“黑色经典”系列之《ARM 嵌入式系统开发典型模块》

第 11 章 ARM 开发环境 ADS 1.2

11.1 ADS 1.2 简介

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 1.1 和 1.0，除了可以安装在 windows NT4、windows 2000、windows 98 和 windows95 操作系统下，还支持 windows ME 和 windows XP 操作系统。

ADS 由命令行开发工具、ARM 实时库、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开发的应用程序了。

下面介绍 ADS 的各个组成部分。

11.1.1 命令行开发工具

这些工具完成将源代码编译、链接成可执行代码的功能。

ADS 提供以下命令行开发工具。

1．armcc

armcc 是 ARM C 编译器。这个编译器通过 Plum Hall C Validation Suite 为 ARSI C 的一致性测试。armcc 用于将用 ANSIC 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，下面对其进行详细的介绍。

在命令控制台环境下，输入命令：

armcc-help

可以查看 armcc 的语法格式和一些常用的操作选项。

armcc 的最基本的用法为：armcc [option] file 1 file2 …file n。这里[option]是编译器所需要的选项，file 1 file2 …file n 是相关的文件名。

l 一些常用的选项功能如下：

-c：表示只进行编译不链接文件。

-C：（注意是大写）表示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，symbol 默认为 1。

-E：仅对 C 源代码产生警告时停止预处理。

-g<option>：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 的搜索路径列表当中去。

-J<directory>：用 directory 所指的路径代替#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 后面的数字不同，表示优化级别不同，-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。

有关更详细的说明，读者可以查看 ADS 软件的在线帮助文件。

下面为读者介绍上述 4 种 ARM C 和 C++编译器的命令通用语法。

compiler[PCS-options][source-language] [search-paths] [preprocessor-options] [output-format] [target-options] [debug-options] [code-generation-options] [warning-options] [additional-checks] [error-options] [source]

用户可以通过命令行操作选项控制编译器的执行。所有的选项都是以符号“-”开始，有些选项后面还跟有参数。在大多数情况下，ARM C 和 C++编译器允许在选项和参数之间存在空格。

命令行中各个选项出现顺序可以任意。

这里的 compiler 是指 armcc，tcc，armcpp 和 tcpp 中的一个。

PCS-options：指定了要使用的过程调用标准。

source-language：指定了编译器可以接受的编写源程序的语言种类。对于 C 编译器默认的语言是 ANSI C，对于 C++编译器默认是 ISO 标准 C++。

search-paths：该选项指定了对包含的文件（包括源文件和头文件）的搜索路径。

preprocessor-options：该选项指定了预处理器的行为，其中包括预处理器的输出和宏定义等特性。

output-format：该选项指定了编译器的输出格式，可以使用该项生成汇编语言输出列表文件和目标文件。

target-options：该选项指定目标处理器或 ARM 体系结构。

debug-options：该选项指定调试信息表是否生成，和该调试信息表生成时的格式。

code-generation-options：该选项指定了例如优化，字节顺序和由编译器产生的数据对齐格式等选项。

warning-options：该选项决定警告信息是否产生。

additional-checks：该选项指定了几个能用于源码的附加检查，例如检查数据流异常，检查没有使用的声明等；

error-options：该选项可以关闭指定的可恢复的错误，或者将一些指定的错误降级为警告。

source：该选项提供了包含有 C 或 C++源代码的一个或多个文件名，默认的，编译器在当前路径寻找源文件和创建输出文件。如果源文件是用汇编语言编写的（也就是说该文件的文件名是以.s 作为扩展名），汇编器将被调用来处理这些源文件。

如果操作系统对命令行的长度有限制，可以使用下面的操作，从文件中读取另外的命令行选项。

-via filename

该命令打开文件名为 filename 的文件，并从中读取命令行选项。用户可以对-via 进行嵌套调用，即，在文件 filename 中又通过-via finlename2 包含了另外一个文件。

在下面的例子中，从 input.txt 文件中读取指定的选项，作为 armcpp 的操作选项。

armcpp -via input.txt source.c 以上是对编译器选项的一个简单概述。它们（包括后面还要介绍的其他一些命令工具）既可以在命令控制台环境下使用，同时由于它们被嵌入到了 ADS 的图形界面中，所以也可以在图形界面下使用。

2．armcpp

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

3．tcc

tcc 是 Thumb C 编译器，该编译器通过了 Plum Hall C Validation Suite 为 ARSI C 的一致性测试。tcc 将 ARSIC 源代码编译成 16 位 Thumb 指令代码。

4．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。

5．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。

下面简要介绍要涉及到的一些术语。

映像文件（image）：是指一个可执行文件，在执行的时候被加载到处理器中。一个映像文件有多个线程。它是 ELF（Executable and linking format）格式的。

