第5章 ARM混合编程和ADS 1.2集成开发环境

CHAPTER 5


视频讲解

5.1 C语言和汇编语言混合编程方式

在应用系统的程序设计中，若所有的编程任务均用汇编语言完成，其工作量是可想而知的。事实上，ARM体系结构支持C/C++以及与汇编语言的混合编程，在一个完整的程序设计中，除了初始化部分用汇编语言完成以外，其主要的编程任务一般都用C/C++完成。汇编语言与C/C++的混合编程主要有以下几种情况。

1. 在C中内嵌汇编语言

在C中内嵌的汇编指令包含大部分的ARM和Thumb指令，不过其使用与汇编文件中的指令有些不同，存在一些限制，主要表现在如下几个方面：

(1) 不能直接向PC寄存器赋值，程序跳转要使用B或者BL指令。

(2) 在使用物理寄存器时，不要使用过于复杂的C表达式，以避免物理寄存器冲突。

(3) R12和R13可能被编译器用来存放中间编译结果，计算表达式值时可能将R0～R3、R13及R14用于子程序调用，因此要避免直接使用这些物理寄存器。

(4) 一般不要直接指定物理寄存器，而让编译器进行分配。

下面通过一个例子来说明如何在C中内嵌汇编语言：

```
#include <stdio.h>
void my_strcpy(const char *src, char *dest)   //声明一个函数
{
    char ch;                                  //声明一个字符型变量
    __asm                                     //调用关键词__asm
    {
    LOOP                                  ;循环入口
        LDRB CH,[src],#1                  ;Thumb指令,将无符号src地址的数送入CH,src+1;
        STRB CH,[dest],#1                 ;Thumb指令,将无符号CH数据送入[dest]存储, dest+1
        CMP CH, #0                        ;比较CH是否为零,否则循环,总共循环256次
        BNE LOOP;                         ;B指令跳转,NE为Z位清零不相等
    }
}
int main()                                ;C语言主程序
{
 char *a = "forget it and move on!";      //声明字符型指针变量
 char b[64];                              //字符型数组
 my_strcpy(a, b);                         //调用子函数,进行复制
 printf("original: %s", a);               //屏幕输出,a的数值
 printf("copyed: %s", b);                 //屏幕输出,b的数值
 return 0;
}
```

在这里 C 和汇编语言之间的值传递是用 C 的指针来实现的，因为指针对应的是地址，所以汇编语言中也可以访问。

2. 在汇编语言中使用 C 程序全局变量

内嵌汇编不用单独编辑汇编语言文件，比较简洁，但是有诸多限制，当汇编代码较多时一般要放在单独的汇编文件中。这就需要在汇编程序和 C 程序之间进行一些数据传递，最简便的办法就是使用全局变量。具体的汇编程序中访问 C 程序变量的方法如下：

(1) 使用 IMPORT 伪操作声明该全局变量。

(2) 使用 LDR 指令读取该全局变量的内存地址，通常该全局变量的内存地址值存放在程序的数据缓冲池(literal pool)中。

(3) 根据该数据的类型，使用相应的 LDR/STR 指令读取/修改该全局变量的值。

下面通过一个例子来说明如何在汇编程序中访问 C 程序的全局变量。

```
AREA asmfile,CODE,READONLY          ;建立一个汇编程序段
EXPORT asmDouble                    ;声明可以被调用的汇编函数 asmDouble
IMPORT gVar_1                       ;调用 C 语言中声明的全局变量
asmDouble                           ;汇编子函数入口
LDR R0, = gVar_1                    ;将等于 gVar_1 地址的数据送入 R0 寄存器
LDR R1,[R0]                         ;将 R0 中的值为地址的数据送给 R1
MOV R2, #10                         ;将立即数 10 送给 R2
ADD R3, R1, R2                      ;R3 = R1 + R2,实现了 gVar_1 = gVar_1 + 10
STR R3,[R0]                         ;将 R3 中的数据送给 R0
MOV PC, LR                          ;子程序返回
END
```

3. 在 C 程序中调用汇编的函数

在 C 程序中调用汇编文件中的函数，主要工作有两个：一是在 C 中声明函数原型，并加 extern 关键字；二是在汇编中用 EXPORT 导出函数名，并用该函数名作为汇编代码段的标识，最后用“MOV PC,LR”返回。然后，就可以在 C 程序中使用该函数了。

下面是一个 C 程序调用汇编程序的例子，其中汇编程序 strcpy 实现字符串复制功能，C 程序调用 strcpy 完成字符串复制的工作。

```
/* C 程序 */
#include <stdio.h>
extern void asm_strcpy(const char *src, char *dest);  //声明可以被调用的函数
int main()                                            //C 语言主函数
{
    const char *s = "seasons in the sun";             //声明字符型指针变量
    char d[32];                                       //声明字符型数组
    asm_strcpy(s,d);                                  //调用汇编子函数
    printf("source: %s",s);                           //屏幕显示,s 的值
    printf("destination: %s",d);                      //屏幕显示,d 的值
    return 0;
}
/* 汇编程序 */
AREA asmfile,CODE,READONLY                            ;声明汇编语言程序段
EXPORT asm_strcpy                                     ;声明可被调用函数名称
asm_strcpy                                            ;函数入口地址
LOOP                                                  ;循环标志
LDRB R4, [R0], #1                                     ;R0 的地址中数字送给 R4 , 地址加 1 后
CMP R4, #0                                            ;比较 R4 是否为零
BEQ OVER                                              ;为零跳转到结束
STRB R4, [R1], #1                                     ;R4 的值送入 R1, R1 地址加 1
B LOOP                                                ;跳转到循环位置
OVER                                                  ;跳出标志位
```

```
MOV PC, LR        ;子函数返回
END
```

4. 在汇编程序中调用 C 程序的函数

在汇编中调用 C 程序的函数，需要在汇编程序中使用伪指令 IMPORT 声明将要调用的 C 函数。下面是一个汇编程序调用 C 程序的例子。其中在汇编程序中设置好各参数的值，本例有 3 个参数，分别使用寄存器 R0 存放第 1 个参数，R1 存放第 2 个参数，R2 存放第 3 个参数。

```
EXPORT asmfile                                    ;可被调用的汇编段
AREA asmfile,CODE,READONLY                        ;声明汇编程序段
IMPORT cFun                                       ;声明调用 C 语言的 cFun 函数
ENTRY                                             ;主程序起始入口
MOV R0, #11                                       ;将 11 放入 R0
MOV R1, #22                                       ;将 22 放入 R1
MOV R2, #33                                       ;将 33 放入 R2
BL cFun                                           ;调用 C 语言子函数
END
/*C 语言函数, 被汇编语言调用*/
int cFun(int a, int b, int c)                     //声明一个函数
{
    return a + b + c;                             //返回 a+b+c 的值
}
```

5.2 ADS 集成开发环境

ADS 全称是 ARM Developer Suite，是一款由 ARM 公司提供的专门用于 ARM 相关应用开发和调试的综合性软件。ADS 在易用性上比上一代的 SDT 开发环境有较大提高，是一套功能强大又易于使用的开发环境，成熟 ADS 包括一系列的应用，并有相关的文档和实例的支持。使用者可以用 ADS 来编写和调试各种基于 ARM 家族 RISC 处理器的应用，也可以使用 ADS 来编辑、编译、调试包括用 C、C++以及 ARM 汇编语言编写的程序。

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。如图 5-1 所示是 ADS 的组成结构图。

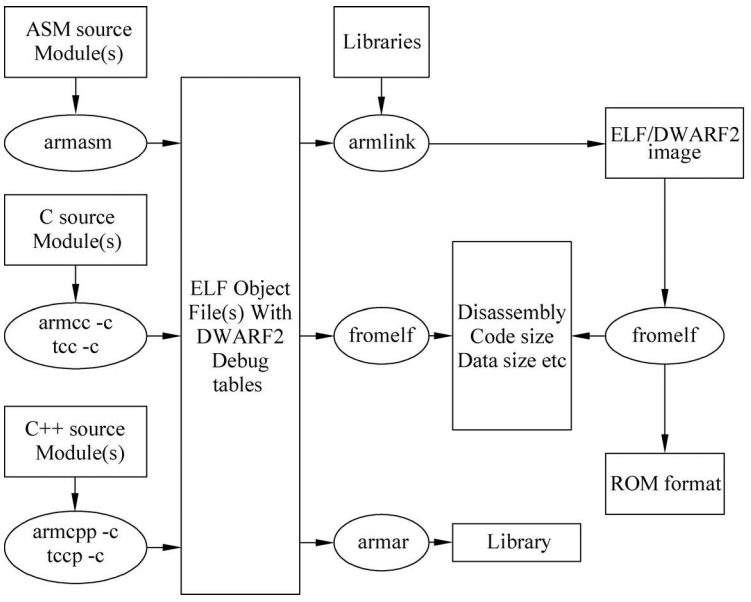

图 5-1 ADS 的组成结构图

图 5-1 ADS 组成结构图包括：

- ANSIC 编译器——armcc and tcc。
- ISO / Embedded C++编译器——armcpp and tcpp。
- ARM/Thumb 汇编器——armasm。
- 格式目标文件——ELF。
- ARM/Thumb 链接器——Linker-armlink。
- 格式转换器——fromelf。
- 库管理器——armar。
- C and C++库——Library 和 Libraries。
- 镜像文件——image。
- 烧写文件——ROM format。

另外，还包含 Windows 集成开发环境 CodeWarrior 和 ARM/Thumb 调试器 AXD Debugger、ARM Firmware Suite、ARM Application Library 和 RealMonitor 等组件。

在工程中接触最多最直接的就是 CodeWarrior 和 AXD Debugger 这两个组件。如图 5-2 所示是 CodeWarrior 的基本界面。在工程中通过在 CodeWarrior 下建立工程，进行编译和链接，最终生成二进制文件，接着在 AXD Debugger 下进行下载和调试仿真。这里没有对其他组件进行详细介绍并不意味着其他模块没有发挥作用或者不重要。

