generation
Enable ARM/Thumb Interworking
Software Stack-Checking
Read-Only Position Independen
Split Load and Store Multiple
Read-Write Position Independe
Thumb Mode
No Warnings
Include Paths
Misc Controls
Assembler control string
--device DARMATS9 -g --apcs=interwork -I "d:\Keil\ARM\INC\Atmel\SAM926x" --pd
"REMAP SETA 1" -o "*.o"
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Err
R/O Base: 0x00300000
R/W Base 0x00314000
disable
Scatter File
Edit...
Misc controls
Linker control string
*.o --strict --scatter ".\Int_RAM\Blinky.sct"
--map --xref --callgraph --symbols
确定 取消 Defaults 帮助

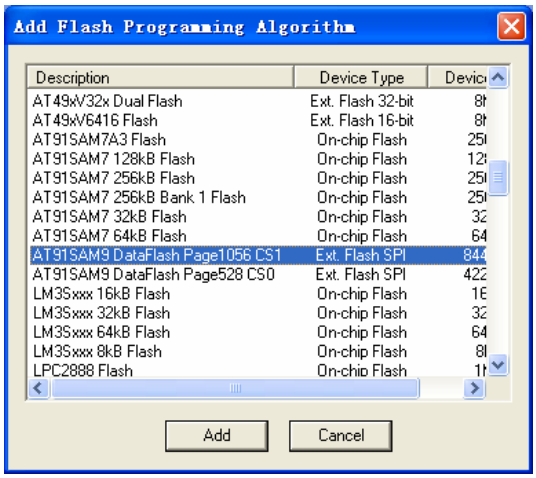


在 3.05 及之前的版本具备 RDI 接口，可以通过该接口来支持第三方仿真器，J-Link 是很不错的 RDI 接口仿真器，不过 Multi-ICE 并不能在 keil 下使用。

在 3.05 以后版本则不再开放 RDI 接口，用户只能使用 Keil 公司的 ULINK2 仿真器。ULINK2 可以支持 10M JTAG 时钟，性能较上一代 ULINK 有所提高。

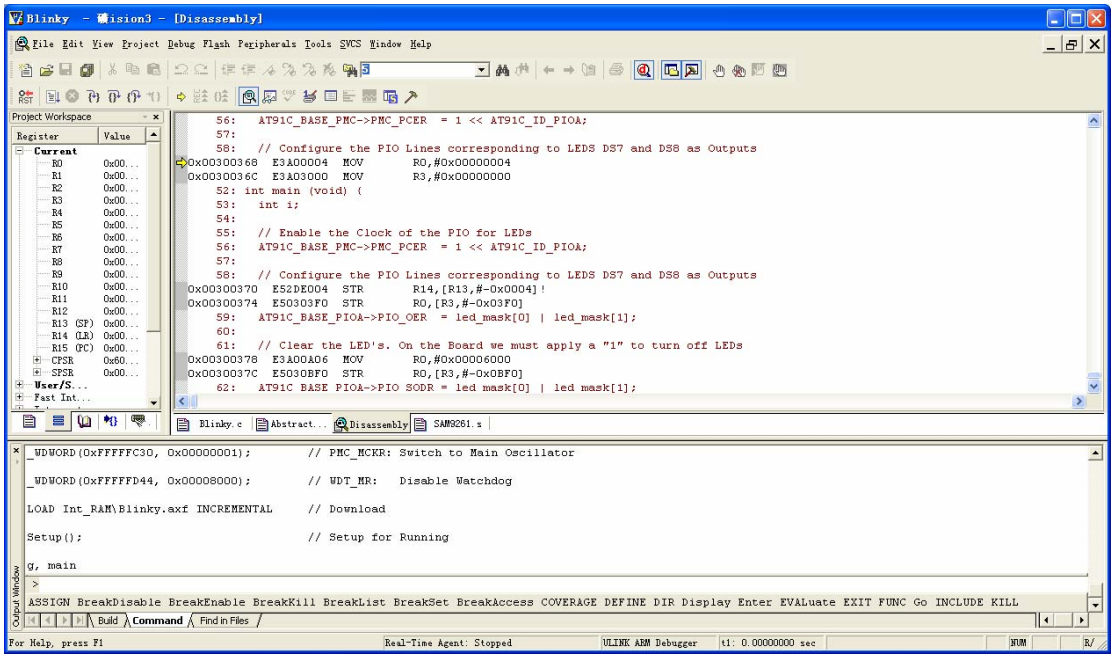






2，调试

设置无误后可以通过  按钮进入调试状态，如下图：



程序进入调试状态后停在 main 函数处，可以通过菜单来控制调试状态，并设置和取消断点。

按下 RUN 按钮  ，程序即开始运行，AT91SAM9261-EK 板上两个 LED 即开始闪烁，说明程序运行正常。

第九章：SAM9261 上的 MMU 的使用

—基于 Realview

本文叙述 SAM9261 芯片上的 MMU 的使用。MMU 的相关代码使用 realview 工具链进行编译，重点在代码的调试过程。调试软件选择了 ARM 的 RVD，仿真器选择的是 Banyan-UE，因为调试 MMU 的代码需要仿真器的支持，而目前不少仿真器并不支持 MMU 的调试。

一， 准备工作

1. 安装调试软件

PC 上的调试软件建议安装 ADS 或者 RealView 2.2，以后者为好。软件的安装请参照软件压缩包内的说明文件。

2. 安装调试器及相关软件

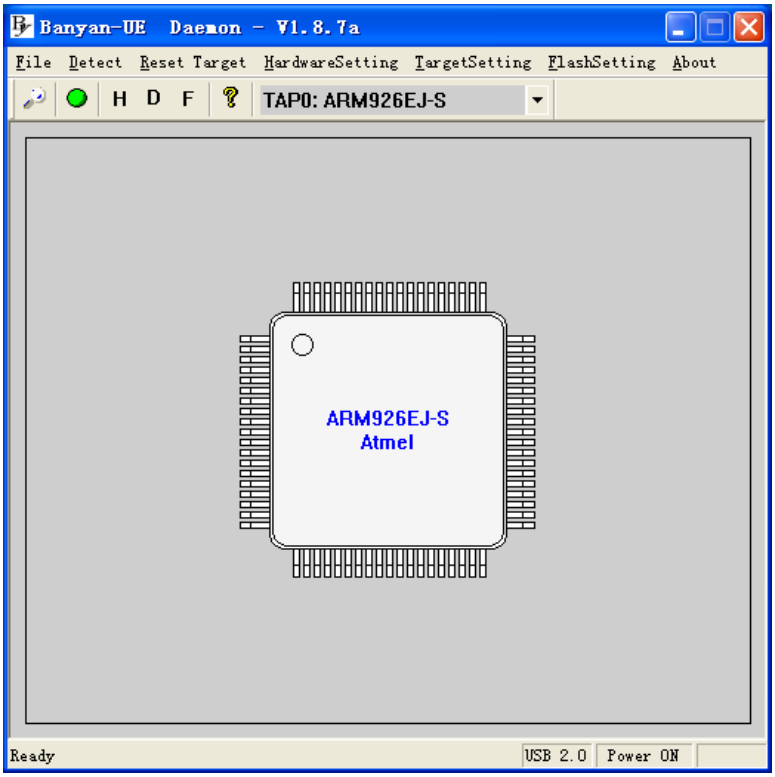
在使用 banyan 的之前，需要安装 banyan 的软件，该软件包含了调试代理软件，也包含了 banyan 的驱动。

连接上 banyan 到 PC 后选择自动安装驱动即可。

然后连接好 banyan 到 9261 板子 JTAG 口的 20 芯排线，给目标板上电。

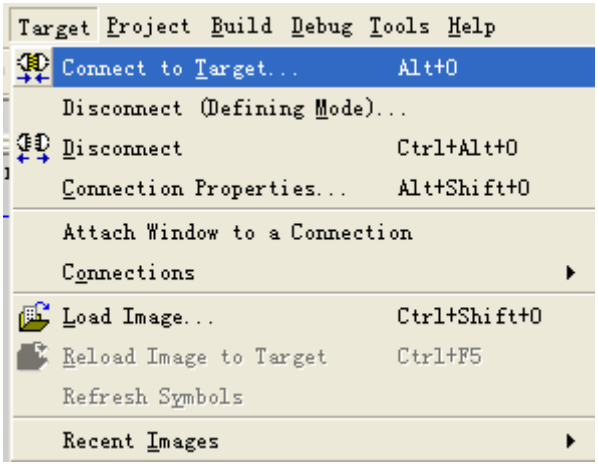
运行 banyan 的软件：

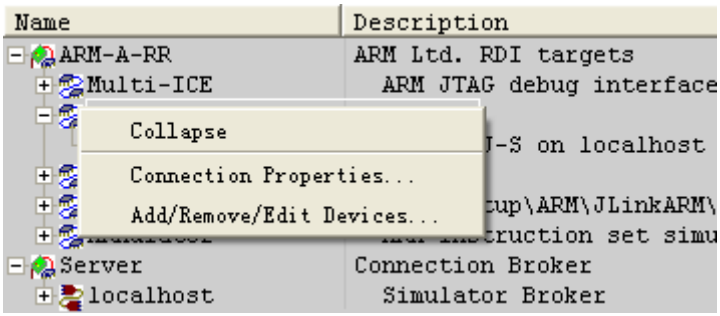
Banyan 将识别出 9261 的核心：



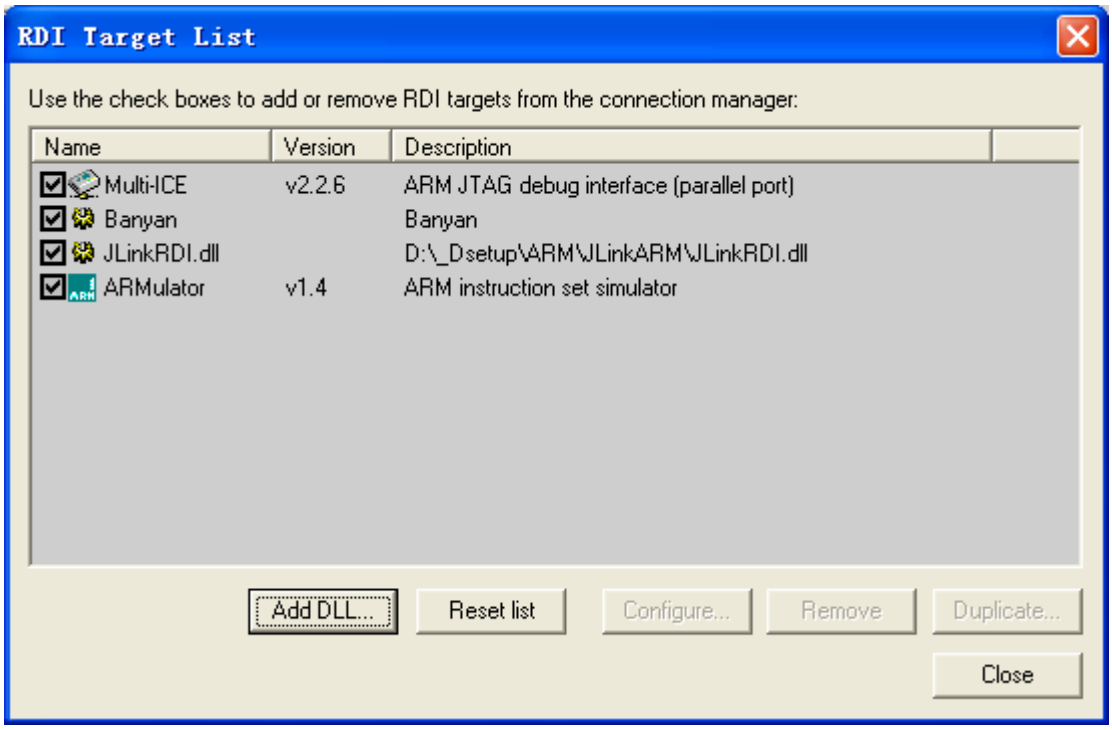
最小化后 banyan 将最小化到系统托盘：

打开 RVD，安装 banyan 的 target 到 RVD：

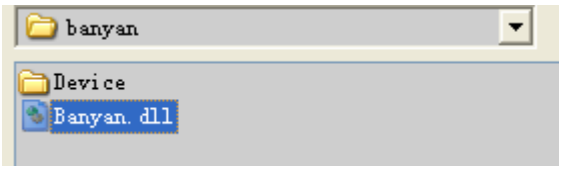




选择 Add/Remove/Edit Devices:



选择 Add DLL，添加 banyan 的 dll：



选择该 target 选择 OK 即可以该配置连接目标版。

3. 获取 MMU 的相关文档

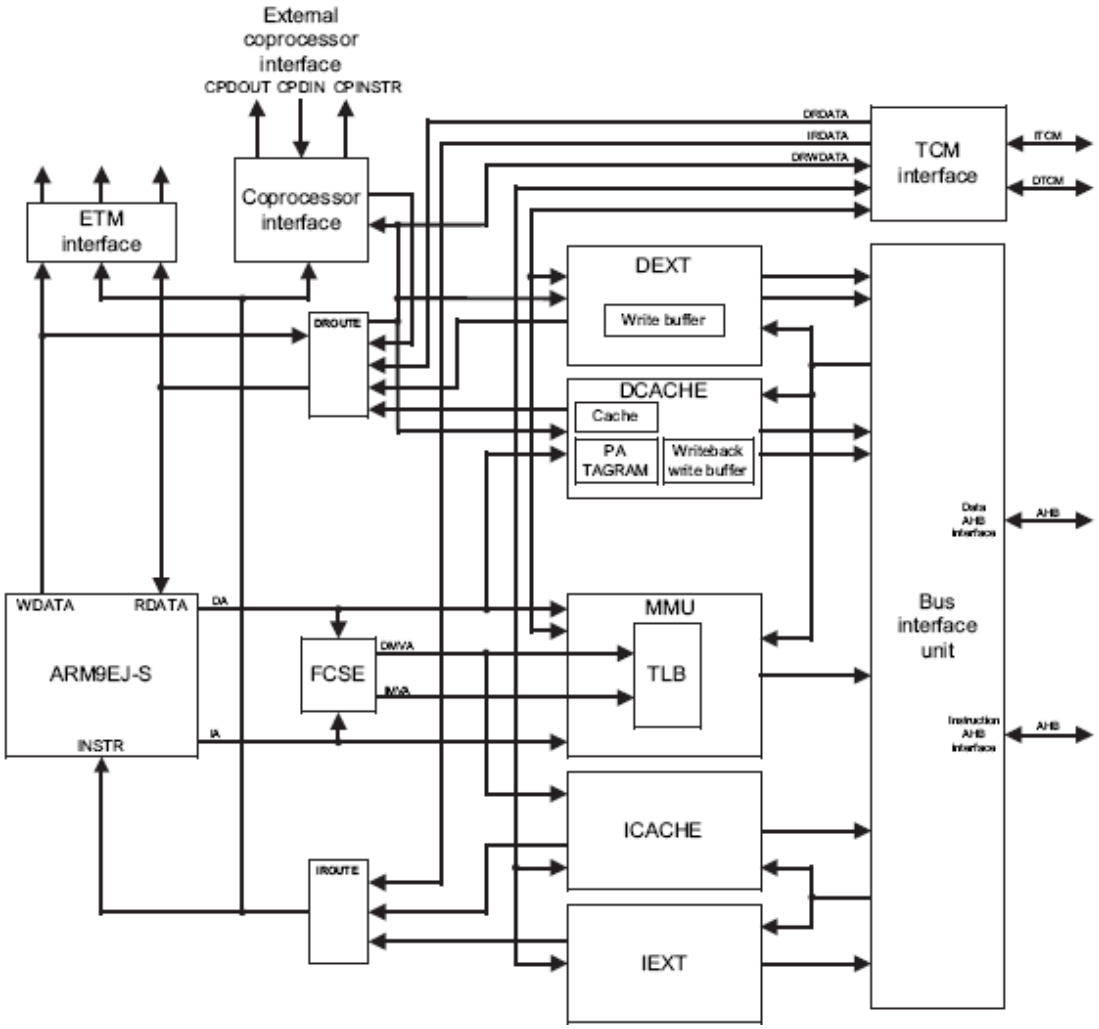
DDI0100E_ARM_ARM.pdf (ARM Architecture Reference Manual)

DDI0198D_926_TRM.pdf (ARM926EJ-S Technical Reference Manual)

二， MMU 简介

1. MMU

Memory Management Unit (MMU)可称为内存管理单元，负责管理内存访问的权限，以及虚拟内存到物理内存的映射。下图显示了ARM926内部的结构：



嵌入式系统中，存储系统差别很大，可包含多种类型的存储器件，如FLASH，SRAM，SDRAM，ROM等，这些不同类型的存储器件速度和宽度等各不相同；在访问存储单元时，可能采取平板式的地址映射机制对其操作，或需要使用虚拟地址对其进行读写；系统中，需引入存储保护机制，增强系统的安全性。为适应如此复杂的存储体系要求，ARM