段（Section）：描述映像文件的代码或数据块。

RO：是 Read-only 的简写形式。

RW：是 Read-write 的简写形式。

ZI：是 Zero-initialized 的简写形式。

输入段（input section）：它包含着代码，初始化数据或描述了在应用程序运行之前必须要初始化为 0 的一段内存。

输出段（output section）：它包含了一系列具有相同的 RO、RW 或 ZI 属性的输入段。

域（Regions）：在一个映像文件中，一个域包含了 1～3 个输出段。多个域组织在一起，就构成了最终的映像文件。

Read Only Position Independent（ROPI）：它是指一个段，在这个段中，代码和只读数据的地址在运行时候可以改变。

Read Write Position Independent（RWPI）：它是指一个段，在该段中的可读/写的数据地址在运行期间可以改变。

加载时地址：是指映像文件位于存储器（在该映像文件没有运行时）中的地址。

运行时地址：是指映像文件在运行时的地址。

6．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。

11.1.2 ARM 运行时库

下面为读者介绍一下 ARM C/C++库方面的相关内容。

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。

（1）ANSI C 库函数

这个 C 函数库是由以下几部分组成。

① 在 ISO C 标准中定义的函数。

② 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信。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。

③ 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。

（2）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++库的内部本身不包含与目标相关的部分。这个库是由以下几部分组成的。

① 版本为 2.01.01 的 Rogue Wave Standard C++库；

② C++编译器使用的支持函数；

③ 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 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，适当地裁剪 C 库函数。

除此之外，用户还可以针对应用程序的要求，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时，用户必须指定一些最基本的操作选项。

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 2 个子目录。假设，ADS 软件安装在 e:\arm\adsv1_2 目录，则在 e:\arm\adsv1_2\lib 目录下的 2 个子目录 armlib 和 cpplib 是 ARM 的库所在的路径。

（1）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 e:\arm\adsv1_2\include 目录中。

（2）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 e:\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 2 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 2 个子目录。

这里需要让读者特别注意以下几点。

① ARM C 库函数是以二进制格式提供的。

② 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。

③ 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。

④ 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。

11.1.3 GUI 开发环境（Code Warrior 和 AXD）

1．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地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，因此节省了在操做工具上花的时间，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 配置以上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 2 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地跳转。
- 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上面介绍的各种命令。

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 和 Registers。

② Project 菜单不能使用的菜单选项是 Precompile 子菜单，因为 ARM 编译器不支持预编译的头文件。

③ Debug 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。

2．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）是 ARM 扩展调试器。
- armsd（ARM Symbolic Debugger）是 ARM 符号调试器。
- ADW/ADU（Application Debugger Windows/Unix）是与老版本兼容的 Windows 或 Unix 下的 ARM 调试工具。

11.1.4 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。

1．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。

2．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。

3．Flash downloader

这是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。

11.1.5 支持的软件

ADS 为用户提供下面的软件，使用户可以在软件仿真的环境下或者在基于 ARM 的硬件环境调试用户应用程序。

ARMulator 是一个 ARM 指令集仿真器，集成在 ARM 的调试器 AXD 中，它提供对 ARM 处理器的指令集的仿真，为 ARM 和 Thumb 提供精确的模拟。用户可以在硬件尚未做好的情况下，开发程序代码。

11.2 使用 ADS 1.2 的系统开发实例

本节和下一节用一个实例，简要地介绍应用 ADS1.2 进行系统开发的实例以及程序的调试过程。

11.2.1 建立一个工程

工程将所有的源码文件组织在一起，并能够决定最终生成文件存放的路径，输出的格式等。

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 11.1 所示的对 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 Executable 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。

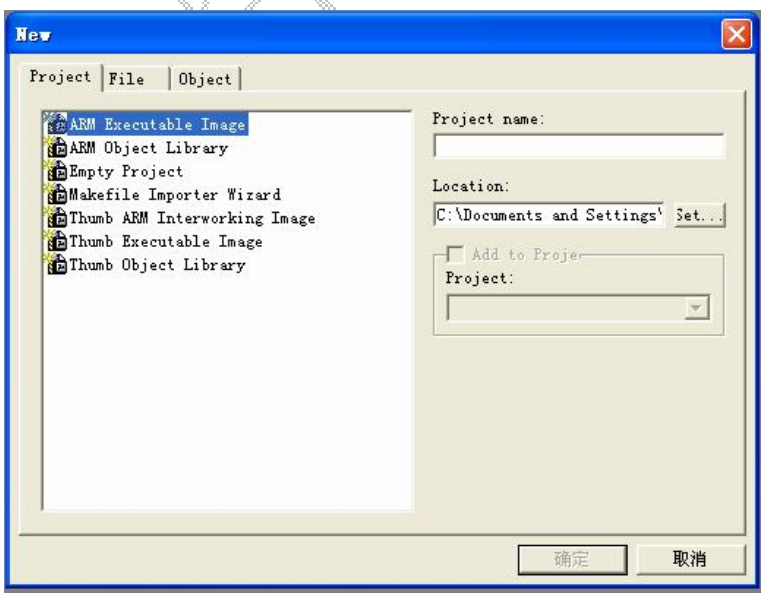

图 **11.1** 新建工程对话框

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 Interworking 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，例如“Myhelloworld”，点击“Location:”文本框后的“Set…”按钮，浏览选择想要将该工程保存的路径，设置好后，点击“确定”按钮，即可建立一个新的名为 Myhelloworld 的工程。

这个时候会出现 Myhelloworld.mcp 的窗口，如图 11.2 所示，有 3 个标签页，默认的是显示第一个标签页 Files，在该标签页单击鼠标右键，选中“Add Files…”可以把要用到的源程序添加到工程中。