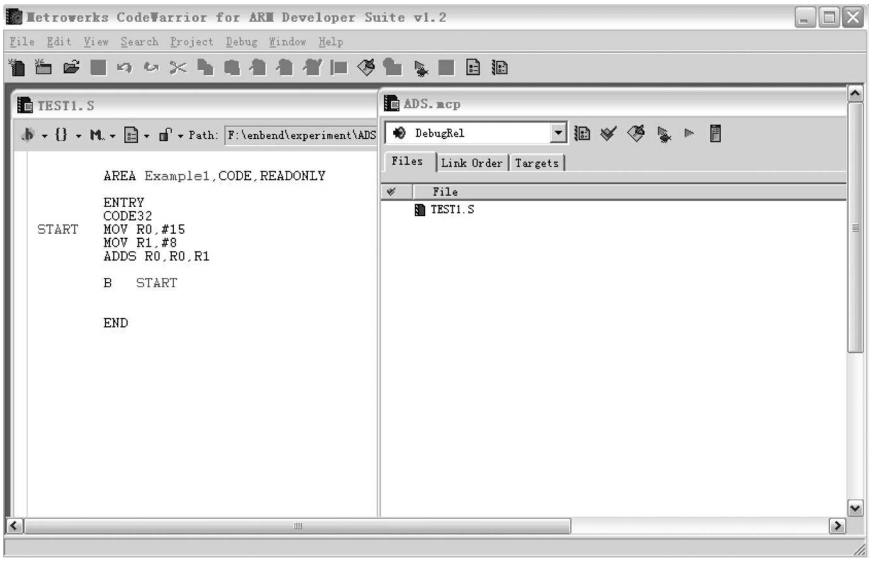

图 5-2　CodeWarrior 的基本界面

5.2.1　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，可加速并简化嵌入式开发过程中的每一个环节，使得开发人员只须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，开发人员无须离开 CodeWarrior 开发环境，因此节省了在操作工具转换上花费的时间，使开发人员有更多的精力投入到代码编写中。

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

编语言编写的程序代码。

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 中为 ARM 配置环境和参数，实现对工程代码的编译、汇编和链接。

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

1. 目标系统(Target system)

目标系统特指代码要运行的环境是基于 ARM 的硬件。例如，要为 ARM 开发板编写要运行的程序，这个开发板就是目标系统。

2. 生成目标(Build target)

生成目标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、库文件和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，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，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，用户可以利用字符串通配符，在多个文件中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不同文本文件的内容。

ADS 的 CodeWarrior IDE 是基于 Metrowerks CodeWarrior IDE 4.2 版本的，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 下配置各种 ARM 开发工具的能力，这样用户不用在命令控制台下就能够使用各种命令。

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 中，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。

熟悉 CodeWarrior IDE 的用户会发现，有许多功能已经从 CodeWarrior IDE For ARM 中移走，例如快速应用程序开发模板等。

在 CodeWarrior IDE For ARM 中有很多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。

包括 Processes、Expressions、Global Variable、Breakpoints、Registers。

(2) Project 菜单下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。

Debug 菜单中没有一个子菜单是可以使用的。

(4) Browser 菜单下不能使用的菜单选项。

包括 New Property、New Method 和 New Event Set。

(5) Help 菜单下不能用于 ADS 的菜单选项。

包括 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,此处不再赘述。

5.2.2　ADS 调试器

调试器本身是一种软件,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变量的查看、断点的控制等调试操作。

ADS 中包含以下 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。
- ADW/ADU(Application Debugger Windows/UNIX)——与老版本兼容的 Windows 或 UNIX 下的 ARM 调试工具。

下面对在调试映像文件中涉及的一些术语进行简单介绍。

1. 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,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时,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。

2. Debug agent

Debug agent 执行调试器发出的命令动作,例如,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,Debug agent 有以下几种方式: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,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。

后面的章节会结合具体实例为读者介绍如何使用 AXD 调试器。

5.3 ADS 使用入门

5.3.1 ADS 调试器的使用

1. 新建工程

通过选择“开始”→“所有程序”→ARM Developer Suite v1.2→CodeWarrior for ARM Developer Suite 命令打开开发软件，如图 5-3 所示。

图 5-3 打开开发软件

启动 Metrowerks CodeWarrior for ARM Developer Suite v1.2 后界面如图 5-4 所示。

图 5-4 启动后的界面

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，如图 5-5 所示；也可以在 File 菜单中选择 New 命令，如图 5-6 所示。

在打开的 New 对话框中有 Project、File 和 Object 这 3 个选项卡，现在新建工程，故选择 Project 选项卡。该选项卡中为用户提供了 7 种可选择的工程类型，如图 5-7 所示。

这里选择 ARM Executable Image 工程类型，在 Project name 文本框中输入工程名，如 ADS，单击 Location 文本框右侧的 Set 按钮，浏览该工程所要保存的路径。如存放在 F:\enbend\experiment\ADS\文件夹中，修改名称后，单击“确定”按钮即可建立一个新的名为 ADS 的工程，这时会出现 ADS.mcp 对话框，如图 5-8 所示。

