Hi3510 HiBoot 移植应用

Application Notes

文档版本	02
发布日期	2006-12-20
BOM编码	N/A

深圳市海思半导体有限公司

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为电气生产中心　邮编：518129

网址：　http://www.hisilicon.com

客户服务电话：　0755-28788858

客户服务传真：　0755-28788838

客户服务邮箱：　support@hisilicon.com

商标声明

 、Hisilicon、海思，均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本升级或其它原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

插图目录

表格目录

前 言

概述

本节介绍本文档的内容、对应的产品版本、适用的读者对象、行文表达约定、历史修订记录等。

产品版本

与本文档相对应的产品版本如下所示。

产品名称	产品版本
Hi3510 通信媒体处理器芯片（简称 Hi3510）	Hi3510 V100
	Hi3510 V101
	Hi3510 V110
Hi3510 DVS 方案	Hi3510 DMS V100R001
Hi3510 VPhone 方案	Hi3510 DMS V200R001

读者对象

本指南为程序员和系统工程师而编写，描述了基于 Hi3510 媒体解决方案平台（Digital Media Solution，简称 DMS）如何进行移植开发。使用本书的程序员应该：

- 掌握相关硬件基本知识
- 熟练掌握 C 语言
- 掌握基本的 Linux 环境编程

内容简介

本文描述在 Hi3510 VSEVB 评估板上如何移植 HiBoot（Hi3510 媒体解决方案平台中的 Bootloader）和烧写 HiBoot 的相关操作及工具使用。内容组织如下：

章节	内容
1 概述	简单描述 Hi3510 VSEVB 评估板的 Bootloader－－HiBoot。

章节	内容
2　目录结构	介绍 HiBoot 代码结构和 HiBoot 目录的描述。
3　移植 HiBoot	介绍如何在 Hi3510 VSEVB 移植 HiBoot，主要内容包括存储器的配置、Flash 驱动、网口 PHY 芯片驱动的修改。
4　烧写 HiBoot	介绍如何使用 Mutil-ICE 和 ADS 工具将新移植的 HiBoot 烧写到单板的 Flash 中及在 ADS 环境中烧写工具和系统初始化脚本的修改。
5　如何使用 ARM 调试工具	介绍如何使用 Mutil-ICE 和 ADS 工具，包括 Magic-ICE 仿真器、Mutil-ICE Server、AXD Debugger。

约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
警告	以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
注意	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

通用格式约定

格式	说明
宋体	正文采用宋体表示。
黑体	一级、二级、三级标题采用黑体。

格式	说明
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。
“Terminal Display”格式	“Terminal Display”格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修改日期	版本	修改说明
2006-08-31	01（第一次版本发布）	-

1 概述

HiBoot 为 Hi3510 VSEVB（Video Solution Evaluation Board）评估板的 Bootloader，当选用与评估板不同的外围芯片时，需要修改待移植单板对应目录中相关硬件的驱动代码，主要包括存储器配置、Flash 驱动、网口 PHY 芯片驱动。本文详细介绍 HiBoot 的移植过程，同时叙述了 HiBoot 的烧写以及 ARM 相关调试工具的使用。

2 HiBoot 目录结构

HiBoot 的主要目录结构如表 2-1 所示，详细目录说明请阅读 HiBoot 目录下的 README 文档。

表2-1 HiBoot 主要目录结构

目录名	描述
cpu	各种 CPU 相关代码，HiBoot 入口代码
board	各种单板相关代码，主要内容包括内存和 Flash 驱动等
board/hisilicon	海思评估板相关代码
lib_xxx	各种体系结构，如 ARM、MIPS 的通用代码
include	头文件
include/configs	各种单板的配置文件
common	各种功能（命令）代码
drivers	网口、串口等驱动代码
drivers/hisilicon	海思芯片部分模块驱动代码
net	网络协议代码
fs	文件系统代码