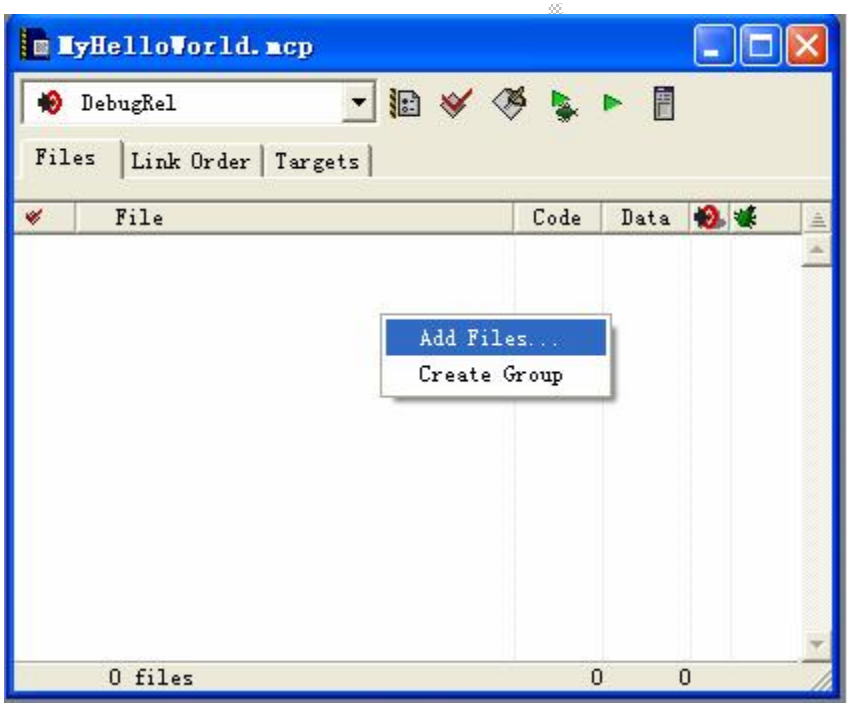

图 **11.2** 新建工程打开窗口

在本例当中，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。

选中“Add Files”命令后，弹出“Add Files”对话框，在 File name 中输入要创建的文件名，输入“Init.s”，单击“确定”按钮关闭窗口。

在打开的文件编辑框中输入下面的汇编代码。

```
; ****************************************************************
; File Name：  Init.s
; Description：
```

```
; Author：
; Date：
; ****************************************************************
    IMPORT Main
    AREA Init，CODE，READONLY
    ENTRY
    LDR R0, =0x3FF0000
    LDR R1, =0xE7FFFF80 ;配置 SYSCFG, 片内 4K Cache,4K SRAM
    STR R1, [R0]
    LDR SP, =0x3FE1000 ;SP 指向 4K SRAM 的尾地址，堆栈向下生成
    BL Main
    B
    END
```

在这段代码中，伪操作 IMPORT 告诉编译器，符号 Main 不是在该文件中定义的，而是在其他源文件中定义的符号，但是本源文件中可能要用到该符号。接下来用伪指令 AREA 定义段名为 Init 的段为只读的代码段，伪指令 ENTRY 指出了程序的入口点。下面就是用汇编指令实现了配置 SYSCFG 特殊功能寄存器，将 S3C4510B 片内的 8KB 一体化的 SRAM 配置为 4KB Cache，4KB SRAM，并将用户堆栈设置在片内的 SRAM 中。

4KB SRAM 的地址为 0x3FE,0000～(0x3FE,1000-1)，由于 S3C4510B 的堆栈由高地址向低地址生成，将 SP 初始化为 0x3FE,1000。

完成上述操作后，程序跳转到 Main 函数执行。

保存 Init.s 汇编程序。

用同样的方法，再建立一个名为 main.c 的 C 源代码文件。具体代码内容如下。

```
//*********************************************************************
//File Name：  main.c
//Description：
//Author：
//Date：
//*********************************************************************
#define IOPMOD (*(volatile unsigned *)0x03FF5000) //IO port mode register
#define IOPDATA (*(volatile unsigned *)0x03FF5008) //IO port data register
void Delay(unsigned int);
int Main()
{
    unsigned long LED;
    IOPMOD=0xFFFFFFFF; //将 IO 口置为输出模式
    IOPDATA=0x01;
    for(;;)
```

```
    {
      LED=IOPDATA;
      LED=(LED<<1);
      IOPDATA=LED;
      Delay(10);
      if(!(IOPDATA&0x0F))
            IOPDATA=0x01;
      }
        return(0);
    }
void Delay(unsigned int x)
{
    unsigned int i,j,k;
    for(i=0;i<=x;i++)
        for(j=0;j<0xff;j++)
            for(k=0;k<0xff;k++);
}
```

该段代码首先将 I/O 模式寄存器设置为输出模式，为 I/O 数据寄存器赋初值 0x1，通过将 I/O 数据寄存器的数值进行周期性的左移，实现使接在 P0~P3 口的 LED 显示器轮流被点亮的功能。注意这里的 if 语句，是为了保证当 I/O 数据寄存器中的数在移位过程中，第 4 位为数字“1”时，使数字 1 通过和 0xFF 相与，又重新回到 I/O 数据寄存器的第 0 位，从而保证了数字 1 一直在 I/O 数据寄存器的低 4 位之间移位。