处理器中引入了存储管理单元来管理存储系统。

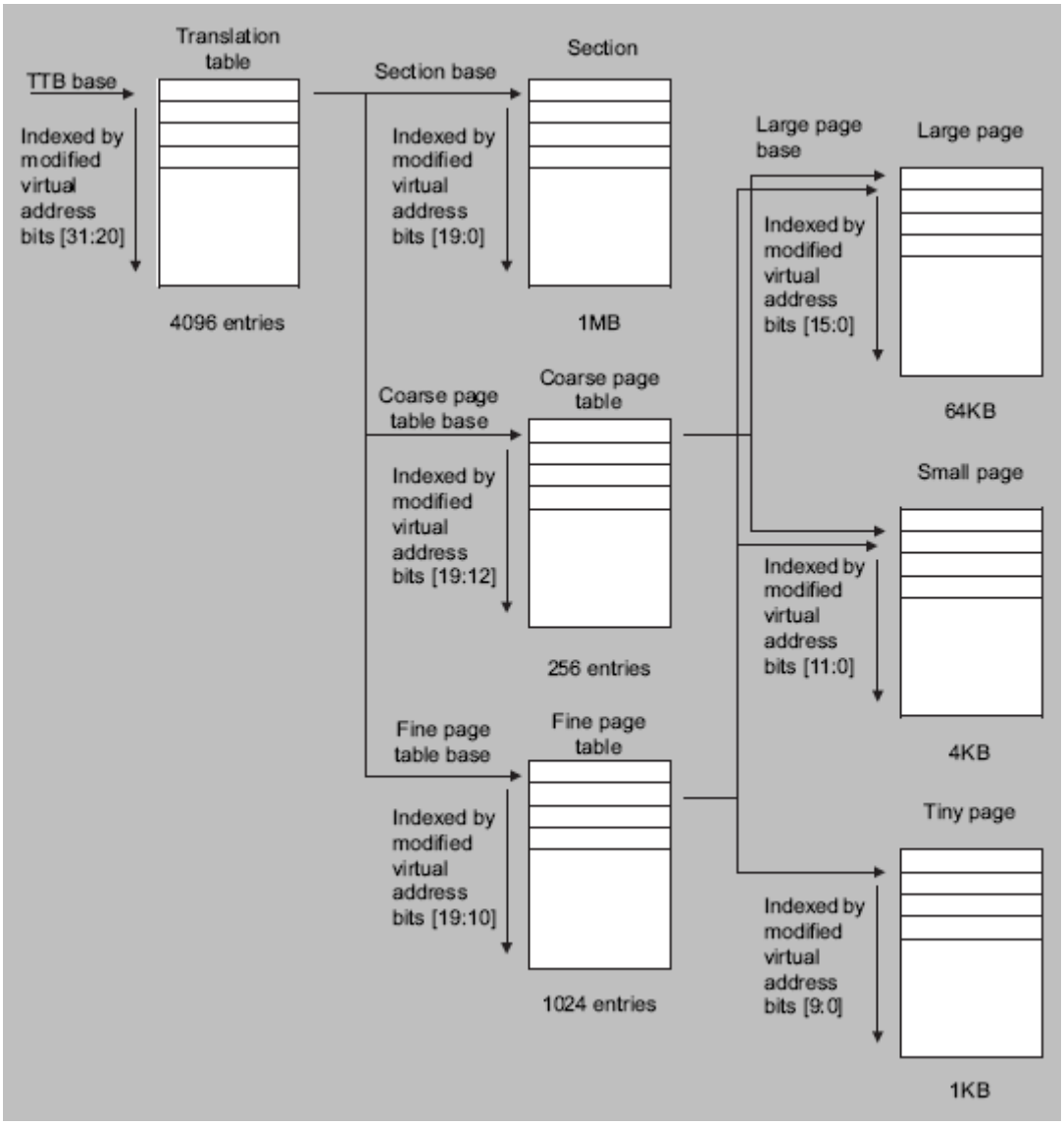
在ARM存储系统中，使用MMU实现虚拟地址到实际物理地址的映射。为何要实现这种映射？首先就要从一个嵌入式系统的基本构成和运行方式着手。系统上电时，处理器的程序指针从0x0（或者是由0xffff_0000处高端启动）处启动，顺序执行程序。程序指针（PC）的启动地址，属于非易失性存储器空间范围，如ROM、FLASH等。然而与上百兆的嵌入式处理器相比，FLASH、ROM等存储器响应速度慢，已成为提高系统性能的一个瓶颈。而SDRAM具有很高的响应速度，为何不使用SDRAM来执行程序呢？为了提高系统整体速度，可以这样设想，利用FLASH、ROM对系统进行配置，把真正的应用程序下载到SDRAM中运行，这样就可以提高系统的性能。然而这种想法又遇到了另外一个问题，当ARM处理器响应异常事件时，程序指针将要跳转到一个确定的位置，假如发生了IRQ中断，PC将指向0x18(如果为高端启动，则相应指向0vxffff_0018处)，而此时0x18处仍为非易失性存储器所占据的位置，则程序的执行还是有一部分要在FLASH或者ROM中来执行的。那么我们可不可以使程序完全都SDRAM中运行那？答案是肯定的，这就引入了MMU，利用MMU，可把SDRAM的地址完全映射到0x0起始的一片连续地址空间，而把原来占据这片空间的FLASH或者ROM映射到其它不相冲突的存储空间位置。例如，FLASH的地址从0x0000_0000－0x00ff_ffff,而SDRAM的地址范围是0x3000_0000－0x31ff_ffff，则可把SDRAM地址映射为0x0000_0000－0x1fff_ffff而FLASH的地址可以映射到0x9000_0000－0x90ff_ffff（此处地址空间为空闲，未被占用）。映射完成后，如果处理器发生异常，假设依然为IRQ中断，PC指针指向0x18处的地址，而这个时候PC实际上是从位于物理地址的0x3000_0018处读取指令。通过MMU的映射，则可实现程序完全运行在SDRAM之中。

在实际的应用中，可能会把两片不连续的物理地址空间分配给SDRAM。而在操作系统中，习惯于把SDRAM的空间连续起来，方便内存管理，且应用程序申请大块的内存时，操作系统内核也可方便地分配。通过MMU可实现不连续的物理地址空间映射为连续的虚拟地址空间。

操作系统内核或者一些比较关键的代码，一般是不希望被用户应用程序所访问的。通过MMU可以控制地址空间的访问权限，从而保护这些代码不被破坏。

MMU的实现过程，实际上就是一个查表映射的过程。建立页表（translate table）是实现MMU功能不可缺少的一步。页表是位于系统的内存中，页表的每一项对应于一个虚拟地址到物理地址的映射。每一项的长度即是一个字的长度（在ARM中，一个字的长度被定义为4字节）。页表项除完成虚拟地址到物理地址的映射功能之外，还定义了访问权限和缓冲特性等。

MMU的结构可以参考926TRM的Chapter 3，页表的顺序如下图：



本文章中的例子只完成一级section表的创建和使用，并且使用一对一映射，不采用虚拟地址。

2. MMU 与 cache

Cache是高性能CPU解决总线访问速度瓶颈的方法，然而它的使用却是需要权衡的，因为缓存本身的动作，如块拷贝和替换等，也是很消耗CPU时间的。MMU的重要性勿庸置疑，ARM926集成了MMU是其一个卖点；有了MMU，高级的操作系统（虚拟地址空间，平面地址，进程保护等）才得以实现。二者都挺复杂，并且在926中又高度耦合，相互配合操作，所以需要结合起来研究。同时，二者的操作对象都是内存，内存的使用是使用MMU/Cache的关键。另外，MMU和Cache的控制寄存器不占用地址空间，CP15是操纵MMU/Cache的唯一途径。

Cache通过预测CPU即将要访问的内存地址（一般都是顺序的），预先读取大块内存供CPU访问，来减少后续的内存总线上的读写操作，以提高速度。然而，如果程序中长跳转的次数很多，Cache的命中率就会显著降低，随之而来，大量的替换操作发生，于是，过多的内存操作反而降低了程序的性能。

ARM926内部采用哈佛结构，将内部指令总线和数据总线分开，分别连接到ICache和DCache，再通过AMBA总线接口连接到ASB总线上去访问内存。Cache由Line组成，Line是Cache进行块读取和替换的单位。

Writer Buffer是和DCache相逆过程的一块硬件，目的也是通过减少memory bus的访问来提高性能。

DCache 和 Write Buffer 需要MMU的配合来使用，当MMU被禁用时，前两者都无效，而ICache无此限制。

MMU在内存中维护一张或几张表，根据内存的划分粒度。通过CP15指定好转换表的位置，926的硬件会自动将转换表的一部分读到TLB中。CPU每次进行内存读写时，发出虚拟地址，参照TLB中的转换表转换到物理地址，并读取相应entry中的信息，以决定是否可以有权限读写和缓存。

操作MMU，实际上就是如何分配和使用系统内存，并记录在translation table里。

ARM926中，MMU的每条entry包括Cachable和Buffable位来指定相应的内存是否可以用Cache缓存和写缓冲。此处就是MMU与Cache的交互作用处。

MMU 的控制通过协处理器 CP15 来实现，请阅读参考文档了解 CP15 的寄存器。

3. TCM

Tightly-coupled Memory (TCM)称为紧密耦合内存，也是 ARM926 的一个特点。TCM 可以分为 ITCM 与 DTCM，这两部分内存直接连接到 ARM926ej-s 的 core，可以实现单周期访问，而不有额外的总线访问消耗，对于那些时间上有严格要求的代码和数据及其有利。SAM9261 实现了 TCM。需要配置 TCM 的话，需要完成以下几点：

i. 在 SAM9261 的总线矩阵控制中配置内部 RAM 为 TCM 并分配大小。
ii. 在 CP15 中配置 TCM
iii. 编写程序的时候，选择相关的代码，并将其连接到 TCM 对应的地址。运行的时候需要将这些代码加载到对应位置(可以通过 scatter loader 实现)。

注意：TCM 并不一定是最佳选择，有时使用 cache 更有效：

TCMs are intended for storing certain types of critical code or data, where low latency, deterministic access is required. TCMs are not necessarily the best choice for all types of such code or data, if code or data exhibit a high degree of spatial or temporal locality better performance may be obtained by using cache memory. (See Chapter 4 *Caches and Write Buffer*).

三， MMU 的使用

1. 使能 ICache

使能 ICache 很简单，置位 CP15 控制寄存器 C1 中的 I 位即可：

31 … 19	18	17	16	15	14	13	12	11 10	9	8	7	6 … 3	2	1	0
SBZ	SBO	SBZ	SBO	L4	RR	V	I	SBZ	R	S	B	SBO	C	A	M

代码片断如下：

```
CP15_ICACHE_ENABLE
    MRC   p15, 0, r0, c1, c0, 0
    ORR   r0, r0, #(CP15_C1_BIT_ICACHE)                    // enable I Cache
    MCR   p15, 0, r0, c1, c0, 0
    BX    LR

CP15_ICACHE_DISABLE
    MRC   p15, 0, r0, c1, c0, 0
    BIC   r0, r0, #(CP15_C1_BIT_ICACHE)                    // disable I Cache
    MCR   p15, 0, r0, c1, c0, 0
    BX    LR
```

访问协处理器的寄存器需要通过 MRC 与 MCR 指令。

2. 使能 DCache

对 DCache 的使能也可以通过置位 C1 中的 C 位来实现。但是 DCache 依赖于 MMU，即使置位了 DCache，但是 MMU 没有使能，DCache 一样无效：

Cache	MMU	Behavior
ICache disabled	Enabled or disabled	All instruction fetches are from external memory (AHB).
ICache enabled	Disabled	All instruction fetches are cachable, with no protection checks. All addresses are flat mapped. That is VA = MVA = PA.
ICache enabled	Enabled	Instruction fetches are cachable or noncachable, and protection checks are performed. All addresses are remapped from VA to PA, depending on the MMU page table entry. That is, VA translated to MVA, MVA remapped to PA.
DCache disabled	Enabled or disabled	All data accesses are to external memory (AHB).
DCache enabled	Disabled	All data accesses are noncachable nonbufferable. All addresses are flat mapped. That is VA = MVA = PA.
DCache enabled	Enabled	All data accesses are cachable or noncachable, and protection checks are performed. All addresses are remapped from VA to PA, depending on the MMU page table entry. That is, VA translated to MVA, MVA remapped to PA.

但是仍然可以编写如下的代码片断：

```
CP15_DCACHE_ENABLE
    BX    LR

CP15_DCACHE_DISABLE
    MRC   p15, 0, r0, c1, c0, 0
    BIC   r0, r0, #(CP15_C1_BIT_DCACHE)                    // disable D Cache
    MCR   p15, 0, r0, c1, c0, 0
    BX    LR
```

代码中的使能函数，是个空函数，实际也可以通过置位一个 bit 来实现，但考虑到和 MMU 的绑定，将代码放到了 MMU 的使能当中。

3. 使能 MMU

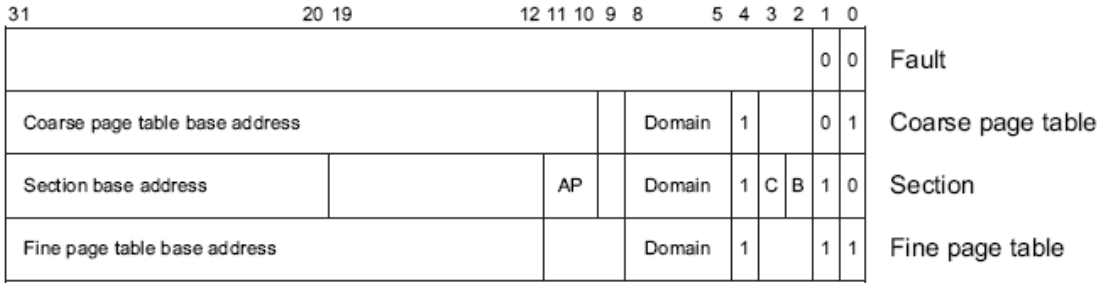
MMU 使能的代码片断如下：

```
CP15_MMU_ENABLE
    MOV   r1, #0
    MCR   p15, 0, r1, c8, c7, 0                    // Invalidate TLB
    MRC   p15, 0, r0, c1, c0, 0
    ORR   r0, r0, #(CP15_C1_BIT_DCACHE)            // enable D Cache
    ORR   r0, r0, #(CP15_C1_BIT_MMU)               // enable MMU
    MCR   p15, 0, r0, c1, c0, 0
    NOP
    NOP
    BX    LR

CP15_MMU_DISABLE
    MRC   p15, 0, r0, c1, c0, 0
    BIC   r0, r0, #(CP15_C1_BIT_DCACHE)            // disable D Cache
    BIC   r0, r0, #(CP15_C1_BIT_MMU)               // disable MMU
    MCR   p15, 0, r0, c1, c0, 0
    NOP
    NOP
    BX    LR
```

需要注意的问题是，在使能 MMU 之前，必须要将 translation table 的首地址设置到 TTB 中，也就是 C2。配置 TTB 的代码如下：

```
CP15_TTB_SET
    LDR   r1, =0x00003FFF
    BIC   r0, r0, r1                  // [31...14] TTB, [13...0] SBZ.
    MCR   p15, 0, r0, c2, c0, 0
    BX    LR
```

在测试代码里面，将 table 设置到 0x20000000。除了设置 TTB，还需要建立 translation table。测试代码中，只建立一级的 section table：



Section table 为每个 1MB 的内存空间设置映射关系和访问控制，因此共需要 4GB/1MB = 4096 个表项。参照手册，为 SAM9261 系统建立一个 section table，该 table 中只包含有意义的部分，其余无意义的部分需要使用不可访问来初始化。本例子中的表项使用虚拟地址和实际地址一样的配置。

表项的代码片断如下：

```
typedef struct _section_tbl_item
{                               /* setcion size is 1MB */
    unsigned int section_id;    /* section index, from 0  to  4096-1 */
  unsigned int section_pa;     /* physical address for this section */
  unsigned int section_ctrl;  /* section property: C, B, AP, domain */
}Section_tbl_Item, pSection_tbl_Item;

const Section_tbl_Item Valid_Section_Table[] = {{0x000, 0x00000000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                                                    (1 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
                                                                    CP15_MMU_LVL1_DESC_BIT_C}, /*  vector  */
                        {0x003, 0x00300000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                                                    (1 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
                                                                    CP15_MMU_LVL1_DESC_BIT_C}, /*  Internal RAM */
                        {0x004, 0x00400000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                                                    (1 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
                                                                    CP15_MMU_LVL1_DESC_BIT_C}, /*  Internal ROM  */
                        {0x005, 0x00500000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                                                    (1 << CP15_MMU_DOMAIN_CTRL_OFFSET)}, /*  UHP  */
                        {0x006, 0x00600000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                                                    (1 << CP15_MMU_DOMAIN_CTRL_OFFSET)}, /*  LCD */
                                                {0x200, 0x20000000, (AP_PRV_RW_USR_NO << CP15_MMU_AP_OFFSET)| \
                                            (1 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
                                      CP15_MMU_LVL1_DESC_BIT_C}, /* Section Table */
                        {0x215, 0x21500000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                            (0 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
                                           CP15_MMU_LVL1_DESC_BIT_C | \
                                           CP15_MMU_LVL1_DESC_BIT_B}, /* code area */
                        {0x216, 0x21600000, (AP_PRV_RW_USR_RW << CP15_MMU_AP_OFFSET)| \
                                            (0 << CP15_MMU_DOMAIN_CTRL_OFFSET) | \
```

Translation table 的初始化代码片断如下：

```
 section_tbl = (unsigned int *)pTTB;

 /* 1. set Translation Table Base */
   CP15_TTB_SET((unsigned int)pTTB);

 /* 2. clear all the items in the table */
 for (; i < CP15_MMU_SECTION_TBL_SIZE; i++)
 {
       section_tbl[i] = 0;
   }

   /* 3. set valid items */
   len = sizeof(Valid_Section_Table) / sizeof(Section_tbl_Item);

 for (i = 0; i < len; i ++)
 {
       section_tbl[Valid_Section_Table[i].section_id] = Valid_Section_Table[i].section_pa | \
                                                        Valid_Section_Table[i].section_ctrl | \
                             (1 << 4) | \
                             CP15_MMU_LVL1_DESC_SECTION;
 }

 CP15_ACCESS_CTRL((CP15_C3_ACCESS_MANAGER << 2) | CP15_C3_ACCESS_CLIENT);

 CP15_MMU_ENABLE();
```

还需要配置 domain access 控制寄存器 C3。代码片断如下：

```
CP15_ACCESS_CTRL_READ
    MRC    p15, 0, r0, c3, c0, 0
    BX     LR

CP15_ACCESS_CTRL_WRITE
    MCR    p15, 0, r0, c3, c0, 0
    BX     LR
```

Domain access control 将整个 ARM 的 4GB 内存空间配置成 16 个 domain，对每个 domain 可以独立控制。

完成这些操作后，MMU 可以使用。

四， 编译和调试 MMU 代码

1. 编译代码

代码在硬件初始化完成后，需要使能 MMU，代码片断如下：

```
#if defined(ROM_RUN)
    CP15_ENABLE_MMU((void *)0x20000000); /*  hold the section table in 0x20000000 */
#endif
```

