北 京 邮 电 大 学

计算机与信息网络实验教学中心

嵌入式系统（本科）
实验指导手册
V1.1

2015 年 06 月

1 说明....4
2 实验时间与方式....4
3 怎样获取实验设备....4
4 TQ2440 开发板说明....4
4.1 开发板接口说明....4
4.2 开发板接线说明....7
5 ADS 开发简介....8
5.1 ADS 集成开发环境组成介绍....9
5.1.1 命令行开发工具....9
5.1.2 ARM 运行时库....10
5.1.3 CodeWarrior 集成开发环境....12
5.1.4 ADS 调试器....14
5.1.5 实用程序....15
5.2 使用 ADS 创建工程....15
5.2.1 建立一个工程....15
5.2.2 配置生成目标....18
5.2.3 编译连接工程....21
5.3 工程的调试....22
5.3.1 调试工具条....22
5.4 JTAG 仿真器及超级终端....23
5.4.1 J-LINK 仿真器的连接....23
5.4.2 超级终端的使用....24
6 实验参考资料....28
7 实验环境准备....28
7.1 认识 TQ2440 开发板....28
7.2 连接开发板与宿主机....29
7.3 宿主机软件环境准备....29
7.4 启动系统....30
7.5 实验注意事项....30
8 实验 1：GPIO 实验—LED 与按键中断....31
9 实验 2：UART 实验....31
10 实验 3：ucOSII 多任务、信号量实验....32
11 实验 4：ucOSII 多任务、信号量、消息队列及 S3c2440 I/O port 综合实验....33
12 实验 1 详细实验步骤....34
12.1 分析实验电路....34
12.2 理解实验原理....37
12.2.1 GPIO 寄存器....37
12.2.2 中断寄存器....40
12.3 连接实验板....41
12.4 打开 PC 端软件....42
12.5 建立 ADS 工程....42
12.6 添加或编写代码....44
12.7 配置工程....46
12.8 调试代码....53

12.9 运行结果.....55
13 实验问题.....57
13.1 J-LINK 固件升级.....57
13.2 ADS 无法调试和运行代码.....58
13.3 J-LINK 仿真器烧写 Nor Flash 失败.....60
13.4 双击打不开 mcp 工程文件.....63
13.5 AXD 调试/运行时提示找不到 2440Init.s.....64
13.6 AXD 调试/运行时找不到.ses 文件.....66

1 说明

本实验手册为修嵌入式系统课程实验的同学提供简要的指导。

2 实验时间与方式

请关注嵌入式系统课程相关通知。

3 怎样获取实验设备

首先请在主楼 9 层值班室（出电梯处的玻璃门）办理实验登记和设备借用手续。

910 有不同实验课程用的多种实验设备，本实验使用 2440 开发板（910 2 号柜，蓝色实验板），请向值班室老师指明要哪种设备。

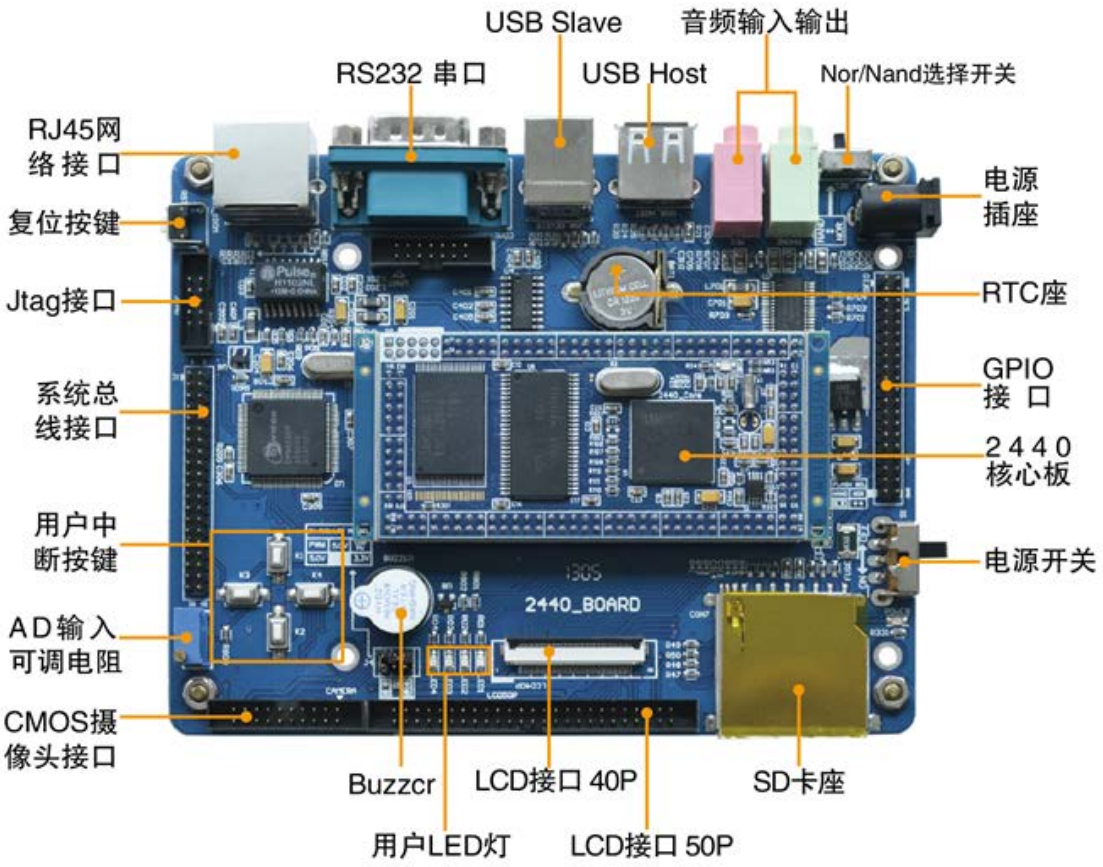
实验后，请将设备整理好，放回实验柜，去值班室交还钥匙。

4 TQ2440 开发板说明

TQ2440 是一款基于三星 S3C2440 16/32 位 RISC 处理器（ARM920T）的实验开发板。

4.1 开发板接口说明

如下图所示，标识出了 TQ2440 上面的常用接口：



接口说明：

◆ **电源接口**

请一定注意输入电压不要大于 7V，TQ2440 标配的电源适配器是 5V 供电。在不清楚适配器输出电压时，请向适配器的供应商确认输出电压或自行用电压表测试输出电压；以防止因为输入电压过高而导致开发板出现损坏的情况。

◆ **Nor/Nand 选择开关**

当从 Nor Flash 启动时，请在开发板开机前将该开关拨到远离绿色接口的地方。

当从 Nand Flash 启动时，请在开发板开机前将该开关拨到靠近绿色接口的地方。

◆ **音频输入输出接口**

TQ2440 提供的音频接口完全按照标准接口提供，绿色为音频输出接口，红色为音频输入接口。

◆ **USB 接口（USB Host 和 USB Slave）**

在 TQ2440 开发板中有两个 USB 接口，一个是 USB A 口（开发板上面的 USB_HOST 接口，作为 HOST 使用，主要用于接 U 盘，USB 摄像头等设备）；另外一个是 USB B 口（开发板上面的 USB_Deive 接口，使用标配的 USB 延长线，连接到 PC，用于传输数据）。

当使用 USB 下载功能时，需要连接标配提供的 USB 延长线到开发板和 PC 直接。

◆ **RTC 备份电池**

没有安装电池。

◆ **串口接口（串口 0（RS232）和三个串口扩展接口（3.3V 电平））**

TQ2440 开发板提供的标配串口线是直连串口线，在使用开发板时请把串口线的一段接开发板的串口接口，另外一端接 PC 的串口接口，然后就可以通过串口进行交互等操作了。

三串口扩展接口引出的串口的 TX 和 RX 引脚均是 3.3V 电平，如果需要使用 232 电平，请扩展 MAX3232 或者 SP3232EEN 芯片转成 232 电平。

◆ **网卡接口（RJ45 网络座子）**

TQ2440 开发板提供了 100M 网卡接口。在启动操作系统后，接上网线就可以进行上网等操作；在 uboot 的下载模式下面可以使用 TFTP 下载数据到开发板。

◆ **复位按钮**

硬件重启开发板时使用。

◆ **Jtag 接口**

在 TQ2440 开发板中，Jtag 的用途是当 Nand Flash 或 Nor Flash 中没有 uboot 时，使用它烧写 uboot 进去；或者是进行仿真时使用它。

使用时请接上 J-LINK 仿真器到开发板的 Jtag 接口和 PC 的 USB 接口，然后再使用 Jtag 软件进行烧写或仿真操作。

◆ **系统总线接口**

引出了数据总线和地址总线等。

◆ **用户按钮**

使用中断功能的 4 个用户按钮。

◆ **AD 输入测试电阻**

ADC 接口（AIN2 管脚）。

◆ **Camera 接口**

接 camera。

◆ **蜂鸣器（PWM 控制）**

使用 PWM 控制的蜂鸣器。

◆ **蜂鸣器/LCD 电压选择接口**

打开或关断蜂鸣器的电源，选择 LCD 供电电压。

◆ **用户 LED 灯**

使用 GPIO 口控制的 LED 灯。

◆ **LCD 接口（LCD 接口 1 和 2）**

在接 LCD 接口时请注意排线接口的定位方向，请不要接反了。

说明：这两个接口是使用的相同的数据线和控制线，唯一不同的是一个是 FPC 接口而另外一个是插针形式的接口。

注意：在使用东芝屏的时候，请注意不要碰到 LCD 驱动板背后的高压包了，虽然天嵌科技已经对高压包的高压输出脚打胶隔离了，但是可能会被电到，造成不必要的麻烦。

◆ **SD 卡接口**

接 SD 卡时，接口面朝下插入即可，支持 SD 卡和 SDHC 卡。

◆ **电源指示灯**

正常开机后，将会点亮红色的 LED 灯。

◆ **电源开关**

控制着整个开发板的供电，需要开机时请拨动该开关到靠近电源指示灯的方向，关机时拨动该开关到反方向。

◆ **AT24C02**

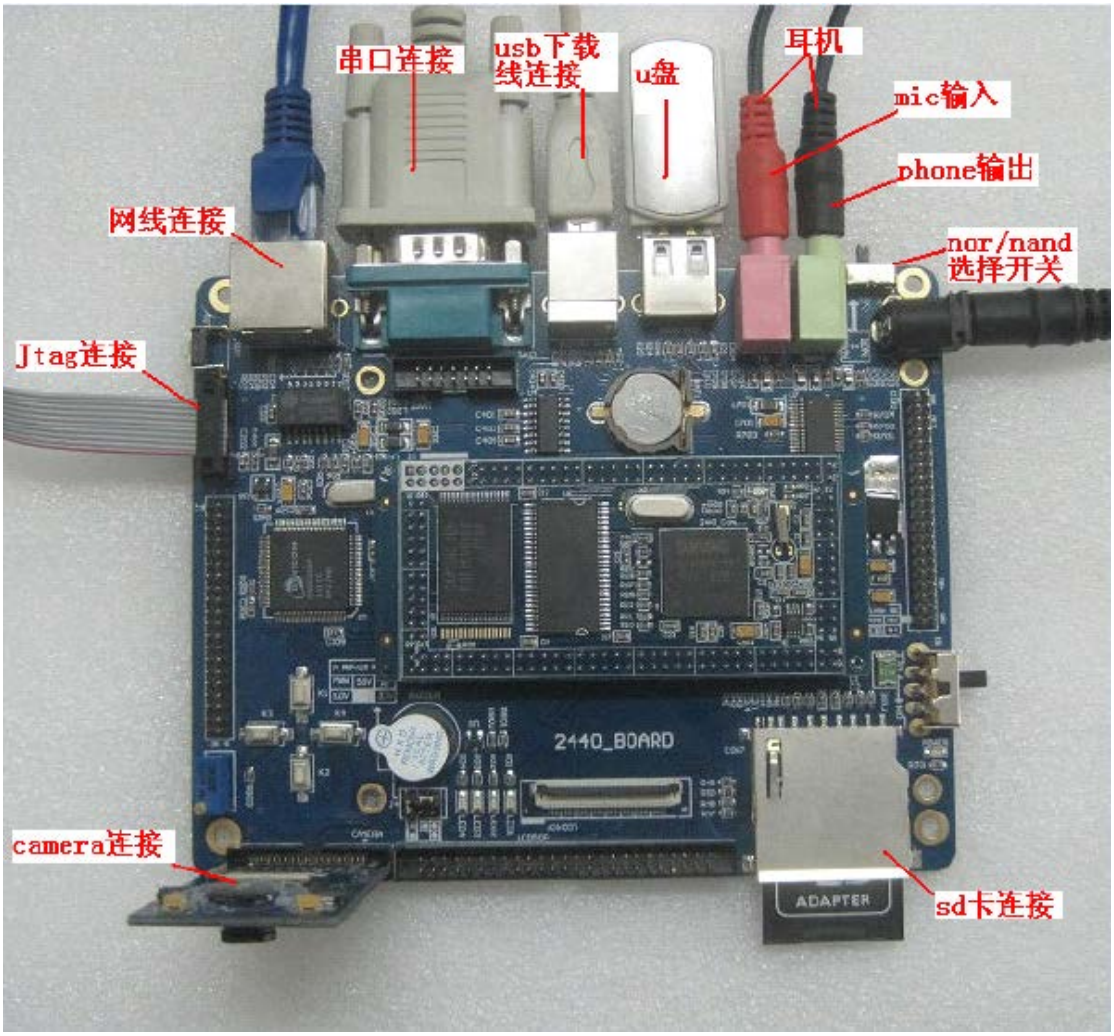
IIC 接口的 EEPROM 芯片，用于测试 IIC 接口。

◆ **GPIO 接口**

不光包含 GPIO 口，还包含了 AIN0 到 AIN3、SPI、IIC 等接口。

4.2 开发板接线说明

下图是 TQ2440 接上所有的线的实例图：



接线说明：

- 当需要使用 USB 烧写镜像到 TQ2440 时，需要连接串口线、USB 下载线和电源线；
- 当需要使用网络烧写镜像到 TQ2440 时，需要连接串口线，网线和电源线；
- 当需要使用 Jtag 烧写镜像到 TQ2440 时，需要连接 Jtag 线和电源线；

5 ADS 开发简介

在这一章里，将介绍 ARM 开发软件 ADS（ARM Developer Suite）。通过学习如何在 CodeWarrior IDE 集成开发环境下编写，编译一个工程的例子，使读者能够掌握在 ADS 软件平台下开发用户应用程序。本章还描述了如何使用 AXD 调试工程，并通过 JTAG 仿真，最终在超级终端中查看结果。

5.1 ADS 集成开发环境组成介绍

ARM ADS 全称为 ARM Developer Suite。是 ARM 公司推出的新一代 ARM 集成开发工具。现在 ADS 的最新版本是 1.2，它取代了早期的 ADS1.1 和 ADS1.0。它除了可以安装在 Windows NT4，Windows 2000，Windows 98 和 Windows 95 操作系统下，还支持 Windows

XP 和 Windows Me 操作系统。

ADS 由命令行开发工具，ARM 时实库，GUI 开发环境（Code Warrior 和 AXD），实用程序和支持软件组成。有了这些部件，用户就可以为 ARM 系列的 RISC 处理器编写和调试自己的开发应用程序了。

下面就详细介绍一下 ADS 的各个组成部分。

5.1.1 命令行开发工具

命令行开发工具完成将源代码编译，链接成可执行代码的功能。ADS 提供下面的命令行开发工具：

armcc：armcc 是 ARM C 编译器。这个编译器通过了 Plum Hall C Validation Suite 为 ANSI C 的一致性测试。armcc 用于将用 ANSI C 编写的程序编译成 32 位 ARM 指令代码。

因为 armcc 是我们最常用的编译器，所以对此作一个详细的介绍。

在命令控制台环境下，输入命令：

armcc –help

可以查看 armcc 的语法格式以及最常用的一些操作选项

armcc 最基本的用法为： armcc [options] file1 file2 ... filen

这里的 option 是编译器所需要的选项，fiel1，file2…filen 是相关的文件名。这里简单介绍一些最常用的操作选项。

-c：表示只进行编译不链接文件；

-C：（注意：这是大写的 C）禁止预编译器将注释行移走；

-D<symbol>：定义预处理宏，相当于在源程序开头使用了宏定义语句#define symbol ，这里 symbol 默认为 1；

-E：仅仅是对 C 源代码进行预处理就停止；

-g<options>：指定是否在生成的目标文件中包含调试信息表；

-I<directory>：将 directory 所指的路径添加到#include 的搜索路径列表中去；

-J<directory>：用 directory 所指的路径代替默认的对#include 的搜索路径；

-o<file>：指定编译器最终生成的输出文件名。

-O0：不优化；

-O1：这是控制代码优化的编译选项，大写字母 O 后面跟的数字不同，表示的优化级别就不同，-O1 关闭了影响调试结果的优化功能；

-O2：该优化级别提供了最大的优化功能；

-S：对源程序进行预处理和编译，自动生成汇编文件而不是目标文件；

-U<symbol>：取消预处理宏名，相当于在源文件开头，使用语句#undef symbol;

-W<options>：关闭所有的或被选择的警告信息；

有关更详细的选项说明，读者可查看 ADS 软件的在线帮助文件。

armcpp：armcpp 是 ARM C++编译器。它将 ISO C++ 或 EC++ 编译成 32 位 ARM 指令代码。

tcc：tcc 是 Thumb C 编译器。该编译器通过了 Plum Hall C Validation Suite 为 ANSI 一致性的测试。tcc 将 ANSI C 源代码编译成 16 位的 Thumb 指令代码。

tcpp：tcpp 是 Thumb C++ 编译器。 它将 ISO C++ 和 EC++ 源码编译成 16 位 Thumb 指令代码。

armasm：armasm 是 ARM 和 Thumb 的汇编器。它对用 ARM 汇编语言和 Thumb 汇编语言写的源代码进行汇编。

armlink：armlink 是 ARM 连接器。该命令既可以将编译得到的一个或多个目标文件和相关的一个或多个库文件进行链接，生成一个可执行文件，也可以将多个目标文件部分链接成一个目标文件，以供进一步的链接。ARM 链接器生成的是 ELF 格式的可执行映像文件。

armsd：armsd 是 ARM 和 Thumb 的符号调试器。它能够进行源码级的程序调试。用户可以在用 C 或汇编语言写的代码中进行单步调试，设置断点，查看变量值和内存单元的内容。

5.1.2 ARM 运行时库

1．运行时库类型和建立选项

ADS 提供以下的运行时库来支持被编译的 C 和 C++代码：

ANSI C 库函数：这个 C 函数库是由以下几部分组成： 在 ISO C 标准中定义的函数；

在 semihosted 环境下（semihosting 是针对 ARM 目标机的一种机制，它能够根据应用程序代码的输入/输出请求，与运行有调试功能的主机通讯。这种技术允许主机为通常没有输入和输出功能的目标硬件提供主机资源）用来实现 C 库函数的与目标相关的函数；

被 C 和 C++编译器所调用的支持函数。

ARM C 库提供了额外的一些部件支持 C++，并为不同的结构体系和处理器编译代码。

C++库函数：C++库函数包含由 ISO C++库标准定义的函数。C++库依赖于相应的 C 库实现与特定目标相关的部分，在 C++库的内部本身是不包含与目标相关的部分。这个库是由以下几部分组成的：

版本为 2.01.01 的 Rogue Wave Standard C++库；

C++编译器使用的支持函数；

Rogue Wave 库所不支持的其他的 C++函数。

正如上面所说，ANSI C 库使用标准的 ARM semihosted 环境提供例如，文件输入/输出的功能。Semihosting 是由已定义的软件中断（Software Interrupt）操作来实现的。在大多数的情况下，semihosting SWI 是被库函数内部的代码所触发，用于调试的代理程序处理 SWI 异常。调试代理程序为主机提供所需要的通信。Semihosted 被 ARMulator ，Angel 和 Multi-ICE 所支持。用户可以使用在 ADS 软件中的 ARM 开发工具去开发用户应用程序，然后在 ARMulator 或在一个开发板上运行和调试该程序。

用户可以把 C 库中的与目标相关的函数作为自己应用程序中的一部分，重新进行代码的实现。这就为用户带来了极大的方便，用户可以根据自己的执行环境，适当的裁剪 C 库函数。除此之外，用户还可以针对自己的应用程序的要求，对与目标无关的库函数进行适当的裁剪。

在 C 库中有很多函数是独立于其他函数的，并且与目标硬件没有任何依赖关系。对于这类函数，用户可以很容易地从汇编代码中使用它们。在建立自己的用户应用程序的时候，用户必须指定一些最基本的操作选项。例如：

字节顺序，是大端模式（big endian：字数据的高字节存放在低地址，低字节存放在高地址），还是小端模式（little endian：字数据的高字节存放在高地址，低字节存放在低地址）； 浮点支持：可能是 FPA，VFP，软件浮点处理或不支持浮点运算； 堆栈限制：是否检查堆栈溢出；

位置无关（PID）：数据是从与位置无关的代码还是从与位置相关的代码中读/写，代码是位置无关的只读代码还是位置相关的的只读代码。

当用户对汇编程序，C 程序或 C++程序进行链接的时候，链接器会根据在建立时所指定的选项，选择适当的 C 或 C++运行时库的类型。选项各种不同组合都有一个相应的 ANSI

C 库类型。

2．库路径结构

库路径是在 ADS 软件安装路径的 lib 目录下的两个子目录。假设，ADS 软件安装在 D：

\arm\adsv1_2 目录，则在 D：\arm\adsv1_2\lib 目录下的两个子目录 armlib 和 cpplib 是 ARM 的库所在的路径。

armlib：这个子目录包含了 ARM C 库，浮点代数运算库，数学库等各类库

函数。与这些库相应的头文件在 D：\arm\adsv1_2\include 目录中。

cpplib：这个子目录包含了 Rogue Wave C++库和 C++支持函数库。Rogue Wave C++库和 C++支持函数库合在一起被称为 ARM C++库。与这些库相应的头文件安装在 D：

\arm\adsv1_2\include 目录下。

环境变量 ARMLIB 必须被设置成指向库路径。另外一种指定 ARM C 和 ARM C++库路径的方法是，在链接的时候使用操作选项-libpath directory（directory 代表库所在的路径），来指明要装载的库的路径。无需对 armlib 和 cpplib 这两个库路径分开指明，链接器会自动从用户所指明的库路径中找出这两个子目录。

这里需要让读者特别注意的以下几点：

ARM C 库函数是以二进制格式提供的；

ARM 库函数禁止修改。如果读者想对库函数创建新的实现的话，可以把这个新的函数编译成目标文件，然后在链接的时候把它包含进来。这样在链接的时候，使用的是新的函数实现而不是原来的库函数。

通常情况下，为了创建依赖于目标的应用程序，在 ANSI C 库中只有很少的几个函数需要实现重建。

Rogue Wave Standard C++函数库的源代码不是免费发布的，可以从 Rogue Wave Software Inc.，或 ARM 公司通过支付许可证费用来获得源文件。

5.1.3 CodeWarrior 集成开发环境

CodeWarrior for ARM 是一套完整的集成开发工具，充分发挥了 ARM RISC 的优势，使产品开发人员能够很好的应用尖端的片上系统技术。该工具是专为基于 ARM RISC 的处理器而设计的，它可加速并简化嵌入式开发过程中的每一个环节，使得开发人员只需通过一个集成软件开发环境就能研制出 ARM 产品，在整个开发周期中，开发人员无需离开 CodeWarrior 开发环境，因此节省了在操作工具上花的时间，使得开发人员有更多的精力投入到代码编写上来。