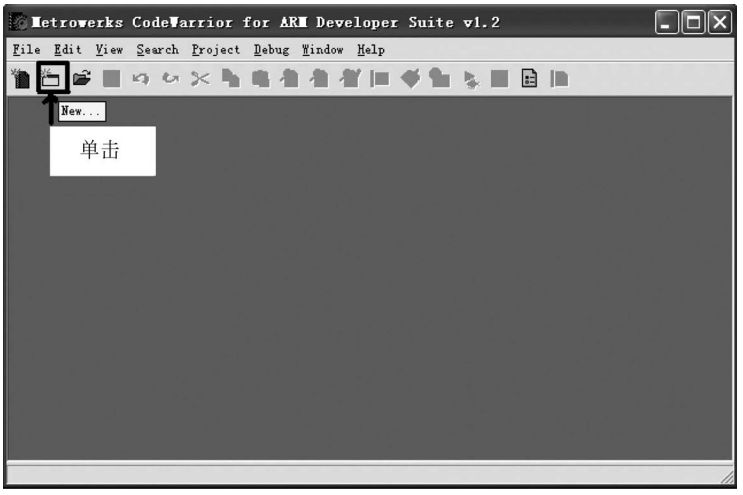

图 5-5　直接单击 New 按钮

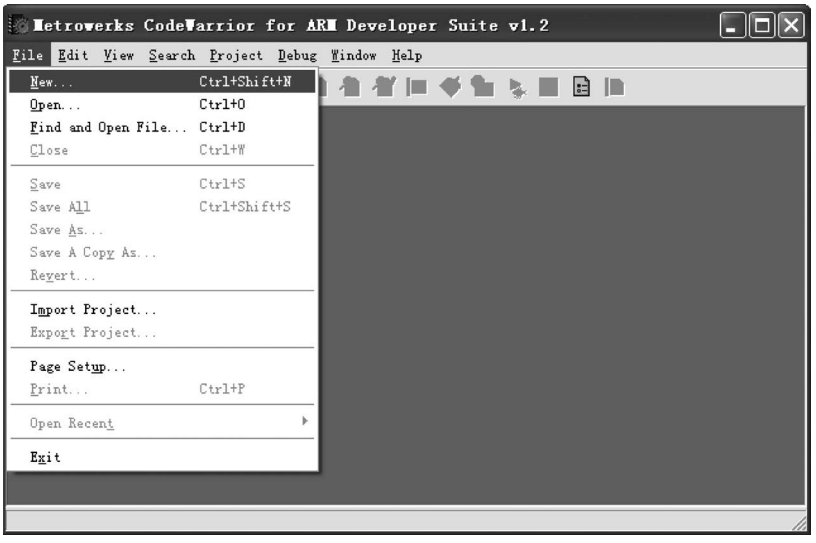

图 5-6　选择 File 菜单中的 New 命令

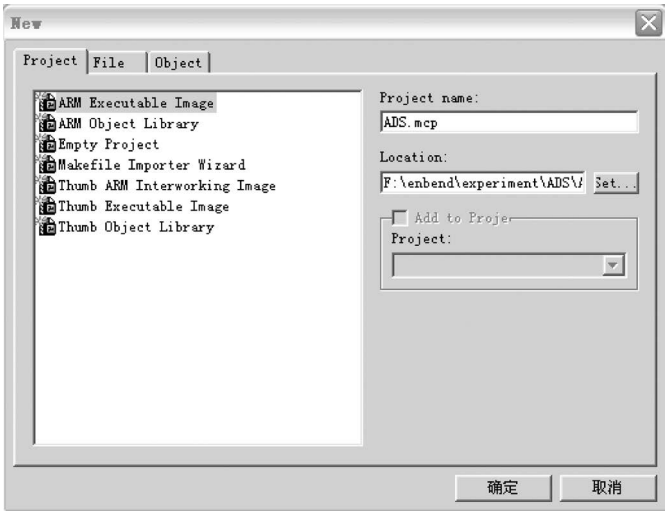

图 5-7　新建 ADS 工程

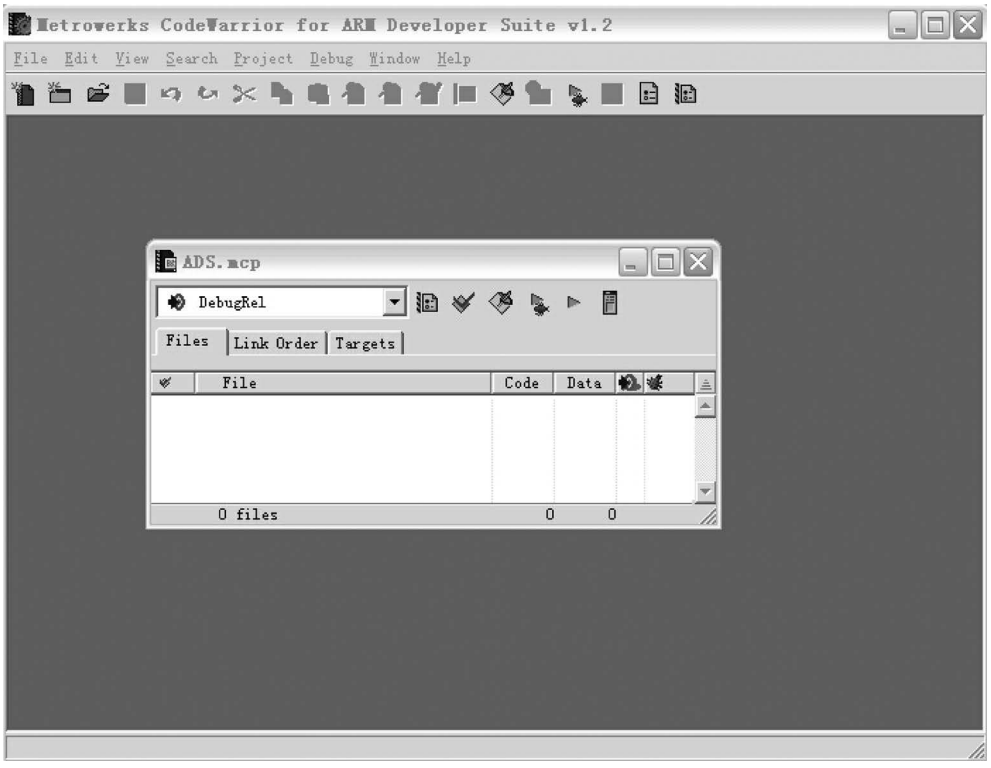

图 5-8　ADS. mcp 的窗口

此时单击“最大化”按钮可以将 ADS. mcp 窗口放大，如图 5-9 所示。

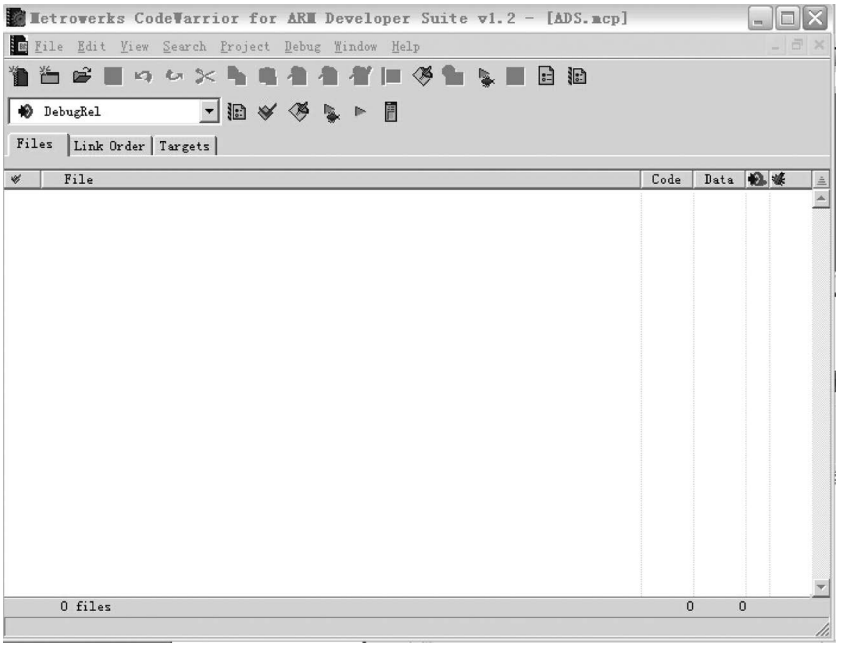

图 5-9　放大的 ADS. mcp 窗口

2. 设置目标及其参数

开发环境要经过设置才能与实验箱配套使用。在工具栏中有一个用于选择目标的下拉列表，如图 5-10 所示。新建工程的默认目标是 DebugRel，另外还有两个可选择的目标，分别是

Debug 和 Release，其含义分别如下：

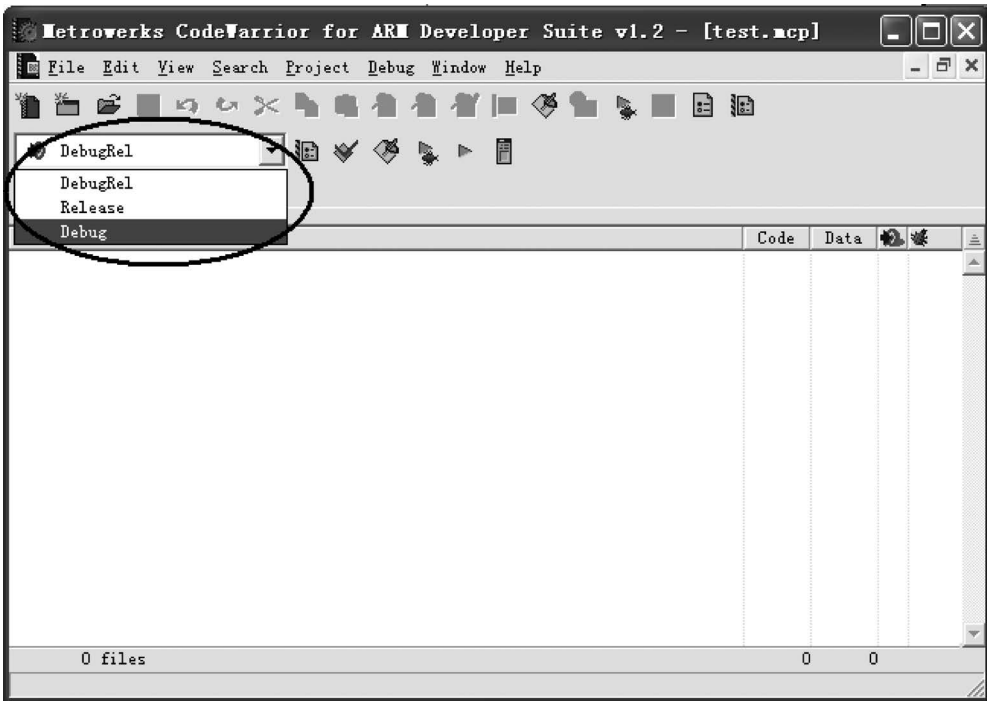

图 5-10　选择目标的下拉列表

- DebugRel——生成目标时，为每一个源文件生成调试信息。
- Release——生成目标时，不生成调试信息。
- Debug——生成目标时，为每一个源文件生成最完全的调试信息。

这里选择 Debug，接下来对 Debug 目标进行参数设置。单击工具栏上的 按钮或选择 Edit→Debug Settings 命令，如图 5-11 所示，打开 Debug Settings 对话框，如图 5-12 所示。

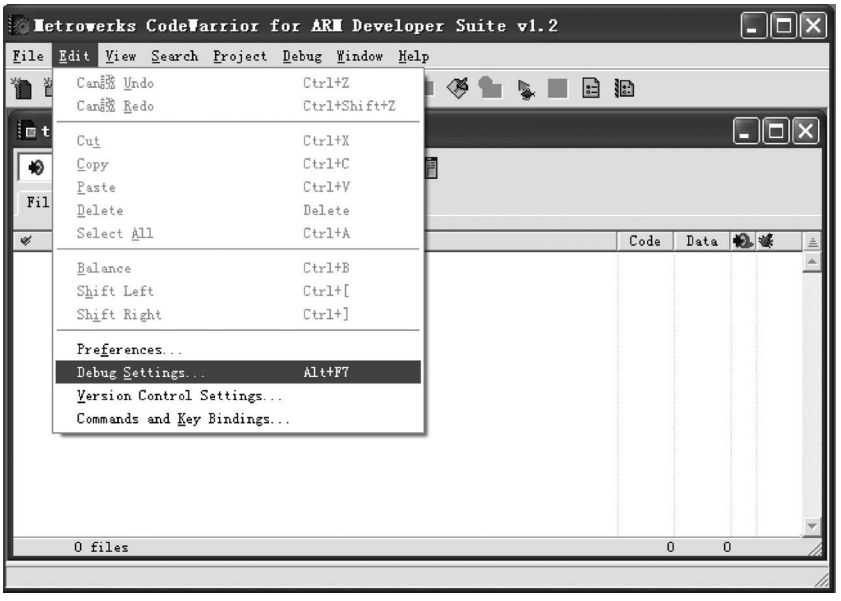

图 5-11　打开 Debug 目标的设置框

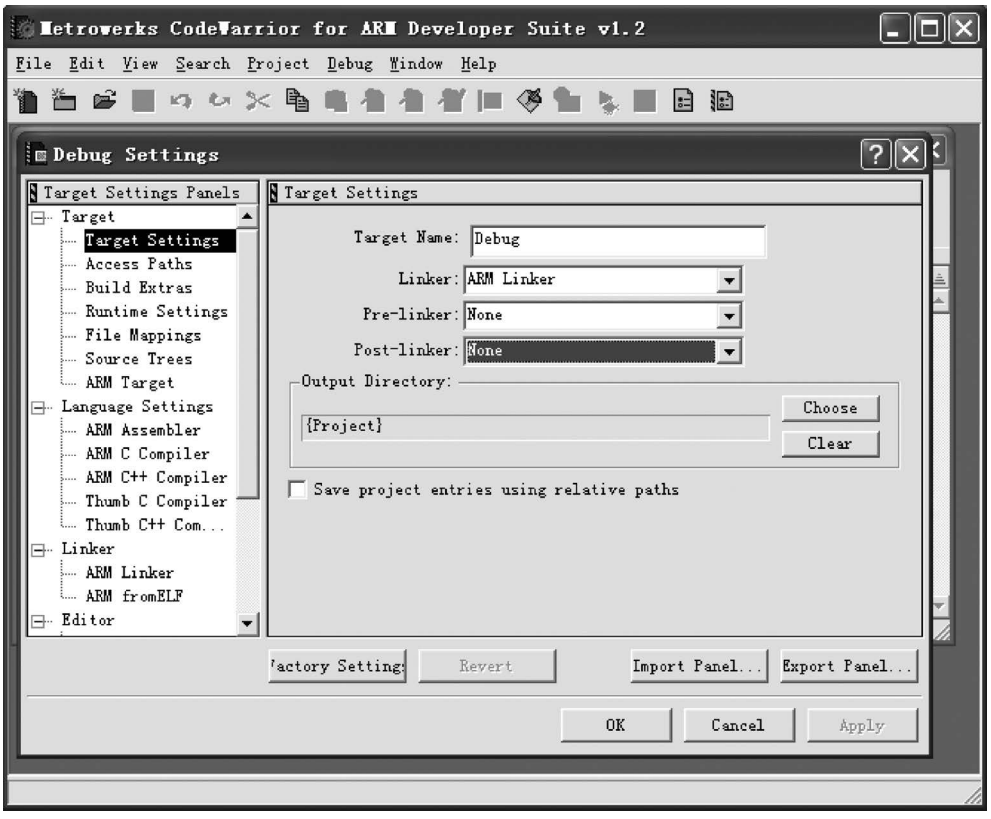

图 5-12　Debug Settings 对话框

在 Debug Settings 对话框中需要设置的内容比较多。设置方法是首先在左侧的树状目录中选中需要设置的对象，然后在右侧的面板中进行相应的设置。下面对经常使用的设置选项进行介绍。