3 移植 HiBoot

本章描述如何在待移植单板上移植 HiBoot。

在 Hi3510 VSEVB 评估板上所选用的外围芯片型号如下：

- SDRAM：MT48LC16M16A2
- DDR SDRAM：HY5DU121622CTP-J
- Flash：S29GL256N10TF01
- PHY：RTL8201CP

如果待移植单板上选用的外围芯片不是以上型号时，需要适当修改 HiBoot，单板才能正常运行。

3.1 创建目录和文件

创建待移植单板相关代码的目录和文件时，不用手动编写所有代码，只需要拷贝参考代码到待移植单板对应目录中即可。在本节的示例中待移植单板对应的目录是 board/hisilicon/myboard，其中 myboard 目录为新建目录，用于存放待移植单板相关代码。

拷贝参考代码

在 Hi3510 VSSDK 中，Hiboot 代码位于 code/linux/boot/u-boot-1.1.4 目录中，因此需要将 board/hisilicon/hi3510-v100-p01-dvs01 目录下的代码拷贝到目录 board/hisilicon/myboard 中，操作如下：

```
#cd code/linux/boot/u-boot-1.1.4/
#cp -a board/hisilicon/hi3510-v100-p01-dvs01 board/hisilicon/myboard  //拷贝 Hi3510dvs 代码到目录 myboard
```

增加新的配置文件

在/include/configs/目录下增加新的单板配置文件，操作如下：

```
#cp include/configs/hi3510-v100-p01-dvs01.h include/configs/myboard.h
```

修改 Makefile

修改 Makefile 增加单板配置的编译选项。

在 Hiboot 代码目录下的 hisilicon.mk 文件中增加配置操作如下：

```
myboard_config:   unconfig
            @./mkconfig $(@:_config=) arm arm926ejs myboard hisilicon
```

完成以上所有操作后，HiBoot 新的移植代码结构就已经完成，剩下的工作就是对驱动代码做相应的修改和配置。

3.2 配置存储器

3.2.1 移植代码

存储器配置文件为 myboad 目录下的 lowlevel_init.S，其主要功能完成系统控制、SDRAM 和 MEMC、DDR SDRAM 和 DDRC、Flash 的配置。当选用不同的 SDRAM 和 DDR SDRAM 时，其对应的电气特性、模式寄存器等需要相应的修改。在 HiBoot 中，将需要配置的参数以宏定义的形式在单板配置文件中进行定义，即在 myboard.h 文件有如下宏定义：

```
/*-----------------------------------------------------------------------
 * Physical Memory Configuration
 */
#define CONFIG_NR_DRAM_BANKS  1          //定义SDRAM的BANK数
#define PHYS_SDRAM_1         0x60000000   //定义SDRAM的物理地址
#define PHYS_SDRAM_1_SIZE     0x04000000   //定义SDRAM的大小
#define SDRAM_CONFIG_VALUE    0x4680       //配置MEMC Dynamic Config寄存器
#define SDRAM_MODE           0x60040000   //定义SDRAM模式寄存器
#define FLASH_CONFIG_VALUE    0x81         //配置MEMC Static Config寄存器

#define DDRC_CONFIG_VALUE     0x00000884   //配置DDRC Dynamic Config寄存器
#define DDR_EXTEND_MODE       0xf0001000   //配置DDR扩展模式寄存器
#define DDR_MODE_REGISTER1    0xf0242000   //定义DDR模式寄存器1
#define DDR_MODE_REGISTER2    0xf0042000   //定义DDR模式寄存器2
```

以上的配置值为 Hi3510 VSEVB 评估板的配置。移植代码只需要根据选用芯片正确配置以上宏定义即可，各个宏定义的详细配置方法在下文中详细说明。

3.2.2 配置 MEMC 和 SDRAM

MEMC 和 SDRAM 配置宏定义

1. CONFIG_NR_DRAM_BANKS

定义 SDRAM 的 BANK 数，设置为 1。

2. PHYS_SDRAM_1

定义 SDRAM 的起始物理地址，设置 0x60000000。

3. PHYS_SDRAM_1_SIZE

定义 SDRAM 的大小，根据实际选用芯片大小设定。

4. SDRAM_CONFIG_VALUE

定义 MEMCDynamicConfig0 寄存器的配置值，根据实际选用芯片型号设定，请参见“设置 SDRAM_CONFIG_VALUE”。

5. SDRAM_MODE

定义 SDRAM 模式寄存器，根据实际选用芯片型号设定，请参见“设置 SDRAM 模式寄存器”。

6. FLASH_CONFIG_VALUE

定义 MEMCStaticConfig0 寄存器的配置值。当 Flash 的总线宽度为 8 位时设置为 0x80；当总线宽度为 16 位时设置为 0x81，当总线宽度为 32 位时设置为 0x82。