CodeWarrior 集成开发环境（IDE）为管理和开发项目提供了简单多样化的图形用户界面。用户可以使用 ADS 的 CodeWarrior IDE 为 ARM 和 Thumb 处理器开发用 C，C++，或 ARM 汇编语言的程序代码。通过提供下面的功能，CodeWarrior IDE 缩短了用户开发项目代码的周期。

全面的项目管理功能；

子函数的代码导航功能，使得用户迅速找到程序中的子函数；

可以在 CodeWarrior IDE 为 ARM 配置在 4.1.1 中介绍的各种命令工具，实现对工程代码的编译，汇编和链接。

在 CodeWarrior IDE 中所涉及到的 target 有两种不同的语义。

目标系统（Target system）：是特指代码要运行的环境，是基于 ARM 的硬

件。比如，要为 ARM 开发板上编写要运行在它上面的程序，这个开发板就是目标系统。

生成目标（Build target）：是指用于生成特定的目标文件的选项设置（包括汇编选项，编译选项，链接选项以及链接后的处理选项）和所用的文件的集合。CodeWarrior IDE 能够让用户将源代码文件，库文件还有其他相关的文件以及配置设置等放在一个工程中。每个工程可以创建和管理生成目标设置的多个配置。例如，要编译一个包含调试信息的生成目标和一个基于 ARM7TDMI 的硬件优化生成目标，生成目标可以在同一个工程中共享文件，同时使用各自的设置。

CodeWarrior IDE 为用户提供下面的功能：

源代码编辑器，它集成在 CodeWarrior IDE 的浏览器中，能够根据语法格式，使用不同的颜色显示代码；

源代码浏览器，它保存了在源码中定义的所有符号，能够使用户在源码中快速方便的跳转； 查找和替换功能，用户可以在多个文件中，利用字符串通配符，进行字符串的搜索和替换；

文件比较功能，可以使用户比较路径中的不同文本文件的内容。

ADS 的 CodeWarrior IDE 是基于 Metrowerks CodeWarrior IDE 4.2 版本的。它经过适当的裁剪以支持 ADS 工具链。针对 ARM 的配置面板为用户提供了在 CodeWarrior IDE 集成环境下配置各种 ARM 开发工具的能力，这样用户可以不用在命令控制台下就能够使用在上文中介绍的各种命令。

以 ARM 为目标平台的工程创建向导，可以使用户以此为基础，快速创建 ARM 和 Thumb 工程。尽管大多数的 ARM 工具链已经集成在 CodeWarrior IDE，但是仍有许多功能在该集成环境中没有实现，这些功能大多数是和调试相关的，因为 ARM 的调试器没有集成到 CodeWarrior IDE 中。由于 ARM 调试器（AXD）没有集成在 CodeWarrior IDE 中，这就意味着，用户不能在 CodeWarrior IDE 中进行断点调试和查看变量。对于熟悉 CodeWarrior IDE 的用户会发现，有许多的功能已经从 CodeWarrior IDE For ARM 中移走，比如快速应用程序开发模板等。

在 CodeWarrior IDE For ARM 中有很多的菜单或子菜单是不能使用的。下面介绍一下这些不能使用的选项。

1．View 菜单下不能使用的菜单选项有：

Processes，Expressions，Global Variable，Breakpoints，Registers。

2．Project 菜单不能使用的菜单选项：

Precompile 子菜单。因为 ARM 编译器不支持预编译的头文件。

3．Debug 菜单

该菜单中没有一个子菜单是可以使用的。

4．Browser 菜单中不能使用的菜单选项：

New Property，New Method 和 New Event Set。

5．Help menu 中不能用于 ADS 的菜单选项有：

CodeWarrior Help，Index，Search 和 Online Manuals。

有关 CodeWarrior IDE 中一些常用菜单的使用，将在后面的举例中具体说明的，在此，不在赘述。

5.1.4 ADS 调试器

调试器本身是一个软件，用户通过这个软件使用 debug agent 可以对包含有调试信息的，正在运行的可执行代码进行比如变量的查看，断点的控制等调试操作。

ADS 中包含有 3 个调试器：

AXD（ARM eXtended Debugger）：ARM 扩展调试器； armsd（ARM Symbolic Debugger）：ARM 符号调试器；

与老版本兼容的 Windows 或 Unix 下的 ARM 调试工具，ADW/ADU（Application

Debugger Windows/Unix）。

下面对在调试映像文件中所涉及到的一些术语做一个简单的介绍。

Debug target：在软件开发的最初阶段，可能还没有具体的硬件设备。如果要测试所开发的软件是否达到了预期的效果，这可以由软件仿真来完成。即使调试器和要测试的软件运行在同一台 PC 上，也可以把目标当作一个独立的硬件来看待。当然，也可以搭建一个 PCB 板，这个板上可以包含一个或多个处理器，在这个板上可以运行和调试应用软件。只有当通过硬件或者是软件仿真所得到的结果达到了预期的效果，才算是完成了应用程序的编写工作。

调试器能够发送以下指令：

1．装载映像文件到目标内存；

2．启动或停止程序的执行；

3．显示内存，寄存器或变量的值；

4．允许用户改变存储的变量值。

Debug agent：Debug agent 执行调试器发出的命令动作，比如：设置断点，从存储器中读数据，把数据写到存储器等。Debug agent 既不是被调试的程序，也不是调试器。在 ARM 体系中，它有这几种方式：Multi-ICE（Multi-processor in-circuit emulator），ARMulator 和 Angel。其中 Multi-ICE 是一个独立的产品，是 ARM 公司自己的 JTAG 在线仿真器，不是由 ADS 提供的。

AXD 可以在 Windows 和 UNIX 下，进行程序的调试。它为用 C，C++，和汇编语言编写的源代码提供了一个全面的 Windows 和 UNIX 环境。

在后面的章节中，会结合具体实例为读者介绍如何使用 AXD 调试器。

5.1.5 实用程序

ADS 提供以下的实用工具来配合前面介绍的命令行开发工具的使用。

fromELF：这是 ARM 映像文件转换工具。该命令将 ELF 格式的文件作为输入文件，将该格式转换为各种输出格式的文件，包括 plain binary（BIN 格式映像文件），Motorola 32-bit S-record format（Motorola 32 位 S 格式映像文件），Intel Hex 32 format（Intel 32 位格式映像文件），和 Verilog-like hex format（Verilog 16 进制文件）。FromELF 命令也能够为输入映像文件产生文本信息，例如代码和数据长度。

armar：ARM 库函数生成器将一系列 ELF 格式的目标文件以库函数的形式集合在一起，用户可以把一个库传递给一个链接器以代替几个 ELF 文件。

FLASH DOWNLOADER：用于把二进制映像文件下载到 ARM 开发板上的 FLASH 存储器的工具。

5.2 使用 ADS 创建工程

本节通过一个具体实例，为读者介绍如何使用该集成开发环境，利用 CodeWarrior 提供的建立工程的模板建立自己的工程，并学会如何进行编译链接，最终生成可执行文件。

5.2.1 建立一个工程

点击 WINDOWS 操作系统的“开始|程序|ARM Developer Suite v1.2 |Code Warrior for ARM Developer Suite”启动 Metrowerks Code Warrior，或双击“ADS 1.2”快捷方式启动。

启动 ADS 1.2 如图 7.1 示：

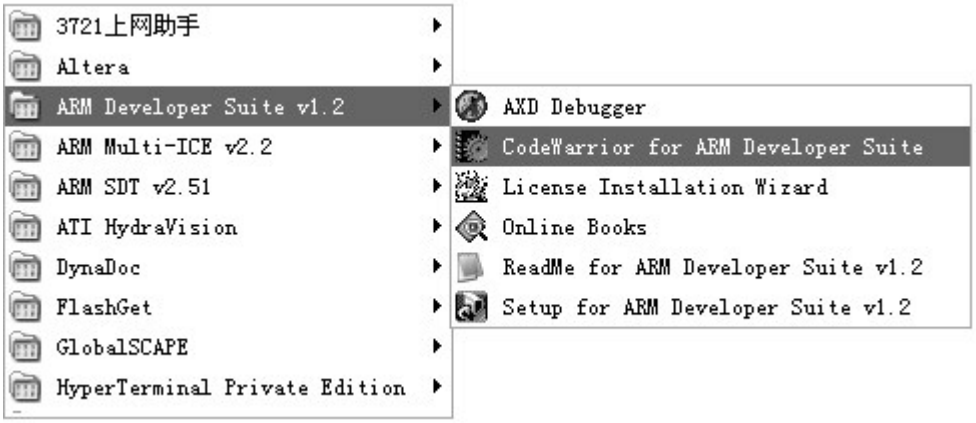


图 7.1 启动 ADS 1.2

在 CodeWarrior 中新建一个工程的方法有两种，可以在工具栏中单击“New”

按钮，也可以在“File”菜单中选择“New…”菜单。这样就会打开一个如图 7.2 所示的对话框。

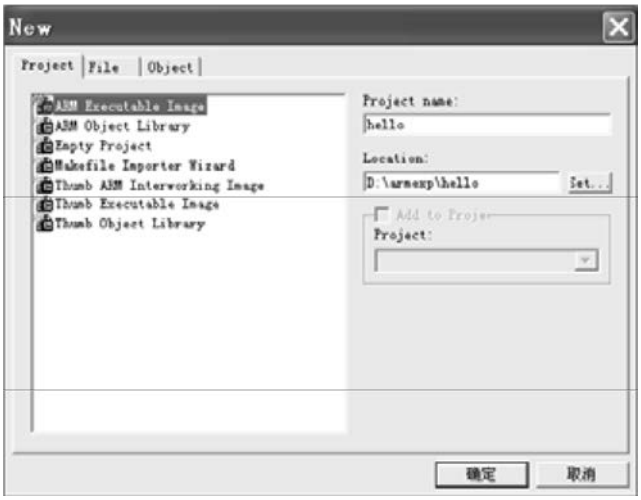


图 7.2　新建工程对话框

在这个对话框中为用户提供了 7 种可选择的工程类型。

ARM Executabl Image：用于由 ARM 指令的代码生成一个 ELF 格式的可执行映像文件；

ARM Object Library：用于由 ARM 指令的代码生成一个 armar 格式的目标文件库； Empty Project：用于创建一个不包含任何库或源文件的工程；

Makefile Importer Wizard：用于将 Visual C 的 nmake 或 GNU make 文件转入到

CodeWarrior IDE 工程文件；

Thumb ARM Executable Image：用于由 ARM 指令和 Thumb 指令的混和代码生成一个可执行的 ELF 格式的映像文件；

Thumb Executable image：用于由 Thumb 指令创建一个可执行的 ELF 格式的映像文件；

Thumb Object Library：用于由 Thumb 指令的代码生成一个 armar 格式的目标文件库。在这里选择 ARM Executable Image，在“Project name：”中输入工程文件名，本例为 “HelloWorld”，点击“Location：”文本框的“Set…”按钮，浏览选择想要将该工程保存的路径，将这些设置好后，点击“确定”，即可建立一个新的名为 HelloWorld 的工程。这个时候会出现 HelloWorld.mcp 的窗口，如图 7.3 所示，有三个标签页，分别为 files，link order，target 默认的是显示第一个标签页 files。通过在该标签页点击鼠标右键，选中“Add

Files…”可以把要用到的源程序添加到工程中。

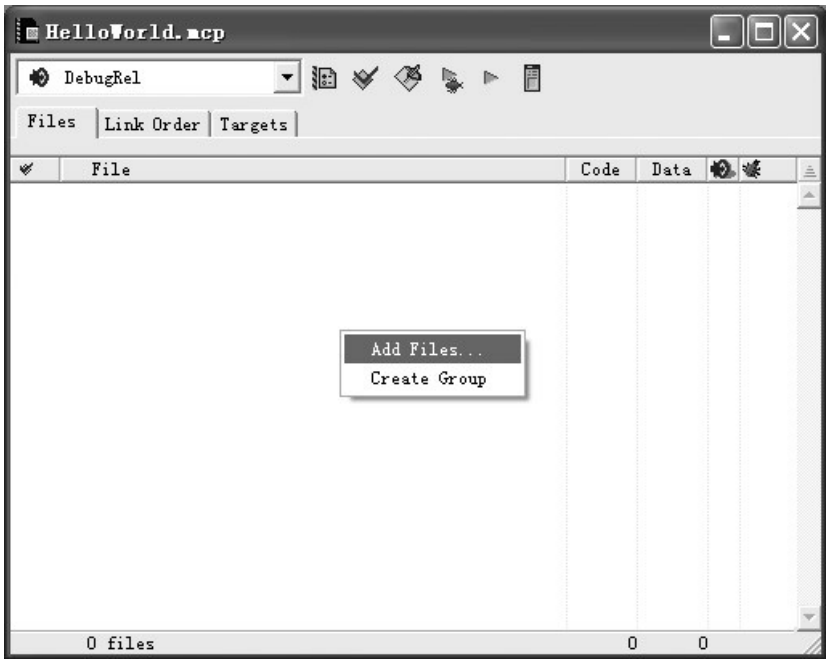


图 7.3　新建工程打开窗口

对于本例，由于所有的源文件都还没有建立，所以首先需要新建源文件。在“File” 菜单中选择“New”，在打开的如图 7.2 所示的对话框中，选择标签页 File，在 File name 中输入要创建的文件名，输入“hello.s”，点击“确定”关闭窗口。在打开的文件编辑框中输入下面的汇编代码：

作为一个最简单的示例，hello.s 源文件如下所示。

```
AREA    Example1，CODE，READONLY    ；声明代码段 Example1
ENTRY                                ；标识程序入口
CODE32                               ；声明 32 位 ARM 指令
START   MOV     R0，#15                      ；设置参数
MOV     R1，#8
ADDS    R0，R0，R1                   ；R0=R0+R1
B       START
END
```

在这里还有一个细节，希望读者注意。在建立好一个工程时，默认的 target 是 DebugRel，还有另外两个可用的 target，分别为 Realse 和 Debug，这三个 target 的含义分别为：

DebugRel：使用该目标，在生成目标的时候，会为每一个源文件生成调试信息；

Debug：使用该目标为每一个源文件生成最完全的调试信息；

Release：使用该目标不会生成任何调试信息。

在本例中，使用默认的 DebugRel 目标。现在已经新建了一个源文件，要把

两个源文件添加到工程中去。

为工程添加源码常用的方法有两种，既可以使用如图 7.3 所示方法，也可以在“Project” 菜单项中，选择“Add Files…”，这两种方法都会打开文件浏览框，用户可以把已经存在的文件添加到工程中来。当选中要添加的文件时，会出现一个对话框，如图 7.4 所示，询问用户把文件添加到何类目标中，在这里，我们选择 DebugRel 目标。把刚才创建的文件添加到工程中来。

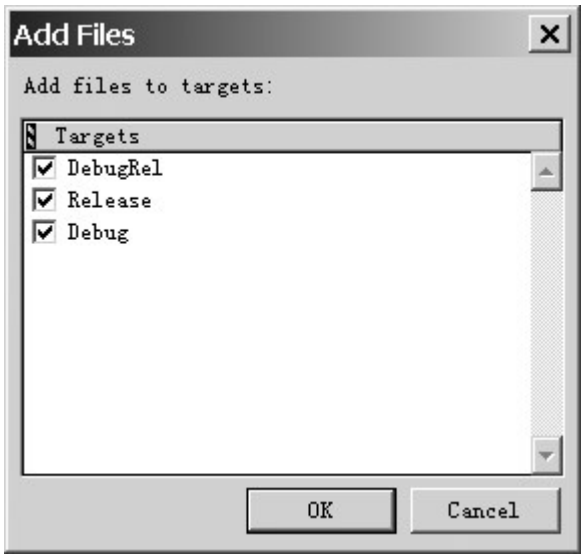


图 7.4　选择添加文件到指定目标

到目前为止，一个完整的工程已经建立，下面该对工程进行编译和链接工作。

5.2.2 配置生成目标

1．打开项目“hello.mcp”，在此项目窗口中，打开目标选择下拉表框，选择 Debug 生成目标，如图 7.5 所示；单击右侧的 Target Setting（此时已变为 Debug Setting）图标。如图 7.6 所示。

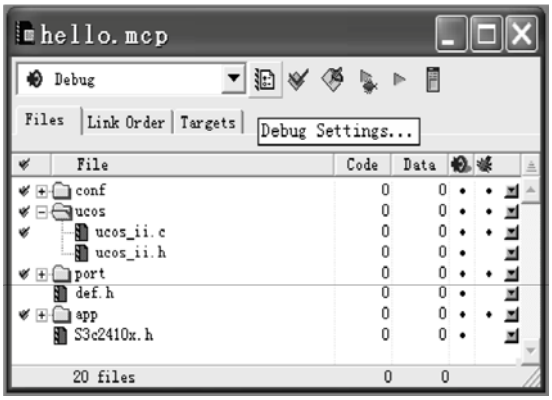


图 7.5 准备设置 Target Setting

2．在 Debug Setting 中包括 6 个面板，这里我们选择如下面板设置相关的生成选项：

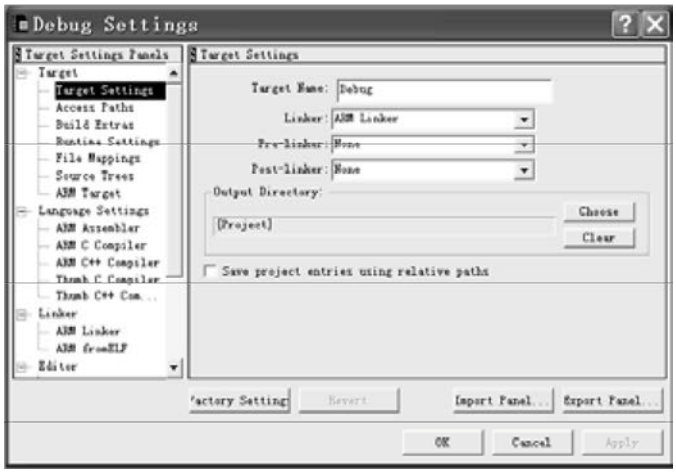


图 7.6 Target Setting 对话框

1) 设置生成目标的基本选项（Target Settings），这里请按照图 7.6 对该选项进行相应的设置。

2) 编译器的选项设置（Language Settings）。Language Settings 目录下选 ARM C Compiler，由于目标板采用的 S3C2410ARM 芯片属于 ARM9 系列，这里需要将该选项中的各个子选项对话框的“Target”或“Target and Source”选项卡下的“Architecture or Processor”对话框设定在“ARM920T”。如图 7.7 所示：

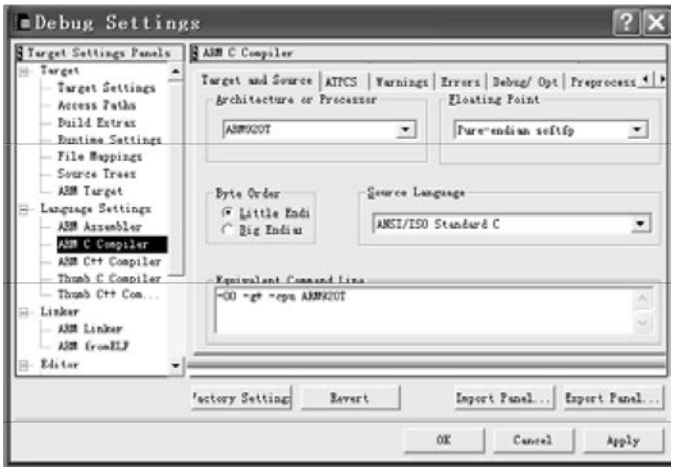


图 7.7 Language Settings 对话框

3) 连接器的选项设置（Linker）在 Target Settings Panels 列表框中选择 Linker 选项，再在其下选择 ARM Linker，即可得到连接器的选项设置对话框，如图 7.8 所示。

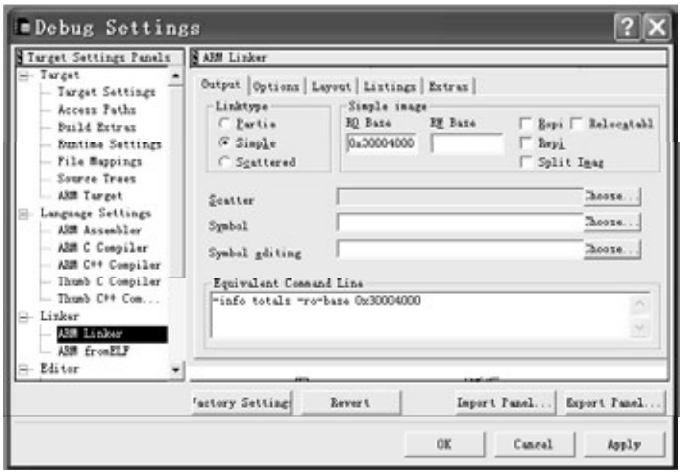


图 7.8 连接器选项设置

OUTPUT 选项卡：该选项卡用来控制连接器进行连接操作的类型。其中 Linktype 选项组中的单选按钮确定使用的连接方式。这里选择 Simple，连接器将根据连接器选项中指定的地址映射方式，生成简单的 ELF 格式的映像文件，所生成的映像文件中的地址映射关系比较简单.当选择 Simple 连接类型时，需要设置下列的连接器选项，如图 7.8 所示。

RO Base 文本框中填入 0x30000000。地址 0x30000000 是开发板上 SDRAM 的真实地址，是由系统的硬件决定的；RW Base 文本框中填入 0x31000000 指的是系统可读写内存的地址。也就是说，在 0x30000000－0x31000000 之间是只读区域，存放程序的代码段，从 0x31000000 开始是程序的数据段。

Layout 选项卡：该选项卡在连接方式位 Simple 时有效，它用来安排一些输入段在映像文件中的位置。Placeat beginning of 选项组用于指定将某个输入段放置在它所在的运行时域的开头。包含复位异常中断处理程序的输入段通常放置在运行时域的开头。

这里，在 Object/Symbol 文本框中指定目标文件的名称 init.o，在 Section 文本框中指定输入段的名称 init，从而确定了 init.s 源文件中的 init 输入段位指定的输入段。如图 7.9 所示。