在使能 cache 和 MMU 之前，需要 flush cache 的表项和 TLB。

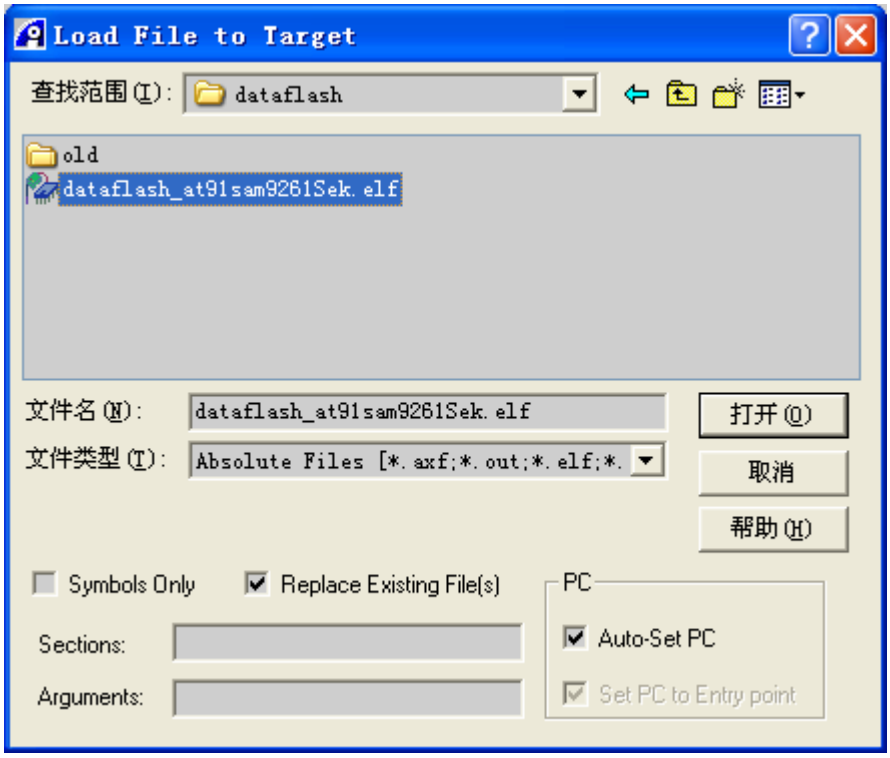
使用 realview 编译代码，完成后生成 axf 文件。

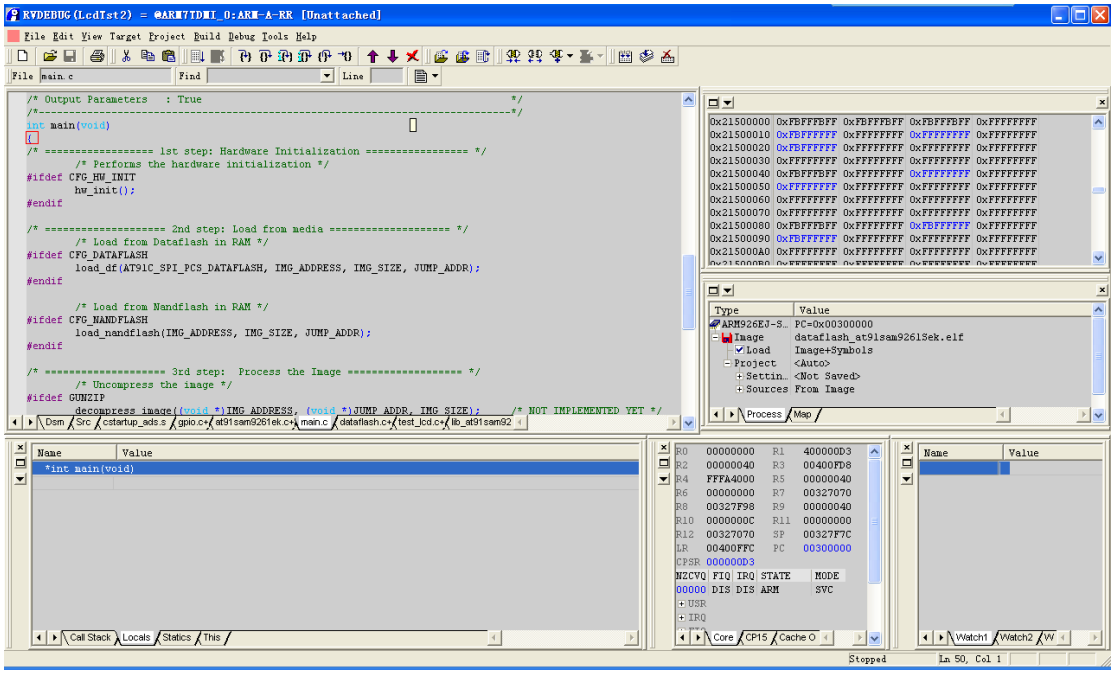
2. 调试 MMU 代码

由于该测试代码需要在 SDRAM 中运行，而板子上的 SDRAM 在没有初始化前不可用，因此可以先 debug 一个能初始化 SDRAM 的代码。可以使用 ATMEL 提供的 bootstrap 的代码。

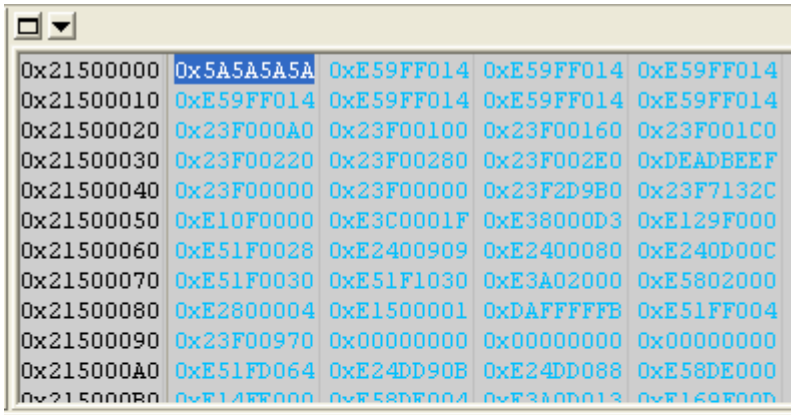
连接好板子上的 DBGU 到 PC 机的终端，设置为 115200,n,8,1。

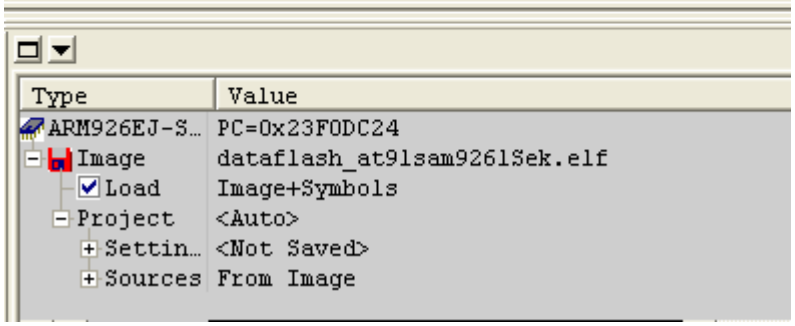
在 RVD 中加载 data flash 使用的 bootstrap 的 elf 文件：



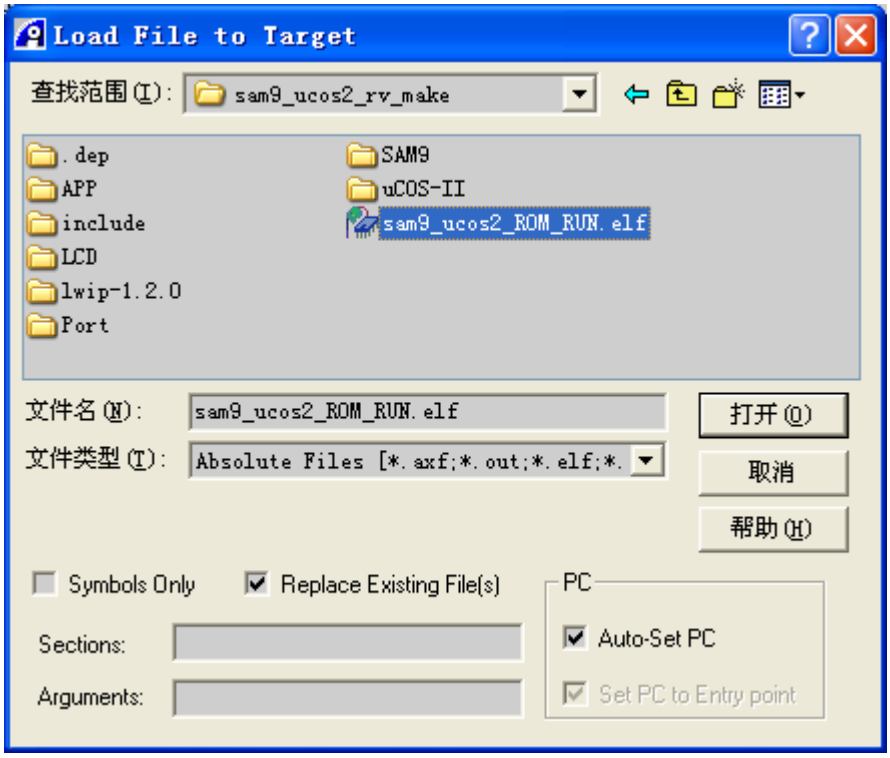


点击运行，使得代码完成对 SDRAM 的初始化，在 memory 窗口中修改一个 SDRAM 单元，以验证其可用：

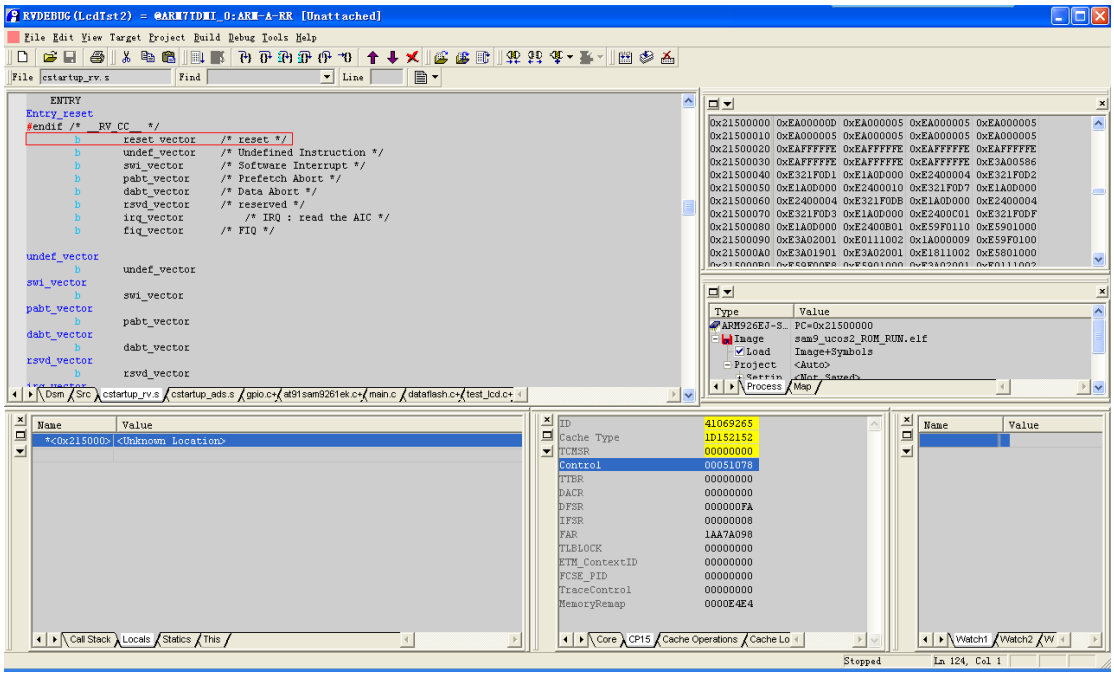




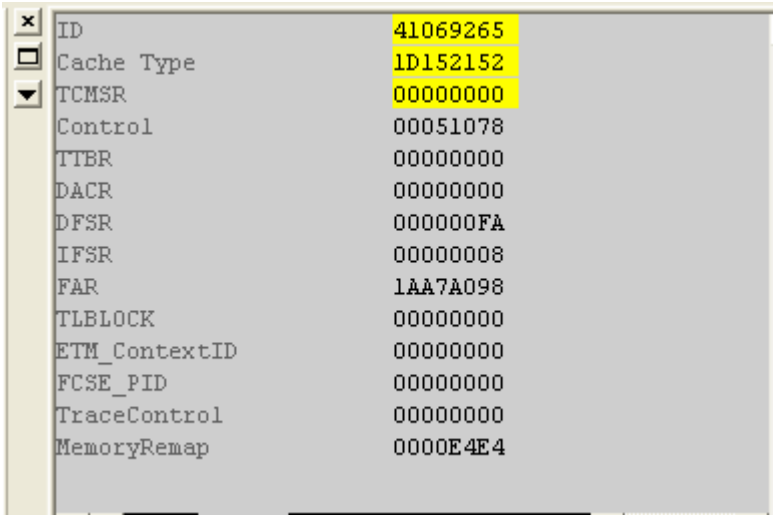
SDRAM 就绪后就可以加载本次测试的代码：



注意此时的窗口布局：



打开寄存器窗口，可以检查 CP15 的寄存器当前的状态：

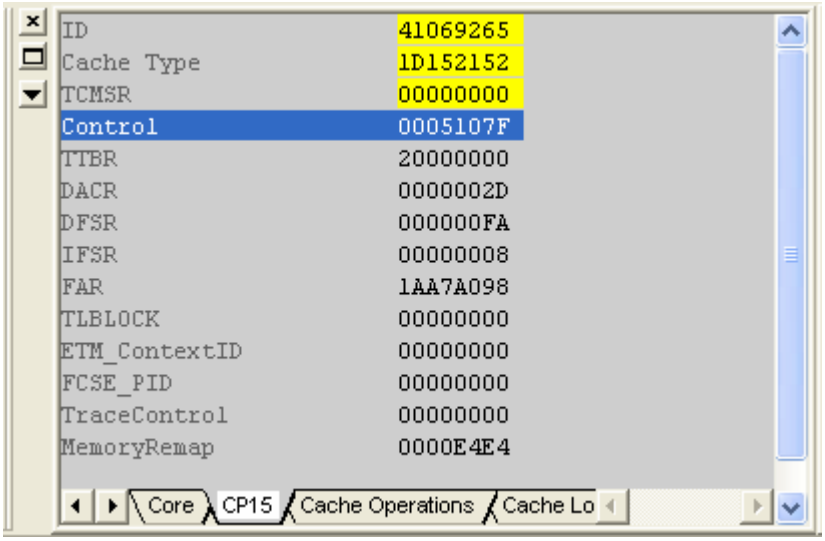


ID	41069265
Cache Type	1D152152
TCMSR	00000000
Control	00051078
TTBR	00000000
DACR	00000000
DFSR	000000FA
IFSR	00000008
FAR	1AA7A098
TLBLOCK	00000000
ETM_ContextID	00000000
FCSE_PID	00000000
TraceControl	00000000
MemoryRemap	0000E4E4

完成 MMU 的初始化代码之后：

```
    printf("\n\rSYS info:\n\r");
    clk = BSP_PCK_Frq();
    printf("PCK = %lOdHz\n\r", clk);
    clk = BSP_MCK_Frq();
    printf("MCK = %lOdHz\n\r", clk);

    //CP15_SET_ITCM(0x00100000, 0x10000);

#if defined(ROM_RUN)
    CP15_ENABLE_MMU((void *)0x20000000); /*  hold the section table in 0x20000000 */
#endif

    CP15 Get Info();
    printf(BANNER);
```

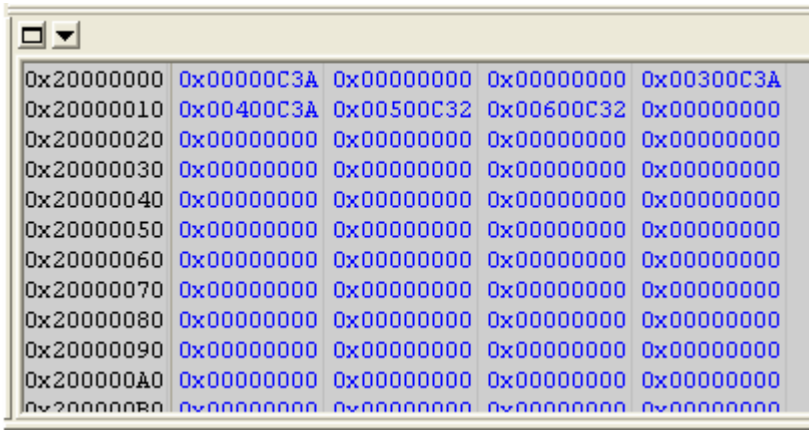
可以发现 CP15 寄存器中的变化：



ID	41069265
Cache Type	1D152152
TCMSR	00000000
Control	0005107F
TTBR	20000000
DACR	0000002D
DFSR	000000FA
IFSR	00000008
FAR	1AA7A098
TLBLOCK	00000000
ETM_ContextID	00000000
FCSE_PID	00000000
TraceControl	00000000
MemoryRemap	0000E4E4

Core | CP15 | Cache Operations | Cache Lo

Translation table 的地址以及 MMU 和 Cache 均已使能，同时 SDRAM 的 0x2000000 处包含 section table：



继续运行代码，让其输出 CP15 的相关信息：

```
SYS info:
PCK =  198656000Hz
MCK =   99328000Hz

CP15 C0 ID code: 0x41069265
ARM Arch. V5TEJ, 926ejs, Rev.5
CP15 C0 cache Type: 0x1d152152
Separate ICache & DCache.
ICache: 16kB, 4-way associate,line length:8words
DCache: 16kB, 4-way associate,line length:8words
DTCM not present.
ITCM not present.
CP15 C1 status: 0x   5107f
Vector starts from 0x00000000.
Replacement strategy for ICache and DCache: Round-robin.
Little-endian operation.
ICache enabled.
DCache enabled.
MMU    enabled.
```

OK，MMU 已经使能。

注意，该测试代码如果使用不支持 MMU 的的仿真器调试，将会出现 data abort。

五， 一些事项

理论上说，使用 cache 和 write buffer 可以大幅提升代码执行性能，使用 MMU 可以为系统提供安全性，但是也是有一些地方需要注意。

1） 当在无 OS 的裸机上开发程序时，初始化运行环境的代码很重要，比如：各种模式堆栈指针的初始化；将代码和 RW data 从 ROM 拷贝到 RAM；初始化.bss 段（zero initialized）空间等。此时会有大量的内存操作，如果 enable 了 Cache，那么在拷贝完代码之后，一定要 invalidate ICache 和 flush DCache。否则将会出现缓存中的代码或数据与内存中的不一致，程序跑飞。

2） 对硬件的操作要小心。很多寄存器值都是被硬件改变的，读写时，要保证确实访问到它的地址。首先，在 C 语言代码中声明为 volatile 变量，以防止内存读写被编译器优化掉；另外，设置好 TLB，使得寄存器映射的地址空间不被缓存。总之，

需要保证 cache 中的数据和主内存中的一致性。

3） 程序员应当很清楚自己的程序中，那里有大量的运算，哪里有无数的循环或递归，而这正是 Cache 的用武之地，将这些空间进行缓存将大大提高运行速度。但是，很多函数或子程序往往仅仅运行很少几次，若是对它们也缓存，只会捡了芝麻丢了西瓜，造成不必要的缓存和替换操作，反而增加了系统负担，降低了整体性能。

第十章：使用 U-boot 加载用户应用

本章叙述通过 U-boot 在本站的 9261 开发板上加载并运行用户代码的过程，包括了代码下载与运行过程。

一、准备工作

1. 编译用户应用

使用任意编译器编译好用户应用。需要注意的是，代码要求是在 SDRAM 中运行，并且要确定好代码的连接地址。这个地址将是 U-boot 加载代码到 SDRAM 的地址。

代码编译完成后，需要使用对应的调试工具，保证代码在 SDRAM 中可以正常运行。这也是进行下面测试的前提。

最后，将代码生成一个 bin 文件。

本文使用 uC/GUI 作为例子，这里将生成 bin 文件 sam9_QVGA_ROM_RUN.bin。这个是在 SDRAM 中运行的代码，连接地址为 0x21500000。