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 3 个 target 的含义分别如下。

- 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。

现在已经新建了两个源文件，接下来要把这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 11.2 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 11.3 所示，询问用户把文件添加到何类目标中，在这里，选择 DebugRel 目标。把刚才创建的两个文件添加到工程中来。

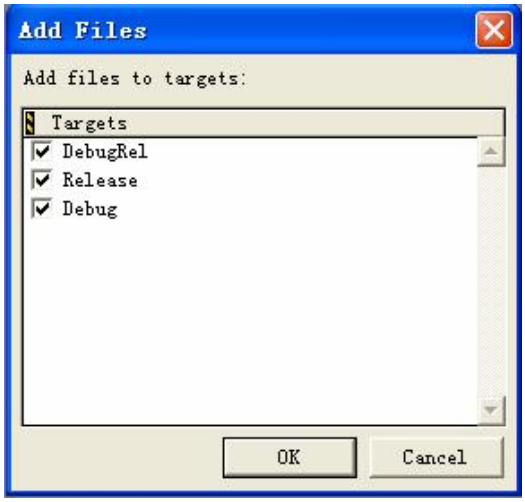

图 **11.3**　选择添加文件到指定目标

至此，一个完整的工程已经建立。

下面对工程进行编译和链接工作。

11.2.2　编译和链接工程

在进行编译和链接前，首先讲述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），出现如图 11.4 所示的对话框。

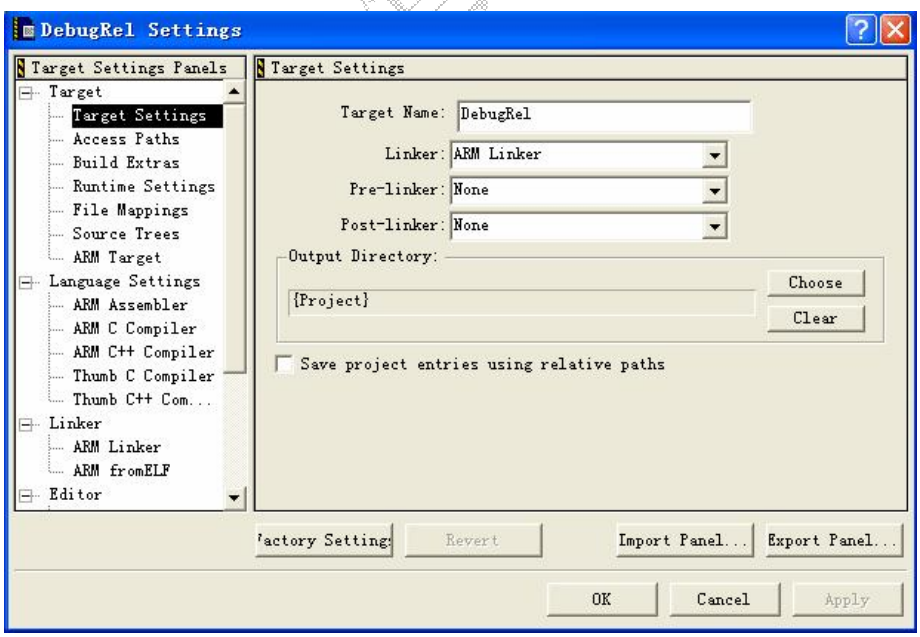

图 **11.4**　**DebugRel** 设置对话框

这个对话框中的设置很多，在这里指介绍一些最为常用的设置选项，读者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 armlink 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 2 个可选项，None 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。这个汇编器实际就是在 11.1 节中谈到的 armasm，默认的 ARM 体系结构是 ARM7TDMI，正好符合目标板 S3C4510B，无需改动。字节顺序默认就是小端模式。其他设置，用默认值即可。

还有一个需要注意的是 ARM C 编译器，它实际就是调用的命令行工具 armcc。使用默认的设置就可以了。

细心的读者可能会注意到，在设置框的右下脚，当对某项设置进行了修改时，该行中的某个选项就会发生相应的改动，如图 11.5 所示。实际上，这行文字就显示的是在 11.1 中介绍的相应的编译或链接选项，由于有了 CodeWarrior，开发人员可以不用再去查看繁多的命令行选项，只要在界面中选中或撤消某个选项，软件就会自动生成相应的代码，为不习惯在 DOS 下键入命令行的用户提供了极大的方便。