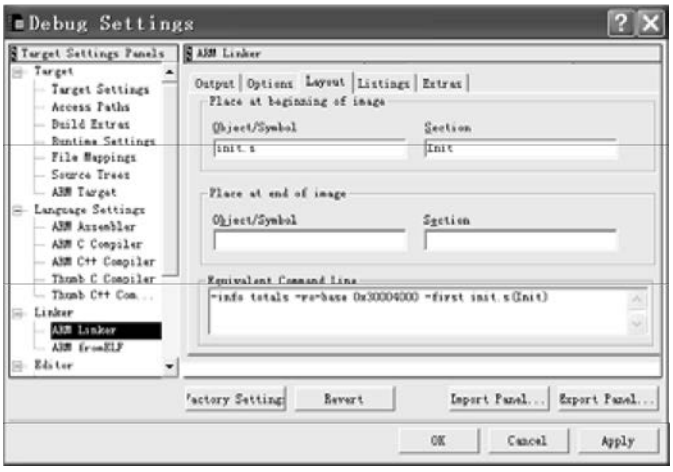


图 7.9 Layout 选项卡中连接器选项

到这里，对 Debug Setting 的设置基本完毕。点击图 7.9 中的 Apply 按钮后点击 OK 按钮，保存所有的设置。

5.2.3 编译连接工程

如图 7.10 所示为工程窗口中的图标按钮，通过这些图标按钮，可以快速的进行工程设置， 编译连接，启动调试等等.它们从左到右分别为

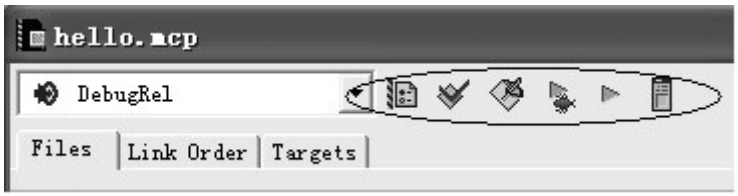


图 7.10 工程窗口中的图标按钮

DebugRel Settings… 工程设置

Synchronize Modification Dates 同步修改日期

Make 编译连接

Debug 启动 ADX 进行调试

Run 启动 ADX 调试，并直接运行

Project Inspector 工程检查，查看和配置工程中源文件的信息

对于简单的软件调试，直接点击工程窗口的”Make”图标按钮，即可完成编译.编译连接输出窗口如图 7.11 所示。

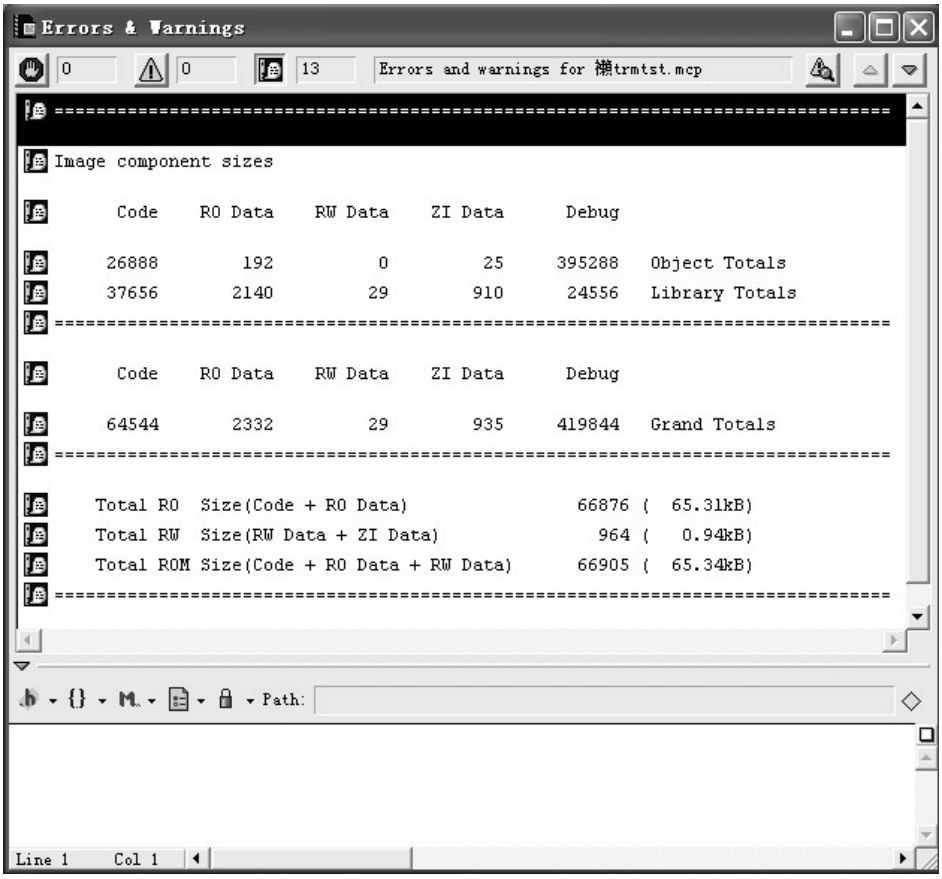


图 7.11 编译连接输出窗口

5.3 工程的调试

5.3.1 调试工具条

AXD运行调试工具条如图 7.12所示，调试观察窗口工具条如图 7.13所示，文件操作工具条如图 7.14 所示.下面具体说明：

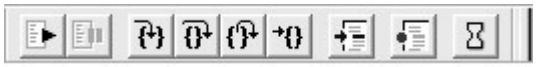


图 7.12 运行调试工具条

全速运行（Go）

停止运行（Stop）

单步运行

运行到光标

设置断点

图 7.13 调试观察窗口工具条


打开寄存器窗口


打开观察窗口


打开变量观察窗口


打开存储器观察窗口


打开反汇编窗口

图 7.14 文件操作工具条


加载调试文件


重新加载文件

5.4 JTAG 仿真器及超级终端

5.4.1 J-LINK 仿真器的连接

J-LINK ARM JTAG 使用 USB 接口与 PC 连接，通过转接板与目标板的连接使用 10 脚的 JTAG 插座。具体连接方法如图 7.15 所示：

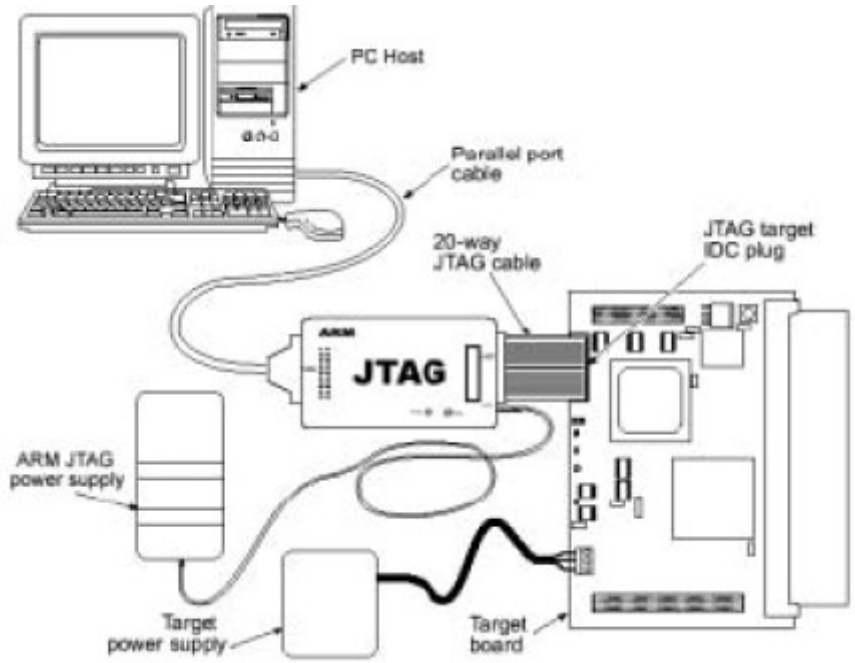


图 7.15 仿真器的连接

在后续的实验中我们还将用到超级终端调试运行实验程序，观察程序运行的结果，因此这里简单介绍一下超级终端的使用。

5.4.2 超级终端的使用

超级终端使用特别广泛，是 Windows xp 自带的内容。以下描述超级终端在 Windows 下的使用。超级终端一般包含在开始→程序→附件→通讯→超级终端文件夹中。如果找不到超级终端，那可能是您在安装 Windows 时将它“删除”。按以下步骤安装超级终端到系统中。（注意：这里要使用到 Windows 启动盘！）

打开控制面板（开始→设置→控制面板）双击添加/删除程序按钮，添加/删除程序的对话框就会打开，如图 7.16 所示。

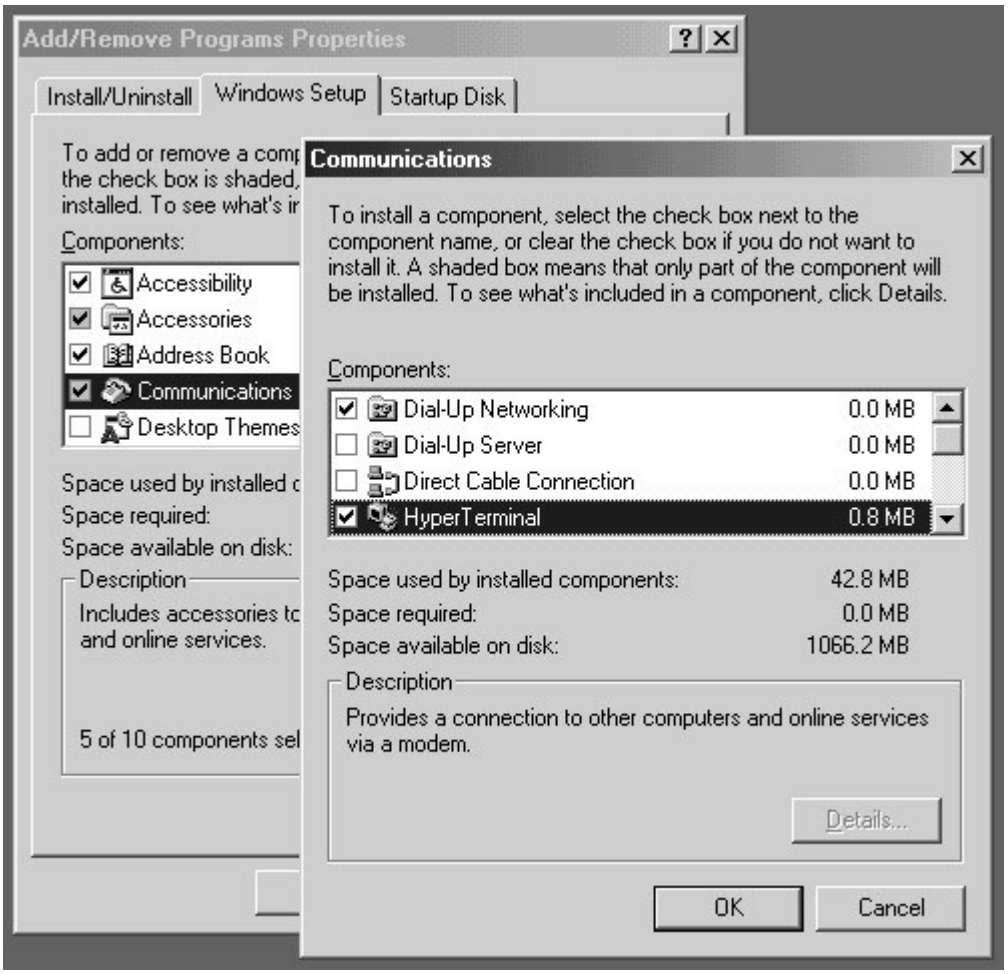


图 7.16　添加超级终端到 Windows 系统中

点击 Windows Setup 面板，浏览安装部件表；选择列表中的通讯并点击具体信息；在通讯窗口中选择超级终端（请划钩）；按 OK 关闭通讯窗口，再按 OK 关闭添加/删除程序。

这时系统可能会要求您插入 Windows CD。然后按屏幕显示的指示继续操作。

启动超级终端进行以下设置：

当连接设置对话框打开后，输入任意名称（例如 EM100），并选择一个图标，按 OK 确认，如图 7.17 所示：

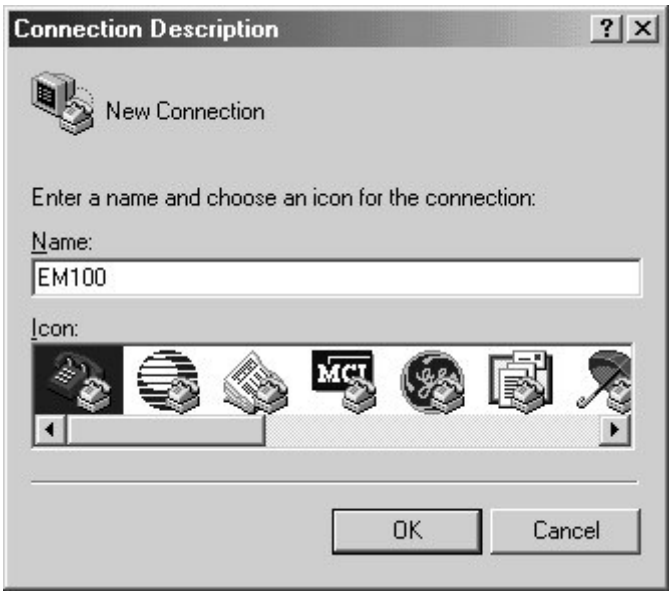


图 7.17　选择连接名称

当连接到对话框打开时，如图 7.18 所示，从 Connect Using 下拉框中选择一个合适的 COM port（适个人串口连接而定，这里假如连接到 PC 的 COM1 Port，那么请选择 COM1）并按 OK 确定。

图 7.18. 选择 COM port

当 COM 属性窗口打开后，设置通讯参数如下 Bits per second：115200；Data bits：8， Parity：None；Stop bits：1；Flow control：None，完成后点 OK 确定。超级终端主窗口就

会打开，如图 7.19 所示：

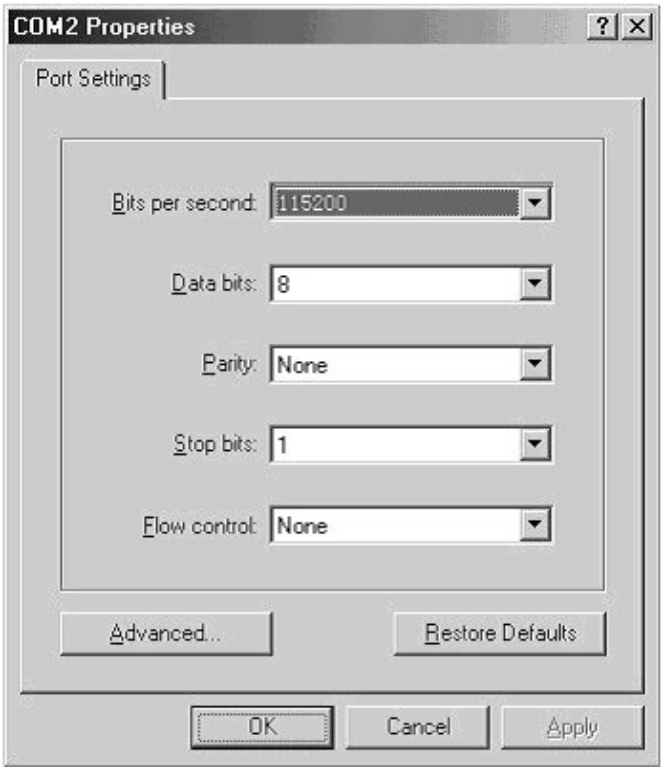


图 7.19　设置 COM port 参数

选择文档→属性，属性对话框打开后，选择设置，按 ASCII Setup 按钮，ASCII Setup 对话框就会打开，如图 7.20 所示。

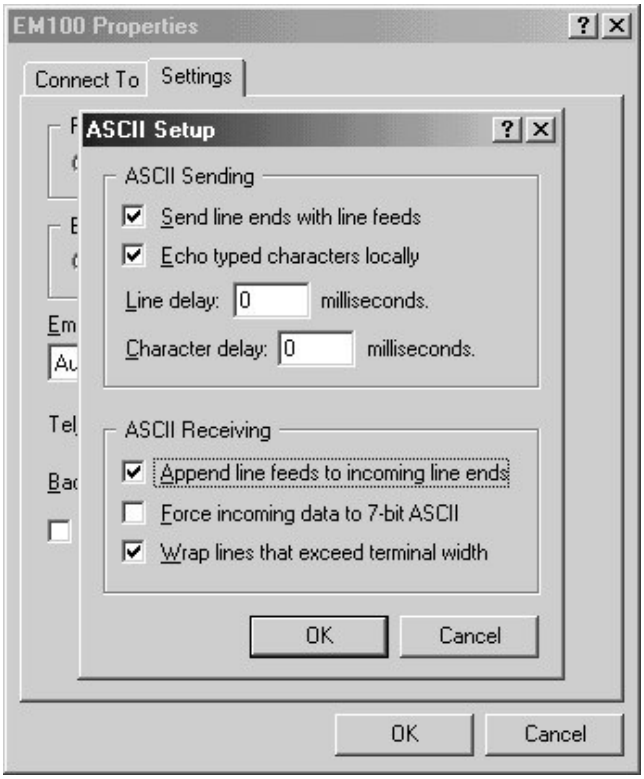


图 7.20　Setting options

对话框内包含以下几个选项，运行不同的实验可以选择不同的显示格式。

ASCII 码发送：

Send line ends with line feeds：以换行符作为发送行末尾 Echo typed characters locally：本地回显键入的字符

ASCII 码接收：

Append line feeds to incoming line ends：将换行符附加到传入行末尾

Force incoming data to 7-bit ASCII：将传入的数据转换为 7 位的 ASCII 码 Wrap lines that exceed terminal width：将超级终端宽度的行自动换行点击 OK 关闭对话框。

此外，如果需要保存该超级终端设置备用，也就不需要再进行设置，从主菜单中选择文件→保存，即可命名保存设置结果。

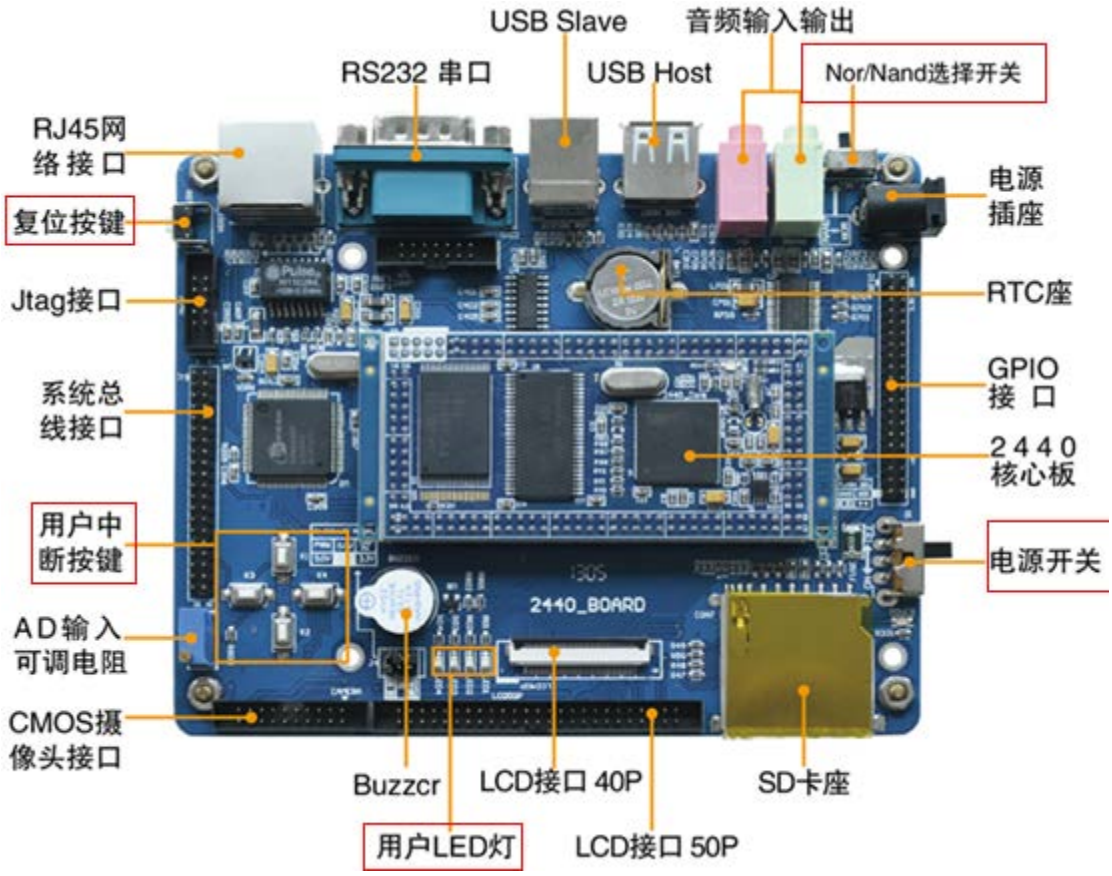
6 实验参考资料

1. 910 实验室机器上开发环境自带。其中 D：\Embedded\TQ2440\下是与 TQ2440 开发板有关的一些资料，包括 TQ2440 的原理图、开发板用户手册、S3C2440 的数据手册、开发板所用外设芯片数据手册、ADS 参考代码、UCOS-II 的参考代码等。
2. Ads 和 axd 相关的资料。可以查阅 ads 和 axd 的帮助文档。可以在 ads 和 axd 的“help”菜单下打开具体的帮助文档索引。

7 实验环境准备

7.1 认识 TQ2440 开发板

需要注意一下的是 TQ2440 的几个控制键和开关的功能。



1. 复位键：可以对整个开发板进行复位；
2. 电源开关：接通或断开开发板电源；
3. 4 个按键：中断按键，按下可以触发一个中断信号；
4. Nor/Nand 选择开关 F_SEL：设置系统的启动方式
   a) S1 拨到外侧（远离绿色接口），复位时由 NOR Flash 启动；
   b) S1 拨到里侧（靠近绿色接口），复位时由 NAND Flash 启动。；
5. 4 个 LED 灯。

本实验课的实验均使用串口 0。从 Nor Flash 启动。

7.2 连接开发板与宿主机

1. 领取 TQ2440 开发板，并检查：串口线、电源、JTAG 仿真器。
2. 将试验箱的 Nor/Nand 选择开关拨到外侧。（设置从 Nor Flash 启动）
3. 连接实验板——串口——PC。
4. 注意：RS232、JTAG 端口不允许带电插拔！

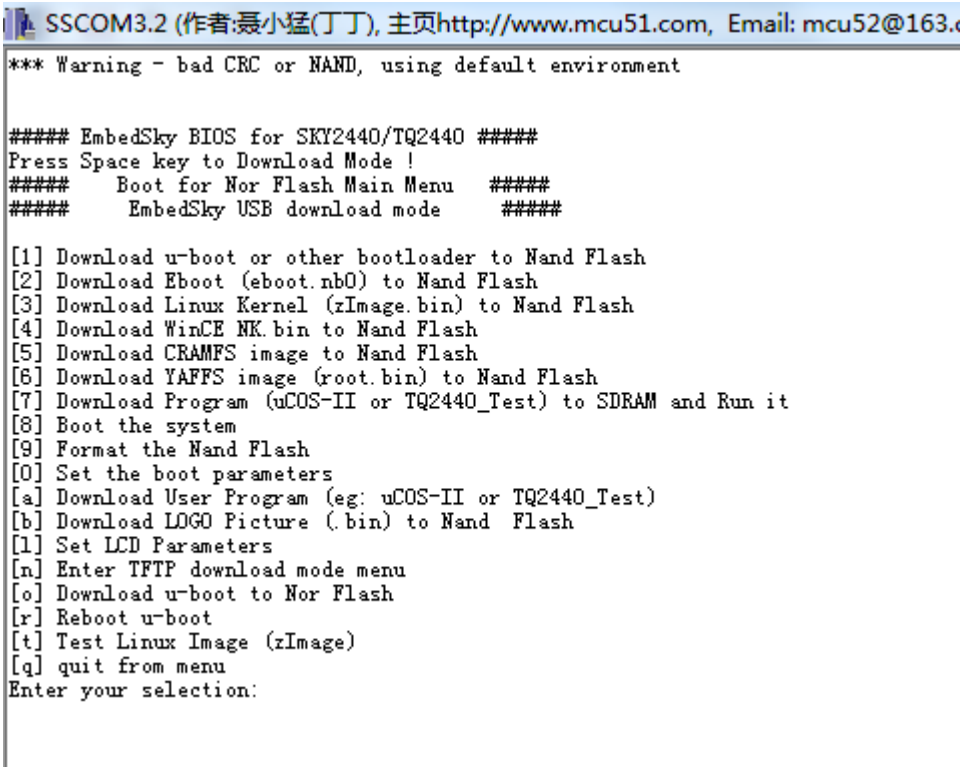
7.3 宿主机软件环境准备

1. 打开 PC 串口终端程序，配置串口参数。
   参数：波特率：115200，数据位：8，停止位：1，校验位：none，流控：none。
   串口终端程序可以使用 windows 所带超级终端或者三方软件 sscom32。