设置 SDRAM_CONFIG_VALUE

1. MEMCDynamicConfig0 寄存器定义

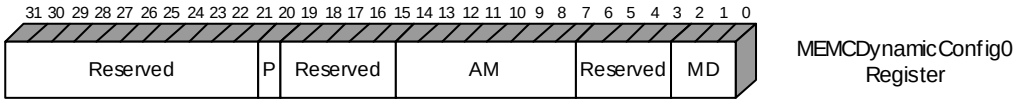


表3-1 MEMCDynamicConfig0 寄存器

位	名称	复位值	描述
[31:21]	Reserved	11’h0	保留位。
[20]	P	1’b0	写保护。 0：没有写保护； 1：有写保护。
[19:15]	Reserved	5’h0	保留位。
[14:7]	AM	8’h0	地址映射。
[6:3]	Reserved	4’h0	保留位。
[2:0]	MD	3’b0	存储器设备类型。 只能写下述值： 000：SDR SDRAM； 010：low-power SDR SDRAM。

表 3-1 中“地址映射”根据表 3-2、表 3-3、表 3-4、表 3-5 配置。

表3-2 16bit 外部总线、High performance SDRAM 地址映射（Bank，Row，Column）

[14]	[13:12]	[11:9]	[8:7]	描述
0	00	000	00	16Mbit (2M×8)，2Banks，Row length=11，Column length=9
0	00	000	01	16Mbit (1M×16)，2Banks，Row length=11，Column length=8
0	00	001	00	64Mbit (8M×8)，4Banks，Row length=12，Column length=9
0	00	001	01	64Mbit (4M×16)，4Banks，Row length=12，Column length=8
0	00	010	00	128Mbit (16M×8)，4Banks，Row length=12，Column length=10
0	00	010	01	128Mbit (8M×16)，4Banks，Row length=12，Column length=9
0	00	011	00	256Mbit (32M×8)，4Banks，Row length=13，Column length=10
0	00	011	01	256Mbit (16M×16)，4Banks，Row length=13，Column length=9
0	00	100	00	512Mbit (64M×8)，4Banks，Row length=13，Column length=11
0	00	100	01	512Mbit (32M×16)，4Banks，Row length=13，Column length=10

表3-3 16bit 外部总线、Low-power SDRAM 地址映射（Bank，Row，Column）

[14]	[13:12]	[11:9]	[8:7]	描述
0	01	000	00	16Mbit (2M×8)，2Banks，Row length=11，Column length=9
0	01	000	01	16Mbit (1M×16)，2Banks，Row length=11，Column length=8
0	01	001	00	64Mbit (8M×8)，4Banks，Row length=12，Column length=9
0	01	001	01	64Mbit (4M×16)，4Banks，Row length=12，Column length=8
0	01	010	00	128Mbit (16M×8)，4Banks，Row length=12，Column length=10
0	01	010	01	128Mbit (8M×16)，4Banks，Row length=12，Column length=9
0	01	011	00	256Mbit (32M×8)，4Banks，Row length=13，Column length=10
0	01	011	01	256Mbit (16M×16)，4Banks，Row length=13，Column length=9
0	01	100	00	512Mbit (64M×8)，4Banks，Row length=13，Column length=11
0	01	100	01	512Mbit (32M×16)，4Banks，Row length=13，Column length=10

表3-4 32bit 外部总线、High performance SDRAM 地址映射（Bank，Row，Column）

[14]	[13:12]	[11:9]	[8:7]	描述
1	00	000	00	16Mbit (2M×8)，2Banks，Row length=11，Column length=9
1	00	000	01	16Mbit (1M×16)，2Banks，Row length=11，Column length=8
1	00	001	00	64Mbit (8M×8)，4Banks，Row length=12，Column length=9