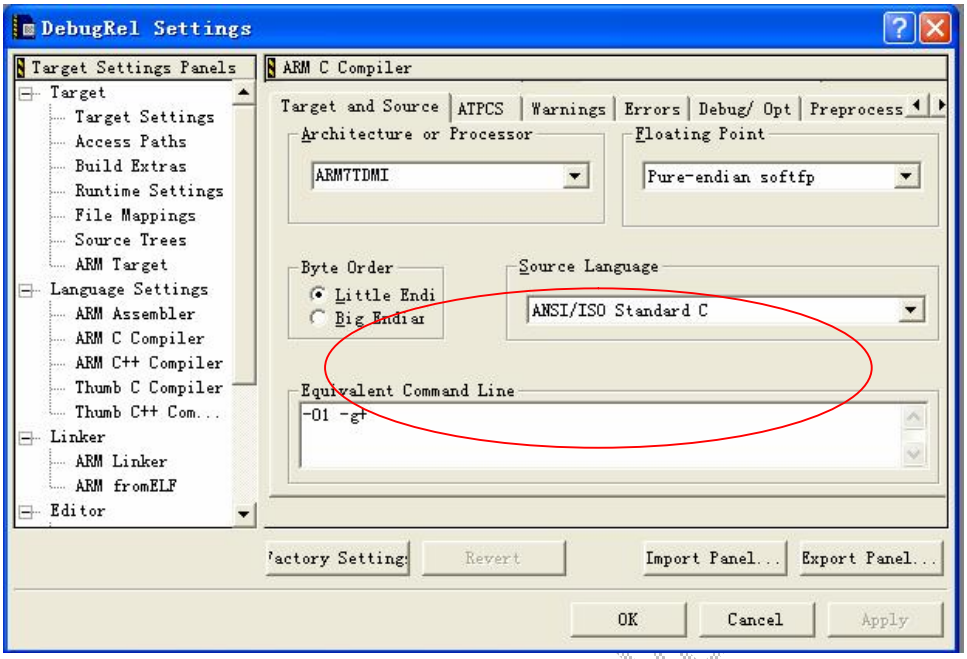

图 **11.5**　命令行工具选项设置

3．Linker 设置

鼠标选中 ARM Linker，出现如图 11.6 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。

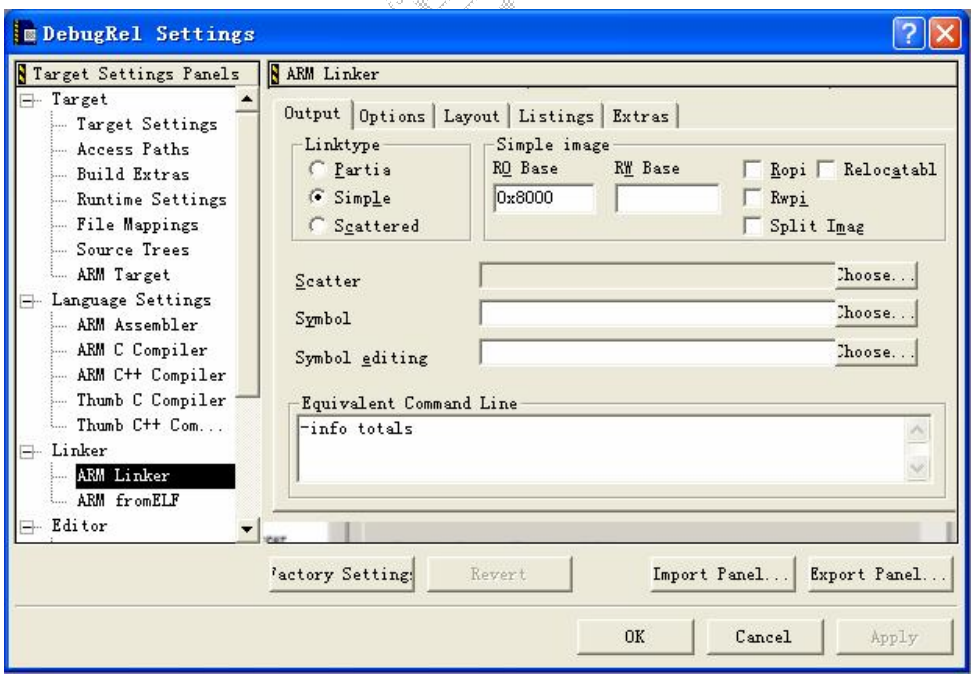

图 **11.6**　链接器设置

在标签页 Output 中，Linktype 中提供了 3 种链接方式。Partia 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，也是使用最为频繁的链接方式，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。一般情况下这个选项使用不太多。

本例选择 Simple 方式就可以了。

在选中 Simple 方式后，就会出现 Simple image。

R0 Base：这个文本框设置包含有 R0 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。针对本书所介绍的目标板，就可以使用这个默认地址值。

RW Base：这个文本框设置包含 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 2 个加载域和 2 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 R0 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 R0 和 RW 的输出段的加载域分成 2 个加载域：一个是包含 R0 输出段的域，另一个是包含 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，例如-entry 0x0。

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。

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。如果该符号有多处定义存在，armlink 将产生出错信息。

-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。

需要注意的是，这里不可以用符号 main 作为入口点地址符号，否则将会出现“Image dose not have an entry point(Not specified or not set due to multiple choice)”的出错信息。

关于 ARM Linker 的设置还有很多，对于想进一步了解的读者，可以查看帮助文件，其中有很详细的介绍。

在 Linker 下还有一个 ARM fromELF，如图 11.7 所示。

fromELF 就是在 11.1 节中介绍的一个实用工具，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，可以被烧写在目标板的 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路径。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，在对工程进行 make 时，CodeWarrior IDE 就会在链接完成后调用 fromELF 来处理生成的映像文件。