已经设置好的串口程序在“开始/所有程序/embedded”路径下。可以直接打开使用。
作用：实验板通过该串口程序打印信息，所以参数必须与实验板要求的一致。

7.4 启动系统

1. 连接电源线，系统启动。
   可以看到开发板上的 LCD 屏幕显示。
   同时在 sscom32 可以看到 TQ2440 的串口输出信息如下：



7.5 实验注意事项

1. 注意：RS232、JTAG 端口不允许带电插拔！
2. 注意：JTAG 端口排线较为脆弱，不要拽排线！
3. 注意用电安全！
4. 实验设备未经允许，不能带出实验室。
5. 实验结束后，请归还实验设备。同时恢复电脑网线连接，便于后续课程使用机房。

8 实验 1：GPIO 实验—LED 与按键中断

实验目的：

熟悉利用 ADS 开发环境建立工程、编写和编译程序

掌握利用 J-LINK 仿真器调试程序

了解 GPIO 配置和控制方式（NON OS 环境）

了解按键中断服务程序编程

实验环境：

硬件：PC 机、TQ2440 开发板、J-LINK 仿真器、串口线、电源线。

软件：PC 机操作系统 WinXP，ADS1.2 集成开发环境和 axd 调试器，J-LINK 仿真器驱动程序，串口终端（超级终端通讯程序或 sscom）。

实验内容：

1. 通过课程讲解及芯片手册，了解 GPIO（LED）和按键中断的控制机理；
2. 阅读 DEMO CODE 的 GPIO（LED）和按键中断相关代码；
3. 在已有 GPIO 工程文件基础上建立实验 1 工程文件；
4. 实验代码可根据按键的按下（自定）切换 GPIO（LED）的显示模式（模式自定，如一个按键对应一个 LED，按键按下则切换 LED 的亮灭）。
5. 在完成以上基本要求的基础上可自行增加功能（验收通过酌情加分）。例如：
   a) 通过 UART 显示当前工作状态
   b) 增加通过按键控制蜂鸣器的功能

验收及提交：

现场运行验收，指导教师随即提问答辩。

实验报告内容：功能描述，实验原理，框图说明系统结构，文字说明系统各部分定义细节（常量变量定义、数据结构、主要函数等）。源代码。给出系统运行的典型截图，并文字说明。对本实验体会及评述。

9 实验 2：UART 实验

实验目的：

熟悉利用 ADS 开发环境建立工程、编写和编译程序

掌握利用 J-LINK 仿真器调试程序

了解 UART 配置和控制方式（NON OS 环境）

实验环境：

硬件：PC 机、TQ2440 开发板、J-LINK 仿真器、串口线、电源线。

软件：PC 机操作系统 WinXP，ADS1.2 集成开发环境和 axd 调试器，J-LINK 仿真器驱动程序，串口终端（超级终端通讯程序或 sscom）。

实验内容：

1. 通过课程讲解及芯片手册，了解 UART 控制机理；
2. 阅读 DEMO CODE 的 UART 相关代码；
3. 在已有 UART 工程文件基础上建立实验 2 工程文件；
4. 实验代码可通过 UART（工作模式自定）输入命令（自定）控制 GPIO（LED）的 2-4 中显示模式（模式自定，如亮、灭、周期闪烁、循环闪烁、间隔闪烁等），每种命令对应一种 LED 显示模式，**并可随时通过命令切换显示模式**。要求 UART 显示当前工作状态。
5. 在完成以上基本要求的基础上可自行增加功能（验收通过酌情加分）。例如：
   a) 增加命令控制蜂鸣器的功能
   b) UART 功能增加中断控制

验收及提交：

现场运行验收，指导教师随即提问答辩。

实验报告内容：功能描述，实验原理，框图说明系统结构，文字说明系统各部分定义细节（常量变量定义、数据结构、主要函数等）。源代码。给出系统运行的典型截图，并文字说明。对本实验体会及评述。

10 实验 3：ucOSII 多任务、信号量实验

实验目的：

基于实验 1 和 2 对 GPIO 及 UART 控制方式（NON OS 环境）的了解，在 uCOSII 环境下建立多 Task 环境，实践 Task 管理、时间管理及信号量通信。

实验环境：

硬件：PC 机、TQ2440 开发板、J-LINK 仿真器、串口线、电源线。

软件：PC 机操作系统 WinXP，ADS1.2 集成开发环境和 axd 调试器，J-LINK 仿真器驱动程序，串口终端（超级终端通讯程序或 sscom）。

实验内容：

1. 了解 uCOSII 的代码结构及 OS 配置方式，包括资源配置、功能配置及 Tick 配置；
2. 建立多 Task 环境。其中至少包括用于接收 UART 的输入命令（自定）的 Task_CON，

以及分别控制 4 个 GPIO（LED）的 Task_LED1、Task_LED2、Task_LED3、Task_LED4。

3. 创建 4 个信号量，用于 Task_CON 分别触发 Task_LEDx 的激活；
4. Task_CON 接收 UART 命令并解析，实现对 4 个 GPIO（LED）的独立显示模式控制。显示模式包括：led ON、led OFF、led 周期闪烁；
5. Task_CON 可接受系统状态查询命令，在 UART 显示当前各 led 工作状态。
6. 在完成以上基本要求的基础上可自行增加功能（验收通过酌情加分）。例如：
   a) 增加读取按键输入的功能
   b) 增加命令控制蜂鸣器的功能
   c) UART 功能增加中断控制

验收及提交：

现场运行验收，指导教师随即提问答辩。

实验报告内容：功能描述，实验原理，框图说明系统结构，文字说明系统各部分定义细节（常量变量定义、数据结构、主要函数等）。源代码。给出系统运行的典型截图，并文字说明。对本实验体会及评述。

11 实验 4：ucOSII 多任务、信号量、消息队列及 S3c2440 I/O port 综合实验

实验目的：

练习多任务实时操作系统下 Task 调度、Task 间主要通信手段、统一消息驱动、软定时器设计、ARM I/O port 等内容，构建合理的实时系统软件框架、并形成嵌入式实时应用软件的良好编程习惯。

实验环境：

硬件：PC 机、TQ2440 开发板、J-LINK 仿真器、串口线、电源线。

软件：PC 机操作系统 WinXP，ADS1.2 集成开发环境和 axd 调试器，J-LINK 仿真器驱动程序，串口终端（超级终端通讯程序或 sscom）。

实验内容：

1. 创建多 Task，每个 Task 有独立的 ID，为每个 Task 创建私有的 Message Queue，每个 Task 只通过自己的私有 Message Queue 接收消息；Task 间消息通信通过向对方私有 Message Queue 发送消息完成。
2. 设计统一的消息头（Message Head）结构，包含必要的信息，如消息编码、发送者 ID、接受者 ID、消息类型、消息长度、内存指针等。
3. Task_Timer：软定时器管理 Task，可管理 10 个左右软定时器。提供 API 使系统中的 Task 可申请、启动、停止、释放软定时器。定时器超时发生时向对应 Task 发送超时消息。
4. Task_Manag：管理 Task。负责系统启动时同步系统中其他 Task 的启动同步，同时

接收各 Task 的告警消息。

5. Task_Cmd：console 命令行接收 Task。接收并分析 console 发来的命令行及参数。自行设置 5 种以上命令，并根据命令的内容向其它发送激励消息。收到非法命令向 Task_Manag 告警。
6. Task_Led：Led 显示模式控制 Task，可控制 2-4 个 led 的多种显示方式。接收 Led 控制消息（命令）。本 task 应负责 led 相关 I/O port 初始化。收到非法 led 控制命令向 Task_Manag 告警。
7. Task_print：console 输出 Task。接收需打印输出的字串消息（命令），输出到 console。收到长度为 0 或超常字串向 Task_Manag 告警。
8. 在以下基本功能完成基础上，自选扩展一项功能（验收通过酌情加分）。例如：
   a) 增加动态内存管理
   b) 增加事件标记组的使用
   c) 增加对其余外设的控制

验收及提交：

现场运行验收，指导教师随即提问答辩。

实验报告内容：功能描述，实验原理，框图说明系统结构，文字说明系统各部分定义细节（常量变量定义、数据结构、主要函数等）。源代码。给出系统运行的典型截图，并文字说明。对本实验体会及评述。

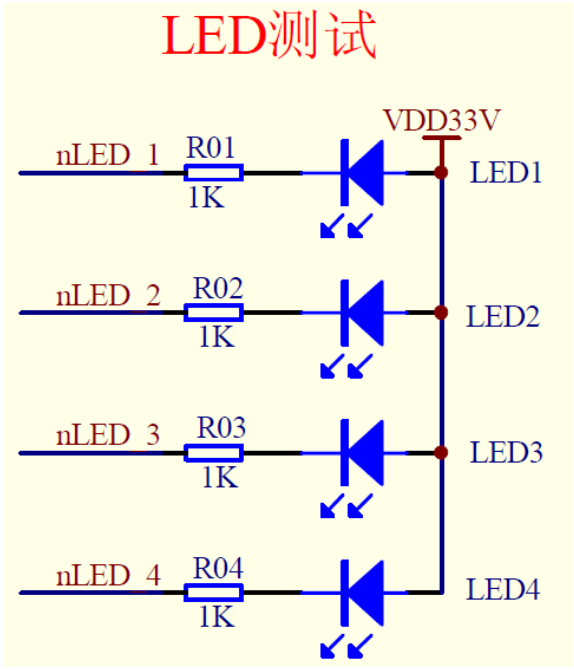
12 实验 1 详细实验步骤

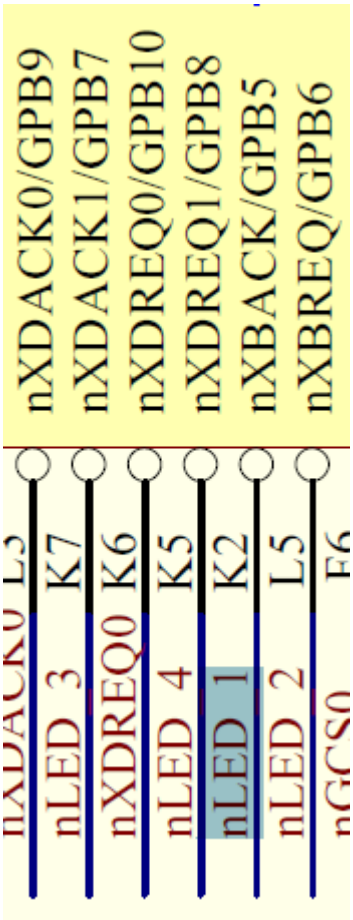
实验 1 的实验内容见第 8 章。本章详述该实验的实验步骤，为之后的 3 个实验做准备。

12.1分析实验电路

找到 TQ2440 的底板原理图（D：\Embedded\TQ2440\ TQ2440 开发板配套电路图\ **TQ2440 底板原理图.pdf**），找到 led 和按键控制部分的原理图。再根据底板图中的 nLED 1 和 EINT0 在核心板原理图（D：\Embedded\TQ2440\ TQ2440 开发板配套电路图**TQ2440_V2 核心板原理图.pdf**）中找到 led 和按键控制部分的原理图。

分别如下：





可见，GPIO 与 LED 的对应关系是：

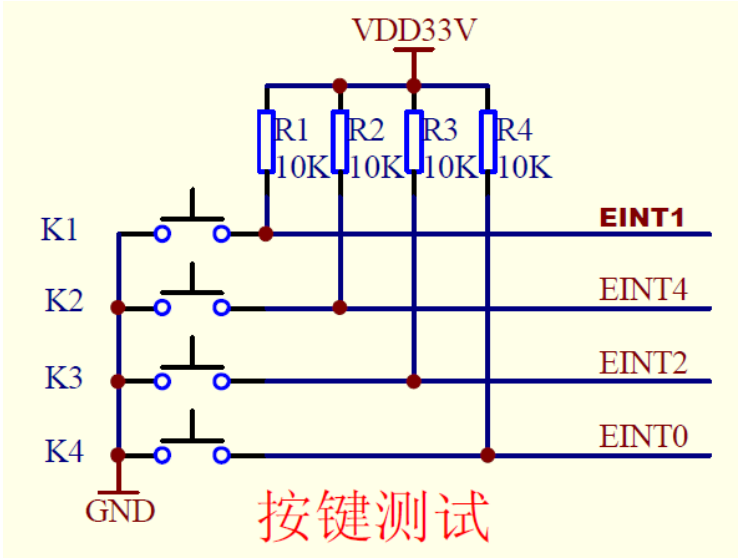
GPB5-nLED1-LED1

GPB6-nLED2-LED2

GPB7-nLED3-LED3

GPB8-nLED4-LED4

即：分别通过控制 GPB5，GPB6，GPB7，GPB8，实现对 LED1，LED2，LED3，LED4 亮灭的控制。



可见，GPIO 与按键的对应关系是：

K1-EINT1-GPF1

K2-EINT4-GPF4

K3-EINT2-GPF2

K4-EINT0-GPF0

即：分别通过读取 GPF1，GPF4，GPF2，GPF0，实现对 K1，K2，K3，K4 开关状态的读取。同时，如果使能了对应的中断（即 EINT1、EINT2、EINT4、EINT0），当按键按下时或改变时（与中断触发模式的设置有关），对应管脚的中断将被触发。

12.2理解实验原理

12.2.1 GPIO 寄存器

Led 通过电源，限流电阻与 ARM 的 I/O 口相连，当 I/O 口为低电平时，电源会通过 led 限流电阻往 ARM 里灌电流，点亮 led。反之 I/O 为高电平时，则没有电流，led 不会亮。注意亮灭之间要有一定的延时（大于人眼分辨率大概十几毫秒），以便人眼能够区分出亮灭。

找到 S3C2440 的数据手册（D：\Embedded\TQ2440\TQ2440 开发板配套芯片手册**S3C2440.pdf**），定位到到第 9 章 IO ports，仔细理解对 GPIO 的控制和相应的寄存器含义。

S3C2440A 芯片上有130个多功能I/O引脚。分别是：

– 端口 A（GPA）：25 位输出端口

– 端口 B（GPB）：11 位输入/输出端口

– 端口 C（GPC）：16 位输入/输出端口

– 端口 D（GPD）：16 位输入/输出端口

– 端口 E（GPE）：16 位输入/输出端口

– 端口 F（GPF）：8 位输入/输出端口

– 端口 G（GPG）：16 位输入/输出端口

– 端口 H（GPH）：9 位输入/输出端口

– 端口 J（GPJ）：13 位输入/输出端口

每个端口都可以通过软件配置寄存器来满足不同系统和设计的需要。在运行主程序之前，必须先对每一个用到的引脚的功能进行设置。如果某些引脚的复用功能没有使用，那么可以先将该引脚设置为 I/O 口。

端口控制描述

端口配置寄存器（GPACON 至 GPJCON）

S3C2440A 中，大多数端口为复用引脚。因此要决定每个引脚选择哪项功能。

PnCON（引脚控制寄存器）决定了每个引脚使用哪项功能。

如果在掉电模式中 PE0 至 PE7 用于唤醒信号，这些端口必须配置为输入模式。

端口数据寄存器（GPADAT 至 GPJDAT）

如果端口配置为输出端口，可以写入数据到 PnDAT 的相应位。如果端口配置为输入端口，可以从 PnDAT 的相应位读取数据。

端口上拉寄存器（GPBUP 至 GPJUP）

端口上拉寄存器控制每个端口组的使能/禁止上拉电阻。当相应位为 0 时使能引脚的上拉电阻。当为 1 时禁止上拉电阻。

如果使能了上拉电阻，那么上拉电阻与引脚的功能设置无关（输入、输出、DATAn、EINTn 等等）

杂项控制寄存器

此寄存器控制睡眠模式，USB 引脚和 CLKOUT 选择的数据端口上拉电阻。

外部中断控制寄存器

24 个外部中断由各种信号方式触发。EXTINT 寄存器为外部中断请求配置信号触发方式为低电平触发、高电平触发、下降沿触发、上升沿触发或双边沿触发。

由于每个外部中断引脚包含一个数字滤波器，中断控制可以确认请求信号是否长于 3 个时钟。

EINT[15：0]可用做唤醒源。

LED 相应寄存器：

端口 B 控制寄存器（GPBCON，GPBDAT，GPBUP）

寄存器	地址	R/W	描述	复位值
GPBCON	0x56000010	R/W	配置端口 B 的引脚	0x0
GPBDAT	0x56000014	R/W	端口 B 的数据寄存器	–
GPBUP	0x56000018	R/W	端口 B 的上拉使能寄存器	0x0
保留	0x5600001C	–	保留	–

GPBCON	位	描述				初始状态
GPB10	[21:20]	00 = 输入	01 = 输出	10 = nXDREQ0	11 = 保留	0
GPB9	[19:18]	00 = 输入	01 = 输出	10 = nXDACK0	11 = 保留	0
GPB8	[17:16]	00 = 输入	01 = 输出	10 = nXDREQ1	11 = 保留	0
GPB7	[15:14]	00 = 输入	01 = 输出	10 = nXDACK1	11 = 保留	0
GPB6	[13:12]	00 = 输入	01 = 输出	10 = nXBREQ	11 = 保留	0
GPB5	[11:10]	00 = 输入	01 = 输出	10 = nXBACK	11 = 保留	0
GPB4	[9:8]	00 = 输入	01 = 输出	10 = TCLK [0]	11 = 保留	0
GPB3	[7:6]	00 = 输入	01 = 输出	10 = TOUT3	11 = 保留	0
GPB2	[5:4]	00 = 输入	01 = 输出	10 = TOUT2	11 = 保留	0
GPB1	[3:2]	00 = 输入	01 = 输出	10 = TOUT1	11 = 保留	0
GPB0	[1:0]	00 = 输入	01 = 输出	10 = TOUT0	11 = 保留	0

GPBDAT	位	描述	初始状态
GPB[10:0]	[10:0]	当端口配置为输入端口时，相应位为引脚状态。当端口配置为输出端口时，引脚状态将与相应位相同。当端口配置为功能引脚，将读取到未定义值。	–

GPBUP	位	描述	初始状态
GPB[10:0]	[10:0]	0：使能附加上拉功能到相应端口引脚 1：禁止附加上拉功能到相应端口引脚	0x0

在 GPFCON 寄存器中设置相应管脚为输入。管脚经按键连接到 GND 上，当按键按下时，按键对应的管脚被拉低，GPFDAT 相应位被置 0。检测 GPFDAT 即可知道哪个按键被按下。

按键相应寄存器：

端口 F 控制寄存器（GPFCON，GPFDAT，GPFUP）

如果 GPF0 至 GPF7 在掉电模式中用于唤醒信号，端口将被设置为中断模式。

寄存器	地址	R/W	描述	复位值
GPFCON	0x56000050	R/W	配置端口 F 的引脚	0x0
GPFDAT	0x56000054	R/W	端口 F 的数据寄存器	–
GPFUP	0x56000058	R/W	端口 F 的上拉使能寄存器	0x00
保留	0x5600005C	–	保留	–

GPFCON	位	描述				初始状态
GPF7	[15:14]	00 = 输入	01 = 输出	10 = EINT[7]	11 = 保留	0
GPF6	[13:12]	00 = 输入	01 = 输出	10 = EINT[6]	11 = 保留	0
GPF5	[11:10]	00 = 输入	01 = 输出	10 = EINT[5]	11 = 保留	0
GPF4	[9:8]	00 = 输入	01 = 输出	10 = EINT[4]	11 = 保留	0
GPF3	[7:6]	00 = 输入	01 = 输出	10 = EINT[3]	11 = 保留	0
GPF2	[5:4]	00 = 输入	01 = 输出	10 = EINT[2]	11 = 保留	0
GPF1	[3:2]	00 = 输入	01 = 输出	10 = EINT[1]	11 = 保留	0
GPF0	[1:0]	00 = 输入	01 = 输出	10 = EINT[0]	11 = 保留	0

GPFDAT	位	描述	初始状态
GPF[7:0]	[7:0]	当端口配置为输入端口时，相应位为引脚状态。当端口配置为输出端口时，引脚状态将与相应位相同。当端口配置为功能引脚，将读取到未定义值。	–

GPFUP	位	描述	初始状态
GPF[7:0]	[7:0]	0：使能附加上拉功能到相应端口引脚 1：禁止附加上拉功能到相应端口引脚	0x00

12.2.2 中断寄存器

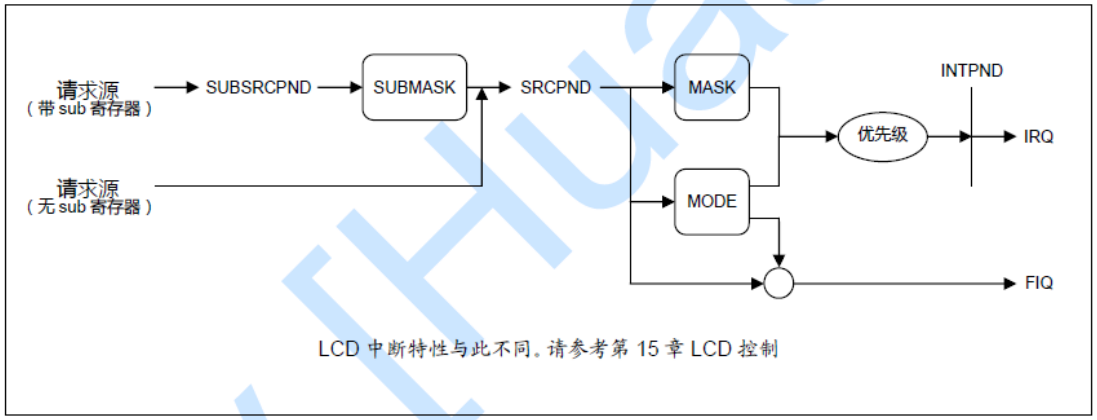
4 个按键使用中断方式触发。

找到 S3C2440 的数据手册（D：\Embedded\TQ2440\TQ2440 开发板配套芯片手册\S3C2440.pdf），定位到到第 14 章 INTERRUPT CONTROLLER，仔细理解对中断的控制和相应的寄存器含义。

S3C2440A 中的中断控制器接受来自 60 个中断源的请求。提供这些中断源的是内部外设，如 DMA 控制器、UART、IIC 等等。在这些中断源中，UARTn、AC97 和 EINTn 中断对于中断控制器而言是“或”关系。

当从内部外设和外部中断请求引脚收到多个中断请求时，中断控制器在仲裁步骤后请求 ARM920T 内核的 FIQ 或 IRQ。

仲裁步骤由硬件优先级逻辑决定并且写入结果到帮助用户通告是各种中断源中的哪个中断发生了的中断挂起寄存器中。



中断控制器操作

程序状态寄存器（PSR）的 F 位和 I 位

如果 ARM920T CPU 中的 PSR 的 F 位被置位为 1，CPU 不会接受来自中断控制器的快中断请求（FIQ）。同样的如果 PSR 的 I 位被置位为 1，CPU 不会接受来自中断控制器的中断请求（IRQ）。因此，中断控制器可以通过清除 PSR 的 F 位和 I 位为 0 并且设置 INTMSK 的相应位为 0 来接收中断。

中断模式

ARM920T 有两种中断模式的类型：FIQ 或 IRQ。所有中断源在中断请求时决定使用哪种类型。

中断挂起寄存器

S3C2440A 有两个中断挂起寄存器：源挂起寄存器（SRCPND）和中断挂起寄存器（INTPND）。这些挂起寄存器表明一个中断请求是否为挂起。当中断源请求中断服务，SRCPND 寄存器的相应位被置位为 1，并且同时在仲裁步骤后 INTPND 寄存器仅有 1 位自动置位为 1。如果屏蔽了中断，则 SRCPND 寄存器的相应位被置位为 1。这并不会引起 INTPND 寄存器的位的改变。当 INTPND 寄存器的挂起位为置位，每当 I 标志或 F 标志被清除为 0 中断服务程序将开始。SRCPND 和 INTPND 寄存器可以被读取和写入，因此服务程序必须首先通过写 1 到 SRCPND 寄存器的相应位来清除挂起状态并且通过相同方法来清除 INTPND 寄存器中挂起状态。