板子需要连接好 LCD 以测试 uC/GUI。

2. 编译 U-boot

请参考本站另一篇文章**《为 SAM926X 编译 U-boot》**为 9261 编译好 u-boot。

注意编译的 U-boot 需要支持 U 盘。

U-boot 就绪后，将一级 boot 和 U-boot 烧写到 9261 板子上，确定 U-boot 可以正确启动。

二、通过 U 盘加载代码

将一个U盘连接到PC，可以格式化为FAT32格式。将sam9_QVGA_ROM_RUN.bin复制到U盘根目录，从PC上弹出U盘后将U盘连接到9261开发板的USB HOST插座。启动U-boot并停在命令行，如下：

```
Hit any key to stop autoboot:  0
U-Boot>
```

运行USB初始化命令：

```
U-Boot> usb start
(Re)start USB...
USB:   scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
U-Boot> _
```

U盘被正确识别和初始化。可以使用下列命令列出U盘上的文件目录：

```
U-Boot> fatls usb 0:1 /
   513544   ucguiexp
   513544   sam9_qvga_rom_run.bin

2 file(s), 0 dir(s)
```

使用下面的命令加载代码到其运行时的地址0x21500000：

```
U-Boot> fatload usb 0:1 21500000 sam9_qvga_rom_run.bin
reading sam9_qvga_rom_run.bin
.................................................

513544 bytes read
```

使用下面命令运行代码：

```
U-Boot> go 21500000
## Starting application at 0x21500000 ...? 櫂括蕙Cゎ? ⊙8? @筝囍E? 蝜8珉? ))))))
```

后面出现的乱码说明串口的设置并不匹配，目前可以不用理会。如果要求看到代码输出，应当使得代码的设置与U-boot一致，或者在用户代码中不再初始化串口而延用U-boot的设置,还需要注意时钟的设置是否一致，波特率与MCK有关。

此时可以看到代码已经启动，LCD上有正确的显示。

可以设置环境变量，使得代码在板子启动时自动加载U盘上的代码运行：

```
U-Boot> set bootcmd run usb_init\;run ker\;go 21500000
U-Boot> saveenv
Saving Environment to dataflash...
U-Boot> printenv
bootdelay=3
baudrate=115200
usb_init=usb start
usb_ls=fatls usb 0:1 /
fs=fatload usb 0:1 21100000 armv5l-uclibc-sam9260
bootargs=mem=64M console=tty0,115200 initrd=0x21100000,4M root=/dev/ram0 rw
ker=fatload usb 0:1 21500000 sam9_qvga_rom_run.bin
stdin=serial
stdout=serial
stderr=serial
bootcmd=run usb_init;run ker;go 21500000
```

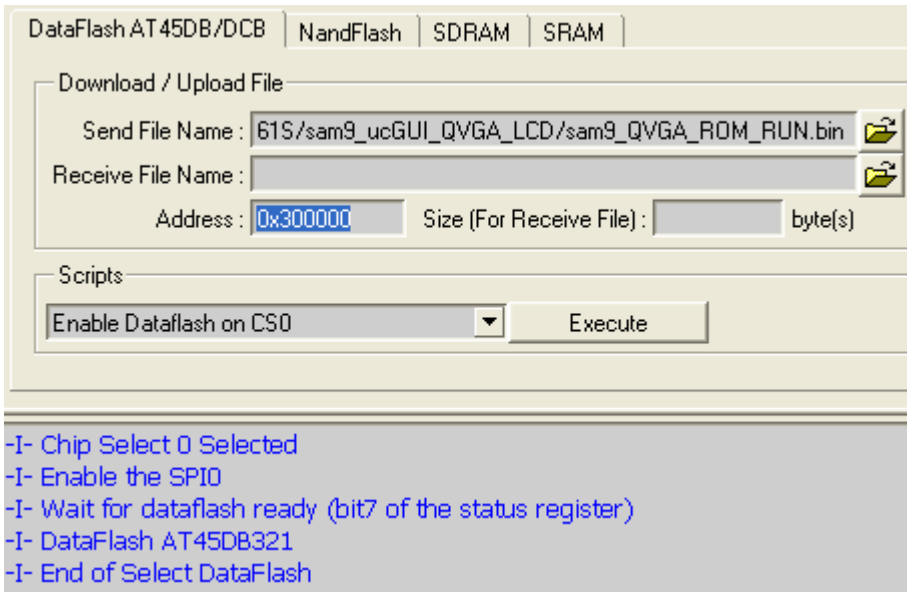
使用saveenv命令保存环境变量。然后按下复位键，等待系统自动启动用户代码。

三、通过板上存储器加载代码

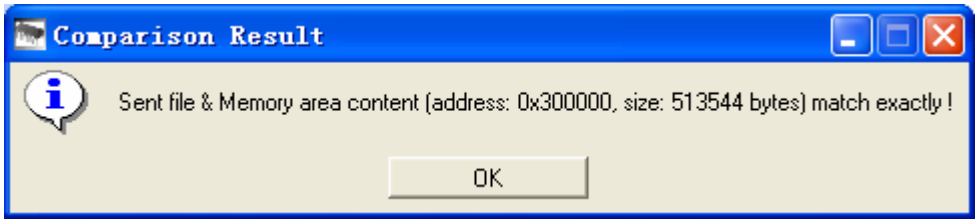
1. 将代码下载到 data flash 并加载运行

如果觉得平时的应用中使用 U 盘加载不方便，可以将代码烧到 data flash。烧写的板子可以有以下几种：使用 SAM－BA 烧写；使用 U-boot 烧写。其中后者可以使用 U 盘或者串口先下载被烧录的代码到 SDRAM 再烧录。使用 U-boot 的烧写方式可以参考本站 U-boot 的应用文档，本文介绍使用 SAM-BA 烧写的过程。

首先将 data flash 启动跳线去除，然后启动 SAM－BA，再连接好 data flash 启动跳线。选择应用文件，并烧录到 data flash 的某个地址，该地址不能覆盖一级 boot，U-boot 以及环境变量，这里设置为 0x300000(45DB321 的 3MB 位置)：



烧写完成后可以进行校验，以保证烧写正确：



关闭 SAM-BA 按下复位键，U-boot 应能正确启动：

```
?  NAND 128MiB 3,3V 8-bit)
NAND: Pagesize: 2048, Blocksize: 128K, OOBsize: 64
128 MiB
DataFlash:AT45DB321
Nb pages:   8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0003FFF (RO)
Area 1: C0004000 to C0007FFF
Area 2: C0008000 to C0037FFF (RO)
Area 3: C0038000 to C041FFFF
In:    serial
Out:   serial
Err:   serial
dm9000 i/o: 0x30000000, id: 0x90000a46
MAC: 00:00:00:00:00:00
could not establish link
Hit any key to stop autoboot:  0
U-Boot>
```

注意 data flash 地址的映射关系。

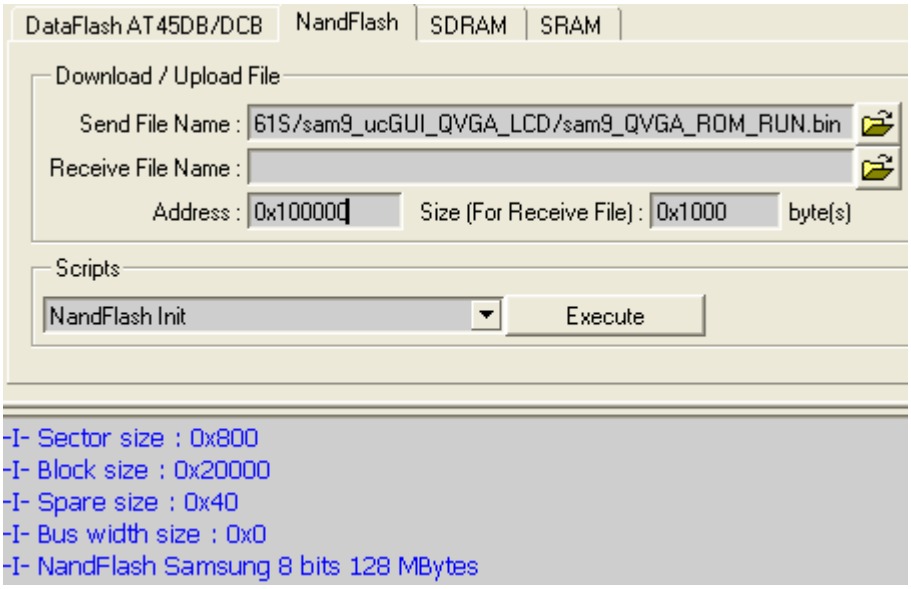
设置如下的环境变量，使得 U-boot 从 data flash 加载用户代码：

```
U-Boot> set bootcmd cp.b c0300000 21500000 100000\;go 21500000
U-Boot> saveenv
Saving Environment to dataflash...
U-Boot> printenv
bootdelay=3
baudrate=115200
usb_init=usb start
usb_ls=fatls usb 0:1 /
fs=fatload usb 0:1 21100000 armv5l-uclibc-sam9260
bootargs=mem=64M console=tty0,115200 initrd=0x21100000,4M root=/dev/ram0 rw
ker=fatload usb 0:1 21500000 sam9_qvga_rom_run.bin
stdin=serial
stdout=serial
stderr=serial
bootcmd=cp.b c0300000 21500000 100000;go 21500000

Environment size: 338/16380 bytes
```

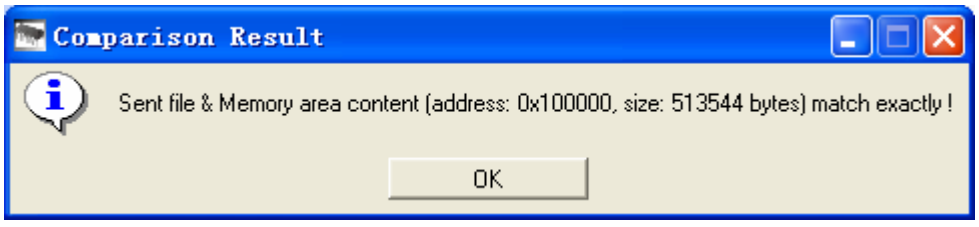
按下复位键将可以看到代码被自动从 data flash 复制到内存并运行。

2. 将代码下载到 nand flash 并加载运行

首先使用 SAM-BA 将代码烧写到 NAND flash 的某个地址，比如 0x100000(1MB)：



烧写完成也可以验证一下：



启动 U-boot 后设置如下的环境变量：

```
U-Boot> set bootcmd nand read 100000 21500000 100000\;go 21500000
U-Boot> saveenv
Saving Environment to dataflash...
U-Boot> printenv
bootdelay=3
baudrate=115200
usb_init=usb start
usb_ls=fatls usb 0:1 /
fs=fatload usb 0:1 21100000 armv5l-uclibc-sam9260
bootargs=mem=64M console=tty0,115200 initrd=0x21100000,4M root=/dev/ram0 rw
ker=fatload usb 0:1 21500000 sam9_qvga_rom_run.bin
stdin=serial
stdout=serial
stderr=serial
bootcmd=nand read 100000 21500000 100000;go 21500000

Environment size: 341/16380 bytes
```

按下复位键即可。

```
Hit any key to stop autoboot:  0

NAND read: device 0 offset 558891008, size 1048576 ...  0 bytes read: ERROR
## Starting application at 0x21500000 ...? 潤栏蕙C$钗(08? @佮轠M? 蟥8皖? )))))))
```

四、一些事项

本文简述了 U-boot 加载和启动用户代码的一些方法，更高级的应用可以在实践中摸索和学习。

使用文中的方式，使用 U-boot 加载的代码，不能引用 U-boot 的功能。

从前文可以看出，使用 U-boot 加载代码运行，使用一级 boot 也可以做到。但是 U-boot 作为一个功能强大的通用 loader，还有很多别的功能，比如使用 U 盘加载代码，可以方便地对代码进行测试。

第十一章：在 SAM9261 上使用 miniGUI

—基于 Virtual PC

本文叙述在 SAM9261 上使用 miniGUI 的过程。包含了 miniGUI 的配置与编译，tslib，jpeg，png lib 的配置与编译，miniGUI 的应用，以及触摸屏的使用。miniGUI 选择根据 GPL 发布的 1.6.10 版，Linux 版本选择了 2.6.26, 开发环境选择 Virtual PC，其它软件可以类推。本文基于 SAM9261，但也适用于本站的 SAM9263。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机
   请参照本站另一篇文档《基于 VPC 建立 ARM_Linux 开发环境》。

2. 下载相关软件包
   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。
   libminigui-1.6.10.tar.gz
   mde-1.6.10.tar.gz
   mg-samples-1.6.10.tar.gz
   minigui-res-1.6.10.tar.gz
   popt-1.7.tar.gz
   tslib-1.0.tar.bz2
   jpegsrc.v6b.tar.gz
   libpng-1.2.26.tar.gz
   edillo.tar.gz

3. 准备相关环境
   请参照本镇另一篇文章《9261 上的 Linux 初步应用 3》设置好主机的 NFS 等环境。

二，编译 miniGUI

1．编译 tslib

展开 tslib 源码包：

```
[root@Linux4SAM miniGUI]# tar xjvf tslib-1.0.tar.bz2
```

运行 autogen.sh:

```
[root@Linux4SAM tslib-1.0]# ./autogen.sh
```

这里需要多做一步，避免后面编译出现问题：

```
[root@Linux4SAM tslib-1.0]# echo "ac_cv_func_malloc_0_nonnull=yes" >arm-none-linux-gnueabi.cache
```

配置，用不到的设备都禁用：

```
[root@Linux4SAM tslib-1.0]# export CFLAGS=-DUSE_INPUT_API
[root@Linux4SAM tslib-1.0]# export CFLAGS="$CFLAGS -mcpu=arm926ej-s"
[root@Linux4SAM tslib-1.0]# echo $CFLAGS
-DUSE_INPUT_API -mcpu=arm926ej-s
[root@Linux4SAM tslib-1.0]#
```

```
[root@Linux4SAM tslib-1.0]# cat arm-none-linux-gnueabi.cache
ac_cv_func_malloc_0_nonnull=yes
[root@Linux4SAM tslib-1.0]# ./configure  --build=i386-linux --host=arm-none-linux-gnueabi --target=arm-none-linux-gnueabi --enable-input --disable-collie --disable-corgi --disable-ucb1x00 --disable-h3600 --disable-mk712 --disable-arctic2 --prefix=$PWD/_install --cache-file=arm-none-linux-gnueabi.cache
```

清楚起见，将编译的输出先放置到_install。然后就是 make 与 make install。安装后在指定目录下生成相关文件：

```
[root@Linux4SAM _install]# pwd
/usr/work/app/miniGUI/tslib-1.0/_install
[root@Linux4SAM _install]# ls
bin  etc  include  lib
[root@Linux4SAM _install]# ls bin/
ts_calibrate  ts_harvest  ts_print  ts_print_raw  ts_test
[root@Linux4SAM _install]# file bin/ts_test
bin/ts_test: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux
s), not stripped
[root@Linux4SAM _install]#
```

复制文件到工具链目录：

```
[root@Linux4SAM _install]# pwd
/usr/work/app/miniGUI/tslib-1.0/_install
[root@Linux4SAM _install]# ls
bin  etc  include  lib
[root@Linux4SAM _install]# ls include/
tslib.h
[root@Linux4SAM _install]# cp -a ./* /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/include/
[root@Linux4SAM _install]# cd lib
[root@Linux4SAM lib]# ls
libts-0.0.so.0  libts-0.0.so.0.1.1  libts.la  libts.so  pkgconfig  ts
[root@Linux4SAM lib]# cp -f libts.la /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib
[root@Linux4SAM lib]# cp -af libts-0.0.so.0* libts.so ts /opt/codesourcery/arm-none-linux-gnueabi/libc/lib/
```

将相关文件传输到 NFS 目录下以测试：

```
[root@Linux4SAM _install]# mkdir /nfs4arm/rootfs/usr/tslib
[root@Linux4SAM _install]# cp bin/* /nfs4arm/rootfs/usr/tslib
[root@Linux4SAM _install]#
```

```
[root@Linux4SAM lib]# ls -al
total 32
drwxr-xr-x    4 root     root         4096 Sep 19 09:31 .
drwxr-xr-x    6 root     root         4096 Sep 19 09:31 ..
lrwxrwxrwx    1 root     root           18 Sep 19 09:31 libts-0.0.so.0 -> libts-0.0.so.0.1.1
-rwxr-xr-x    1 root     root        10297 Sep 19 09:31 libts-0.0.so.0.1.1
-rwxr-xr-x    1 root     root          732 Sep 19 09:31 libts.la
lrwxrwxrwx    1 root     root           18 Sep 19 09:31 libts.so -> libts-0.0.so.0.1.1
drwxr-xr-x    2 root     root         4096 Sep 19 09:31 pkgconfig
drwxr-xr-x    2 root     root         4096 Sep 19 09:31 ts
[root@Linux4SAM lib]# cp libts-0.0.so.0.1.1 /nfs4arm/rootfs/lib/
[root@Linux4SAM lib]# cd /nfs4arm/rootfs/lib/
```

```
[root@Linux4SAM lib]# ln -s libts-0.0.so.0.1.1 libts-0.0.so.0
[root@Linux4SAM lib]# ln -s libts-0.0.so.0.1.1 libts.so
[root@Linux4SAM lib]# ls -al libts*
lrwxrwxrwx    1 root     root           18 Sep 19 09:40 libts-0.0.so.0 -> libts-0.0.so.0.1.1
-rwxr-xr-x    1 root     root        10297 Sep 19 09:39 libts-0.0.so.0.1.1
lrwxrwxrwx    1 root     root           18 Sep 19 09:40 libts.so -> libts-0.0.so.0.1.1
[root@Linux4SAM lib]#
```

```
[root@Linux4SAM _install]# ls
bin  etc  include  lib
[root@Linux4SAM _install]# cd etc/
[root@Linux4SAM etc]# ls
ts.conf
[root@Linux4SAM etc]# cp ts.conf /nfs4arm/rootfs/etc/
[root@Linux4SAM etc]#
```

```
[root@Linux4SAM lib]# cd ts/
[root@Linux4SAM ts]# pwd
/usr/work/app/miniGUI/tslib-1.0/_install/lib/ts
[root@Linux4SAM ts]# ls
dejitter.la  input.la  linear_h2200.la  linear.la  pthres.la  variance.la
dejitter.so  input.so  linear_h2200.so  linear.so  pthres.so  variance.so
[root@Linux4SAM ts]# mkdir /nfs4arm/rootfs/lib/ts
[root@Linux4SAM ts]# cp *.so /nfs4arm/rootfs/lib/ts
[root@Linux4SAM ts]#
```