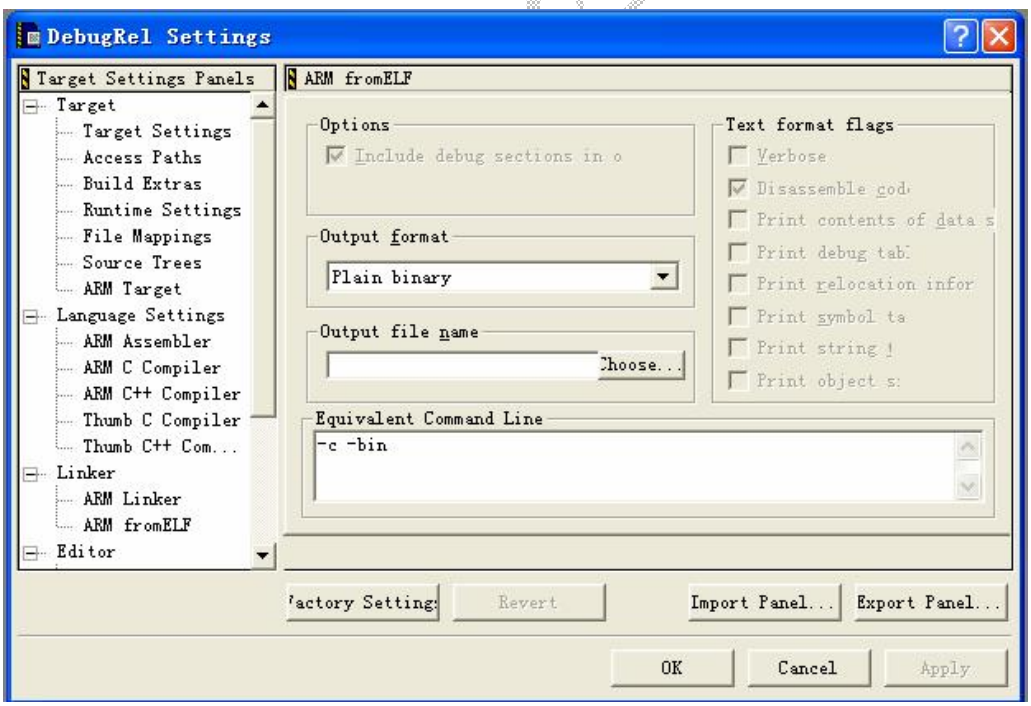

图 11.7　ARM fromELF 可选项

对于本例而言，到此就完成了 make 之前的设置工作。

单击 CodeWarrior IDE 的菜单 Project 下的 make 菜单，就可以对工程进行编译和链接了。

在工程 ledcircle 所在的目录下，会生成一个名为：工程名_data 目录，进入到 DebugRel 目录中，读者会看到 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 S3C4510B 的 Flash 中运行。

11.2.3 使用命令行工具编译应用程序

如果用户开发的工程比较简单，或者只是想用到 ADS 提供的各种工具，而并不想在 CodeWarrior IDE 中进行开发。在这种情况下，再为读者介绍一种不在 CodeWarrior IDE 集成开发环境下，开发用户应用程序的方法，当然前提是用户必须安装了 ADS 软件，因为在编译链接的过程中要用到 ADS 提供的各种命令工具。

这种方法对于开发包含较少源代码的工程是比较实用的。

首先用户可以用任何编辑软件（比如 UltraEdit）编写 11.2.1 中所提到的 2 个源文件 Init.s 和 main.c。接下来，可以利用在第 7 章中介绍的 makefile 的知识，编写自己的 makefile 文件。对于本例，编写的 makefile 文件（假设该 makefile 文件保存为 ads_mk.mk）如下。

```
PAT = e:/arm/adsv1_2/bin
CC = $(PAT)/armcc
LD = $(PAT)/armlink
OBJTOOL = $(PAT)/fromelf
RM = $(PAT)/rm -f
AS = $(PAT)/armasm -keep -g
ASFILE = e:/arm_xyexp/Init.s
CFLAGS = -g -01 -Wa -DNO_UNDERSCORES=1
MODEL = main
SRC = $(MODEL).c
OBJS = $(MODEL).o
all: $(MODEL).axf $(MODEL).bin clean
%.axf:$(OBJS) Init.o
@echo "### Linking ..."
$(LD) $(OBJS) Init.o -ro-base 0x8000 -entry Main -first Init.o -o $@ -libpath e:/arm/adsv1_2/lib
%.bin: %.axf
$(OBJTOOL) -c -bin -output $@ $<
$(OBJTOOL) -c -s -o $(<:.axf=.lst) $<
%.o:%.c
@echo "### Compiling $<"
$(CC) $(CFLAGS) -c $< -o $@
clean:
$(RM) Init.o $(OBJS)
```

由于 ADS 在安装的时候没有提供 make 命令，如 ADS 安装在目录 e:\arm\adsv1_2 下，可以将 make 命令拷贝到 e:\arm\adsv1_2\bin 目录下进行编译和链接。

经过上述编译链接以及链接后的操作，在 e:\arm_xyexp\ledcircle 目录下会生成 2 个新的文件，main.axf 和 main.bin。

用这种方式生成的文件与在 CodeWarrior IDE 界面通过各个选项的设置生成的文件是一样的。

11.3 ADS 1.2 的程序调试

AXD（ARM eXtended Debugger）是 ADS 软件中独立于 CodeWarrior IDE 的图形软件，打开 AXD 软件，默认打开的目标是 ARMulator。这也是调试的时候最常用的一种调试工具，本节主要是结合 ARMulator，介绍在 AXD 中进行代码调试的方法和过程，使读者对 AXD 的调试有初步的了解。