1）目标设置(Target Settings)

在树状目录中选择 Target→Target Settings 选项，在右侧面板的 Post-linker 下拉列表框中选择 ARM fromELF 选项，使得工程链接后通过 fromELF 产生二进制代码，使其可以写到 ROM 中，如图 5-13 所示。

2）语言设置(Language Settings)

开发语言有汇编、C、C++及其混合语言等。在开发前要对其设置，这里主要是对其硬件(架构或处理器)的支持设置，因为实验是在采用 TMS320DM365 处理器的实验箱中进行的，所以具体设置方法是先选中树状目录中 Language Settings 下的开发语言，然后在本语言对应的右侧面板的 Architecture or Processor 下拉列表框中选择 ARM926EJ-S 选项，其他选项保持默认设置。注意，在开发中用到的语言都要进行类似设置。汇编语言的设置过程如图 5-14 所示，其他语言设置方法与此一样。

3）链接器设置(Linker)

在左侧的树状目录中选中 Linker→ARM Linker 选项，出现链接器的设置对话框，此处的设置很重要，下面详细介绍部分选项卡的设置方法。

(1) Output 选项卡(如图 5-15 所示)。

其中，Linktype 选项区域中为链接器提供了 3 种链接类型。

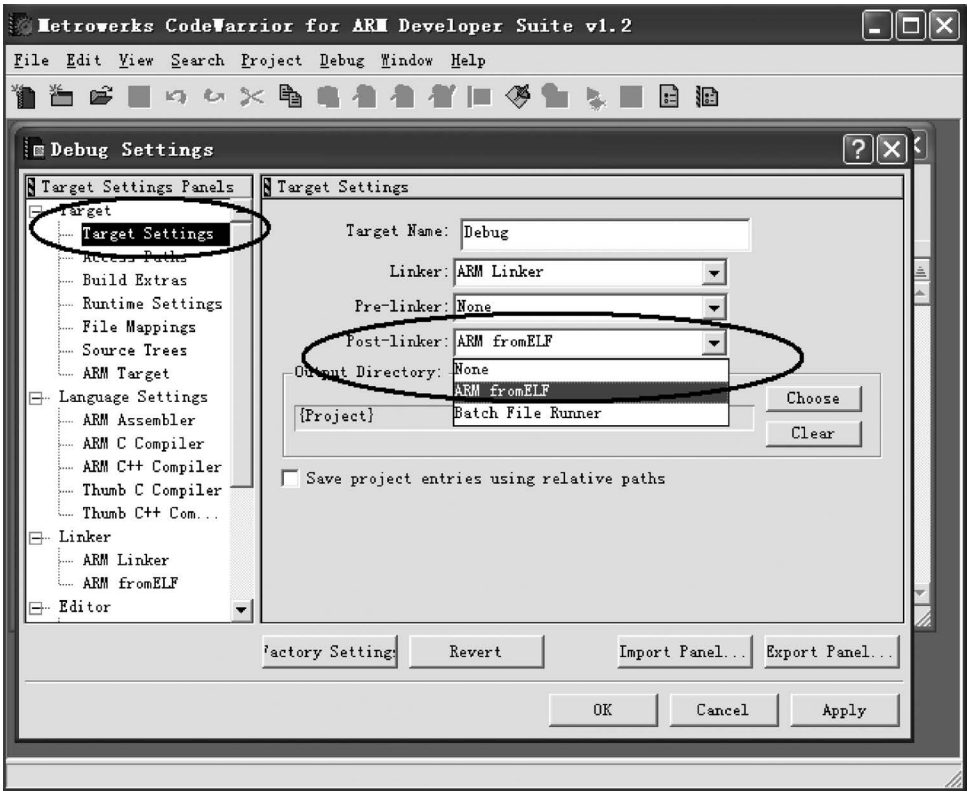

图 5-13　目标设置

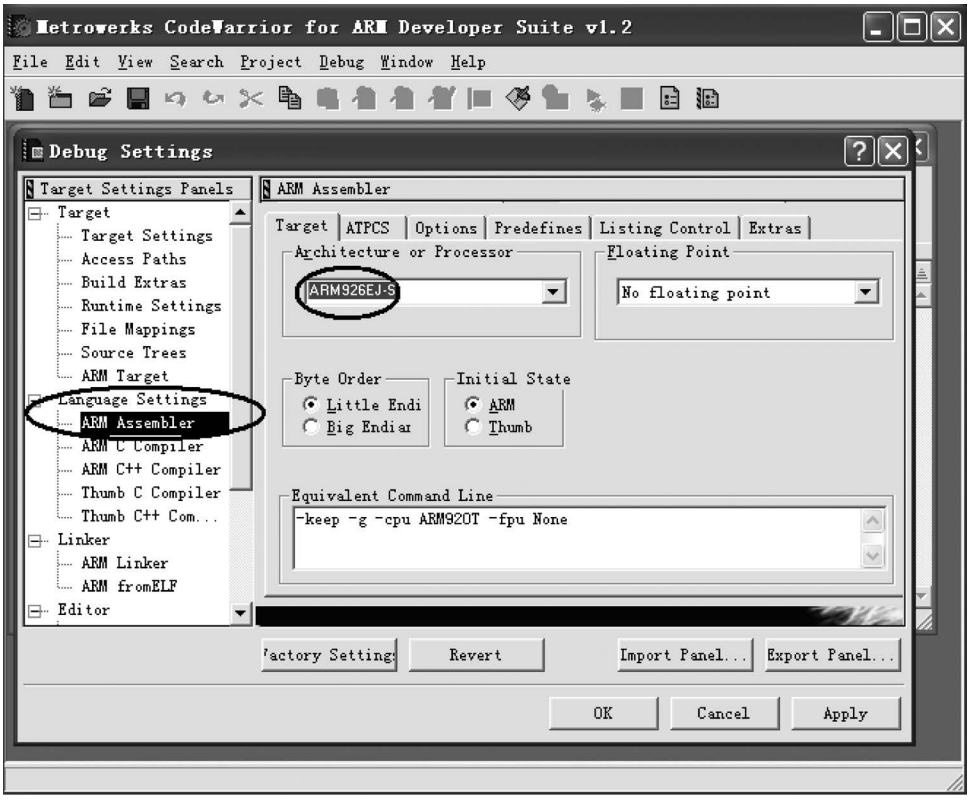

图 5-14　开发语言设置

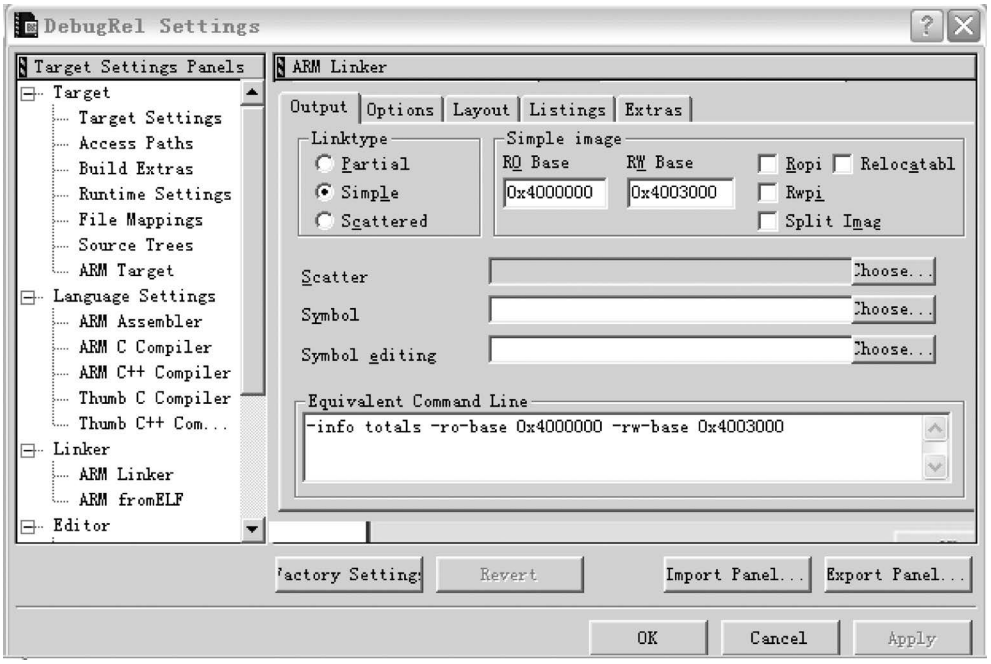

图 5-15　链接器的设置对话框

① Partial：表示链接器只进行部分链接，链接后的目标文件可以作为以后进一步链接的输入文件。

② Simple：表示链接器将生成简单的 ELF 格式的映像文件，地址映射关系在 Simple image 选项区域中设置。

③ Scattered：表示链接器将生成复杂的 ELF 格式的映像文件，地址映射关系在 Scatter 格式的文件中指定。这里选择常用的 Simple 类型。选择 Simple 后，在其右侧 Simple image 选项区域中包含 RO Base 和 RW Base 两个文本框。

- RO Base：用来设置程序代码存放的起始地址。
- RW Base：用来设置程序数据存放的起始地址。

这两项的地址均由硬件决定，并应该在 SDRAM 的地址范围内。本实验箱使用的是 32MB×8 的 SDRAM，其地址范围是 0x4000000～0x4FFFFFF，故采用首地址作为程序代码存放的首地址，即在 RO Base 文本框中输入 0x4000000；RW Base 文本框可由用户自定义，只要保证在 SDRAM 地址空间内，并且是字对齐即可，这里可以输入 0x4003000。

此处的设置表示在地址为 0x4000000～0x4003000 的范围是只读区域，用来存放程序代码，从 0x4003000 开始用来存放程序数据。

（2）Options 选项卡（如图 5-16 所示）。

此处 Options 选项卡只对 Image entry point 进行设置，该项是程序代码的入口地址。如果程序在 SDRAM 中运行，那么针对本实验箱可选择的地址范围为 0x4000000～0x4FFFFFF。通常程序代码的入口地址与 RO Base 中程序代码的首地址相同，这里为 0x4000000。其他选项保持默认设置即可。

（3）Layout 选项卡（如图 5-17 所示）。

该选项卡在链接方式为 Simple 时有效，用来安排一些输入段在映像文件中的位置。即在 Place at beginning of image 选项区域的 Object/Symbol 文本框内填写启动程序的目标文件名