为了在目标板上测试，首先需要配置环境：

```
[root@mcuzone tslib]#export TSLIB_TSDEVICE=/dev/input/event1
[root@mcuzone tslib]#export TSLIB_CALIBFILE=/etc/pointercal
[root@mcuzone tslib]#export TSLIB_CONFFILE=/etc/ts.conf
[root@mcuzone tslib]#export TSLIB_PLUGINDIR=/lib/ts
[root@mcuzone tslib]#export TSLIB_CONSOLEDEVICE=none
[root@mcuzone tslib]#export TSLIB_FBDEVICE=/dev/fb0
[root@mcuzone tslib]#export LD_LIBRARY_PATH=/lib:/usr/lib
[root@mcuzone tslib]#
```

可以将这些变量放置到/etc/profile 中，每次开机都会设置。

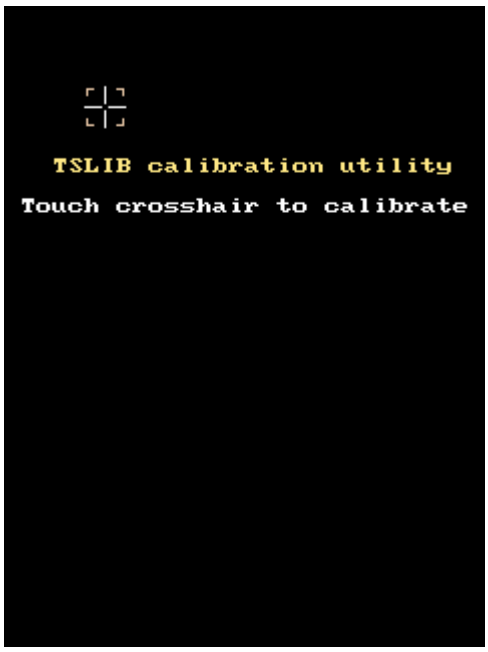
修改配置文件 ts.conf：

```
module_raw input
module pthres pmin=1
module variance delta=30
module dejitter delta=100
module linear
```

运行 ts_calibrate:

```
[root@mcuzone tslib]#ls
ts_calibrate  ts_harvest    ts_print      ts_print_raw  ts_test
[root@mcuzone tslib]#./ts_calibrate
xres = 240, yres = 320
```

屏幕上显示：



使用笔点击对应点，终端会有相应输出：

```
Took 31 samples...
Top left : X = 3006 Y = 3213
```

依次点击校准点，完成校准：

```
Took 31 samples...
Top left : X = 3006 Y = 3213
Took 28 samples...
Top right : X =  931 Y = 3196
Took 21 samples...
Bot right : X = 1043 Y =  864
Took 21 samples...
Bot left : X = 2986 Y =  821
Took 23 samples...
Center : X = 1995 Y = 2024
```

最后程序会记录与屏幕相关的校准常量：

```
261.385864 -0.069618 -0.001330
349.578766 -0.000556 -0.093137
Calibration constants: 17130184 -4562 -87 22909994 -36 -6103 65536
```

并将其保存到相关文件中：

```
[root@mcuzone tslib]#cat /etc/pointercal
-4562 -87 17130184 -36 -6103 22909994 65536[root@mcuzone tslib]#_
```

运行 ts_test 进行测试

为了在 minigui 下使用触摸屏，需要利用 tslib 写一个板子适用的 ial。可以修改 libminigui-1.6.10/src/ial/fxrm9200.c 来实现。这个文件本身是个空文件，这里只是利用编译框架，避免新增文件需要改动 make 相关文件。

编译完成的 tslib 相关的头文件，库需要复制到工具链对应目录。

2．编译 jpeg lib

展开源码包 jpegsrc.v6b.tar.gz。

配置：

```
[root@Linux4SAM jpeg-6b]# cp /usr/share/libtool/config.sub ./
[root@Linux4SAM jpeg-6b]# cp /usr/share/libtool/config.guess ./
[root@Linux4SAM jpeg-6b]#
```

```
[root@Linux4SAM jpeg-6b]# ./configure --prefix=./_install/ CC=arm-none-linux-gnueabi-gcc --enable-shared --enable-stati
c CFLAGS="-mcpu=arm926ej-s -O3"
```

修改生成的 Makefile 中 AR 与 AR2 为对应的工具：

```
MKDIR= mkdir
# library (.a) file creation command
AR= arm-none-linux-gnueabi-ar rc
# second step in .a creation (use "touch" if not needed)
AR2= arm-none-linux-gnueabi-ranlib
# installation program
INSTALL= /usr/bin/install -c
```

执行 make：

```
[root@Linux4SAM jpeg-6b]# ln -s /usr/bin/libtool ./libtool
[root@Linux4SAM jpeg-6b]# make
```

编译完成后运行 make install-lib:

```
[root@Linux4SAM jpeg-6b]# mkdir _install/include
[root@Linux4SAM jpeg-6b]# mkdir _install/lib
[root@Linux4SAM jpeg-6b]# make install-lib
```

需要复制这些库与头文件到工具链的对应目录下：

```
[root@Linux4SAM include]# ls
jconfig.h  jerror.h  jmorecfg.h  jpeglib.h
[root@Linux4SAM include]# cp -a *.h /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/include/
[root@Linux4SAM include]#
```

```
[root@Linux4SAM lib]# ls -al
total 400
drwxr-xr-x    2 root     root         4096 Sep 20 21:54 .
drwxr-xr-x    5 root     root         4096 Sep 20 21:54 ..
-rw-r--r--    1 root     root       212874 Sep 20 21:54 libjpeg.a
-rwxr-xr-x    1 root     root          504 Sep 20 21:54 libjpeg.la
lrwxrwxrwx    1 root     root           17 Sep 20 21:54 libjpeg.so -> libjpeg.so.62.0.0
lrwxrwxrwx    1 root     root           17 Sep 20 21:54 libjpeg.so.62 -> libjpeg.so.62.0.0
-rwxr-xr-x    1 root     root       176078 Sep 20 21:54 libjpeg.so.62.0.0
[root@Linux4SAM lib]# cp libjpeg.so.62.0.0 /opt/codesourcery/arm-none-linux-gnueabi/libc/lib/
[root@Linux4SAM lib]# cd /opt/codesourcery/arm-none-linux-gnueabi/libc/lib/
[root@Linux4SAM lib]# ln -s libjpeg.so.62.0.0 libjpeg.so
[root@Linux4SAM lib]# ln -s libjpeg.so.62.0.0 libjpeg.so.62
```

```
[root@Linux4SAM lib]# cp libjpeg.a libjpeg.la /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib
```

3．编译 png lib

展开源码包 libpng-1.2.26.tar.gz。

配置：

```
[root@Linux4SAM libpng-1.2.26]# mkdir _install
[root@Linux4SAM libpng-1.2.26]# ./configure --host=arm-none-linux-gnueabi --prefix=$PWD/_install CFLAGS="-mcpu=arm926ej
-s -O3"
```

输入 make，make install 即可：

```
[root@Linux4SAM _install]# pwd
/usr/work/app/miniGUI/libpng-1.2.26/_install
[root@Linux4SAM _install]# ls
bin  include  lib  share
[root@Linux4SAM _install]# ls lib/
libpng12.a   libpng12.so    libpng12.so.0.26.0  libpng.la  libpng.so.3       pkgconfig
libpng12.la  libpng12.so.0  libpng.a            libpng.so  libpng.so.3.26.0
[root@Linux4SAM _install]# ls include/
libpng12  pngconf.h  png.h
```

复制相关文件到工具链目录：

```
[root@Linux4SAM lib]# cp -a libpng.so* libpng12.so* /opt/codesourcery/arm-none-linux-gnueabi/libc/lib/
[root@Linux4SAM include]# pwd
/usr/work/app/miniGUI/libpng-1.2.26/_install/include
[root@Linux4SAM include]# ls
libpng12  pngconf.h  png.h
[root@Linux4SAM include]# cp -a ./* /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/include/
[root@Linux4SAM include]#
```

```
[root@Linux4SAM lib]# ls
libpng12.a   libpng12.so    libpng12.so.0.26.0  libpng.la  libpng.so.3       pkgconfig
libpng12.la  libpng12.so.0  libpng.a            libpng.so  libpng.so.3.26.0
[root@Linux4SAM lib]# cp -a *.a *.la /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib
```

4．编译库文件

展开各个压缩包：

```
[root@Linux4SAM miniGUI]# ls
libminigui-1.6.10.tar.gz  mde-1.6.10.tar.gz  mg-samples-1.6.10.tar.gz  minigui-res-1.6.10.tar.gz
[root@Linux4SAM miniGUI]# tar xzvf libminigui-1.6.10.tar.gz
[root@Linux4SAM miniGUI]# tar xzvf mde-1.6.10.tar.gz
[root@Linux4SAM miniGUI]# tar xzvf mg-samples-1.6.10.tar.gz
[root@Linux4SAM miniGUI]# tar xzvf minigui-res-1.6.10.tar.gz
[root@Linux4SAM miniGUI]#
```

首先编译库文件。

建立一个文件夹用于安装 minigui 的编译输出，下面以/usr/local/minigui 为例。

```
[root@Linux4SAM libminigui-1.6.10]# pwd
/usr/work/app/miniGUI/libminigui-1.6.10
[root@Linux4SAM libminigui-1.6.10]# mkdir /usr/local/minigui
```

修改 configure：

```
11200   build_utpmc_ial_engine="no"
11201   build_fxrm9200_ial_engine="yes"
11202   build_abssig_ial_engine="no"
```

配置:

```
[root@Linux4SAM libminigui-1.6.10]# ./configure --disable-lite --prefix=/usr/local/minigui --host=arm-none-linux-gnueab
i --enable-fxrm9200ial CFLAGS="-mcpu=arm926ej-s -O3 -L/opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib -lts -lz -l
png -ljpeg"
```

这一步完成就会生成 Makefile。

输入 make 开始编译，编译完成后输入 make install 安装。

安装之后在指定的安装目录下生成 3 个文件夹：

```
[root@Linux4SAM libminigui-1.6.10]# ls /usr/local/minigui/
etc  include  lib
[root@Linux4SAM libminigui-1.6.10]#
```

```
[root@Linux4SAM libminigui-1.6.10]# ls /usr/local/minigui/etc/
MiniGUI.cfg
[root@Linux4SAM libminigui-1.6.10]# ls /usr/local/minigui/lib/
libmgext-1.6.so.10      libmgext.la          libminigui-1.6.so.10.0.0  libminigui.so            libvcongui.a
libmgext-1.6.so.10.0.0  libmgext.so          libminigui.a              libvcongui-1.6.so.10     libvcongui.la
libmgext.a              libminigui-1.6.so.10 libminigui.la             libvcongui-1.6.so.10.0.0 libvcongui.so
[root@Linux4SAM libminigui-1.6.10]# ls /usr/local/minigui/include/
minigui
[root@Linux4SAM libminigui-1.6.10]# ls /usr/local/minigui/include/minigui/
colordlg.h    endianrw.h   mgext.h             ose_semaphore.h   threadx_pthread.h    vxworks_semaphore.h
colorspace.h  ext          minigui.h           own_malloc.h      threadx_semaphore.h  win32_dirent.h
common.h      filedlg.h    mywindows.h         own_stdio.h       ucos2_pthread.h      win32_pthread.h
control.h     fixedmath.h  newfiledlg.h        psos_pthread.h    ucos2_semaphore.h    win32_sched.h
ctrl          gdi.h        nucleus_pthread.h   psos_semaphore.h  vcongui.h            win32_semaphore.h
dti.c         mgconfig.h   nucleus_semaphore.h  skin.h            vxworks_pthread.h    window.h
[root@Linux4SAM libminigui-1.6.10]#
```

复制相关文件到工具链目录：

```
[root@Linux4SAM include]# pwd
/usr/local/minigui/include
[root@Linux4SAM include]# ls
minigui  popt.h
[root@Linux4SAM include]# cp -a ./* /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/include/
[root@Linux4SAM include]#
```

```
[root@Linux4SAM lib]# cp -a ./* /opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib
```

5．安装资源

在 minigui-res-1.6.10 文件夹下修改 config.linux，指明本次编译的路径信息：

```
11 TOPDIR=/usr/local/minigui
12 prefix = $(TOPDIR)/usr/local
13 exec_prefix = $(prefix)
```

然后运行 make install 即可。

安装后的效果，多了一个文件夹 usr/local/lib/minigui:

```
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/
etc  include  lib  usr
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/usr/
local
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/usr/local/
lib
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/usr/local/lib/
minigui
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/usr/local/lib/minigui/
res
[root@Linux4SAM minigui-res-1.6.10]# ls /usr/local/minigui/usr/local/lib/minigui/res/
bmp  cursor  font  icon  imetab
[root@Linux4SAM minigui-res-1.6.10]#
```

6．编译 mde

编译 mde 需要 popt。

展开 popt 源码包：

```
[root@Linux4SAM miniGUI]# tar xzvf popt-1.7.tar.gz
```

配置：

```
[root@Linux4SAM popt-1.7]# pwd
/usr/work/app/miniGUI/popt-1.7
[root@Linux4SAM popt-1.7]# ./configure --prefix=/usr/local/minigui/ --host=arm-none-linux-gnueabi --enable-shared --ena
ble-static
```

依次运行 make，make install，完成后的效果：

```
[root@Linux4SAM popt-1.7]# ls /usr/local/minigui/
etc  include  lib  man  share  usr
[root@Linux4SAM popt-1.7]# ls /usr/local/minigui/include/
minigui  popt.h
[root@Linux4SAM popt-1.7]#
```

在 mde-1.6.10 文件夹下运行配置程序：

```
[root@Linux4SAM mde-1.6.10]# ./configure --prefix=/usr/local/minigui --host=arm-none-linux-gnueabi --enable-shared --enable-static CFLAGS="-mcpu=arm926ej-s -O3 -L/opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib -lts -lz -ljpeg -lpng"
```

配置完成后即可输入 make，编译完成可以在当前目录下看到生成的各个例子：

```
[root@Linux4SAM mde-1.6.10]# ls
aclocal.m4      bomb          config.status  COPYING   fontdemo     install-sh   mginit        notebook  same
AUTHORS         ChangeLog     config.sub     ctrldemo  gdidemo      Makefile     missing       painter   tools
autogen.sh      config.guess  configure      depcomp   housekeeper  Makefile.am  mkinstalldirs picview   Version
autom4te.cache  config.log    configure.in   dlgdemo   INSTALL      Makefile.in  NEWS          README
[root@Linux4SAM mde-1.6.10]# ls bomb/
bomb  bomb.c  bomb.o  Makefile  Makefile.am  Makefile.in  res
[root@Linux4SAM mde-1.6.10]# file bomb/bomb
bomb/bomb: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs), not stripped
[root@Linux4SAM mde-1.6.10]#
```

将这些例子文件夹里的可执行文件与资源文件夹复制到 NFS 目录下，也请注意保持每个 app 所有文件都在一个文件夹下，例如：

```
[root@Linux4SAM mde-1.6.10]# mkdir /nfs4arm/rootfs/usr/guiapp/bomb
[root@Linux4SAM mde-1.6.10]# cp ./bomb/bomb /nfs4arm/rootfs/usr/guiapp/bomb/
[root@Linux4SAM mde-1.6.10]# cp -a ./bomb/res/ /nfs4arm/rootfs/usr/guiapp/bomb/
[root@Linux4SAM mde-1.6.10]#
```

7．编译 mg samples

在 mg-samples-1.6.10 文件夹下运行配置程序：

```
[root@Linux4SAM mg-samples-1.6.10]# ./configure --host=arm-none-linux-gnueabi --prefix=/usr/local/minigui CFLAGS="-L/opt/codesourcery/arm-none-linux-gnueabi/libc/usr/lib -lz -lts -mcpu=arm926ej-s -O3 -lpthread -lpng -ljpeg"
```

完成后输入 make 即可开始编译。

Src 文件夹会生成一些例子：

```
[root@Linux4SAM src]# ls -t
coolbar          timeeditor.o  cursordemo    newtoolbar     static
mywins           animation     cursordemo.o  newtoolbar.o   static.o
mywins.o         animation.o   drawicon      progressbar    helloworld
treeview         scrollview    bitblt        progressbar.o  helloworld.o
treeview.o       scrollview.o  bitblt.o      trackbar       mycontrol
coolbar.o        scrollwnd     drawicon.o    trackbar.o     mycontrol.o
listview         scrollwnd2    loadbmp       combobox       Makefile
listview.o       scrollwnd2.o  loadbmp.o     combobox.o     res
monthcal         bmpbkgnd      stretchblt    edit           Makefile.in
monthcal.o       bmpbkgnd.o    stretchblt.o  edit.o         caretdemo.c
spinbox          input         capture       listbox        combobox.c
spinbox.o        input.o       capture.o     listbox.o      menubutton.c
gridview         scrollwnd.o   painter       menubutton.o   button.c
gridview_ext     skindemo      painter.o     button         dialogbox.c
gridview_ext.o   skindemo.o    propsheet     button.o       gridview.c
gridview.o       caretdemo     propsheet.o   dialogbox      gridview_ext.c
iconview         caretdemo.o   scrollbar     dialogbox.o    mywins.c
iconview.o       createicon    scrollbar.o   simplekey      scrollwnd2.c
timeeditor       createicon.o  menubutton    simplekey.o    bmpbkgnd.c
[root@Linux4SAM src]# pwd
/usr/work/app/miniGUI/mg-samples-1.6.10/src
[root@Linux4SAM src]#
```

三，运行 miniGUI

1．复制文件

以 NFS 方式启动目标板：

```
Looking up port of RPC 100003/2 on 192.168.1.5
Looking up port of RPC 100005/1 on 192.168.1.5
VFS: Mounted root (nfs filesystem).
Freeing init memory: 132K
init started: BusyBox v1.11.2 (2008-09-09 21:57:17 CST)
starting pid 844, tty '': '/etc/init.d/rcS'
----------mount all----------
----------Starting mdev------
This may take some time ...
```

在主机上将 minigui 相关文件复制到 NFS 所在目录：

```
[root@Linux4SAM rootfs]# pwd
/nfs4arm/rootfs
[root@Linux4SAM rootfs]# ls
bin  dev  etc  home  lib  mnt  proc  sbin  sys  tmp  usr  var
[root@Linux4SAM rootfs]# cp -a /usr/local/minigui/* ./
```

由于是 NFS，空间不是问题，如果需要做成 img 烧到板子上，可以将一些不必要的文件去掉，比如 include 与.a 文件。

在目标板上可以看到这些文件：

```
[root@mcuzone /etc]#ls
MiniGUI.cfg  host.conf    ld.so.cache
boa          hosts        ld.so.conf
fstab        init.d       mdev.conf
group        inittab      mime.types
```

2．测试例程

在主机上将某个 sampe 传输到 NFS 目录：

```
[root@Linux4SAM testapp]# pwd
/nfs4arm/rootfs/usr/testapp
[root@Linux4SAM testapp]# cp /usr/work/app/miniGUI/mg-samples-1.6.10/src/helloworld ./
[root@Linux4SAM testapp]#
```