[14]	[13:12]	[11:9]	[8:7]	描述
1	00	001	01	64Mbit (4M×16)，4Banks，Row length=12，Column length=8
1	00	010	00	128Mbit (16M×8)，4Banks，Row length=12，Column length=10
1	00	010	01	128Mbit (8M×16)，4Banks，Row length=12，Column length=9
1	00	011	00	256Mbit (32M×8)，4Banks，Row length=13，Column length=10
1	00	011	01	256Mbit (16M×16)，4Banks，Row length=13，Column length=9
1	00	100	00	512Mbit (64M×8)，4Banks，Row length=13，Column length=11
1	00	100	01	512Mbit (32M×16)，4Banks，Row length=13，Column length=10

表3-5 32bit 外部总线、Low-power SDRAM 地址映射（Bank，Row，Column）

[14]	[13:12]	[11:9]	[8:7]	描述
1	01	000	00	16Mbit (2M×8)，2Banks，Row length=11，Column length=9
1	01	000	01	16Mbit (1M×16)，2Banks，Row length=11，Column length=8
1	01	001	00	64Mbit (8M×8)，4Banks，Row length=12，Column length=9
1	01	001	01	64Mbit (4M×16)，4Banks，Row length=12，Column length=8
1	01	010	00	128Mbit (16M×8)，4Banks，Row length=12，Column length=10
1	01	010	01	128Mbit (8M×16)，4Banks，Row length=12，Column length=9
1	01	011	00	256Mbit (32M×8)，4Banks，Row length=13，Column length=10
1	01	011	01	256Mbit (16M×16)，4Banks，Row length=13，Column length=9
1	01	100	00	512Mbit (64M×8)，4Banks，Row length=13，Column length=11
1	01	100	01	512Mbit (32M×16)，4Banks，Row length=13，Column length=10

2. 配置示例

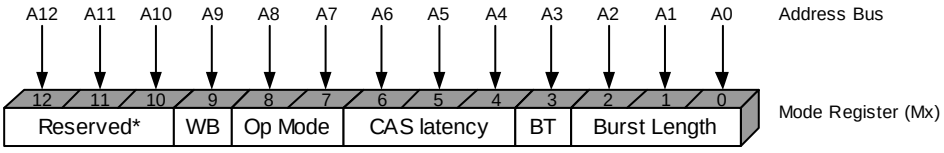
在 Hi3510 VSEVB 评估板中选用 2 片 16 位宽的 32MB 的 SDRAM，配置为 High performance 模式，组成 32bit 外部总线。因此根据表 3-4 选择“256Mbit(16M×16)，4Banks，Row length=13，Column length=9”，即配置 MEMC Dynamic Config0 寄存器的 bit[14:7] = 10001101。因此最终 MEMC Dynamic Config0 寄存器的配置值为 0x4680。

设置 SDRAM 模式寄存器

1. SDRAM 模式寄存器定义

Hi3510 VSEVB 评估板选用 Micron 公司的 MT48LC32M16A2 型号 SDRAM，本文针对 MT48LC32M16A2 型号的 SDRAM 介绍 SDRAM 模式寄存器，关于 MT48LC32M16A2 的详细内容请参考该芯片的相关资料。如果采用其他型号的 SDRAM 存储芯片，请参考对应型号的 SDRAM 相关资料。

SDRAM_MODE 定义 SDRAM 模式控制。SDRAM 模式采用直接读取 SDRAM 固定地址的办法来进行控制，具体的地址值根据芯片模式寄存器进行计算。



*Should program M12, M11, M10 = "0, 0, 0"
to ensure compatibility with future devices.