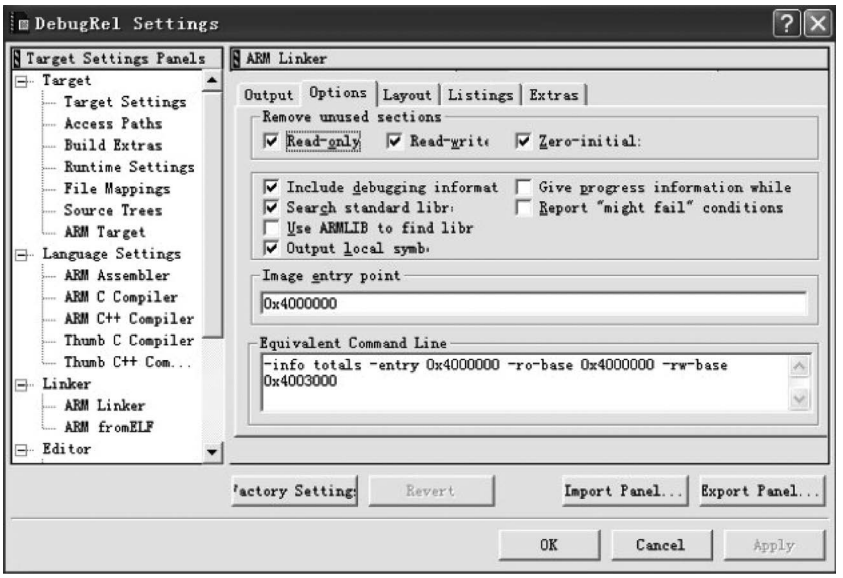

图 5-16　Options 选项卡

Startup.o；在 Section 文本框中填写程序入口起始段的标号 Start，其作用是通知编译器，整个项目从该段开始执行。

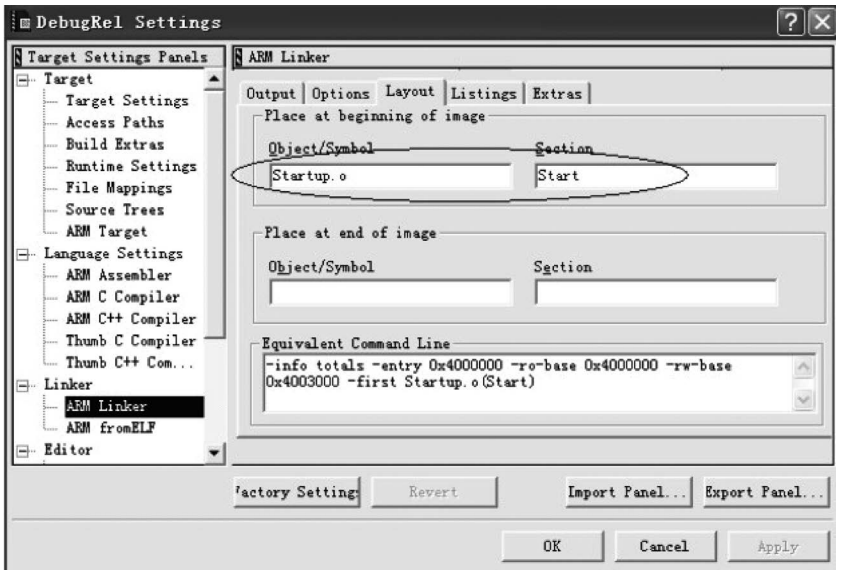

图 5-17　Layout 选项卡

如果希望将编译后生成的二进制文件放到指定文件夹，可以在左侧的树状目录中选择 Linker→ARM fromELF 选项进行设置，如图 5-18 所示。若未见此项，将默认在工程目录下生成二进制文件。该二进制文件可用于以后下载到 Flash(实验箱等硬件)中执行。

至此，对 Debug Settings 对话框的设置基本完成，单击 Apply 按钮，再单击 OK 按钮，保存设置。

3. 向工程中添加源文件

工程创建、设置好以后就会出现 ADS.mcp 的窗口，该窗口包含 Files、Link Order 和 Targets 这 3 个选项卡，默认情况下显示的是 Files 选项卡，此时可以通过选择 Project→Add

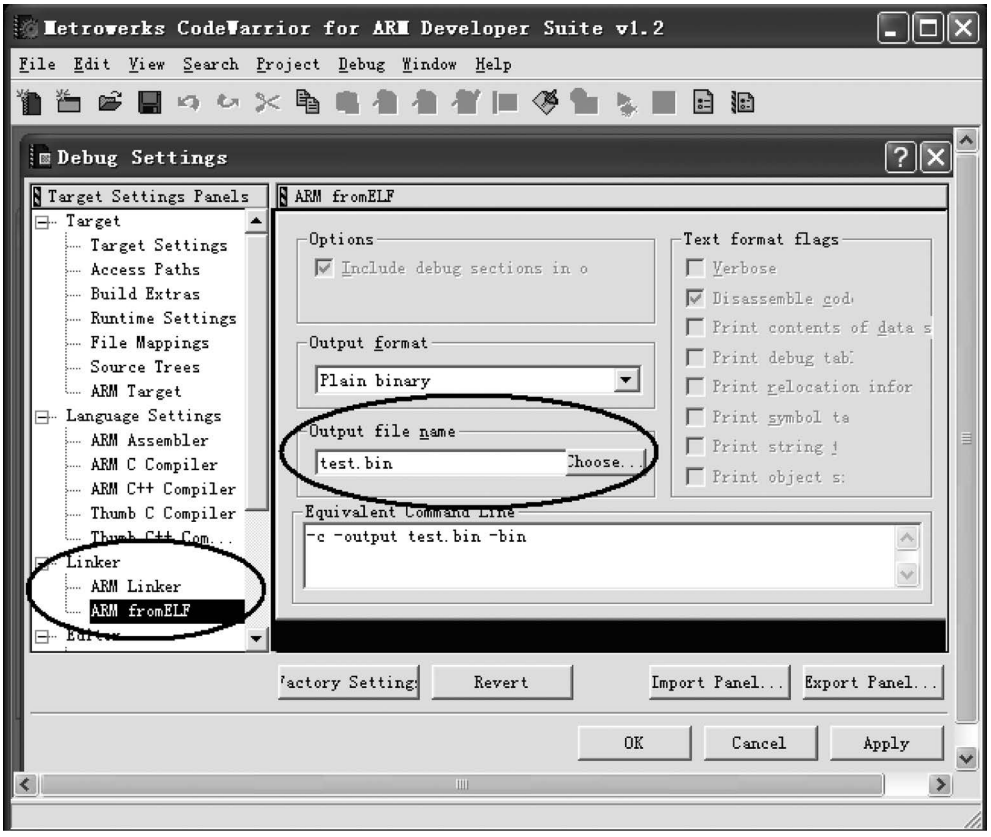

图 5-18　Linker→ARM fromELF 的设置

Files 命令把与工程有关的所有源文件添加到该工程，如图 5-19 所示，或者通过在空白处右击，在弹出的快捷菜单中选择 Add Files 命令来完成，如图 5-20 所示。

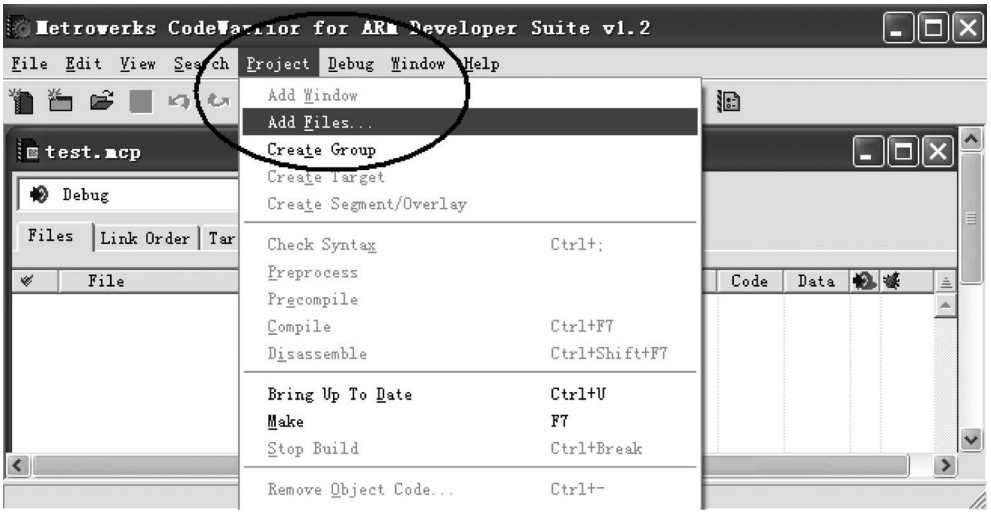

图 5-19　选择 Project→Add Files 命令向工程添加源文件

当没有源文件可用时，首先需要新建源文件。这里以新建文件类型为汇编语言的 TEST1.S 文件为例说明一下过程。选择 File→New 命令，如图 5-21 所示。在弹出的对话框中选择 Files 选项卡；在 File name 文本框中输入新建文件的文件名 TEST1.S(注意：文件名后缀与要使用的开发语言种类有关，如用 C 语言开发时文件扩展名为.c，汇编语言开发时文