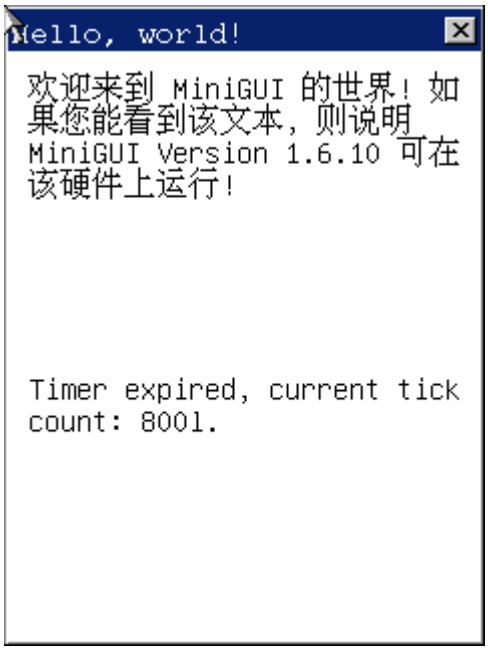
修改/etc/MiniGUI.cfg，修改 GAL，IAL 及显示大小，以适应开发板：

```
[system]
# GAL engine and default options
gal_engine=fbcon
defaultmode=240x320-16bpp

# IAL engine
ial_engine=console
mdev=/dev/input/mouse1
mtype=IMPS2

[fbcon]
defaultmode=240x320-16bpp

[qvfb]
defaultmode=640x480-16bpp
display=0

# The first system font must be a logical font using RBF device font.
```

在目标板上尝试运行：

```
[root@mcuzone testapp]#./helloworld _
```

LCD 显示：

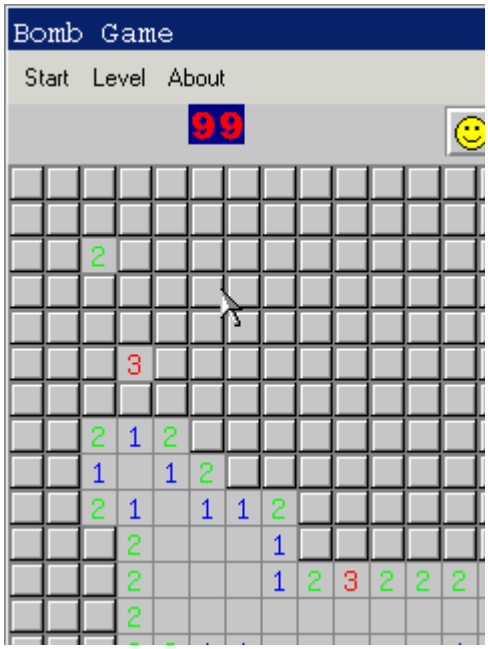


Notebook 的例子：

Bomb 例子(扫雷):

```
[root@mcuzone bomb]#pwd
/usr/guiapp/bomb
[root@mcuzone bomb]#ls
bomb  res
[root@mcuzone bomb]#ldd bomb
        libpthread.so.0 => /lib/libpthread.so.0 (0x40025000)
        libminigui-1.6.so.10 => /lib/libminigui-1.6.so.10 (0x40045000)
        libm.so.6 => /lib/libm.so.6 (0x40140000)
        libc.so.6 => /lib/libc.so.6 (0x401ec000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone bomb]#./bomb
```

使用鼠标即可游戏：



3. USB 鼠标测试

在板子上连接好 USB 鼠标，在/dev/input 下建立相应的节点：

```
[root@mcuzone testapp]#cat /proc/bus/input/devices
```

```
I: Bus=0003 Vendor=046d Product=c50a Version=0110
N: Name="Logitech USB Receiver"
P: Phys=usb-at91-1/input0
S: Sysfs=/class/input/input2
U: Uniq=
H: Handlers=mouse1 event2
B: EV=20017
B: KEY=ffff0000 0 0 0 0 0 0 0 0
B: REL=103
B: MSC=10
B: LED=ff00

[root@mcuzone testapp]#_
```

```
[root@mcuzone testapp]#cat /proc/bus/input/devices
[root@mcuzone testapp]#ls -al /dev/input/
drwxr-xr-x    2 root     root          4096 Sep 18  2008 .
drwxr-xr-x    4 root     root         16384 Sep 18  2008 ..
crw-r--r--    1 root     root      13,  64 Sep 18  2008 event0
crw-r--r--    1 root     root      13,  65 Sep 18  2008 event1
crw-r--r--    1 root     root      13,  66 Sep 18  2008 event2
crw-r--r--    1 root     root      13,  63 Sep 18  2008 mice
crw-r--r--    1 root     root      13,  32 Sep 18  2008 mouse0
crw-r--r--    1 root     root      13,  33 Sep 18  2008 mouse1
[root@mcuzone testapp]#
```

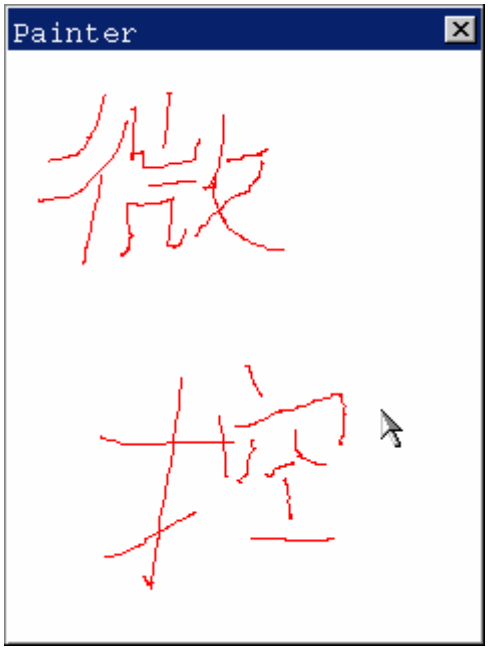
可以看到 USB 鼠标位于/dev/input/mouse1，注意修改 MiniGUI.cfg:

```
# IAL engine
ial_engine=console
mdev=/dev/input/mouse1
mtype=IMPS2
```

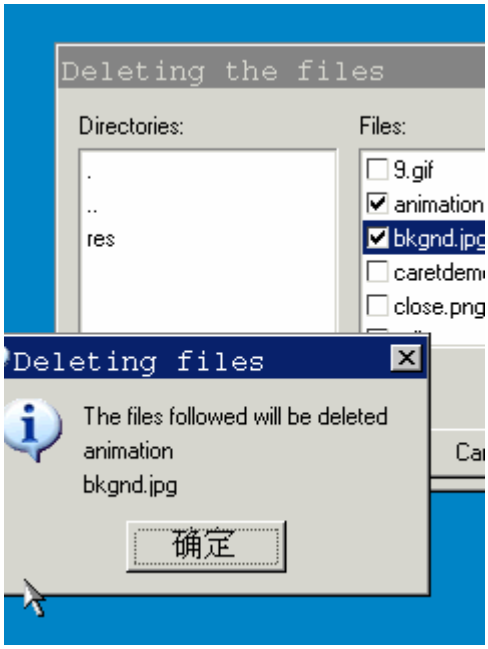
完成后在开发板上运行例程 painter(从 mg samples 的 src 文件夹下复制)：

```
[root@mcuzone testapp]#./painter
```

此时就可以利用鼠标在屏幕上画图：



使用 listbox 测试鼠标点击，拖拽:

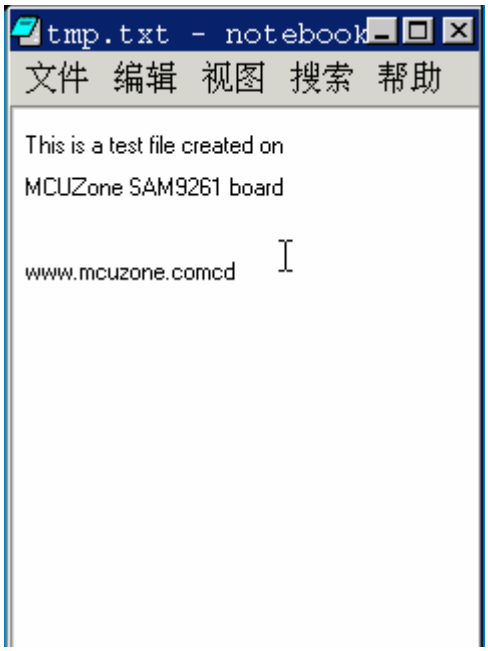


4．USB 键盘测试

连接上 USB 键盘，然后启动 notebook 的例子：

```
[root@mcuzone notebook]#usb 1-2: new low speed USB device using at91_ohci and a
ddress 3
usb 1-2: configuration #1 chosen from 1 choice
input: Dell Dell USB Keyboard as /class/input/input3
input: USB HID v1.10 Keyboard [Dell Dell USB Keyboard] on usb-at91-2

[root@mcuzone notebook]#
[root@mcuzone notebook]#
[root@mcuzone notebook]#
[root@mcuzone notebook]#
[root@mcuzone notebook]#./notebook
```

输入内容：



保存文件：

文件已经保存：

```
[root@mcuzone notebook]#ls
0-1.bmp    0-2.bmp    notebook   res        test1.txt
[root@mcuzone notebook]#cat test1.txt
This is a test file created on
MCUZone SAM9261 board

www.mcuzone.comcd [root@mcuzone notebook]#_
```

5．触摸屏测试

使用 tslib 校准触摸屏后，修改 minigui 配置文件，指定 ial 为修改的 fxrm9200，设为触摸屏对应设备：

```
# IAL engine
ial_engine=fxrm9200
mdev=/dev/input/event1
mtype=IMPS2
```

运行一个例程并点击屏幕，可以看到 fxrm9200ial 输出的 debug 信息：

```
[root@mcuzone testapp]#./helloworld
InitRm9200Input
mouse_update:(0,0)
mouse down, x=161, y=247, pressure=4
mouse_update:(161,247)
```

屏幕上的鼠标指针也有相应的动作。

四，miniGUI 应用

1．浏览器 edillo

配置：

```
[root@Linux4SAM edillo]# ./configure --host=arm-none-linux-gnueabi CFLAGS="-mcpu=arm926ej-s -O3 -L/opt/codesourcery/arm
-none-linux-gnueabi/libc/usr/lib -lz"
```

完成后 make，在 test/simple 将生成例程，将相关文件复制到开发板：

```
[root@mcuzone edillo]#pwd
/usr/guiapp/edillo
[root@mcuzone edillo]#ls
160.gif             demo-help.html      index.html          res
ccontact.html       form.html           registergif.html    simple-edillo
clicensing.html     images              registergif_files
[root@mcuzone edillo]#
```

```
[root@Linux4SAM zlib-1.2.3]# ls -al libz.so*
lrwxrwxrwx    1 nobody   nobody         13 Sep 13 20:13 libz.so -> libz.so.1.2.3
lrwxrwxrwx    1 nobody   nobody         13 Sep 13 20:13 libz.so.1 -> libz.so.1.2.3
-rwxr-xr-x    1 nobody   nobody      84973 Sep 13 20:13 libz.so.1.2.3
[root@Linux4SAM zlib-1.2.3]# cp -a libz.so* /nfs4arm/rootfs/lib/
[root@Linux4SAM zlib-1.2.3]# _
```

```
[root@mcuzone edillo]#ldd simple-edillo
        libz.so.1 => /lib/libz.so.1 (0x40025000)
        libts-0.0.so.0 => /lib/libts-0.0.so.0 (0x40040000)
        libdl.so.2 => /lib/libdl.so.2 (0x4004a000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40055000)
        libm.so.6 => /lib/libm.so.6 (0x40075000)
        libc.so.6 => /lib/libc.so.6 (0x40121000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone edillo]#_
```

运行 simple-edillo，打开网页：

2．编写 minigui 小程序

编写一个 minigui 的小应用，可以参考 minigui 的白皮书，主要流程是新建窗口，输出字符串，并显示 gif 图像。代码完成后编写一个 Makefile：

```
1
2  CC = arm-none-linux-gnueabi-gcc
3  CFLAGS = -mcpu=arm926ej-s -march=armv5te -O3
4  LDFLAGS = -lminigui -lmgext -lpthread
5
6  REMOVE = rm -f
7
8  TARGET = minigui1
9
10 minigui1 :
11   $(CC) $(CFLAGS) $(LDFLAGS) -o $(TARGET) main.c
12
13 clean:
14   $(REMOVE) $(TARGET)
```

主要是需要加入相关的 lib。运行效果：

第十二章：将 uc/GUI 移植到 9261

本文叙述 uC/GUI 在本站 9261 开发板上的运行过程，包括了 uC/GUI 的简要介绍，部分移植工作，使用 realview 工具链进行编译，重点在 uC/GUI 的运行过程，包括在仿真环境下的使用和代码烧写到板子上启动的过程。调试软件选择了 ARM 的 RVD，仿真器选择的是 jlink，其它类型的仿真器可以类推。

一、 准备工作

1. 安装调试软件

PC 上的调试软件建议安装 ADS 或者 RealView 2.2，以后者为好。软件的安装请参照软件压缩包内的说明文件。

2. 安装调试器及相关软件

由于使用的 jlink，首先必须安装 jlink 的软件，可以到 segger(www.segger.com)网站下载。安装完成后需要将 jlink 整合到 axd 及 rvdebug 中。这部分内容可以参考本站编写的 jlink 使用说明。

连接上 jlink 到 PC 后选择自动安装驱动即可。

然后连接好 jlink 到 9261 板子 JTAG 口的 20 芯排线，给目标板上电。

到 jlink 软件安装目录下运行 jlink.exe:

出现：

```
Feature(s) : RDI,FlashDL,FlashBP,JFlash,GDBFull
VTarget = 3.539V
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41069265: ARM, Architecure 5TEJ
Info: CP15.0.1: 0x1D152152: ICache: 16kB (4*128*32), DCache: 16kB (4*128*32)
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x0792603F
Found ARM with core Id 0x0792603F (ARM9)
  ETM V1.3: 4 pairs addr.comp, 2 data comp, 8 MM decs, 2 counters, sequencer
J-Link>
```

那么就说明核心已经能被正确识别并连接。

3. 获得 uC/GUI 的相关源代码

这个可以到网上搜索。

二、 uC/GUI 简介

4. uC/GUI

UCGUI是一种嵌入式应用中的图形支持系统。它被设计用于为任何使用LCD图形显示的应用提供高效的独立于处理器及LCD控制器的图形用户接口，它适用单任务或是多任务系统环境，并适用于任意LCD控制器和CPU下任何尺寸的真实显示或虚拟显示。

它的设计架构是模块化的，由不同的模块中的不同层组成，由一个LCD驱动层来

提供所有对LCD的具体图形操作，UCGUI可以在任何的CPU上运行，因为它是100%的标准C代码编写的。

UCGUI能够适应大多数使用黑白或彩色LCD的应用，它提供非常好的允许处理灰度的颜色管理。还提供一个可扩展的2D图形库及占用极少RAM的窗口管理体系。

5. uC/GUI 对系统资源的占用

对于内存的需求取决于所选用的UCGUI的功能模块以及所使用的目标系统上的编译器的效率。内存的占用量无法估计准确的值，下面就一些的数值适用于多数的目标系统。

小型系统(不含窗口管理功能)

[1].RAM:100字节

[2].栈: 500字节

[3].ROM:10~25K(取决于选用的UCGUI功能模块)

大型系统(包含窗口管理及各种窗体控件功能)

[1].RAM: 2-6 kb (决于选用的应用中建立窗口的数量)

[2].栈: 1200 bytes

[3].ROM: 30-60 kb (决于选用的UCGUI功能模块)

还要注意ROM的需求量随着在应用程序中使用的字体数目而增长,以上的所有值都是粗糙的估计，并不准确.

6. 参考文献

请参考下面文档了解更多关于 uC/GUI 的信息：

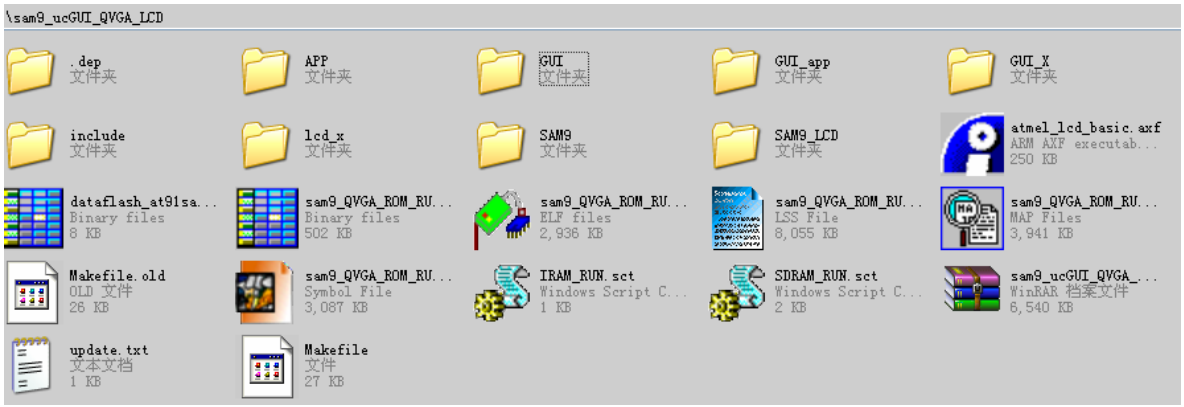
《uCGUI 中文手册》

《uC/GUI Graphical User Interfacewith Graphic Library Manual》

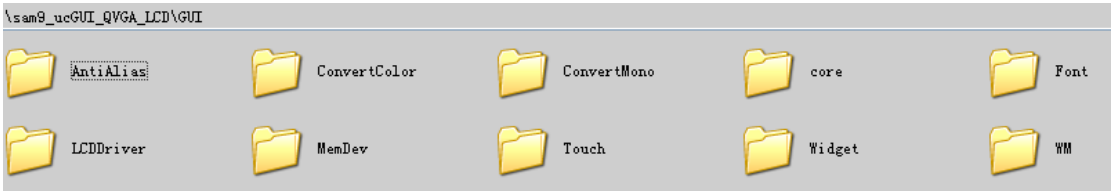
三、 uC/GUI 的移植和编译

1. uC/GUI 工程简介

为 uC/GUI 的工程新建一个文件夹，为了避免使用图形方式的 IDE，也为了方便的控制编译每个文件，可以使用命令行方式编译。如下图：

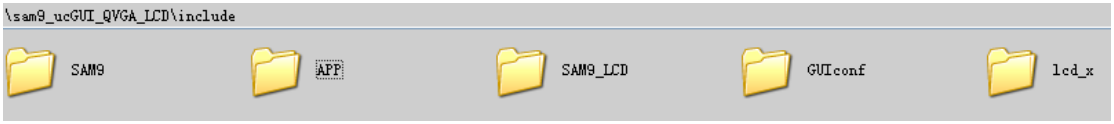


APP 文件夹放置应用程序，GUI 文件就是 uC/GUI 的源代码，如下图：



各子文件夹的意义的可以参考官方的说明书。

GUI_app 放置 GUI 相关的应用。GUI_X 是一些用于 GUI 移植的系统扩展层，比如定时器。Include 包含相应的头文件：



注意其中的 GUIconf 文件夹，里面有 GUI 移植的相关配置。

LCD_x 包含了 uC/GUI 必要的 LCD 驱动层。

SAM9 文件夹包含了 SAM9261 的底层支持，SAM9_LCD 包含了 SAM9 硬件 LCD 的支持。

2．uC/GUI 的移植简述

首先需要配置 uC/GUI，主要就是这些配置文件：

GUIConf.h：

```
*/

#define GUI_OS                    (0)  /* Compile with multitasking support */
#define GUI_WINSUPPORT            (1)  /* Use window manager if true (1)   */
#define GUI_SUPPORT_MEMDEV        (1)  /* Support memory devices */
#define GUI_SUPPORT_TOUCH         (0)  /* Support a touch screen (req. win-manager) */
//#define GUI_SUPPORT_CURSOR        (0)  /* support cursor */
#define GUI_SUPPORT_UNICODE       (1)  /* Support mixed ASCII/UNICODE strings */
#define GUI_SUPPORT_MOUSE         (0)

/*********************************************************************
*
*         Configuration of dynamic memory

Dynamic memory is used for memory devices and window manager.
If you do not use these features, there is no need for dynamic memory
and it may be switched off completely. (This section can be erased)
*/

#define GUI_ALLOC_SIZE          512*1024  /* Size of dynamic memory */


/*********************************************************************
*
*         Configuration of available fonts

Dynamic memory is used for memory devices and window manager.
If you do not use these features, there is no need for dynamic memory
and it may be switched off completely. (This section can be erased)
*/

#define GUI_DEFAULT_FONT    &GUI_Font8x8
```