中断屏蔽寄存器

此寄存器表明如果中断相应的屏蔽位被置位为 1 则禁止该中断。如果某个 INTMSK 的中断屏蔽位为 0，将正常服务中断。如果 INTMSK 的中断屏蔽位为 1 并且产生了中断，将置位源挂起位。

具体寄存器定义略，参见手册。

12.3连接实验板

1. 将开发板的 F_SEL 开关（Nor/Nand 选择开关）拨到外侧（远离绿色接口）。（设置从 NOR Flash 启动）

2. 连线：开发板——串口——PC 机。注意，串口使用串口 0。
3. 连线：开发板——J-link 仿真器——PC 机。注意：仿真器与板子的接口排线比较脆弱，不要拉着排线插拔仿真器，而要抓住插头。
4. 注意：RS232、JTAG 端口不允许带电插拔！
5. 连线：开发板——电源线——电源插座。
6. 如果板子没有上电，将板子电源开关拨到外侧，打开电源。

12.4打开 PC 端软件

打开 PC 串口终端程序，配置串口参数。

参数：波特率：115200，数据位：8，停止位：1，校验位：none，流控：none。

串口终端程序可以使用 windows 所带超级终端或者三方软件 sscom32。

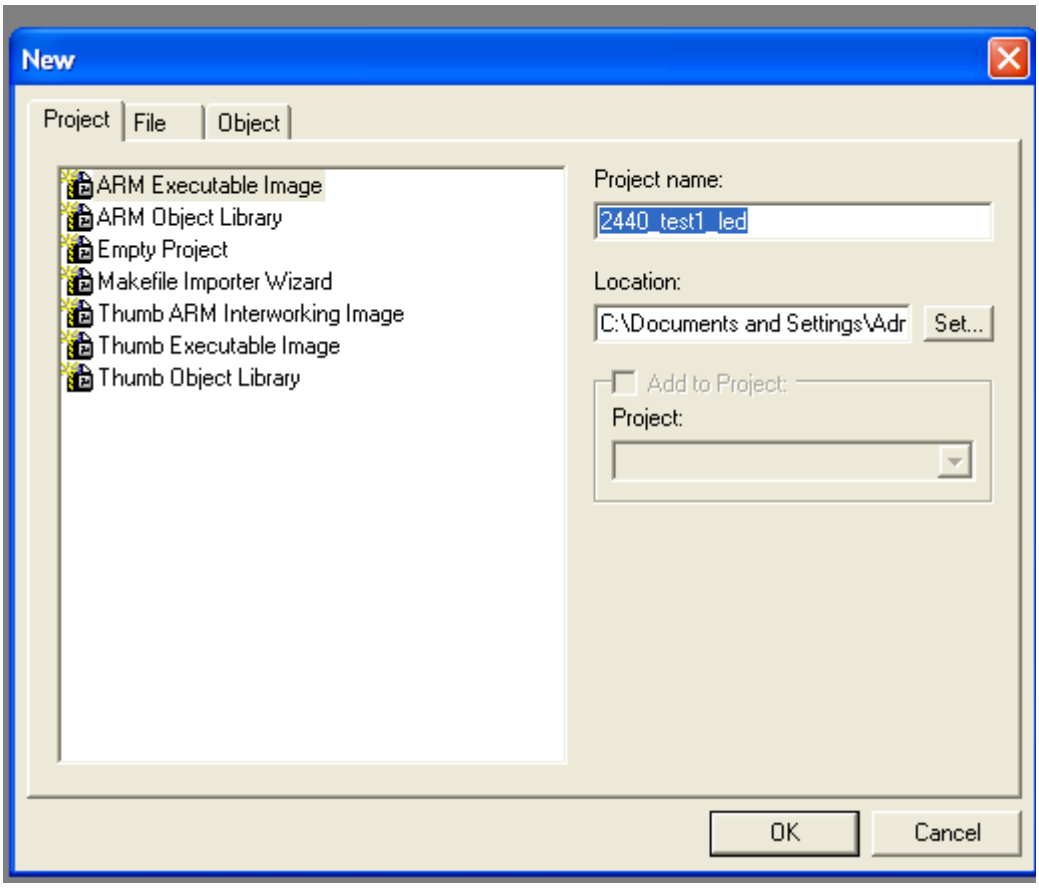
已经设置好的串口程序在“开始/所有程序/embedded”路径下。可以直接打开使用。

作用：实验板通过该串口程序打印信息，所以参数必须与实验板要求的一致。

12.5建立 ADS 工程

首先打开 CodeWarrior，点击 File 菜单下的 New 来创建新工程。Project 对话框中选择 ARM Executable Image 。在 Project name 中输入工程名，本例为：2440_test1_led，点击“Location：”文本框的“Set...”按钮，选择要将工程保存的路径，然后点击确定即可建立一个新的工程。

如下图所示：



在该提示框中我们见到有七种工程类型：

ARM Executable Image：用于由 ARM 指令代码生成一个 ELF 格式的可执行映像文件。

ARM Object Library：用于有 ARM 指令代码生成一个 armar 格式的目标文件库。

Empty Project：创建一个不包含任何库和源文件的空工程。

Makefile Importer Wizard：用于将 Visual C 的 nmake 或者 GNU make 文件转入到 CodeWarrior IDE 工程文件。

Thumb ARM Executale Image：用于由 ARM 指令和 Thumb 指令的混合代码生成一个可执行的 ELF 格式的映像文件。

Thumb Executable Image：用于由 Thumb 指令代码生成一个 ELF 格式的可执行映像文件。

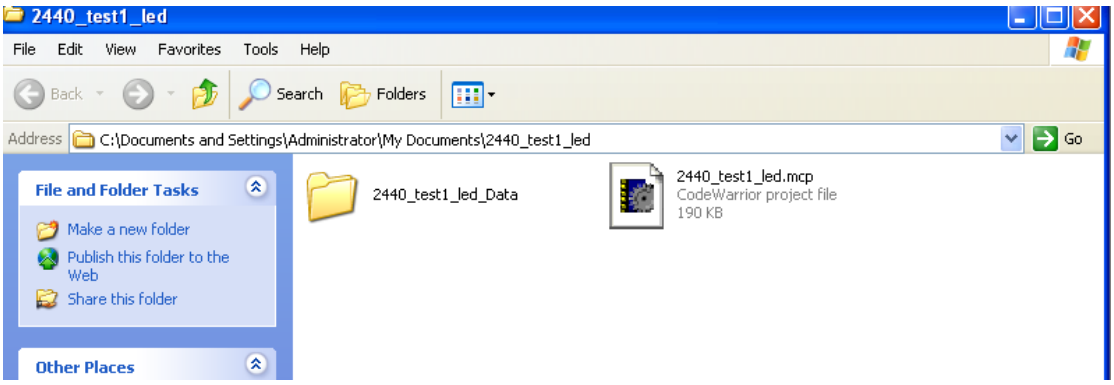
Thumb Object Library：用于由 Thumb 指令代码生成一个 armar 格式的目标文件库。

在这里我们选择第一种工程 ARM Executable Image，在 Project name 中输入工程文件名，我们取名为 2440_test1_led，在 location 处点击 set……选择你的工程文件将保存于何处本例保存于

C：\Documents and Settings\Administrator\My Documents\2440_test1_led，最

后确定就可以了，这时候会出现一个 2440_test1_led.mcp 窗口。

同时 C：\Documents and Settings\Administrator\My Documents\2440_test1_led 该目录下会有两个文件



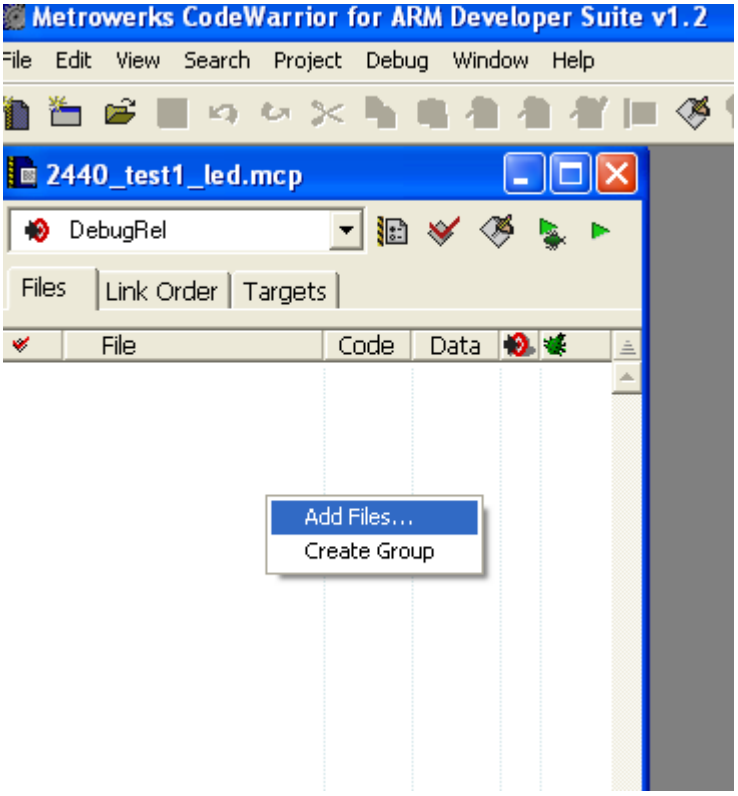
下面开始创建源文件。

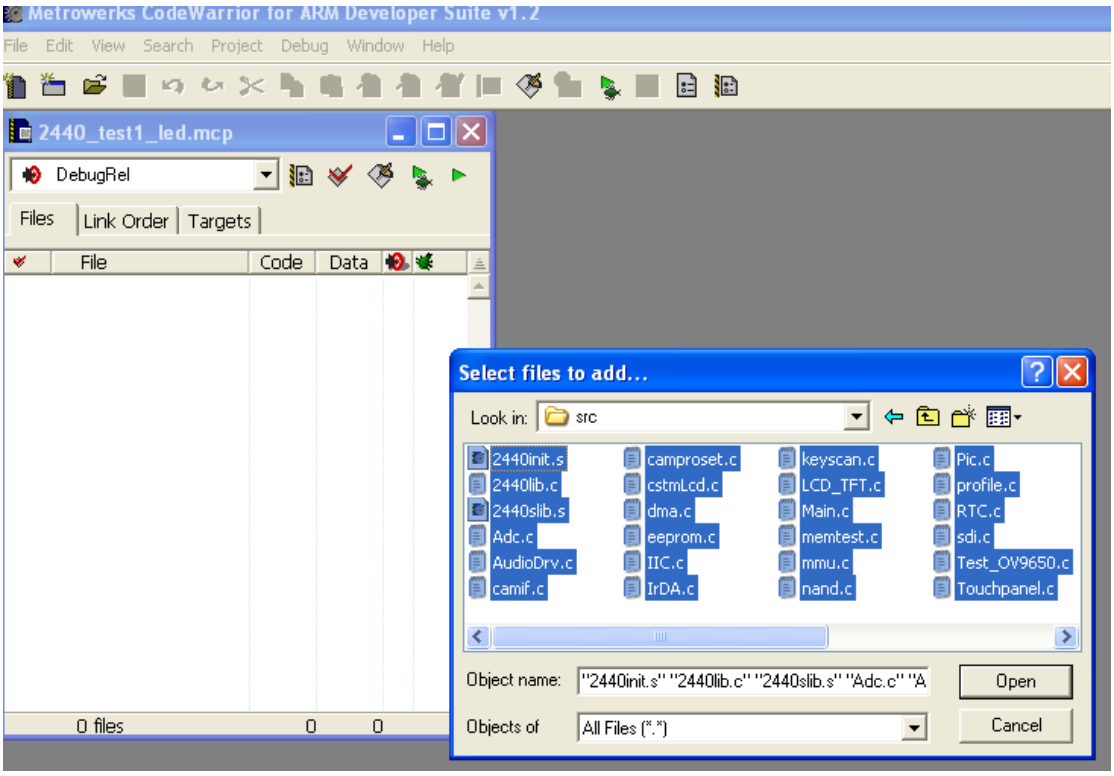
12.6添加或编写代码

可以点击 File 菜单下的 New，选择标签页 File 在 File name 中输入要建立的文件名，如此时选上了 Add to Project,以后就不要添加了。点击确定关闭窗口。

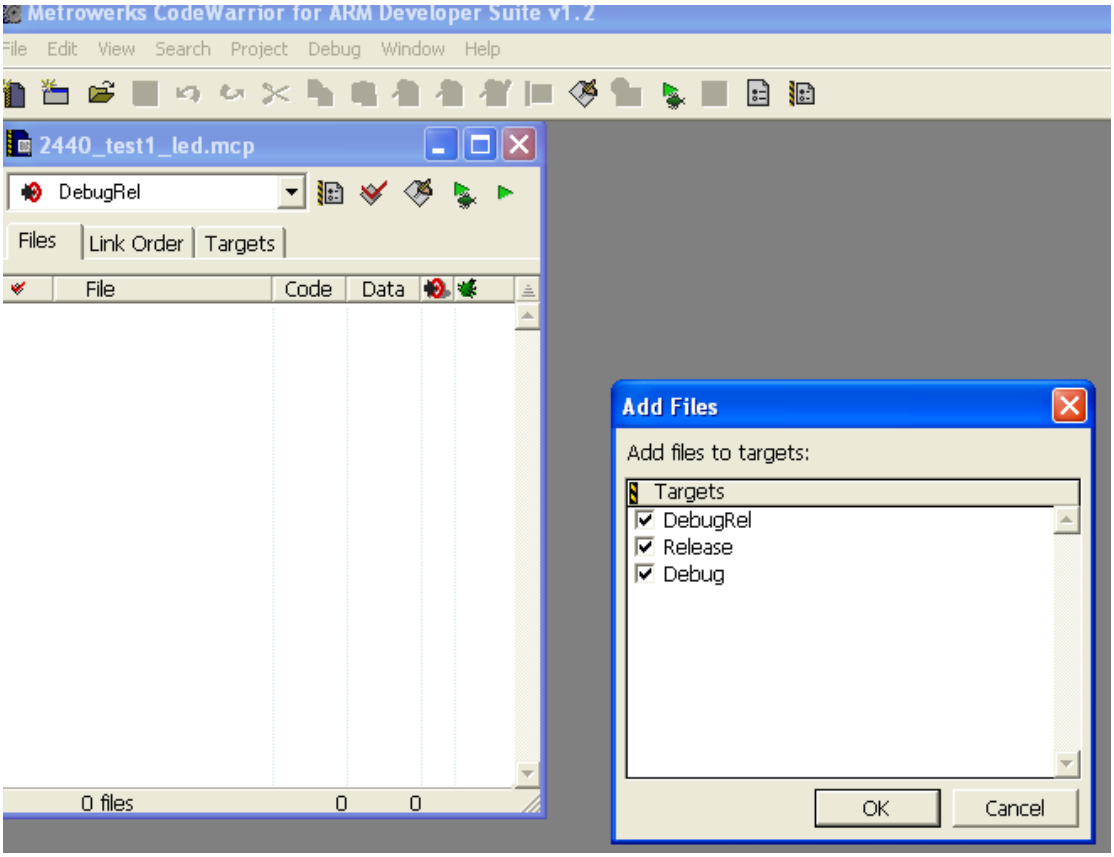
也可以添加现成的代码到工程中，本实验使用此方式。

点击 Project 菜单下的 Add Fils 选择要添加的源文件。如下图：





打开之后会弹出询问添加哪类目标，如图：



此时有三个选项：

DebugRel：使用该目标选项，在生成目标的时候，会为每一个源文件生成调

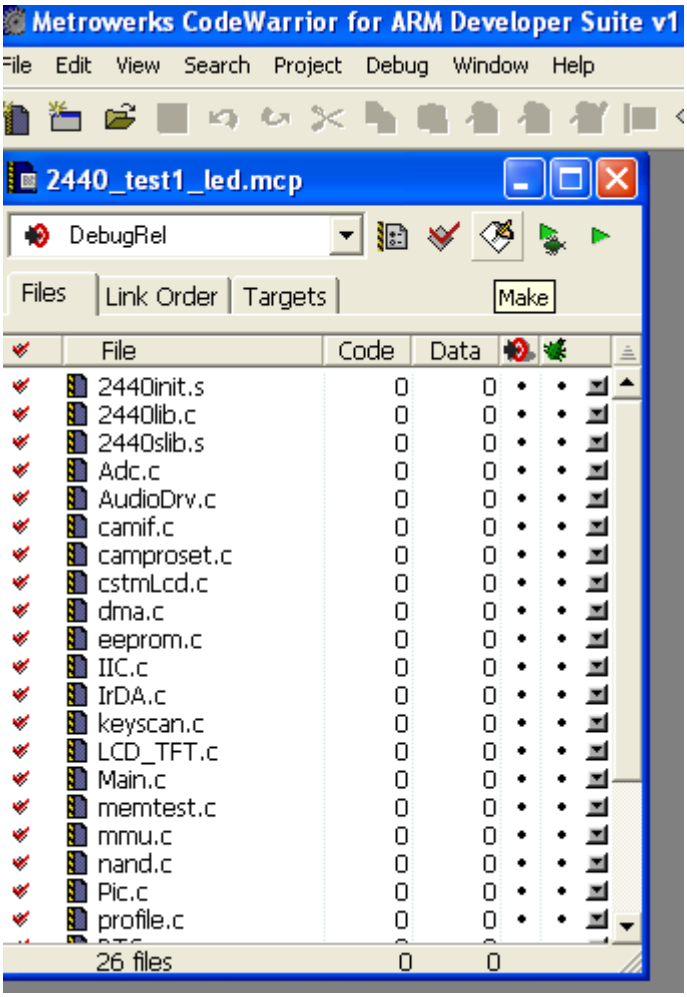
试信息。

Release： 使用该目标选项，在生成目标的时候，不生成任何调试信息。

Debug： 使用该目标选项，在生成目标的时候，会为每一个源文件生成完美的调试信息。

我们全部选择。

也可在工程窗体上的 File 标签页中单击右键--->Add Fils。

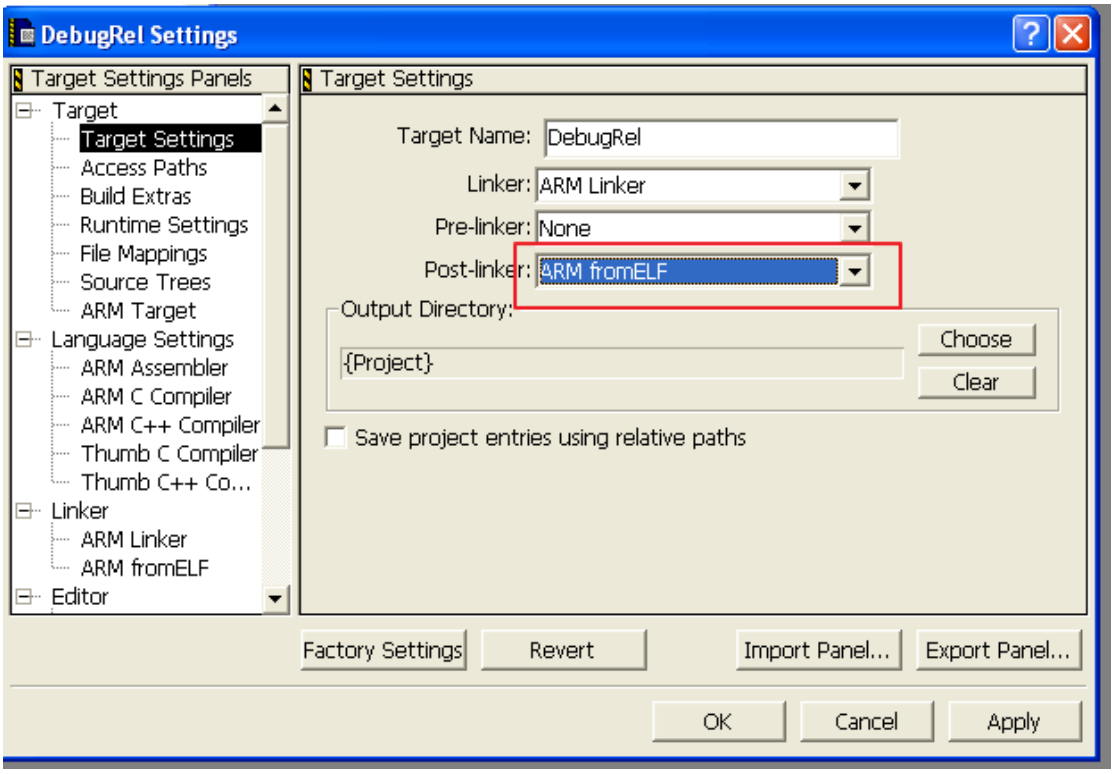


接下来要进行编译和链接的配置。

12.7配置工程

在进行编译和链接前，首先讲述一下如何进行生成目标的配置。

点击 Edit 菜单，选择“DebugRel Settings…”(注意，这个选项会因用户选择的不同目标而有所不同)，出现如下图所示的对话框。



这个对话框中的设置很多，在这里介绍一些最为常用的设置选项，若对其他未涉及到的选项感兴趣，可以查看相应的帮助文件。

1. target 设置选项

Target Name 文本框显示了当前的目标设置。

Linker 选项供用户选择要使用的链接器。在这里默认选择的是 ARM Linker，使用该链接器，将使用 ARM Linker 链接编译器和汇编器生成的工程中的文件相应的目标文件。

这个设置中还有两个可选项：

None 不是不用任何链接器，如果使用它，则工程中的所有文件都不会被编译器或汇编器处理。ARM Librarian 表示将编译或汇编得到的目标文件转换为 ARM 库文件。对于本例，使用默认的链接器 ARM Linker。

Pre-linker：目前 CodeWarrior IDE 不支持该选项。

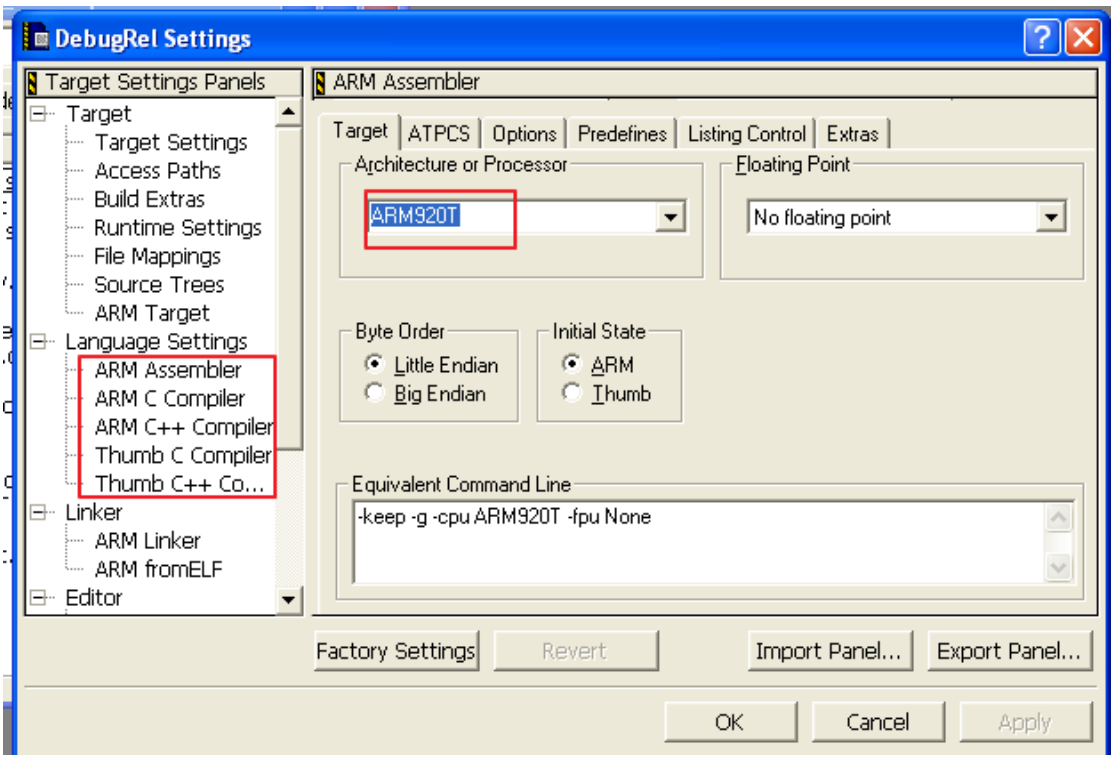
Post-Linker：选择在链接完成后，还要对输出文件进行的操作。因为在本例中，希望生成一个可以烧写到 Flash 中去的二进制代码，所以在这里选择 ARM fromELF，表示在链接生成映像文件后，再调用 FromELF 命令将含有调试信息的 ELF 格式的映像文件转换成其他格式的文件。