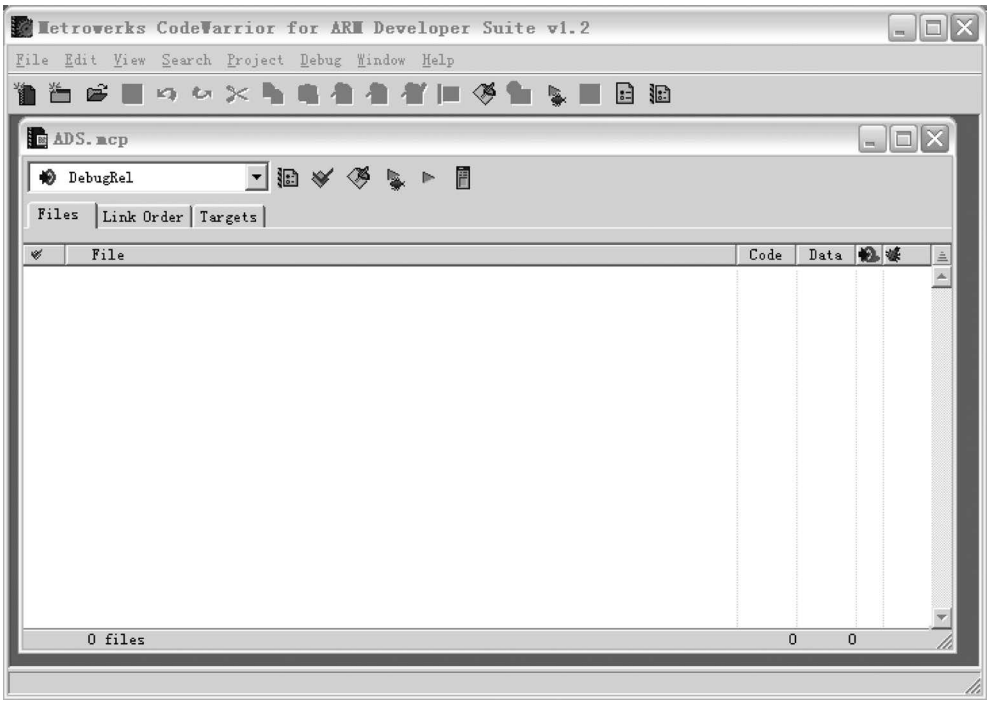

图 5-20　通过快捷菜单添加源文件

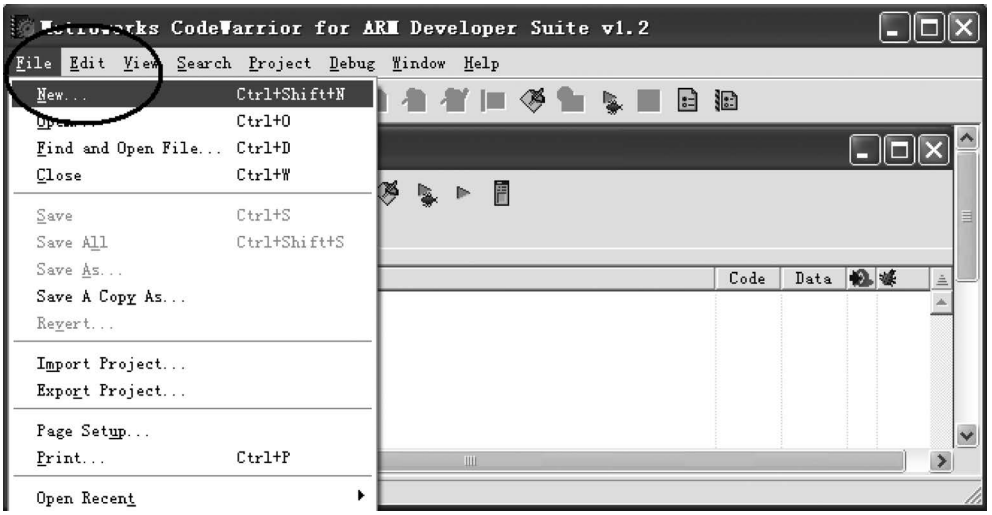

图 5-21　新建源文件

件扩展名为.s)；在 Location 文本框中输入文件的保存位置 D:\ARM\experiment\ADS；选中 Add to Project 复选框；在 Project 下拉列表框中选择将文件添加到的工程 ADS.mcp；在 Targets 选项区域中选中文件要添加的目标，过程如图 5-22 所示。单击“确定”按钮即可将新建的文件添加到工程中，文件添加到工程后的窗口如图 5-23 所示。接下来只需在新建文件中进行编码、保存即可。

工程创建好以后，对其进行编译和链接。选择 Metrowerks CodeWarrior for ARM Developer Suite v1.2 窗口中的 Project→Make 命令或单击 按钮来完成编译和链接。如果有错误或警告，则根据提示更改程序，窗口如图 5-24 所示。

如果没有语法错误，则将在工程所在目录下生成一个名为“工程名_data”的文件夹。如本

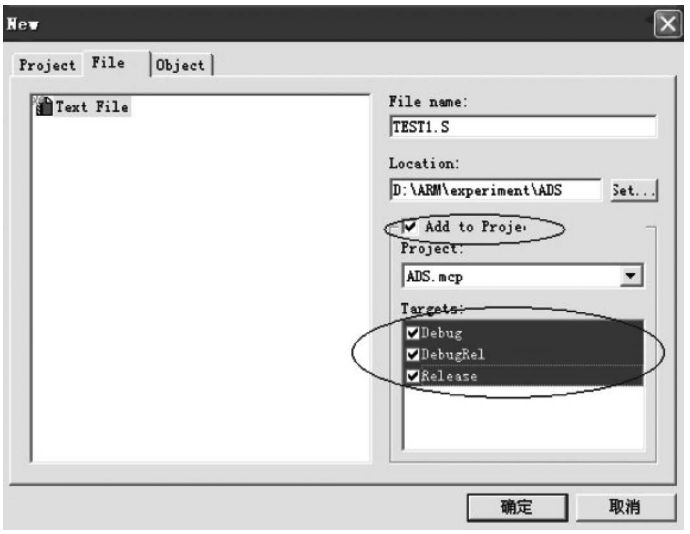

图 5-22　添加到工程

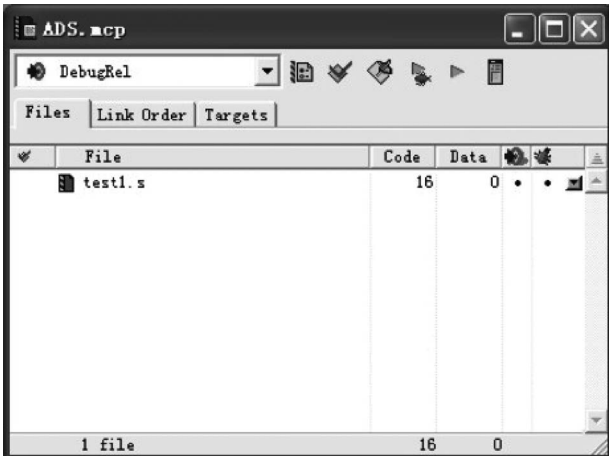

图 5-23　文件添加后的窗口

图 5-24　编译、链接后产生的警告

例的工程名为 ADS.mcp，生成的文件夹名为 ADS_data。在该文件夹下，针对不同类型的目标将生成多个文件夹。本例中由于使用的是 Debug 目标，因此生成的最终文件都在 Debug 文件夹下。进入 Debug 文件夹会看到编译、链接后生成的映像文件（xxx.axf）和二进制文件（xxx.bin）。映像文件用于调试，二进制文件用于烧写到 Flash 中运行。

5.3.2　ADS 1.2 环境下工程的仿真、调试及配置方法

在“开始”菜单中选择“所有程序”→ARM Developer Suite v1.2→AXD Debugger 命令打开调试软件，如图 5-25 所示。

图 5-25　打开调试软件

如果程序代码没有错误或警告，则可以在 Metrowerks CodeWarrior for ARM Developer Suite v1.2 窗口中选择 Project→Debug 命令或单击 按钮或工程窗口的 按钮来直接调出 AXD 调试窗口，如图 5-26 和图 5-27 所示。