这里需要配置 LCD 所支持的特性，以及动态内存的大小，由于板子上的内存很大，

所以可以设置很大。

GUITouchConf.h

```
#define GUI_TOUCH_AD_LEFT   20
#define GUI_TOUCH_AD_RIGHT  140
#define GUI_TOUCH_SWAP_XY    1
#define GUI_TOUCH_MIRROR_X   0
#define GUI_TOUCH_MIRROR_Y   1
```

这里可以配置触摸的相关参数。

LCDConf.h

```
#define LCD_CONTROLLER     9261          /* SAM9261 onchip LCDC */
#define LCD_XSIZE          240           /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE          320           /* Y-resolution of LCD, Logical coor. */
#define LCD_INTERFACEBITS  32            /* select interface bit width*/
#define LCD_BITSPERPIXEL   8
#define LCD_FIXEDPALETTE   323           /* BBBGGRRR */
#define LCD_MAX_LOG_COLORS 256

#define LCD_CACHE          0
#define LCD_USE_BITBLT     0

#define LCD_MIRROR_X       0
#define LCD_MIRROR_Y       0
#define LCD_SWAP_XY        0            /* remember to change the X Y Size if  set */

#define LCD_INIT_CONTROLLER()

#define LCD_OFF()
#define LCD_ON()

#define LCD_VRAM_BASE      0x20000000    /* VRAM base address */
```

根据 LCD 配置 LCD 的尺寸，以及 LCD 显存的位置。

3. uC/GUI 的编译

为了方便编译，可以使用 makefile。整个工程可以参考 WinAVR 的 makefile 文件。以下是 makefile 的片断，供参考。

```
TOOLCHAIN_PREFIX=arm
PATH_TO_LINKSCRIPTS=../RV_lds

#FLASH_TOOL = UVISION
FLASH_TOOL = JLINK

# MCU name and submodel
#MCU      = ARM926EJ-S
MCU      = 5TEJ
SUBMDL   = AT91SAM9261

# CPU endian
CPU_ENDIAN = littleend

# MMU enable/disable
#ARM9_MMU_EN = YES
ARM9_MMU_EN = NO

#USE_THUMB_MODE = YES
USE_THUMB_MODE = NO

## Create ROM-Image (final)
RUN_MODE = ROM_RUN
## Create RAM-Image (debugging)
#RUN_MODE = RAM_RUN

# entry point, must be the same as in the scatter file
ifeq ($(RUN_MODE),RAM_RUN)
ENTRY_ADDR = 0x00300000
```

```
# List C source files here which must be compiled in ARM-Mode.
# use file-extension c for "c-only"-files
SRCARM  = $(ARCH_FOLDER)/init.c
SRCARM += $(LCD_DEV)/lcd_sam9.c
SRCARM += $(LCD_DEV)/touchscreen.c
SRCARM += $(ARCH_FOLDER)/ARM9_excp_c.c
SRCARM += $(ARCH_FOLDER)/BSP.c
```

```
# Define programs and commands.
ASM = $(TOOLCHAIN_PREFIX)asm
CC = $(TOOLCHAIN_PREFIX)cc
CPP = $(TOOLCHAIN_PREFIX)cpp
LD = $(TOOLCHAIN_PREFIX)link
OBJCOPY = fromelf
OBJDUMP = fromelf
SIZE = fromelf -z
NM = fromelf -s
SHELL = sh
REMOVE = rm -f
COPY = cp
```

除了 makefile 之外，还应该设置一个 scatter loader 文件，这里配置代码运行在 SDRAM。

```
CODE_LR 0x21500000 0x00100000 ; code size 1MB
{
    ER_Startup_code +0
    {
        Cstartup_ads.o (reset, +FIRST)
    }

    ER_APP +0x00000000
    {
        main.o      (+RO)
        test_lcd.o (+RO)
    }

    ER_GUI_APP +0x0
    {
        MainTask.o      (+RO)
        MicriumLogo*.o (+RO)
        GUIDEMO*.o      (+RO)
    }

    ER_SAM9_BSP +0
    {
        init.o (+RO)
        ARM9_excp_c.o (+RO)
    }

    ER_OS_SRC +0
    {

    }

    ER_GUI_SRC +0
    {
```

启动一个 cmd 窗口，一路 cd 到工程目录：

```
                    \sam9_ucGUI_QVGA_LCD>ls
APP       IRAM_RUN.sct  SAM9_LCD                  include              sam9_QVGA_ROM_RUN.lss          update.txt
GUI       Makefile      SDRAM_RUN.sct             lcd_x                sam9_QVGA_ROM_RUN.map
GUI_X     Makefile.old  atmel_lcd_basic.axf       sam9_QVGA_ROM_RUN.bin sam9_QVGA_ROM_RUN.sym
GUI_app   SAM9          dataflash_at91sam9261Sek.bin sam9_QVGA_ROM_RUN.elf sam9_ucGUI_QVGA_LCD_rev1.2.rar
```

首先应当检查工具链是否正确：

```
                                    >armcc
ARM/Thumb C/C++ Compiler, RVCT2.2 [Build 349]

Usage:          armcc [options] file1 file2 ... filen
Main options:

--arm           Generate ARM code
--thumb         Generate Thumb code
--c90           Switch to C mode (default for .c files)
--cpp           Switch to C++ mode (default for .cpp files)
-O0             Minimum optimization
-O1             Restricted optimization for debugging
-O2             High optimization
-O3             Maximum optimization
-Ospace         Optimize for codesize
-Otime          Optimize for maximum performance
--cpu <cpu>     Select CPU to generate code for (eg. ARM9E/ARM10E)
--cpu list      Output a list of all the selectable CPUs
-o <file>       Name the final output file of the compilation
-c              Compile only, do not link
--asm           Output assembly code as well as object code
-S              Output assembly code instead of object code
--interleave    Interleave source with disassembly (use with --asm or -S)
-E              Preprocess the C source code only
-D<symbol>      Define <symbol> on entry to the compiler
-g              Generate tables for high-level debugging
-I<directory>   Include <directory> on the #include search path
```

一切正常。

使用 makefile 还需要 GNU make 的支持，可以通过安装 WinARM，WinAVR 或者 Cygwin 的环境来实现：

```
                              sam9_ucGUI_QVGA_LCD>make -v
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i386-pc-mingw32
```

首先输入 make clean，然后输入 make all，即可编译，如果有编译的问题，请修改代码，然后完成编译。

编译完成之后将会生成相关的文件，比如 elf，bin，map，lss 文件。

```
                    \sam9_ucGUI_QVGA_LCD>ls -a sam9_QVGA_ROM_RUN.*
sam9_QVGA_ROM_RUN.bin  sam9_QVGA_ROM_RUN.elf  sam9_QVGA_ROM_RUN.lss  sam9_QVGA_ROM_RUN.map  sam9_QVGA_ROM_RUN.sym
```

四、 运行 uC/GUI

1. 在调试环境中运行 uC/GUI

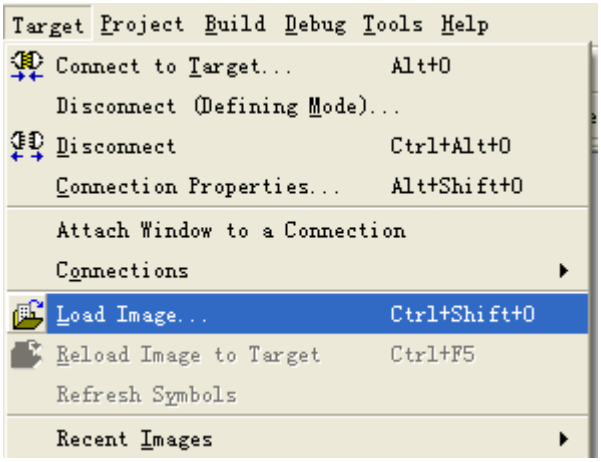
将板子上的启动跳线从 data flash 跳开，配置板子为内部 ROM 启动。

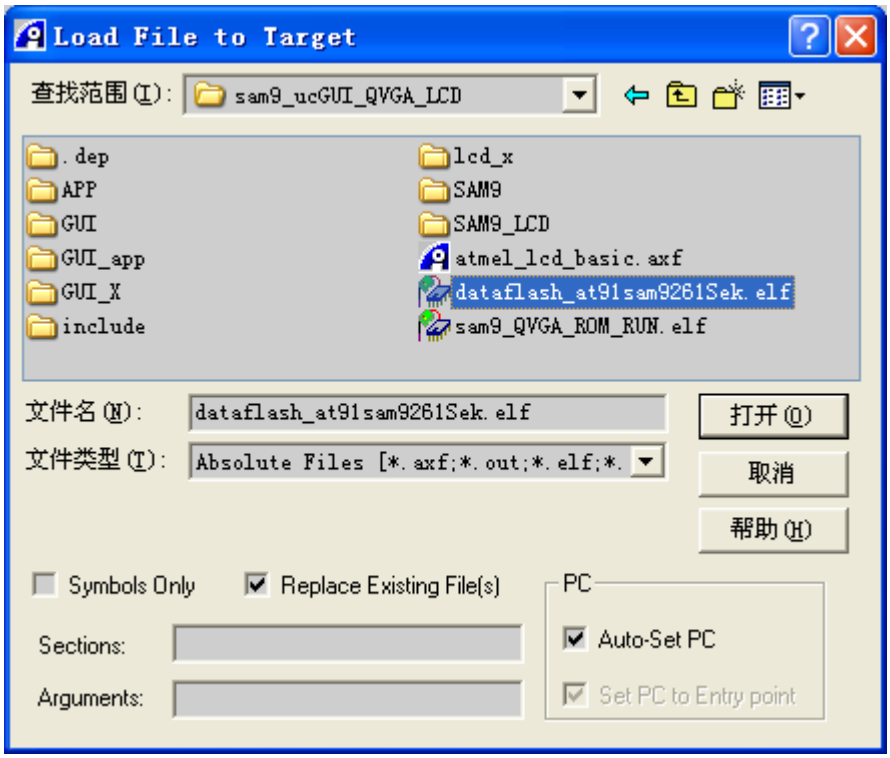
连接上 JTAG 连线到板子，验证了连接可靠性后，打开 RVD。

uC/GUI 代码被编译到 SDRAM 中运行，由于 SDRAM 在没有初始化的情况下无

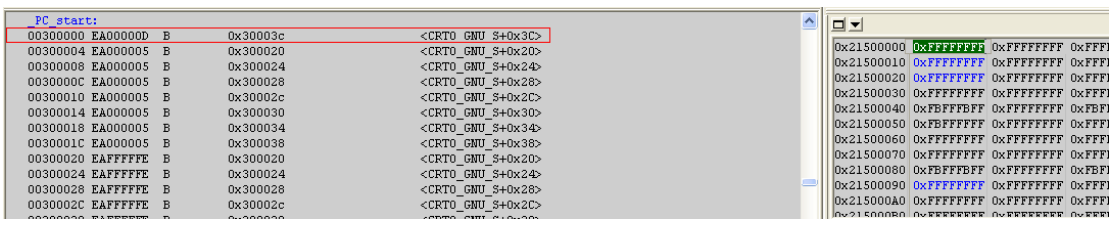
法使用，因此必须先初始化。初始化的过程可以使用初始化脚本，也可以采用先运行别的在 RAM 中运行但是可以初始化 SDRAM 的代码片断。前面一个办法可以参考本站另一篇文档《在 SAM9261 上调试 U-boot.doc》。这里采用后一种办法，引导代码选择了 ATMEL 官方提供的 bootstrap 代码。该代码正常情况下会被 9261 的内部 boot 加载到内部 ram 内运行，该代码会初始化 SDRAM。

首先加载 bootstrap 的 elf 文件：





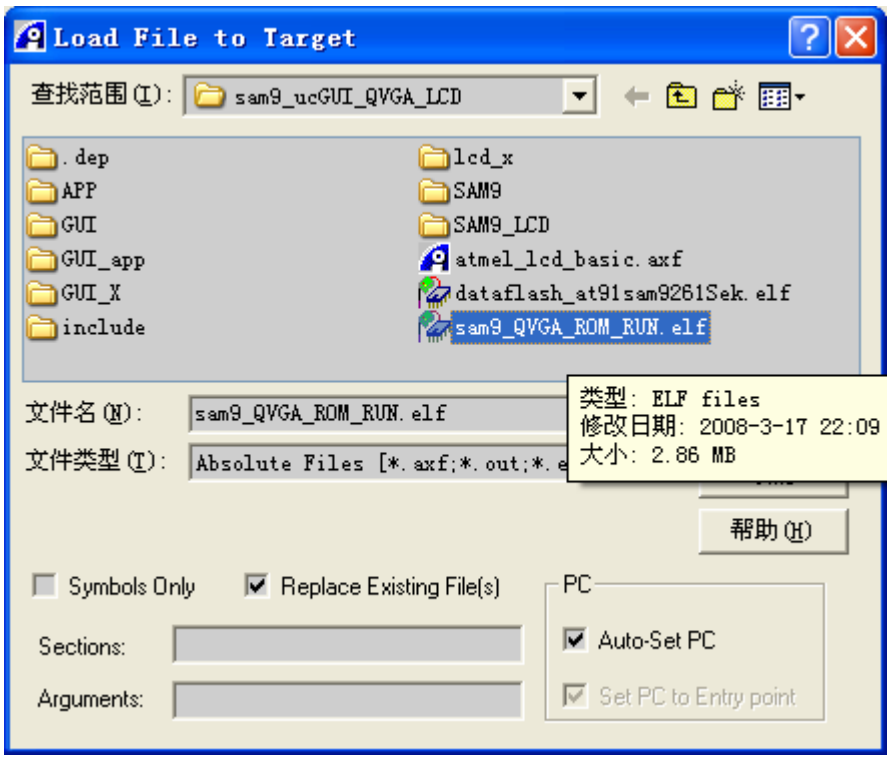
此时可以发现，SDRAM 区域是不能修改的：



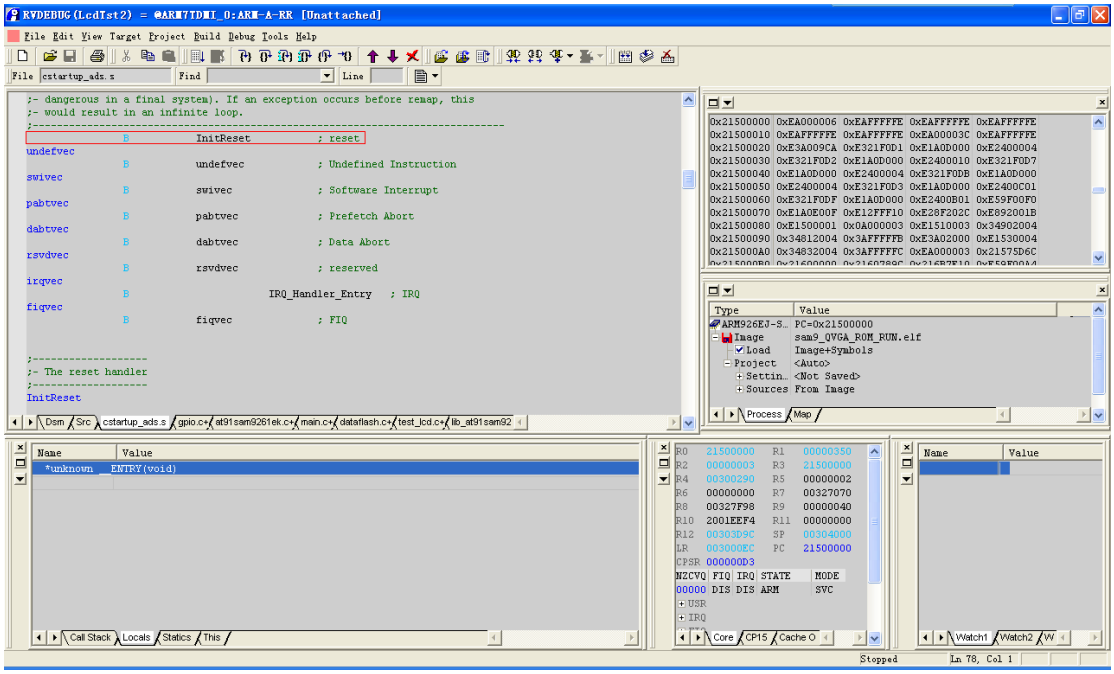
直接点击运行，由于 bootstrap 会试图加载相关代码并跳转，而在本次调试中不能加载到正确代码，系统运行会失败。不过此时 SDRAM 已经可以使用，可以写入任意数字来验证其可用性：

0x21500000	0x5A5A5A5A	0x5990863A	0xBC9BDC47	0xF8711300
0x21500010	0x00800020	0x00800020	0x0E13A982	0x00020205
0x21500020	0x756A694C	0x2E320D78	0x30322E36	0x00000000
0x21500030	0x00000000	0x00000000	0x00000000	0x00000000
0x21500040	0xE1A00000	0xE1A00000	0xE1A00000	0xE1A00000
0x21500050	0xE1A00000	0xE1200000	0xE0A00000	0xE1A00000
0x21500060	0xEA000002	0x016F2818	0x00000000	0x0011E1F8
0x21500070	0xE1A07001	0xE1A08002	0xE10B2000	0xE3120003
0x21500080	0x1A000001	0xE3A00007	0xE7123456	0xE10F2000
0x21500090	0xE38020C0	0xE121F000	0x00000000	0x00000000
0x215000A0	0xE28E00CC	0xE890307E	0xE0500001	0x0A00000A
0x215000B0	0xE0851000	0xE0826000	0xC08CC000	0xE0822000