2. Language Settings

因为本例中包含有汇编源代码，所以要用到汇编器。首先看 ARM 汇编器，默认的 ARM 体系结构是 ARM7TDMI，将其改为 ARM920T。字节顺序默认就是小端模式。其他设置，就用默认值即可。

本工程中还包含 C 语言代码，因此还有必要设置 ARM C Compiler 选项，

点击该选项会在右侧出现相应的设置选项，将默认的 ARM 体系结构 ARM7TDMI 改为 ARM920T，字节顺序仍然是小端模式，其它设置采取默认值即可，如图：



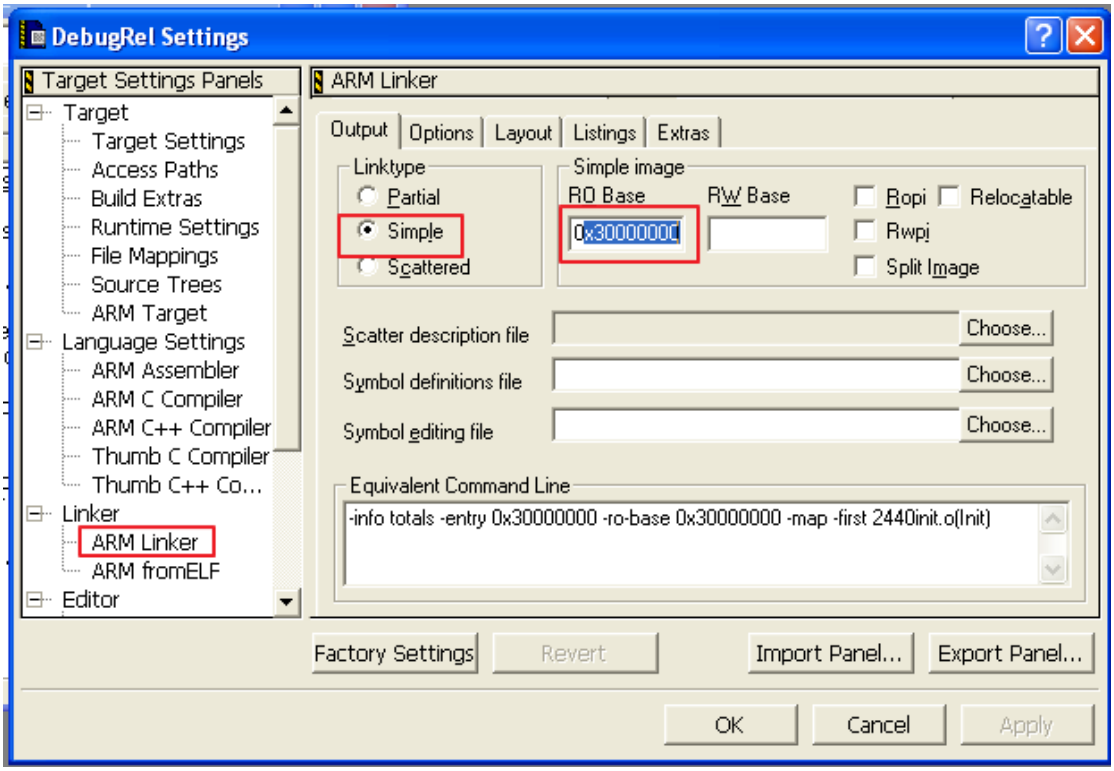
注意在设置框的下边有一个命令行，当对上边某个设置进行修改时，该行中的某个参数就会跟着发生相应的改变，实际上，这行文字显示的就是相应的编译或者连接选项，由于有了 CodeWarrior，开发人员可以不再去记数量繁多的命令行选项，只要在界面中选择或撤销某个选项，软件就会自动生成相应的代码。不过该命令框也为习惯 DOS 下键入命令行的用户提供了极大方便。

3.Linker 设置

鼠标选中 ARM Linker，出现如下图所示对话框。这里详细介绍该对话框的主要的标签页选项，因为这些选项对最终生成的文件有着直接的影响。

在标签页 Output 中，Linktype 中提供了三种链接方式。Partial 方式表示链接器只进行部分链接，经过部分链接生成的目标文件，可以作为以后进一步链接时的输入文件。Simple 方式是默认的链接方式，也是最为频繁使用的链接方式，它链接生成简单的 ELF 格式的目标文件，使用的是链接器选项中指定的地址映射方式。Scattered 方式使得链接器要根据 scatter 格式文件中指定的地址映射，生成复杂的 ELF 格式的映像文件。这个选项一般情况下，使用不太多。

因为我们所举的例子比较简单，选择 Simple 方式就可以了。在选中 Simple 方式后，就会出现 Simple image。



RO Base：这个文本框设置包含有 RO 段的加载域和运行域为同一个地址。默认是 0x8000。

这里用户要根据自己硬件的实际 SDRAM 的地址空间来修改这个地址，保证在这里填写的地址，是程序运行时，SDRAM 地址空间所能覆盖的地址。对于 OK 开发板而言程序小的话可以用 CPU 内部 4K（0x0—0x1000）,大的话可以用外部 64M（0x30000000—0x34000000）,如果想把编译出的二进制文件烧写进开发板，此处地址应该填写为 0x30000000。

RW Base：这个文本框设置了包含 RW 和 ZI 输出段的运行域地址。如果选中 split 选项，链接器生成的映像文件将包含两个加载域和两个运行域，此时，在 RW Base 中所输入的地址为包含 RW 和 ZI 输出段的域设置了加载域和运行域地址。

Ropi：选中这个设置将告诉链接器使包含有 RO 输出段的运行域位置无关。使用这个选项，链接器将保证下面的操作：

检查各段之间的重定址是否有效；

确保任何由 armlink 自身生成的代码是只读位置无关的。

Rwpi：选中该选项将会告诉链接器使包含 RW 和 ZI 输出段的运行域位置无关。如果这个选项没有被选中，域就标识为绝对。每一个可写的输入段必须是读写位置无关的。如果这个选项被选中，链接器将进行下面的操作：

检查可读/可写属性的运行域的输入段是否设置了位置无关属性；

检查在各段之间的重地址是否有效；

在 Region$$Table 和 ZISection$$Table 中添加基于静态存储器 sb 的选项。

该选项要求 RW Base 有值，如果没有给它指定数值的话，默认为 0 值。

Split Image：选择这个选项把包含 RO 和 RW 的输出段的加载域分成 2 个加载域：一个是包

含 RO 输出段的域，一个是包含 RW 输出段的域。

这个选项要求 RW Base 有值，如果没有给 RW Base 选项设置，则默认是-RW Base 0。

Relocatable：选择这个选项保留了映像文件的重定址偏移量。这些偏移量为程序加载器提供了有用信息。

在 Options 选项中，需要读者引起注意的是 Image entry point 文本框。它指定映像文件的初始入口点地址值，当映像文件被加载程序加载时，加载程序会跳转到该地址处执行。如果需要，用户可以在这个文本框中输入下面格式的入口点：

入口点地址：这是一个数值，仿真时该地址不写问题不大，如果要烧进 flash，该地址需设置为 0x30000000。(见下图红框部分)

符号：该选项指定映像文件的入口点为该符号所代表的地址处，比如：

-entry int_handler

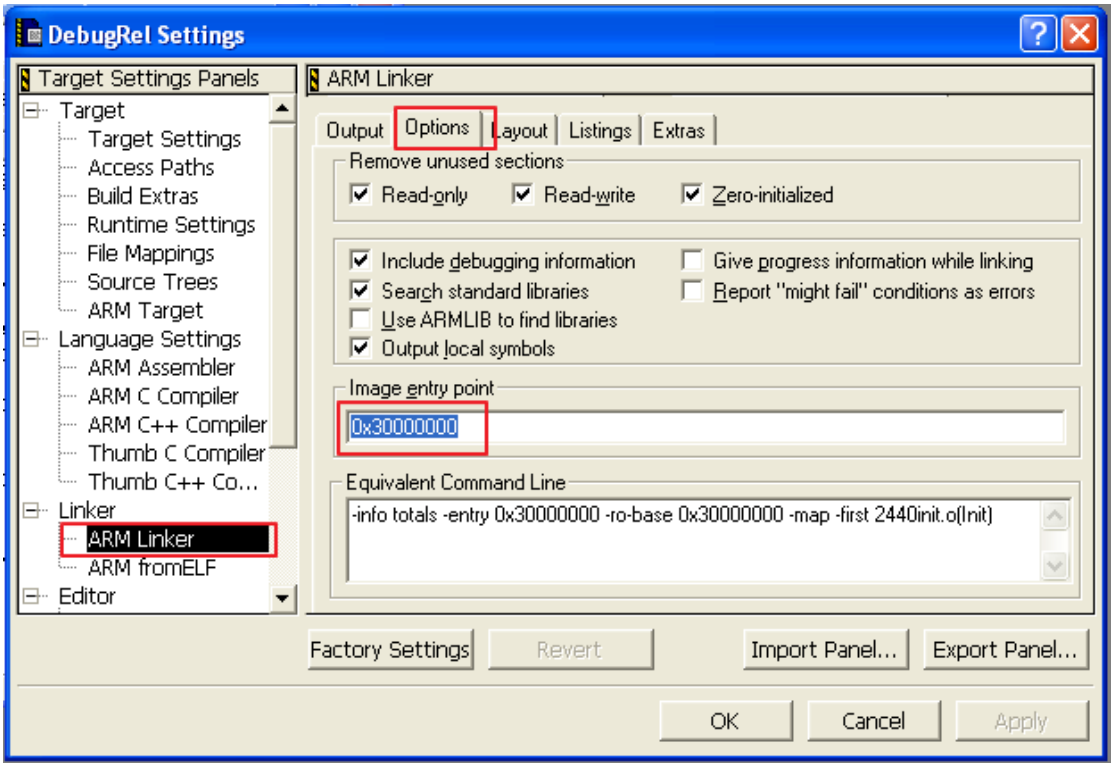
如果该符号有多处定义存在，armlink 将产生出错信息。

offset+object(section)：该选项指定在某个目标文件的段的内部的某个偏移量处为映像

文件的入口地址，例如：

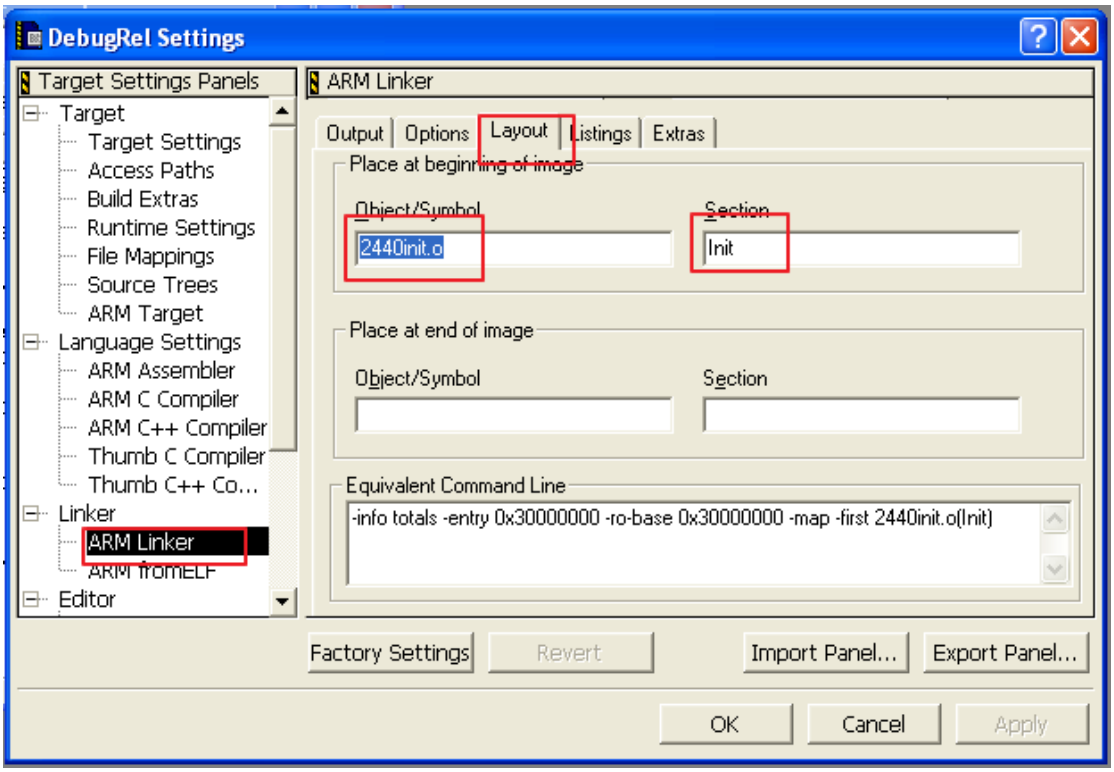
-entry 8+startup(startupseg)

在此处指定的入口点用于设置 ELF 映像文件的入口地址。



需要引起注意的是，这里不可以用符号 main 作为入口点地址符号，否则会出现类似“Image dose not have an entry point(Not specified or not set due to multiple choice)”的错误信息。

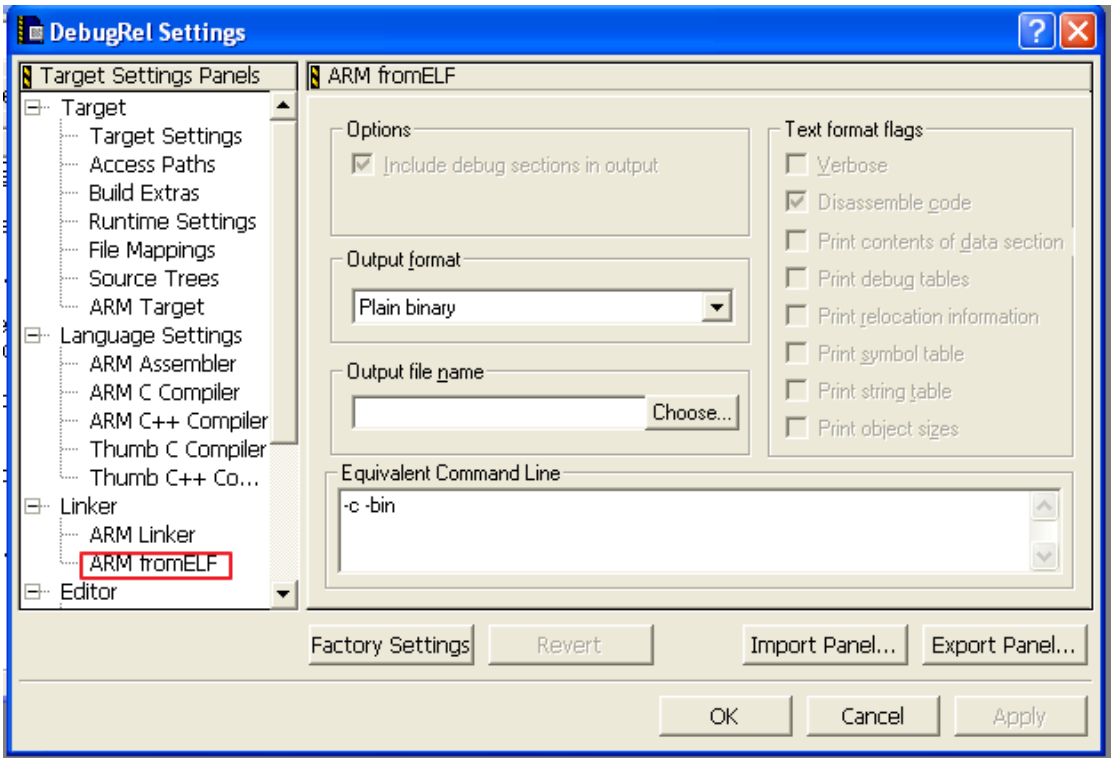
在 Layout 选项中，需要读者引起注意的是 Object/symbol 和 section 文本框。



关于 ARM Linker 的设置还有很多，对于想进一步深入了解的，可以查看帮

助文件，都有很详细的介绍。

在 Linker 下还有一个 ARM fromELF，如下图所示：



fromELF 是一个实用工具，它实现将链接器，编译器或汇编器的输出代码进行格式转换的功能。例如，将 ELF 格式的可执行映像文件转换成可以烧写到 ROM 的二进制格式文件；对输出文件进行反汇编，从而提取出有关目标文件的大小，符号和字符串表以及重定址等信息。

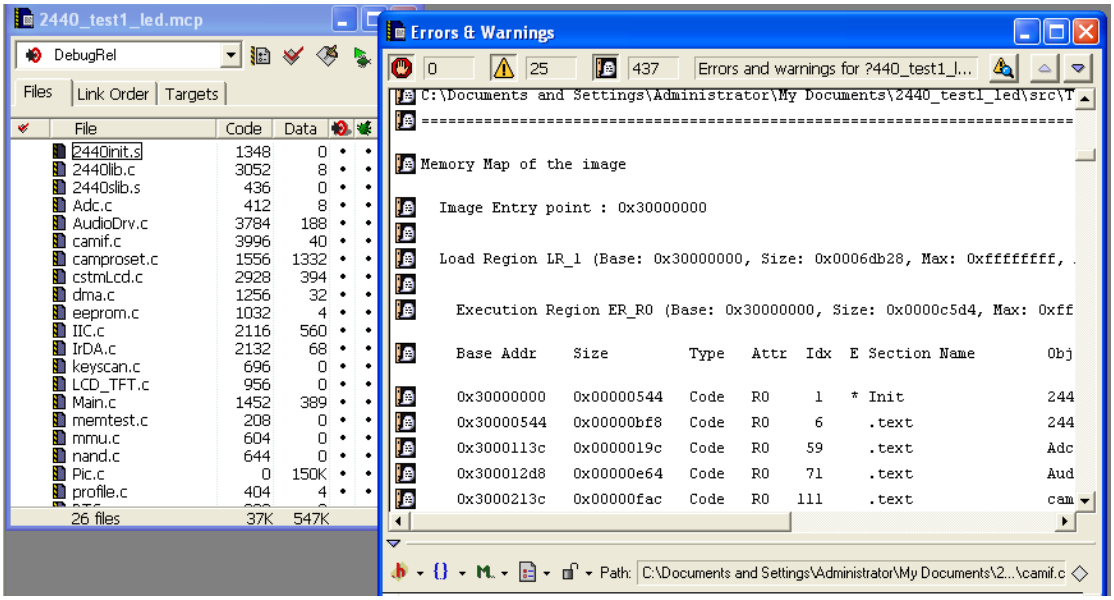
只有在 Target 设置中选择了 Post-linker，才可以使用该选项。

在 Output format 下拉框中，为用户提供了多种可以转换的目标格式，本例选择 Plain binary，这是一个二进制格式的可执行文件，可以被烧些的目标板的 Flash 中。

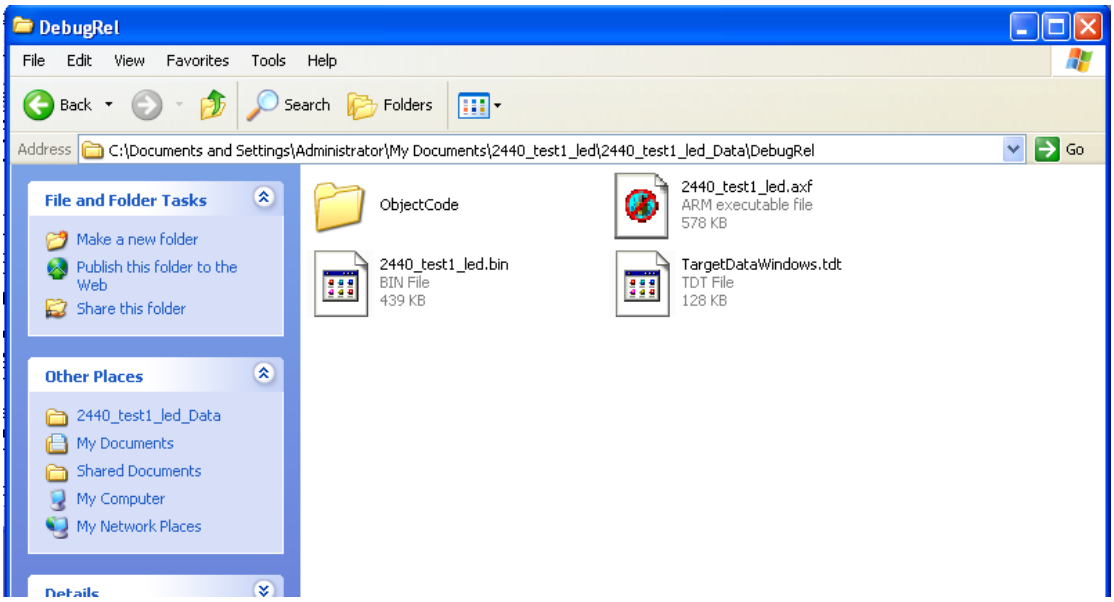
在 Output file name 文本域输入期望生成的输出文件存放的路径，或通过点击 Choose...按钮从文件对话框中选择输出文件。如果在这个文本域不输入路径名，则生成的二进制文件存放在工程所在的目录下。

进行好这些相关的设置后，以后在对工程进行 make 的时候，CodeWarrior IDE 就会在链接完成后调用 fromELF 来处理生成的映像文件。

对于本例的工程而言，到此，就完成了 make 之前的设置工作了。点击 CodeWarrior IDE 的菜单 Project 下的 make 菜单，对工程进行编译和链接。整个编译链接过程下图所示：