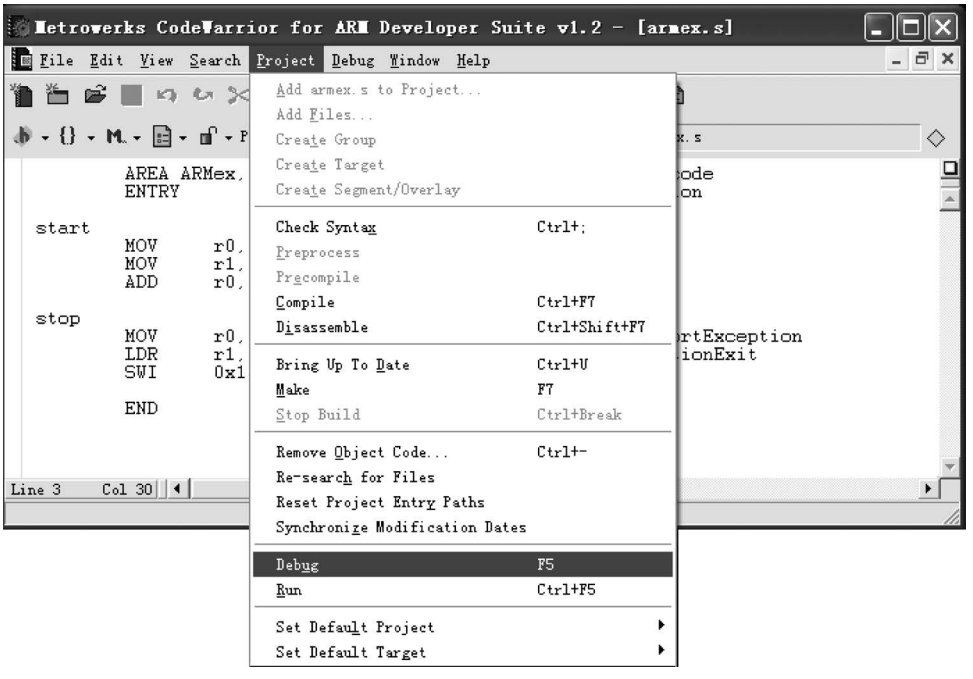

图 5-26　直接调出 AXD 调试窗口方法 1

AXD 调试窗口如图 5-28 所示。

第一次使用需要对 AXD 进行配置，具体方法如下：

初次运行 AXD，左侧的目标平台为 ARM7TDMI。实验箱采用的 CPU 为 ARM920，所以

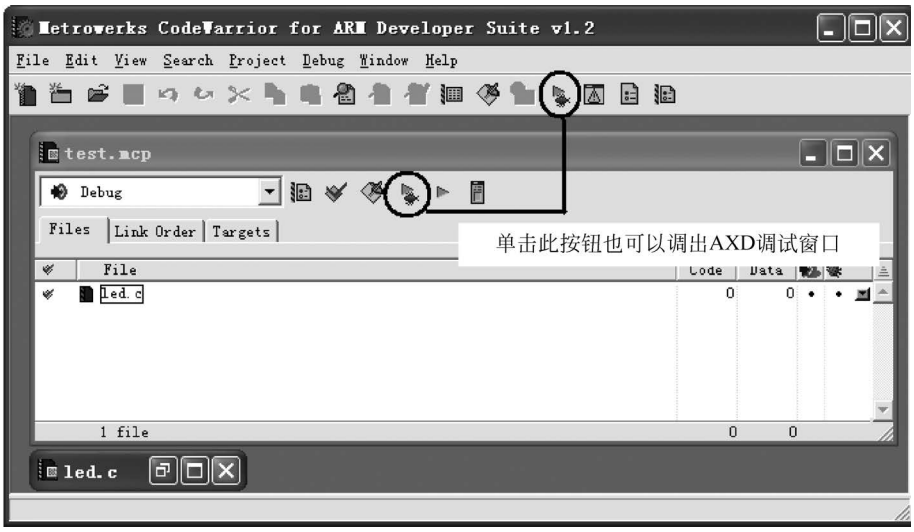

图 5-27　直接调出 AXD 调试窗口方法 2

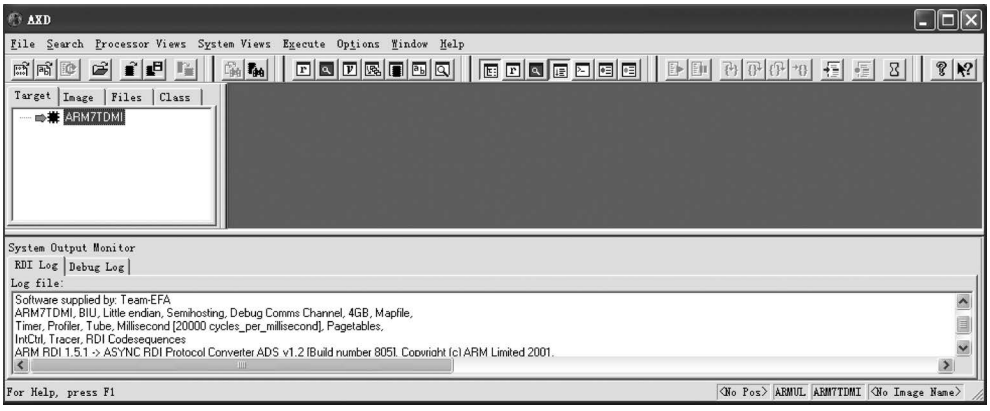

图 5-28　AXD 调试窗口

需要配置 AXD 使之匹配。方法为在 AXD 窗口中选择 Options→Configure Target 命令，如图 5-29 所示。

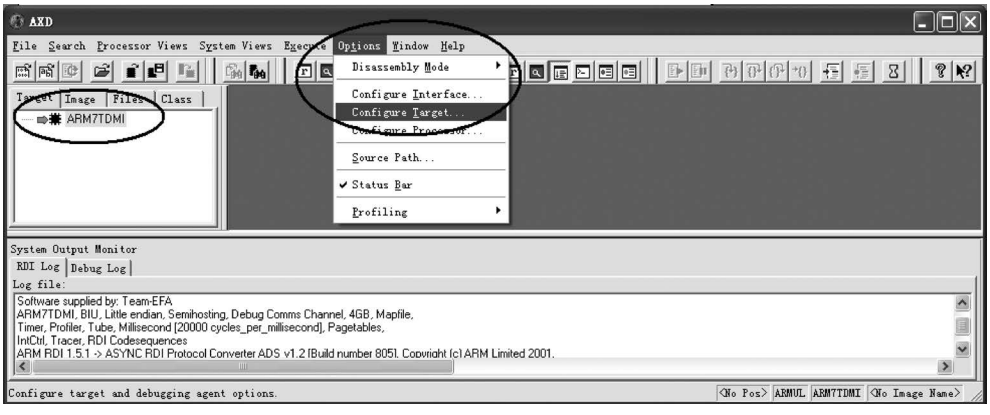

图 5-29　AXD 的配置

上述操作将调出 Choose Target 对话框，如图 5-30 所示。在该对话框中，Target 栏代表不同的目标 CPU。ADP 和 ARMUL 是默认的设置。此处选择 ARMUL，表示使用软件仿真，

因为此时 PC 可以不连接任何目标板，ARM 系统中 CPU 的行为完全由软件模拟。

要设置 CPU 类型需双击 ARMUL，然后在弹出的对话框中的 Processor 区域选择 Variant 下拉列表框，选择 ARM920T，然后单击 OK 按钮，再单击 Choose Target 对话框中的 OK 按钮即可。设置过程如图 5-31 所示。

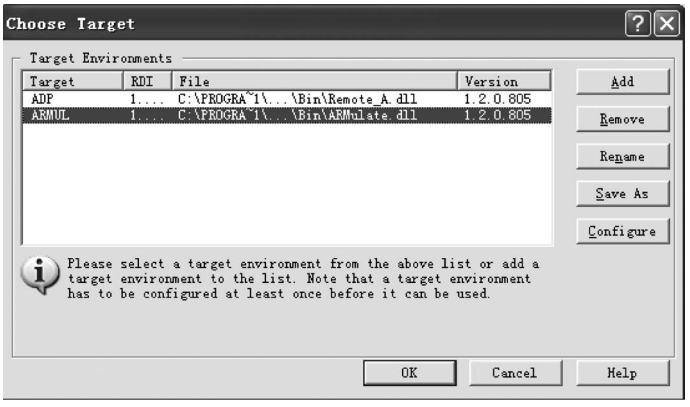

图 5-30　Choose Target 对话框

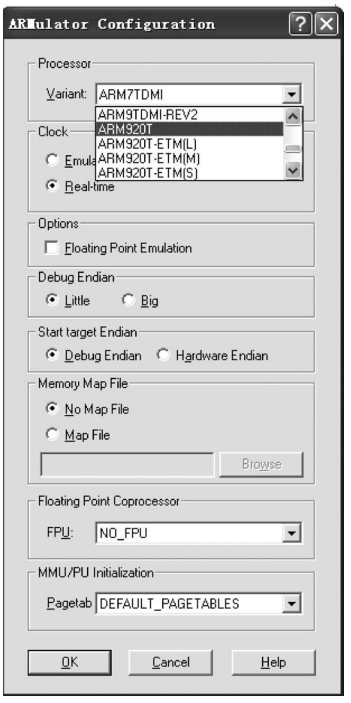

图 5-31　CPU 的设置

设置好的 AXD 界面左侧会显示 ARM920T。现在可以向 AXD 调试软件中添加工程的映像文件，方法为选择 AXD 窗口中的 File→Load Image 命令，选择要加载的映像文件(扩展名为.axf)，如图 5-32 所示。

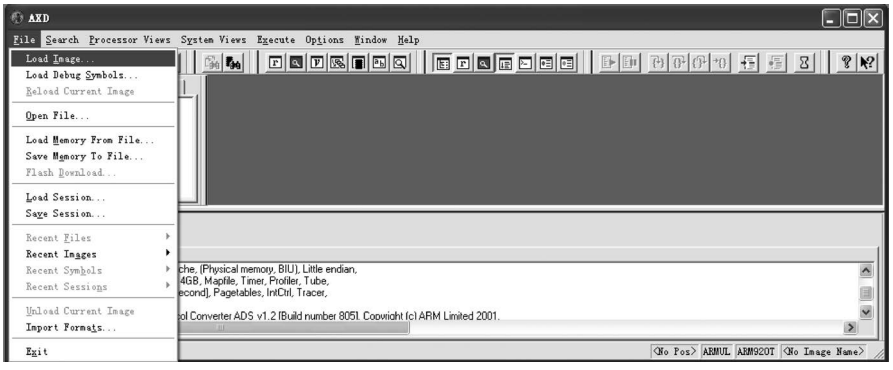

图 5-32　向 AXD 中添加工程的映像文件

加载完映像文件就可以对程序代码进行调试了。下面介绍 AXD 界面的一些常用工具和窗口，如图 5-33 所示。

1. 文件操作工具条

部分按钮作用介绍如下：

——加载调试文件。

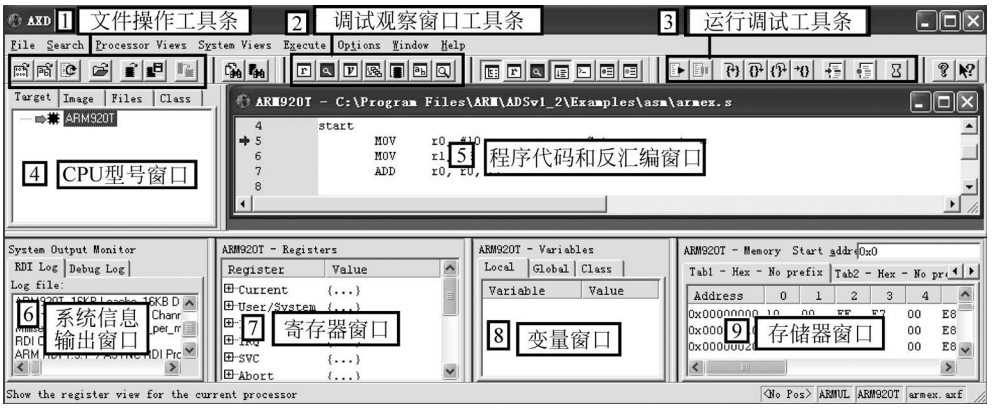

图 5-33　AXD 的常用工具和窗口

——重新加载文件。

2. 调试观察窗口工具条

部分按钮作用介绍如下：

——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。

3. 运行调试工具条

部分按钮作用介绍如下：

——全速运行(GO)，直到结束或断点停止。

——停止运行(Stop)。

——单步运行，遇到函数调用则转入函数内部。

——单步运行，遇到函数调用则不进入函数内部。

——单步运行，从被调函数中返回。

——运行到光标处停止。

——设置或取消断点。

4. CPU 型号窗口

显示当前目标运行 CPU 的型号。

5. 程序代码和反汇编窗口

显示当前调整程序代码和反汇编代码。该窗口可是在调试时，实时显示调试的代码位置。

6. 系统信息输出窗口

显示程序运行过程中输出的提示信息或错误信息。可以通过选择 System Views→Output 命令设置为显示或隐藏。

7. 寄存器窗口

用于查看和修改 CPU 中各寄存器的值。在不同模式下，不同窗口对应不同的寄存器。通过双击寄存器的值可对其进行修改。可以通过选择 Processor Views→Registers 命令设置为显示或隐藏。