要使用 AXD 必须首先生成包含有调试信息的程序，前面 11.2 的实例中已经生成的 MyHelloWorld.axf 或 main.axf 就是含有调试信息的可执行 ELF 格式的映像文件，本节应用前节的实例简单介绍一下程序的调试过程。

11.3.1 在 AXD 中打开调试文件

在菜单 File 中选择“Load image…”选项，打开 Load Image 对话框，找到要装载的.axf 映像文件，点击“打开”按钮，就把映像文件装载到目标内存中了。

在打开的映像文件中会有一个蓝色的箭头指示当前执行的位置。对于本例，打开映像文件后，如图 11.8 所示。

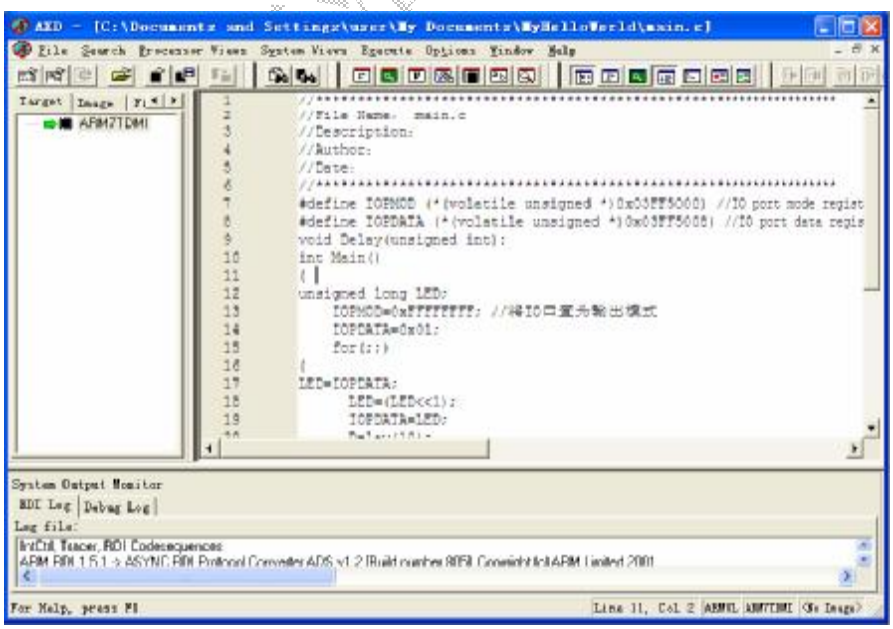

图 11.8 在 AXD 下打开映像文件

在菜单 Execute 中选择“Go”，将全速运行代码。要想进行单步的代码调试，在 Execute 菜单中选择“Step”选项，或用 F10 即可，窗口中蓝色箭头会发生相应的移动。

有时候，用户可能希望程序在执行到某处时，查看一些所关心的变量值，此时可以通过断点设置实现。将光标移动到要进行断点设置的代码处，在 Execute 菜单中，选择“Toggle Breakpoint”或按 F9，就会在光标所在位置出现一个实心圆点，表明该处为断点。

也可以在 AXD 中查看寄存器值，变量值，某个内存单元的数值等等。

11.3.2 查看存储器内容

在程序运行前，可以先查看 2 个宏变量 IOPMOD 和 IOPDATA 的当前值。从 Processor Views 菜单中选择“Memory”选项。

在 Memory Start address 选择框中，读者可以根据要查看的存储器的地址输入起始地址。因为 I/O 模式控制寄存器和 I/O 数据控制寄存器，都是 32 位的控制寄存器，所以从 0x03ff5000 开始的连续 4 个地址空间存放的是 I/O 模式控制寄存器的值，可程序运行到 for 循环处时，再查看这两个寄存器的内容，以对比变化。

11.3.3 设置断点

可以在 for 循环语句的“Delay(10);”语句处设置断点，将光标定位在该语句处，如图 11.9 所示，使用快捷键 F9 在此处设置断点，按 F5 键，程序将运行到断点处，如果读者想查看子函数 Delay 是如何运行的，可以在 Execute 菜单中选择“Step In”选项，或按下 F8 键，进入到子函数内部进行单步程序的调试。