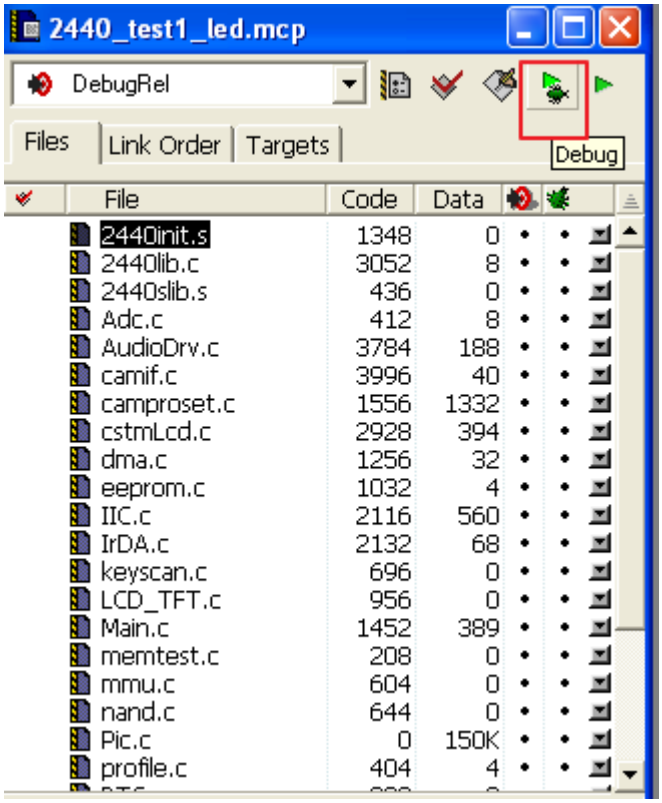


在工程 2440_test1_led 所在的目录下，会生成一个名为：工程名_data 目录，在本例中就是 2440_test1_led _Data 目录，在这个目录下不同类别的目标对应不同的目录。在本例中由于我们使用的是 DebugReL 目标，所以生成的最终文件都应该在该目录下。进入到 DebugRel 目录中去，会看到 make 后生成的映像文件和二进制文件，映像文件用于调试，二进制文件可以烧写到 TQ2440 的 Flash 中运行。

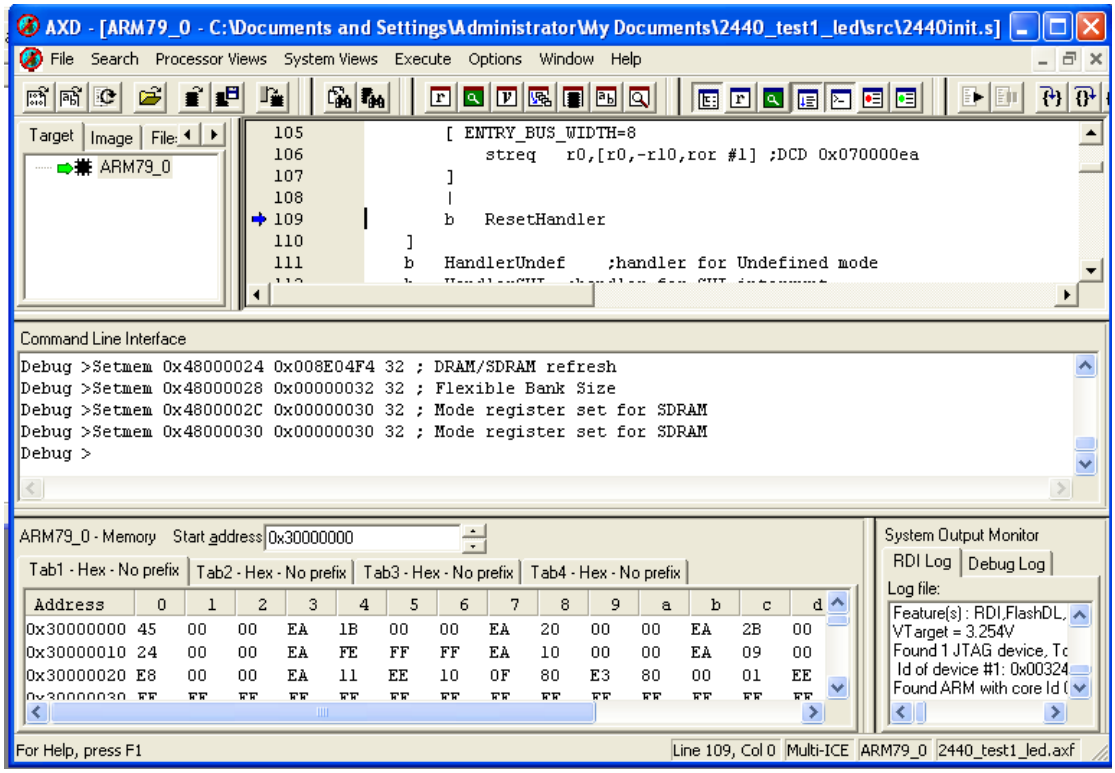


12.8调试代码

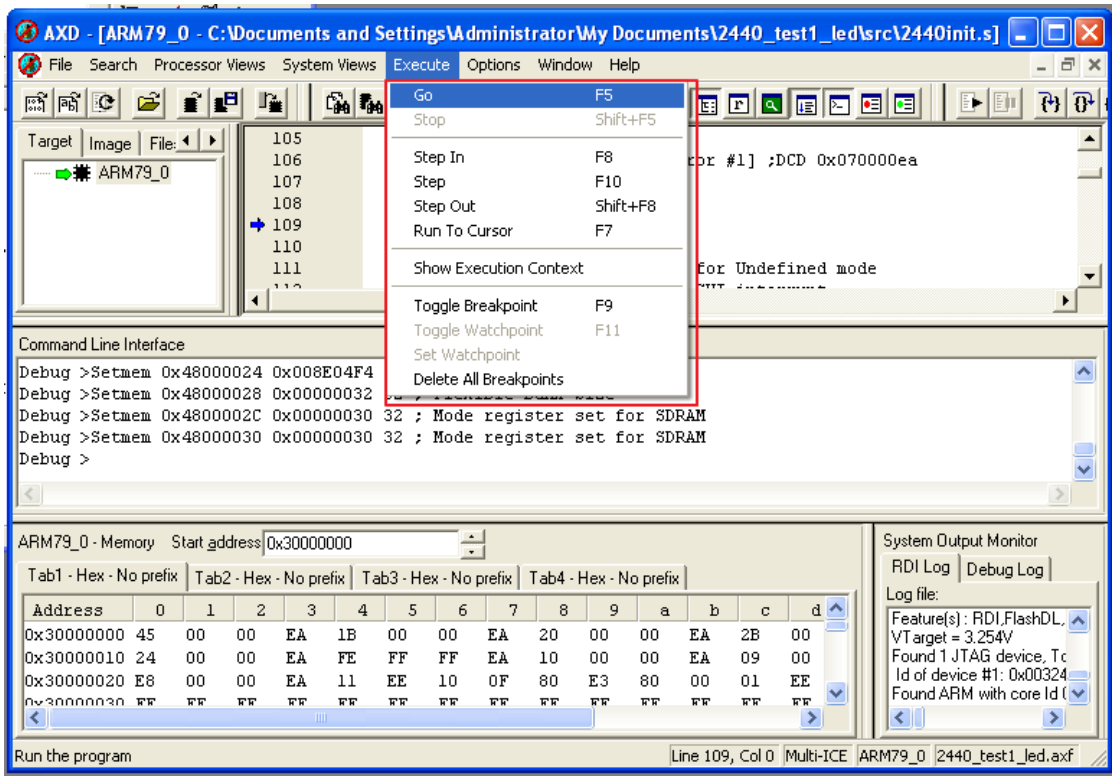
点击调试：



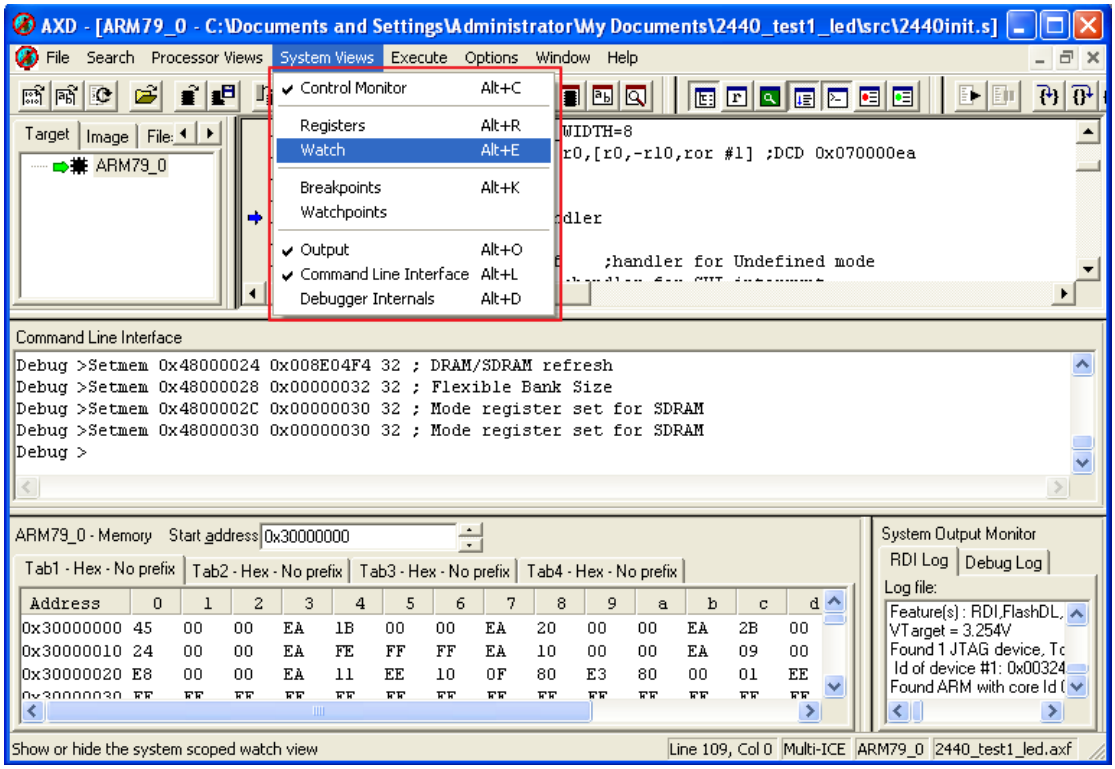
弹出 axd 调试界面：



“execute”目录可以控制程序运行：单步、断点、暂停等：

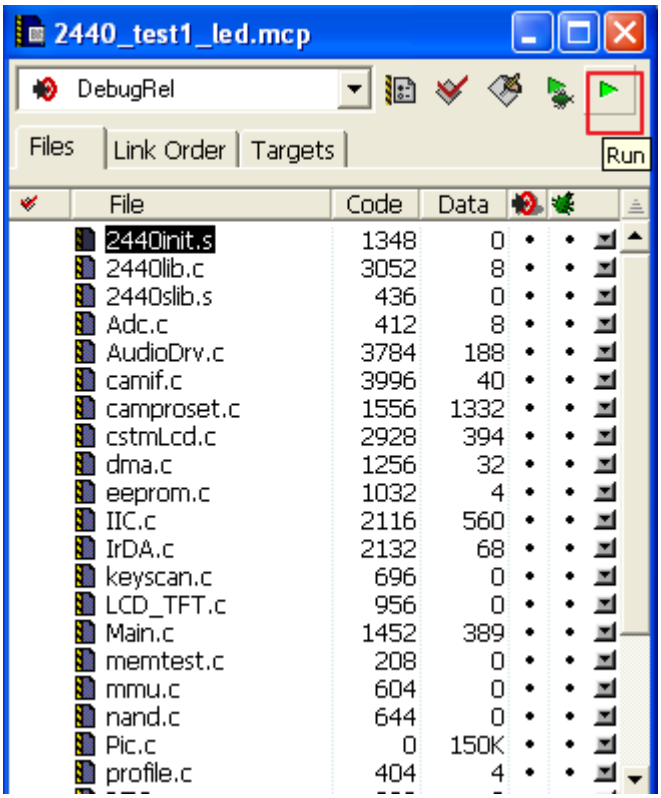


“system views”检查程序状态：内存、寄存器等。



12.9运行结果

点击 run 可以直接运行程序：


2440_test1_led.mcp
DebugRel
Run
Files
Link Order
Targets
File
Code
Data
2440init.s 1348 0
2440lib.c 3052 8
2440slib.s 436 0
Adc.c 412 8
AudioDrv.c 3784 188
camif.c 3996 40
camproset.c 1556 1332
cstmLcd.c 2928 394
dma.c 1256 32
eeprom.c 1032 4
IIC.c 2116 560
IrDA.c 2132 68
keyscan.c 696 0
LCD_TFT.c 956 0
Main.c 1452 389
memtest.c 208 0
mmu.c 604 0
nand.c 644 0
Pic.c 0 150K
profile.c 404 4

结果，在串口显示如图：

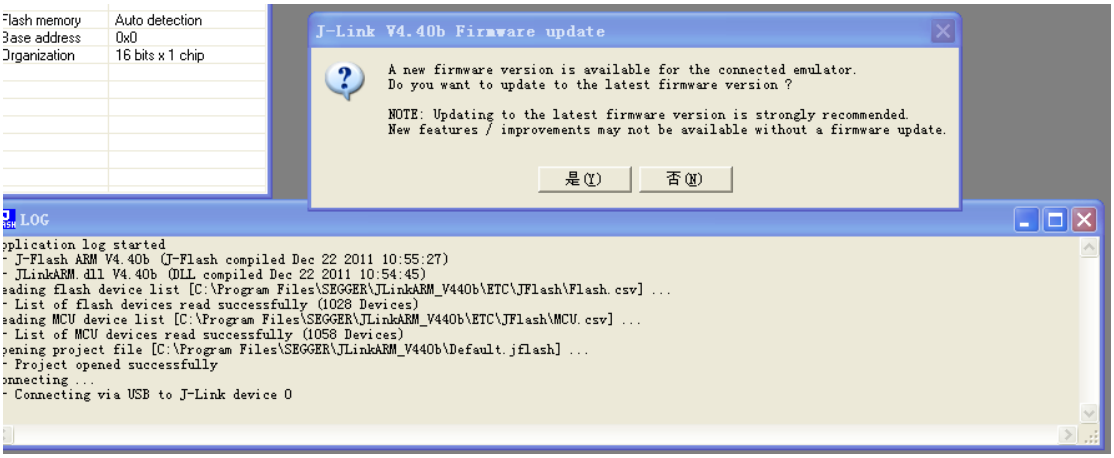
```
<****************************************>
               TQ2440 Test Program
                www.embedsky.net
<****************************************>

Please select your LCD type  :
0:              W35
1:              S35
2:              T35
3:              W43
4:              LCD57
5:              VGA
6:              A70
7:              LCD104

con1=0x678, con2=0xc3bc105, con3=0xb13f21, con4=0xd2c, con5=0xb09

Please select function :
0 : Please input 1-11 to select test
1 : Test PWM
2 : RTC time display
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test TFT LCD
7 : Test IIC EEPROM
8 : UDA1341 play music
9 : UDA1341 record voice
10 : Test SD Card
11 : Test CMOS Camera
4
Key Scan Test, press ESC key to exit !
Interrupt occur... K1 is pressed!
Interrupt occur... K1 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K1 is pressed!
Interrupt occur... K1 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K1 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K1 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K2 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K4 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K3 is pressed!
Interrupt occur... K3 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
Interrupt occur... K3 is pressed!
Interrupt occur... K3 is pressed!
Interrupt occur... Key is released!
Interrupt occur... Key is released!
```

13 实验问题

13.1J-LINK 固件升级

问题：

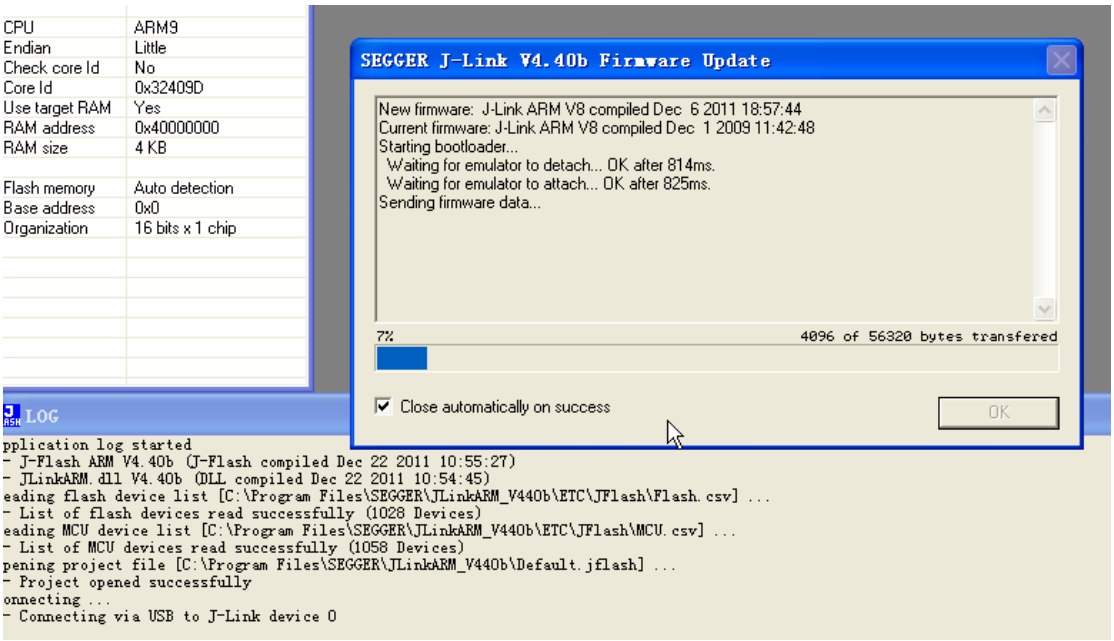
如果将 J-link 仿真器连接到 PC，打开 J-Flash ARM 程序时，弹出提示升级 j-link 固件的对话框。



解决：

请选择“yes”，然后等待 j-link 的固件升级，中途不要拔出 j-link，大约半分钟后，升级完成，j-link 上的绿灯停止闪烁。

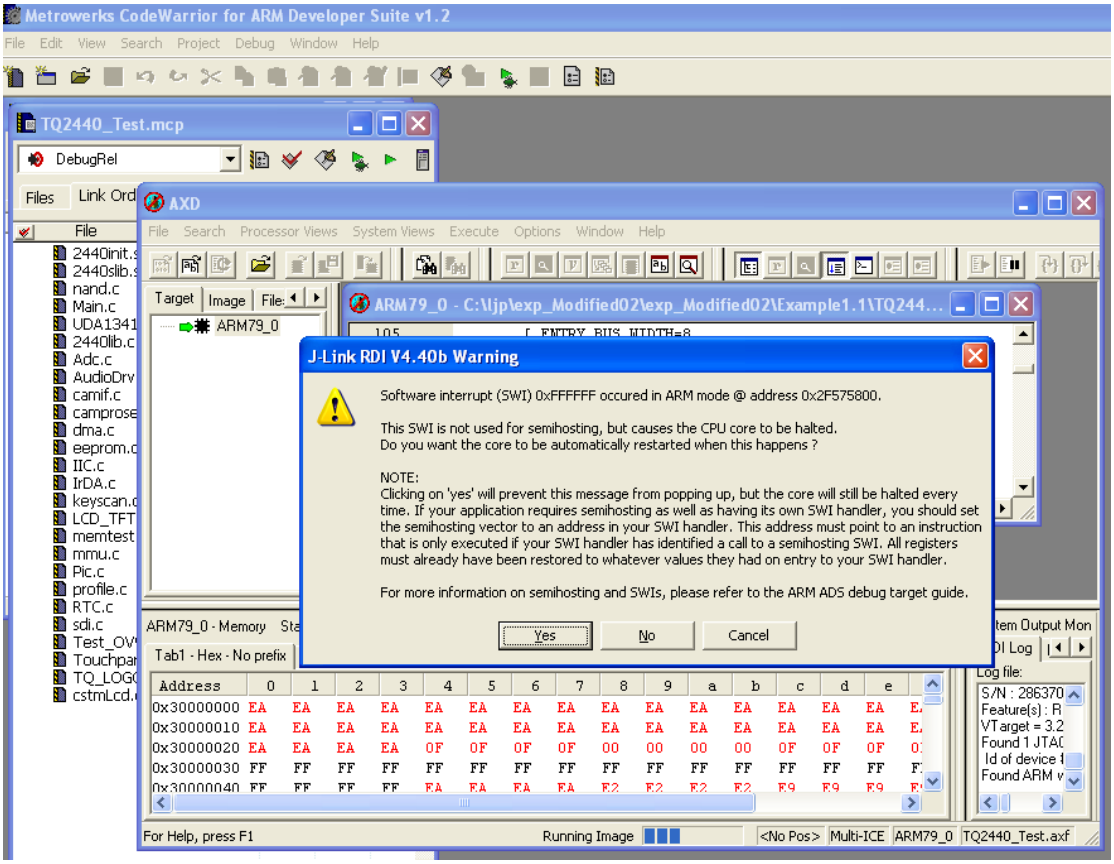
升级完后，J-link 可以继续使用，



13.2ADS 无法调试和运行代码

问题：

复位开发板后，在 ADS 中点击调试后无法运行程序（之前能运行的程序），可能的一种情况是弹出如下界面：



分析：

如果单步运行，第一条语句就发生错误。

如果显示 0x30000000 处的内存值，不是 0xea000045，而是 0xeaeaeaea 之类的错误值。

这基本上是因为开发板 SDRAM 没有初始化（SDRAM 基地址是 0x30000000），而 ADS 调试时，程序是下载到 SDRAM 中的，因为 SDRAM 没有初始化，所以不能下载正确的代码，所以程序也就无法运行。

解决：

有 2 种办法，思路都是在下载程序前先将 SDRAM 初始化好：

1. 打开 J-Flash ARM 程序，点击菜单“target-connect”，通过仿真器连接开发板；然后点击菜单“target-start application”或者 F9 键，此动作将复位开发板。因为开发板内已经固化有程序，该程序启动后会初始化 SDRAM。此时，再用 ads 调试就正常了。
2. 或者在 axd 中增加一个初始化文件，使得 axd 在下载程序前先通过命令初始化 SDRAM，然后再下载。

具体方法是，在 axd 的“Option->Config Interface->session file->run configuration script”处添加一个文件：D:\Embedded\TQ2440\sample_code\tq2440_axd_init.txt。