8. 变量窗口

用于查看程序运行过程中各变量值的变化。可以通过选择 Processor Views→Variables 命令设置为显示或隐藏。

9. 存储器窗口

用于查看相应存储器地址中的数据。用户可以输入地址，查看相应地址内的数据，如果输入地址是无效的，则显示错误的数据。可以通过选择 Processor Views→Memory 命令设置为显示或隐藏。

5.4　JTAG 介绍

JTAG(Joint Test Action Group，联合测试行动小组)是一种国际标准测试协议(IEEE 1149.1 兼容)，主要用于芯片内部测试。现在多数的高级器件都支持 JTAG 协议，如 DSP、FPGA 等。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO，分别为模式选择、时钟、数据输入和数据输出线。

JTAG 最初是用来对芯片进行测试的，基本原理是在器件内部定义一个 TAP (Test Access Port，测试访问口)，通过专用的 JTAG 测试工具对内部节点进行测试。JTAG 测试允许多个器件通过 JTAG 接口串联在一起，形成一个 JTAG 链，能实现对各个器件分别测试。现在，JTAG 接口还常用于实现 ISP(In-System Programmable，在线编程)，对 Flash 等器件进行编程。

目前在 ARM 调试系统中普遍采用的方式是通过 JTAG 接口调试目标板系统。由于 JTAG 调试的目标程序是直接在目标板上执行，因此仿真更接近于目标硬件，且 JTAG 调试功能较强大。在现实应用中，基于 JTAG 的 ARM 调试工具主要有：

- 简单的 JTAG 电缆。

通过协议转换软件来实现调试，虽然电缆制作简单，软件免费，但是调试速度慢，而且软件检错能力不够完善。

- 基于 JTAG 的在线仿真器。

采用的是 ICD 技术，利用串口、并口等实现与主机之间的通信。这类仿真器虽然仿真速度快，但是价格比较贵。

- 全功能在线仿真器。

采用仿真头完全取代目标板上的 CPU 来实现完全仿真 ARM 芯片行为。这类仿真器的调试和检错功能十分强大，价格也最贵，一般是为某一固定速度的处理器专门开发的，速度是没有办法扩展的。

如图 5-34 所示是实验箱 JTAG 接口示意图，该图主要是根据 Multi-ICE 的接口来定义的。

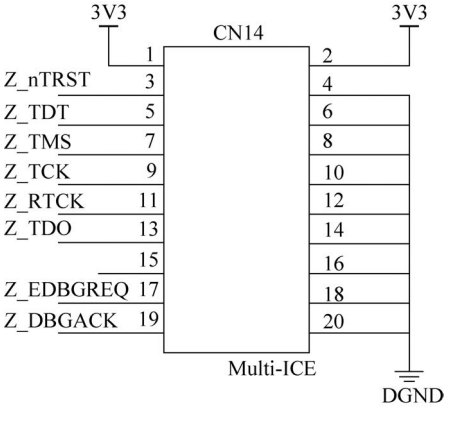

图 5-34　实验箱 JTAG 接口示意图

5.5　Multi-ICE 仿真器

Multi-ICE 是 ARM 公司自己的 JTAG 在线仿真器。Multi-ICE 的 JTAG 链时钟可以设置为 5kHz～10MHz，JTAG 操作的一些简单逻辑由 FPGA 实现，使得并行口的通信量最小，

以提高系统的性能。Multi-ICE 硬件支持低至 1V 的电压。Multi-ICE 2.1 还可以外部供电，不需要消耗目标系统的电源，这对调试类似手机等便携式、电池供电设备是很重要的。

Multi-ICE 2.x 支持该公司的实时调试工具 MultiTrace。MultiTrace 包含一个处理器，因此可以跟踪触发点前后的轨迹，并且可以在不终止后台任务的同时对前台任务进行调试，在微处理器运行时改变存储器的内容，所有这些特性使延时降到最低。

Multi-ICE 2.x 支持 ARM7、ARM9、ARM9E、ARM10 和 Intel Xscale 微结构系列，通过 TAP 控制器串联，提供多个 ARM 处理器以及混合结构芯片的片上调试，还支持低频或变频设计以及超低压核的调试，并且支持实时调试。

Multi-ICE 提供支持 Windows NT 4.0、Windows 95/ 98/2000/ME、HPUX 10.20 和 Solaris V2.6/7.0 的驱动程序。

如图 5-35 所示是 Multi-ICE 运行示意图。将实验箱与 Multi-ICE 正确连接后，通上电，双击 Multi-ICE Server 图标，如果能够显示出 FA5/FA6，则说明 Multi-ICE 已经与实验箱确立了连接关系，接下来就能够用 ADS 进行裸机调试了。

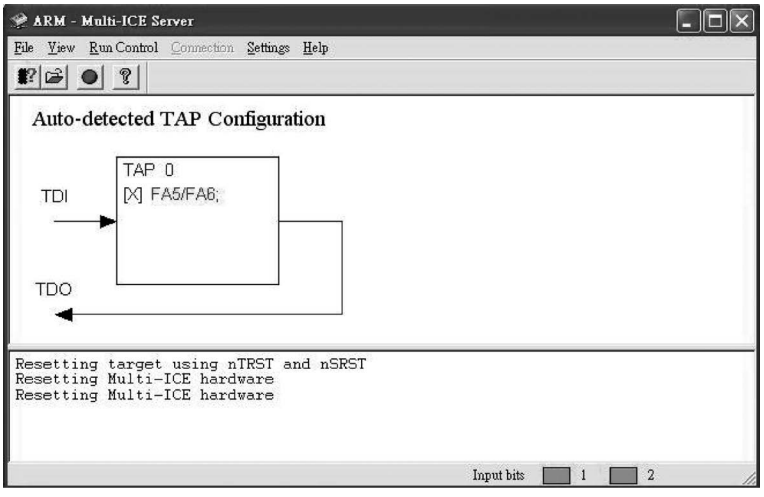

图 5-35　Multi-ICE 运行示意图

本章习题

一、ARM 中汇编语言和 C 语言混合编程方法

1. 汇编语言的源程序由哪几部分组成？
2. ARM(Thumb)汇编程序所支持的变量形式有几种？
3. 如何定义全局数字变量、全局逻辑变量、全局字符串变量？如何赋值？
4. 如何定义局部数字变量、局部逻辑变量、局部字符串变量？如何赋值？
5. 列举变量代换符“$”的作用。
6. 说明下面的数字为哪种进制数？

0X11A,2356,&042,2_01101111,8_54231067,'A',0x41

7. 数字常量怎么定义？
8. 认识算术运算符、逻辑运算符、关系运算符，指出下面的一段代码中所用的运算符。

```
MOV R5,#0xFF00:MOD:0xF:ROL:2;
IF R5:LAND:R6 <= R7;
```

```
MOV R0,#0x00;
ELSE
MOV R0,#0xF
```

9. 认识汇编程序基本结构，请说明以下汇编代码的含义。

```
AREA example,CODE,READONLY ;
ENTRY ;
Start
MOV R0, #40 ;
MOV R1,#16 ;
ADD R2,R0,R1 ;
MOV R0,#0x18 ;
LDR R1, = 0x20026 ;
SWI 0x123456 ;
END
```

10. 认识子程序调用方法，请说明以下汇编代码的含义。

```
AREA Init,CODE,READONLY ;
ENTRY ;
LOOP1 MOV R0,#412 ;
MOV R1,# 106 ;
MOV R2,# 64 ;
MOV R3,# 195 ;
BL SUB1 ;
MOV R0,#0x18 ;
LDR R1, = 0x20026;
SWI 0x123456 ;
SUB1 SUB R0,R0,R1 ;
SUB R0,R0,R2 ;
SUB R0,R0,R3 ;
MOV PC,LR ;
END
```

11. 以下是C语言内嵌汇编的例子，请说明代码的含义。

```
#include < stdio.h >
void my_strcpy(const char * src, char * dest)
{
    char ch;
    __asm
    {
    LOOP;
    LDRB CH,[SRC],#1 ;
    STRB CH,[dest],#1 ;
    CMP CH, #0;
    BNE LOOP;
    }
}
int main() ;
{
    char * a = "forget it and move on!";
    char b[64];
    my_strcpy(a, b);
    printf("original: %s", a);
    printf("copyed: %s", b);
    return 0;
}
```

12. 以下是汇编调用C中的全局变量的例子，其中gVar_1为C程序中定义的全局变量，请说明代码的含义。

```
AREA asmfile,CODE,READONLY ;
EXPORT asmDouble ;
IMPORT gVar_1 ;
asmDouble ;
LDR R0, = gVar_1 ;
LDR R1,[R0] ;
MOV R2, #10 ;
ADD R3, R1, R2 ;
STR R3,[R0] ;
MOV PC, LR ;
END
```

13. 以下是 C 语言调用汇编函数的例子,请说明代码的含义。

```
#include < stdio.h >
extern void asm_strcpy(const char * src, char * dest);
int main()
{
    const char * s = "seasons in the sun";
    char d[32];
    asm_strcpy(s,d);
    printf("source: %s",s);
    printf(" destination: %s",d);
    return 0;
}
AREA asmfile,CODE,READONLY ;
EXPORT asm_strcpy ;
asm_strcpy ;
LOOP ;
LDRB R4, [R0], #1 ;
CMP R4, #0 ;
BEQ OVER ;
STRB R4, [R1], #1 ;
B LOOP ;
OVER ;
MOV PC, LR ;
END
```

14. 以下是汇编语言调用 C 函数的例子,请说明代码的含义。

```
EXPORT asmfile ;
AREA asmfile,CODE,READONLY;
IMPORT cFun ;
ENTRY ;
MOV R0, #11 ;
MOV R1, #22 ;
MOV R2, #33 ;
BL cFun;
END
/ * C 语言函数, 被汇编语言调用 * /
int cFun(int a, int b, int c);
{
    return a + b + c;
}
```

二、ADS 集成开发软件

1. ADS 的英文全称是什么?
2. ADS 软件支持的编译器一般有几种? 请列举。
3. ADS 中可生成多少种目标文件? 请列举,并说明它们的作用。
4. ADS 软件建立工程时,对应的 3 种目标输出分别是 DebugRel、Debug、Release,它们代

表什么含义？

5. Project 下的 Make 菜单项主要实现什么功能？

6. ADS 软件创建工程时，需要对哪几项关键的部分进行设置？

7. 对 Link 进行设置时，需要对哪几项关键的部分进行设置？

8. fromELF 实现什么功能？

9. 在 ARM 系统中，主机和目标板的连接接口一般有哪几种？