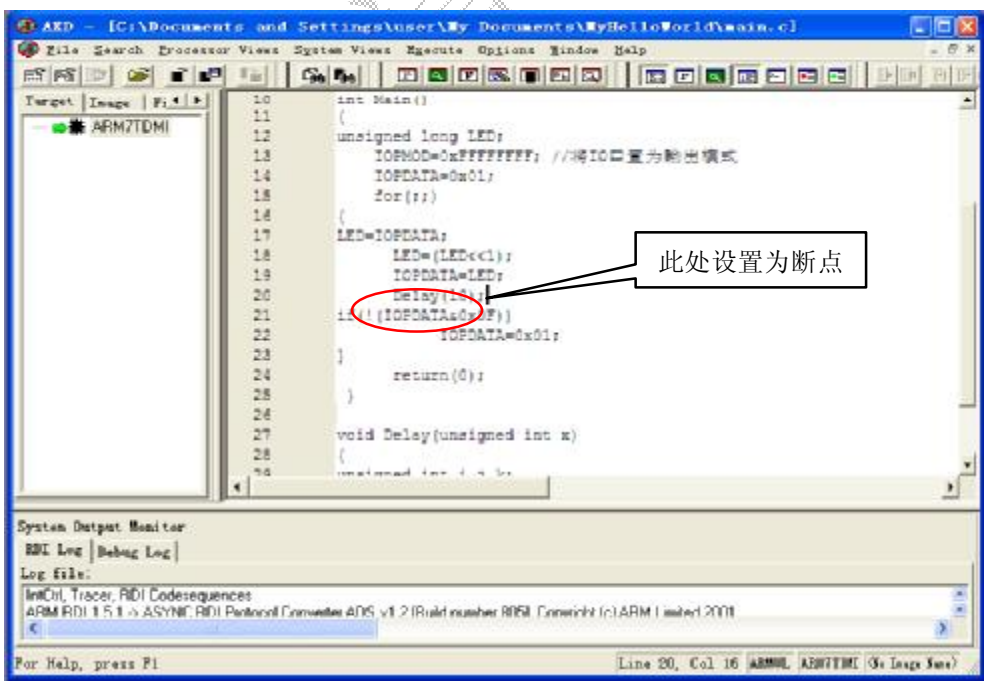

图 **11.9** 设置断点

11.3.4 查看变量值

在 Delay 函数的内部，如果用户希望查看某个变量的值，比如查看变量 i 的值，可以在 Processor Views 菜单中选择“Watch”，会出现如图 11.10 所示的 watch 窗口，然后右键单击变量 i，在快捷菜单中选中“Add to watch”，这样变量 i 默认是添加到 watch 窗口的 Tab1 中。程序运行过程中，用户可以看到变量 i 的值在不断的变化。默认显示变量数值是以十六进制格式显示的，如果用户对这种显示格式不习惯的话，可以通过在 watch 窗口单击鼠标右键，在弹出的快捷菜单中选择“Format”选项选择所查看的变量显示格式。如果用户想从 Delay 函数中跳出，到主函数中去，最简单的方法就是将光标定位到想跳转到的主函数处，在 Execute 菜单中选择“Run to Cursor”选项，则程序会从 Delay 函数中跳转到光标所在位置。

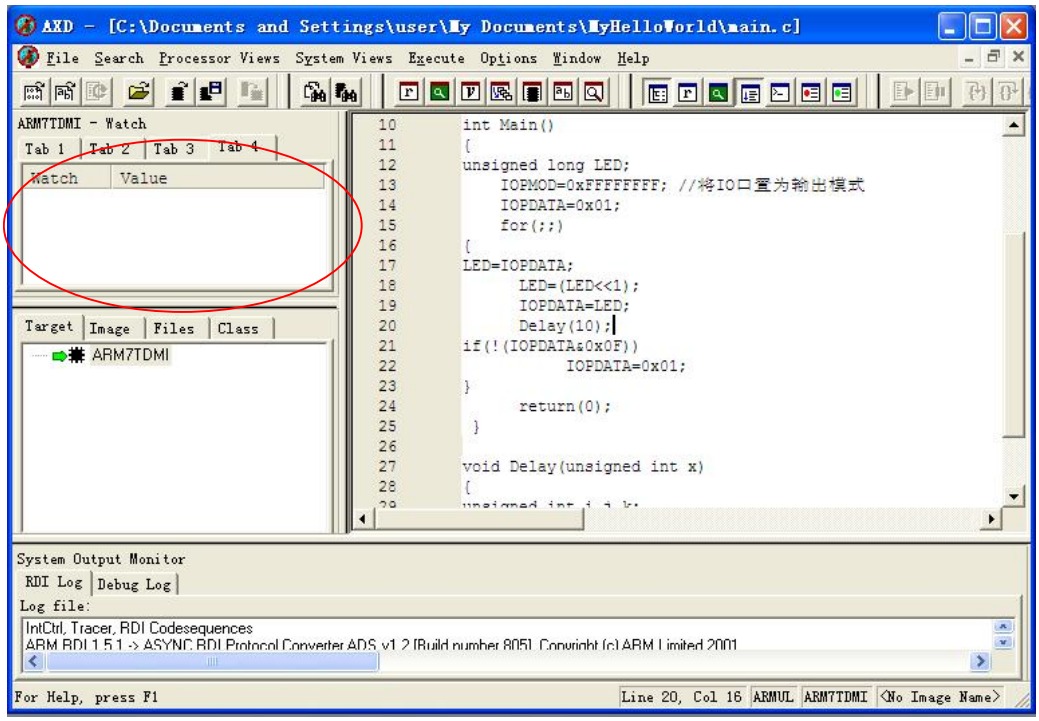

图 **11.10** 查看变量

11.4 本 章 小 结

本章主要介绍了 ADS 软件。首先介绍了 ADS 软件的基本组成部分，接着重点地介绍了最常用的 2 个命令工具 armcc 和 armlink 的使用语法和各个操作选项。然后结合一个具体的应用实例，介绍如何在 CodeWarrior IDE 环境下建立自己的新工程，编译和链接工程生成可以调试的映像文件和二进制文件的过程，同时补充了一种不在

CodeWarrior IDE 集成开发环发环境下，利用有关 make 的知识，利用 ADS 提供的编译和链接等命令工具编写 makefile 文件，来开发和编译程序的方法。最后介绍了如何使用 ADS 的调试软件 AXD 调试应