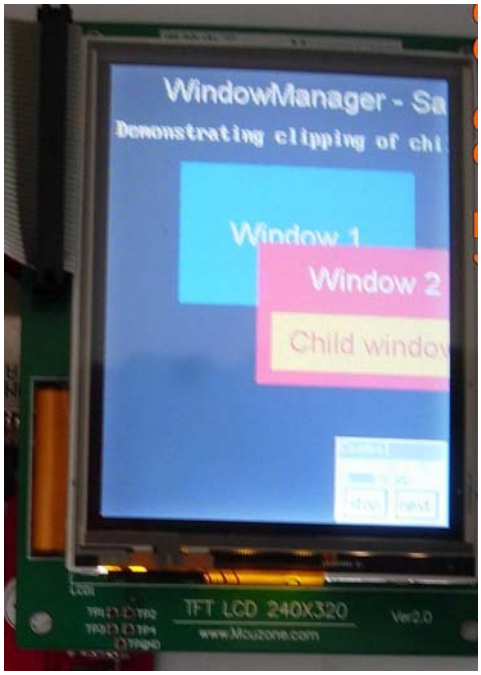
下面可以加载真正的 uC/GUI 的代码，

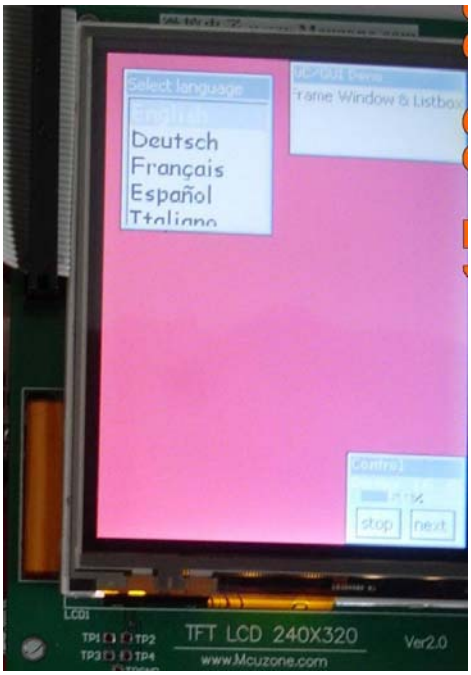


选择对应的 elf 后，即可加载：



直接点击运行，即可运行 uC/GUI 的代码。由于该代码基于 ATMEL 的触摸屏的例子，代码运行后会点击一下 LCD，完成后即可运行 uC/GUI：





2. 固化运行 uC/GUI

如果需要让代码从板子上直接运行，则需要将代码烧写到板子的 data flash 上。

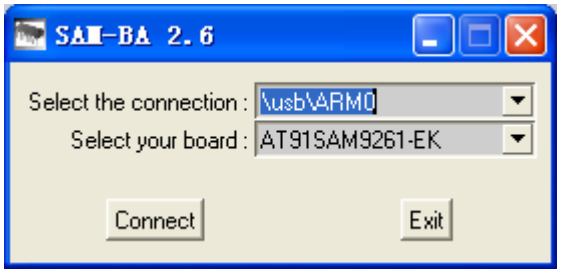
首先需要编译对应的引导代码，为了方便，可以直接使用 data flash 的 boot，而 uC/GUI 则烧写到 data flash。

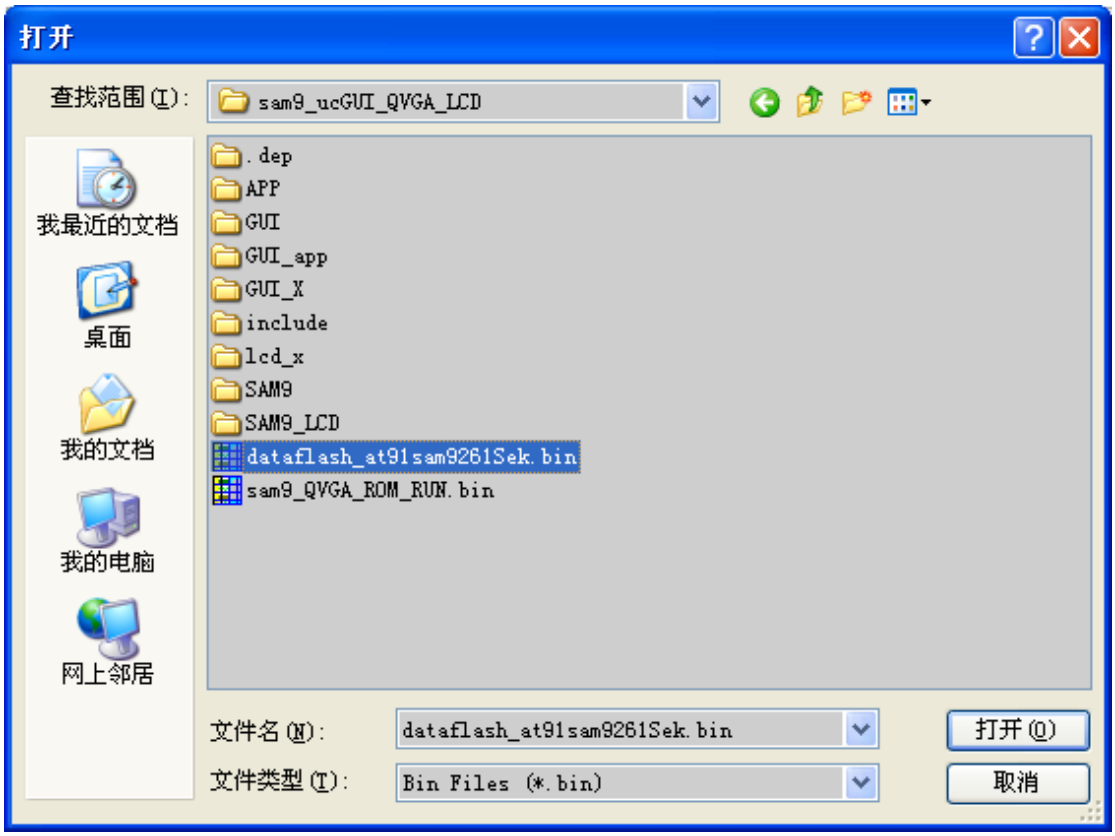
在编译 loader 之前，需要检查一级 loader 的参数：

```
#define AT91C_SPI_PCS_DATAFLASH   AT91C_SPI_PCS0_DATAFLASH  /* Boot on SPI NCS0 */

#define IMG_ADDRESS     0x8000      /* Image Address in DataFlash */
#define IMG_SIZE        0xA0000     /* Image Size in DataFlash    */

#define MACH_TYPE       0x350       /* AT91SAM9261-EK */
#define JUMP_ADDR       0x21500000  /* Final Jump Address         */
```

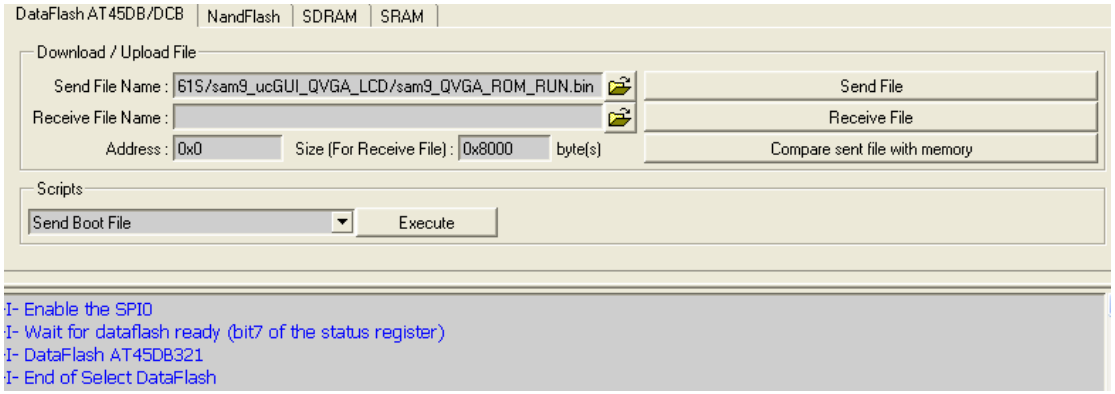
IMG_ADDRESS 就是 uC/GUI 在 dataflash 上烧写的地址，而 JUMP_ADDR 就是 uC/GUI 被加载到 SDRAM 的地址，必须与 uC/GUI 连接时的地址一致。

编译完成之后，使用 SAM-BA，将一级 loader 使用 send boot file 的命令烧写到 data flash。烧写之前请跳上 data flash 的跳线。





烧写完成后，将 uC/GUI 的代码烧写到 data flash 的 0x8000 的位置，与 loader 的定义一样。



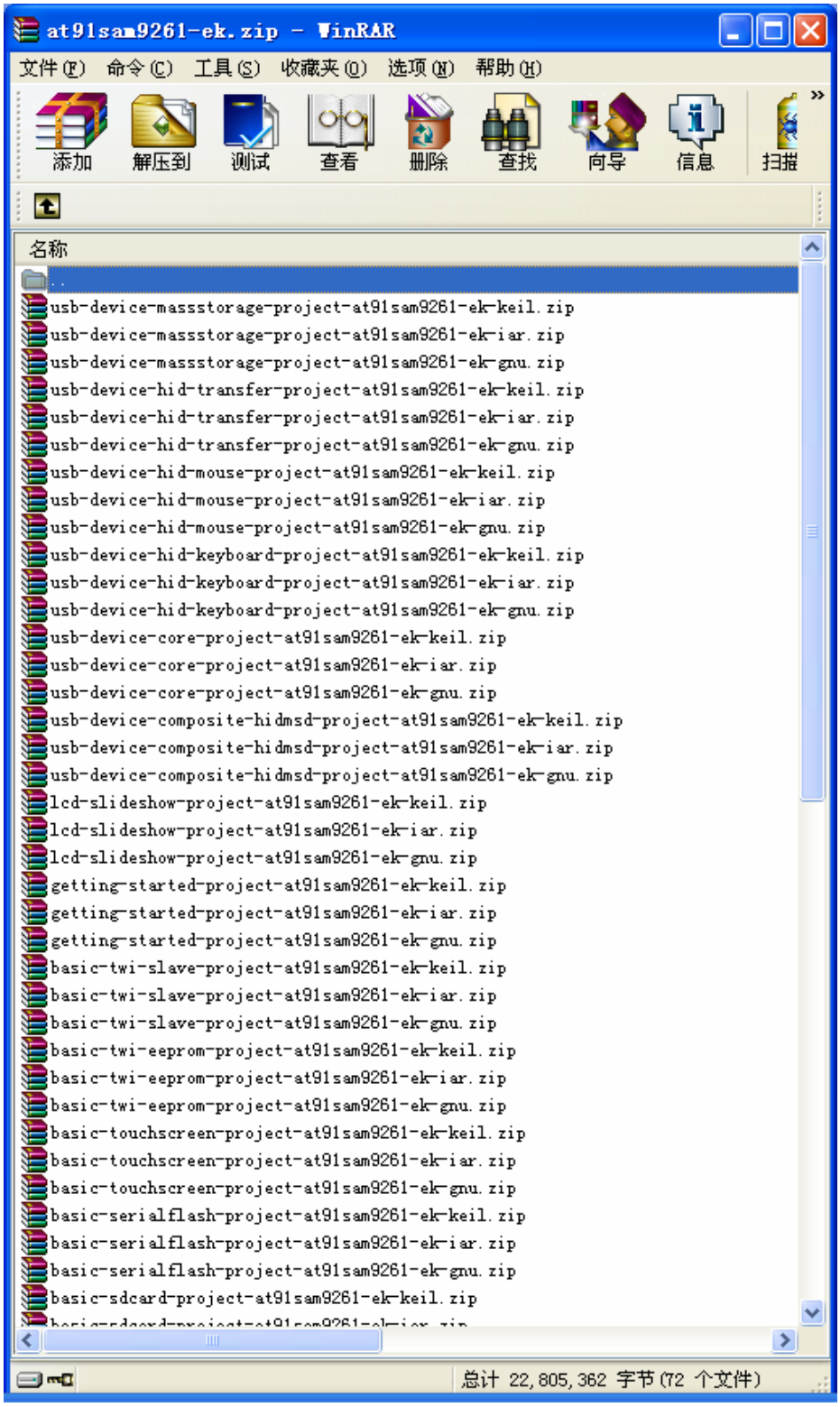
发送完成后按复位键即可从板子上直接运行 uC/GUI。

五、 一些事项

uC/GUI 的驱动对其运行的效率有很大影响，需要根据处理器的特性编写，采用最高效率的代码。

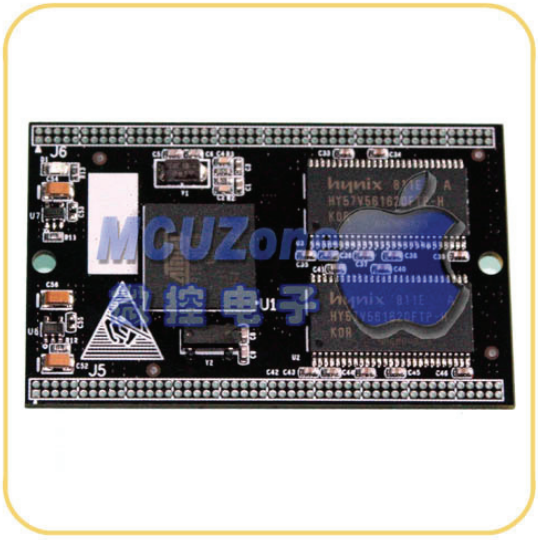
第十三章：AT91 softpack 软件包

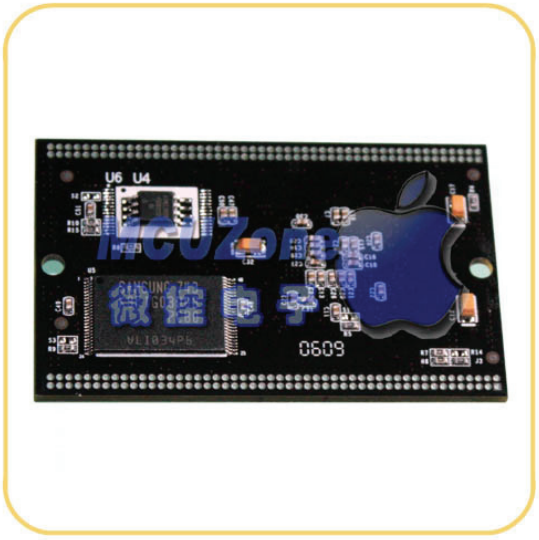
为了方便大家使用 AT91 芯片，ATMEL 发布了一个 AT91 softpack 软件包，有了该软件包，我们可以更容易的使用 AT91 的外设，而且该软件包会随编译器的升级而不定期升级，建议大家使用。下面我们来看一下这个软件包里面有哪些例子：



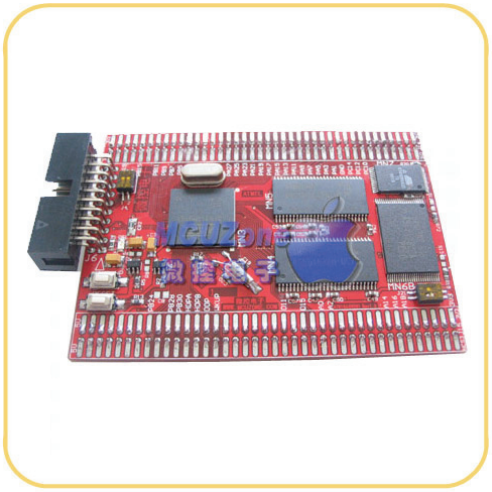
可以看到例子很丰富，而且每个例子都分 GNU、Keil、IAR 三个版本，方便不同平台的用户使用。

MDK9261 核心板

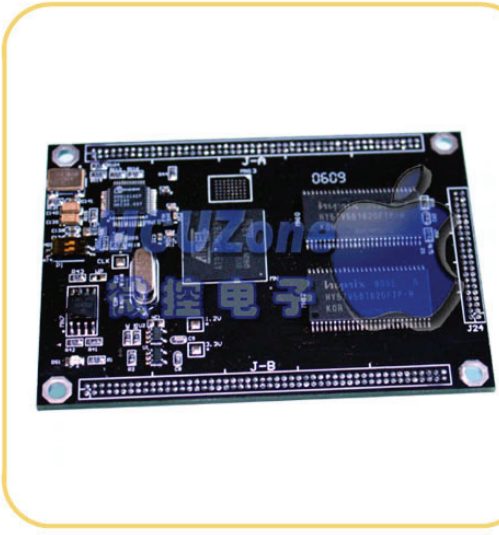




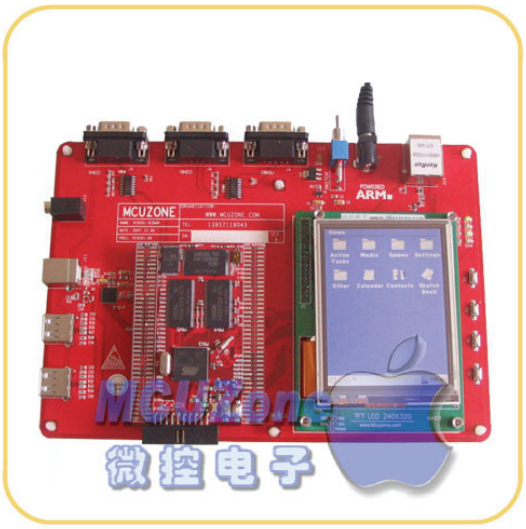
VC9261 核心板



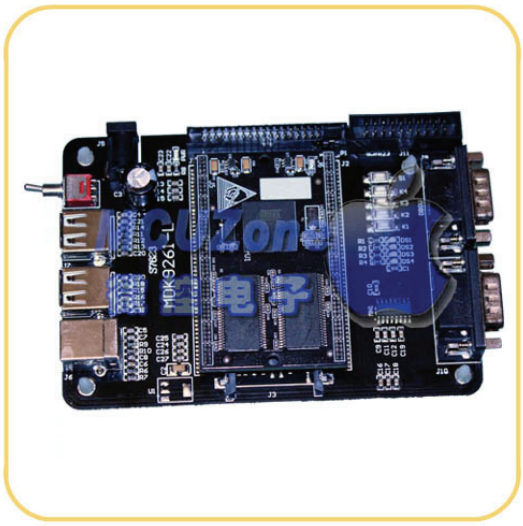
MDK9263 核心板



VC9261-EK 开发板



MDK9261-L 开发板（￥400）



AT9261-EK 开发板