文件内容如下：

Setmem 0x53000000 0x00000000 32；pWTCON，看门狗定时器控制寄存器

Setmem 0x4A000008 0xFFFFFFFF 32 ; INTMSK，中断屏蔽寄存器

Setmem 0x4A00001C 0x000007FF 32 ; INTSUBMSK，针对 INTMAK 具体化的一个中断请求屏蔽寄存器

Setmem 0x53000000 0x00000000 32；pWTCON，看门狗定时器控制寄存器

Setmem 0x56000050 0x000055AA 32 ; rGPFCON , Port F control

Setmem 0x4C000014 0x00000005 32；CLKDIVN , CPU 时钟分频控制寄存器

Setmem 0x4C000000 0x00FFFFFF 32 ; LOCKTIME，锁时计数寄存器

Setmem 0x4C000004 0x0005C011 32 ; MPLLCON , MPLL 寄存器

Setmem 0x4C000008 0x00038022 32 ; UPLLCON , UPLL 寄存器

Setmem 0x48000000 0x2201D552 32 ; Bus width & wait status

Setmem 0x48000004 0x00000700 32 ; Boot ROM control

Setmem 0x48000008 0x00000700 32 ; BANK1 control

Setmem 0x4800000C 0x00000700 32 ; BANK2 control

Setmem 0x48000010 0x00001F4C 32 ; BANK3 control

Setmem 0x48000014 0x00000700 32 ; BANK4 control

Setmem 0x48000018 0x00000700 32 ; BANK5 control

Setmem 0x4800001C 0x00018005 32 ; BANK6 control

Setmem 0x48000020 0x00018005 32 ; BANK7 control

Setmem 0x48000024 0x008E04F4 32 ; DRAM/SDRAM refresh

Setmem 0x48000028 0x00000032 32 ; Flexible Bank Size

Setmem 0x4800002C 0x00000030 32 ; Mode register set for SDRAM

Setmem 0x48000030 0x00000030 32 ; Mode register set for SDRAM

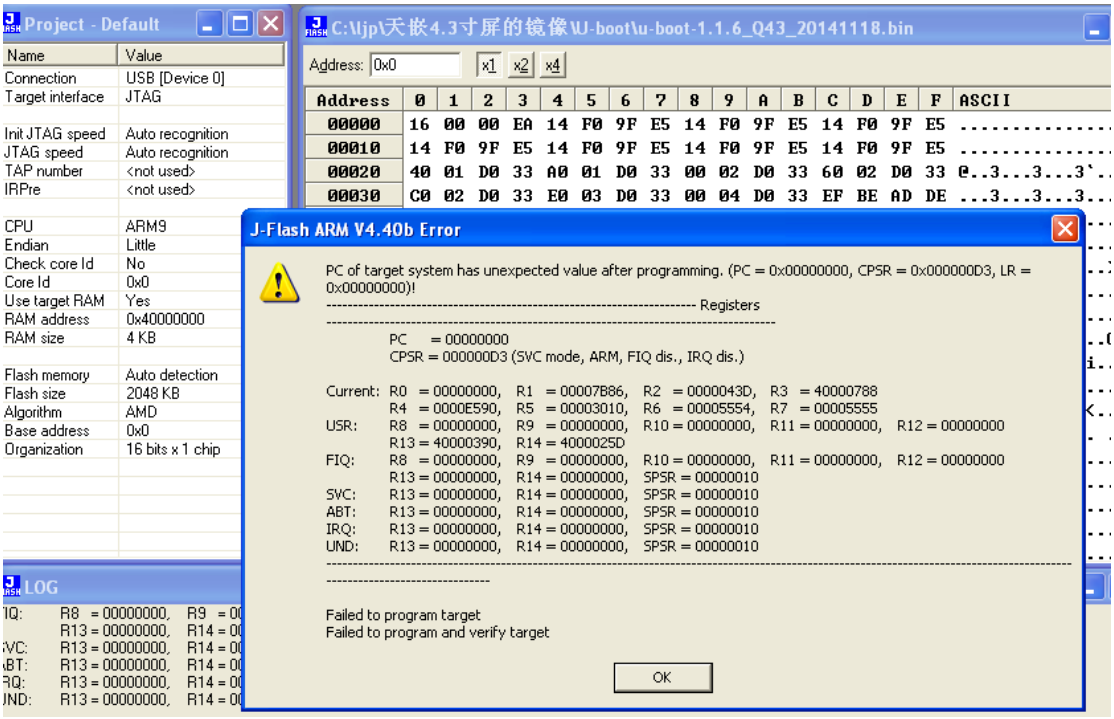
13.3J-LINK 仿真器烧写 Nor Flash 失败

问题：

用 J-Flash ARM 烧写 TQ2440 板子上的 Nor Flash 时，出现错误：

PC of target system has unexpected value after programming

如图：



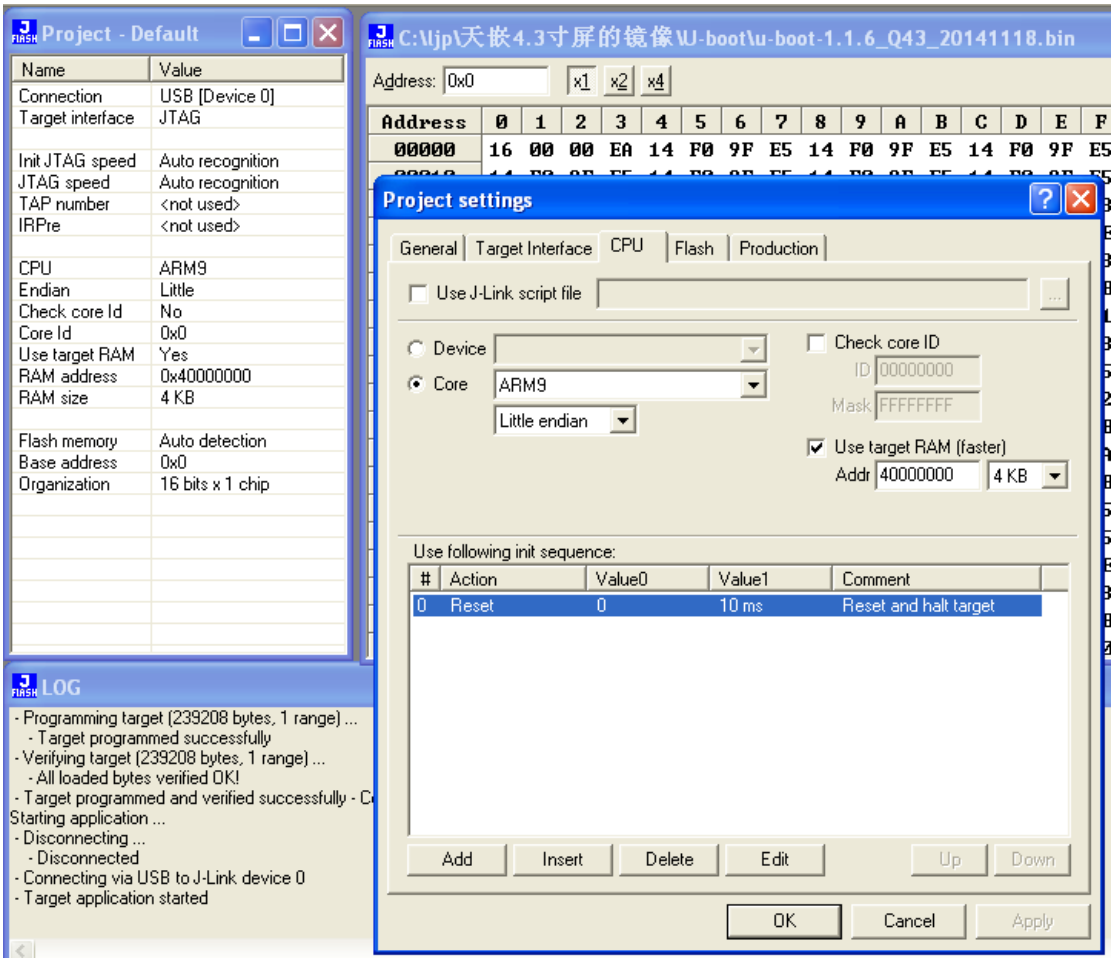
解决：

1. 修改 J-Flash ARM 的配置：

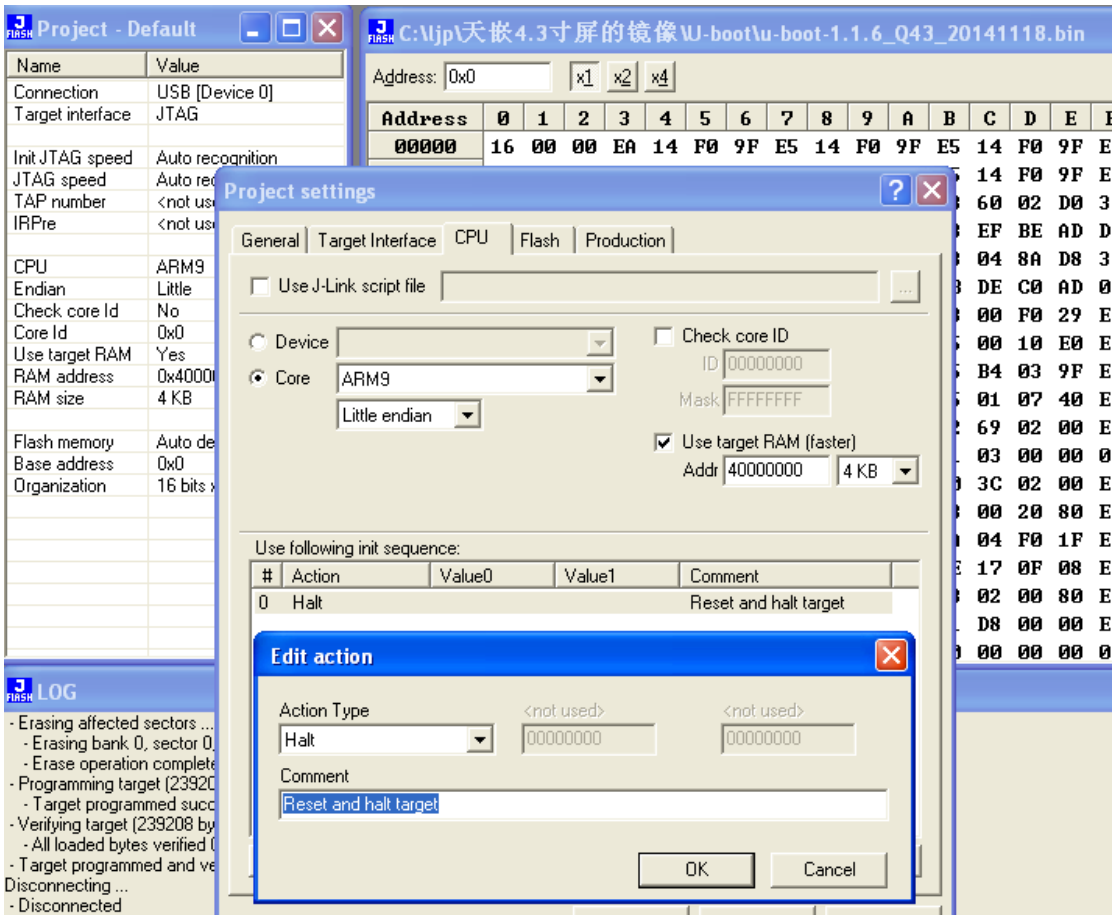
Options -> Project Settings -> CPU -> 'Use following init sequence:'中，默认只有一行：

0 reset　0 0ms reset and Halt target,

然后选中该行，点击 Edit，修改 Delay 为 10ms，确定，即可。



2. 或者，Options -> Project Settings -> CPU -> 'Use following init sequence:'这一行，把'Use following init sequence 中的动作从默认的 reset 改为 Halt，即使 cpu 暂停，使得 CPU 不会乱跑，然后接下来去烧写 nor flash，也就正常了。2 种办法目的都是等待系统稳定。目的相同，实现方法不同而已。



3. 还有一种可能性是之前烧录在 nor flash 中的程序修改了处理器状态，而 JLINK connect 时没有完全复位芯片，导致系统状态不稳定。例子之一是程序在 CLKCON 中禁止了 nand flash，从而导致 bootsram 被禁止，使得 JLINK 无法使用 bootsram 来下载程序进行烧录。此种情形的解决办法是：1.JLINK 不使用 bootsram，缺点是导致烧录速度非常慢。或者 2.在 Options -> Project Settings -> CPU -> 'Use following init sequence:增加使能 CLKCON 中 nandflash 的初始化设置，或者 3.修改 nor flash 初始化程序。

13.4双击打不开 mcp 工程文件

问题：

双击 mcp 文件，提示如下错误，打不开 mcp 工程文件

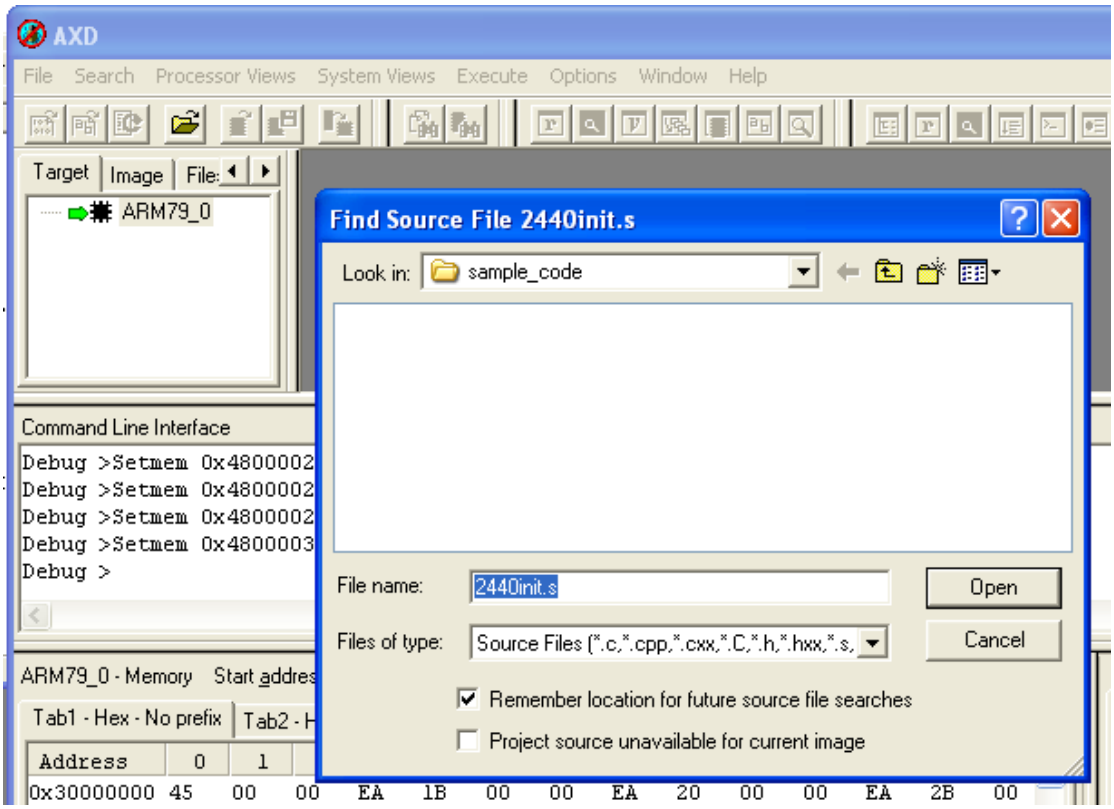
解决：

一般是因为 ADS 工程路径中有中文字符，移到英文路径就 OK 了。或者可以将 mcp 拖入 ADS，就可以打开。

13.5AXD 调试/运行时提示找不到 2440Init.s

问题：

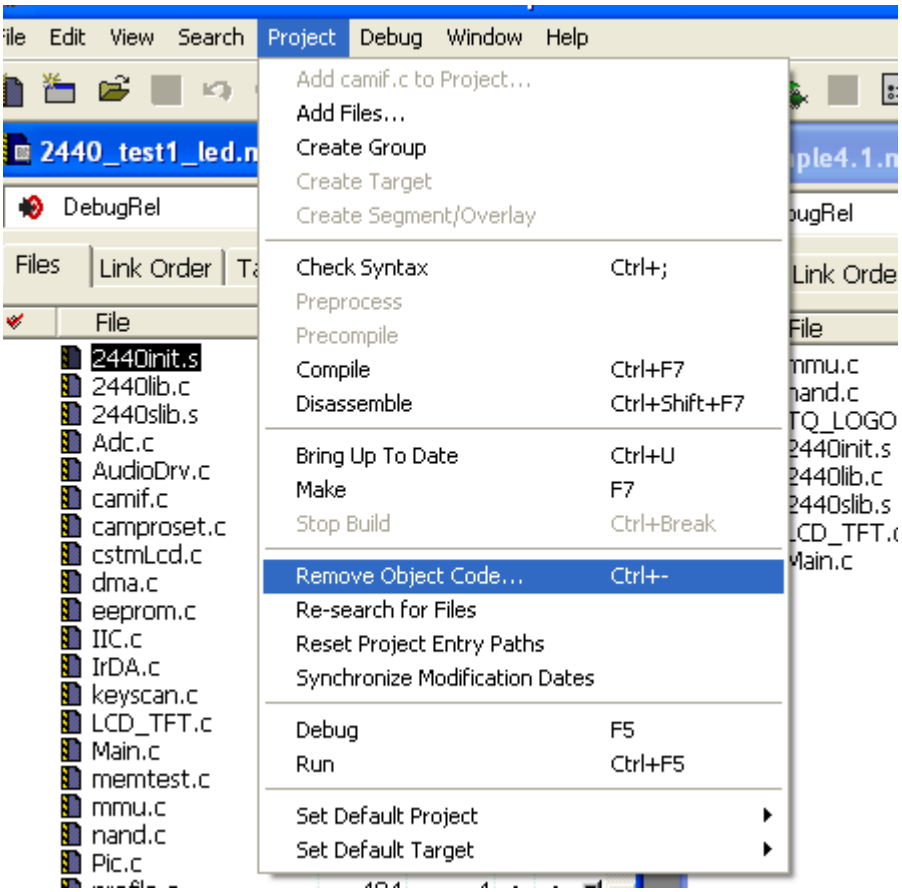
在 ADS 中点击 debug 或者 run 显示：

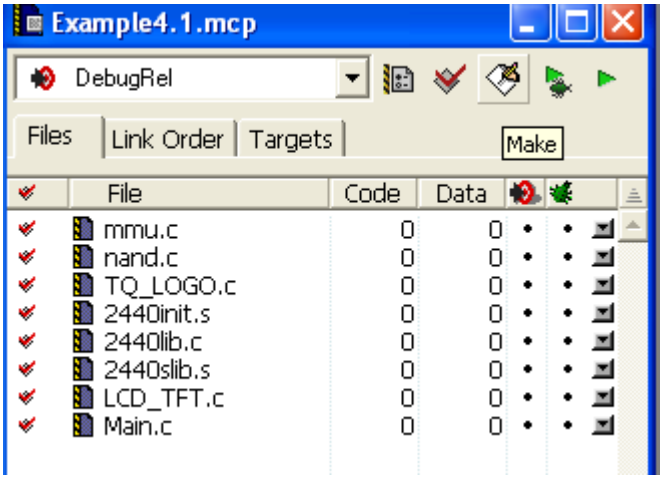


解决：

这是因为要调试的 axf 文件中带有目标调试文件路径信息，如果这个工程被移动后，没有重新 make（即一般出现在 make 已有代码时），就会出现这样的错误。

按照如下图所示删除已经生成的 object 文件，重新 make 一次，再重新 debug/run 即可。

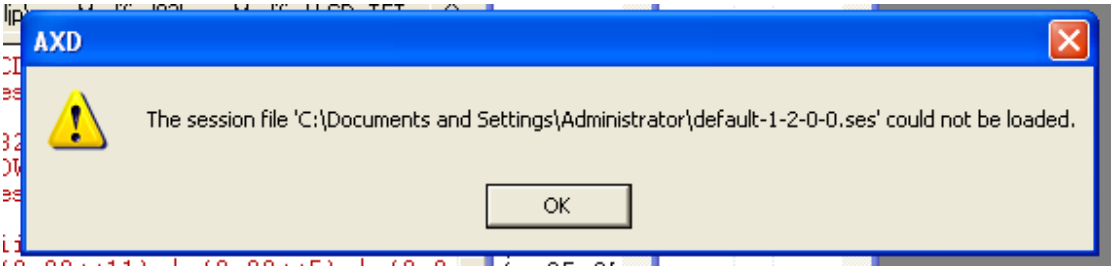




13.6AXD 调试/运行时找不到.ses 文件

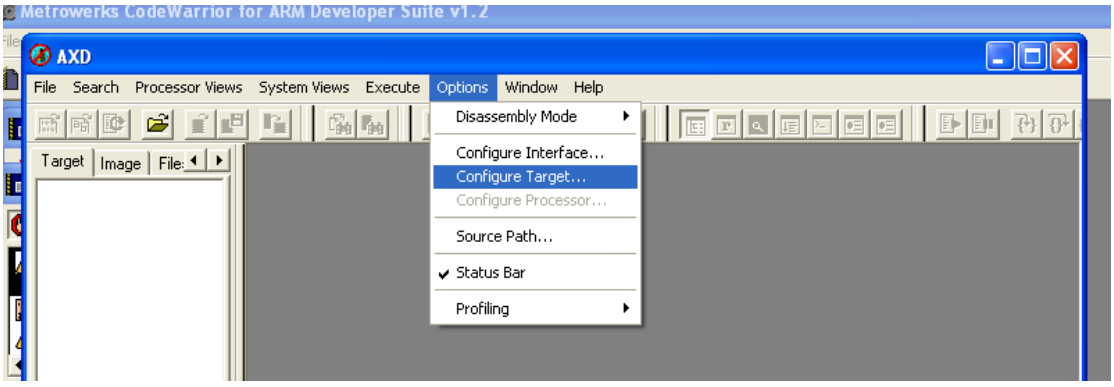
问题：

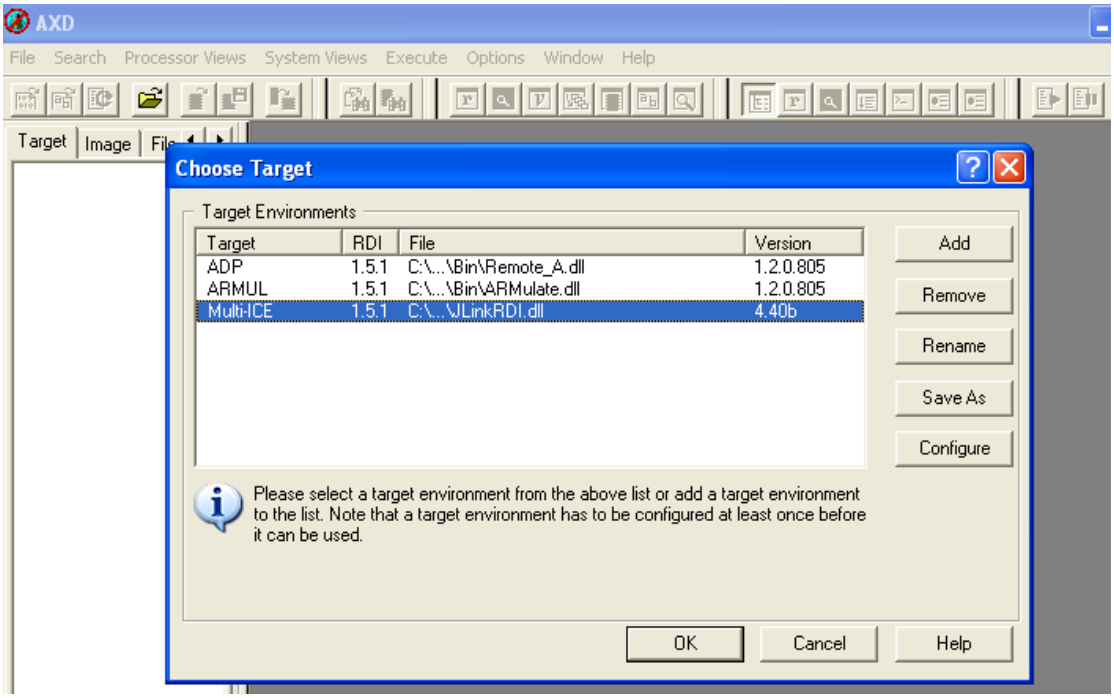
在 ADS 中点击 debug 或者 run 显示：



解决：

1. 可能是 ADS 工程目录带中文路径。由于 ADS 对中文的支持不是很完善，所以不要把工程放在中文目录下（路径中不能有中文）。如果有中文的话，在你运行 AXD 进行调试的时候会提供.SES 文件找不到。
2. 如果问题继续出现，或者没有中文路径，可设置 AXD->Options->ConfigureTarget...，在弹出的对话框里选择你的调试目录（选 JLinkRDI.dll）。然后关掉 AXD 再来一遍。




AXD
File Search Processor Views System Views Execute Options Window Help
Target Image File
Choose Target
Target Environments
Target RDI File Version
ADP 1.5.1 C:\...\Bin\Remote_A.dll 1.2.0.805
ARMUL 1.5.1 C:\...\Bin\ARMulate.dll 1.2.0.805
Multi-ICE 1.5.1 C:\...\JLinkRDI.dll 4.40b
Add
Remove
Rename
Save As
Configure
Please select a target environment from the above list or add a target environment to the list. Note that a target environment has to be configured at least once before it can be used.
OK
Cancel
Help