表3-6 Burst Length 的定义

M2 M1 M0	Burst Length	
	M3 = 0	M3 = 1
0 0 0	1	1
0 0 1	2	2
0 1 0	4	4
0 1 1	8	8
1 0 0	Reserved	Reserved
1 0 1	Reserved	Reserved
1 1 0	Reserved	Reserved
1 1 1	Full Page	Reserved

表3-7 BT 的定义

M3	Burst type
0	Sequential
1	Interleaved

表3-8 CAS latency 的定义

M6 M4 M5	CAS latency
0 0 0	Reserved
0 0 1	Reserved
0 1 0	2
0 1 1	3
1 0 0	Reserved

M6 M4 M5	CAS latency
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

表3-9 Op Mode 的定义

M8	M7	M6～M0	Operating Mode
0	0	Defined	Standard operation
-	-	-	All other states reserved

表3-10 WB 的定义

M9	Write Burst Mode
0	Programmed burst length
1	Single location access

2. 配置示例

Hi3510 VSEVB 评估板的 SDRAM 采用标准操作模式，CAS 延迟 2 个时钟周期，Burst 方式为 Sequential 且 Burst 长度为 1。读写地址时按照 Row、Bank、Column 格式组合地址数据（其中 Row、Bank、Column 的顺序根据不同型号的芯片会有所不同，详细描述请参考《Hi3510V100 通信媒体处理器芯片 用户指南》）。

Row(12:0)	Bank(1:0)	Column(8:0)	补充位

地址填充在 Row 位置，根据表 3-2 选择“256Mbit(16M×16)，4Banks，Row length=13，Column length=9”，补充位根据芯片位数而定。如果使用 32 位存储器，补充位为“00”；如果使用 16 位存储器，补充位为“0”。

如果使用 32 位地址，高位采用 0x6000，即 SDRAM 起始物理地址。由此可见，应该通过读取存储器 0x60040000 地址来进行配置寄存器的配置。

3.2.3 配置 DDRC 和 DDR SDRAM

DDRC 和 DDR SDRAM 配置宏定义

修改 DDR 存储器的配置与 SDRAM 类似，地址结果也相同。不同的是，DDR 需要修改扩展模式寄存器。模式寄存器需配置 2 次，以完成对 DDR 内部的 DLL 初始化。

1. DDRC_CONFIG_VALUE

定义 DDRC_CONFIG_VALUE 寄存器的配置值，根据实际选用芯片型号设定，请参见“设置 DDRC_CONFIG_VALUE”

2. DDR_EXTEND_MODE

定义 DDR SDRAM 扩展模式寄存器，根据实际选用芯片型号设定，请参见“设置 DDR_EXTEND_MODE”。

3. DDR_MODE_REGISTER1 和 DDR_MODE_REGISTER2

定义 DDR SDRAM 模式寄存器，根据实际选用芯片型号设定，需要配置 2 个值，请参见“设置 DDR SDRAM 模式寄存器”。

设置 DDRC_CONFIG_VALUE

1. DDRC_CONFIG_VALUE 寄存器定义

Hi3510 的 DDRC 只支持 16 位 DDR SDRAM。

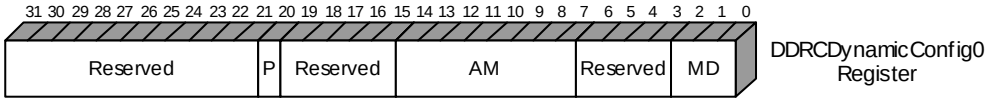


表3-11 DDRC Dynamic Config0 定义

位	名称	复位值	描述
[31:21]	Reserved	11’h0	保留位。
[20]	P	1’b0	写保护。 0：没有写保护； 1：有写保护。
[19:15]	Reserved	5’h0	保留位。
[14:7]	AM	8’h0	地址映射。
[6:3]	Reserved	4’h0	保留位。
[2:0]	MD	3’b0	存储器设备类型。 100：DDR SDRAM，必须设置为 3’b100。

表 3-11 中的地址映射根据表 3-12 进行配置。

表3-12 16bit 外部总线地址映射（Bank，Row，Column）

[14]	[13:12]	[11:9]	[8:7]	描述
0	00	000	00	16Mbit (2M×8)，2Banks，Row length=11，Column length=9
0	00	000	01	16Mbit (1M×16)，2Banks，Row length=11，Column length=8
0	00	001	00	64Mbit (8M×8)，4Banks，Row length=12，Column length=9

[14]	[13:12]	[11:9]	[8:7]	描述
0	00	001	01	64Mbit (4M×16)，4Banks，Row length=12，Column length=8
0	00	010	00	128Mbit (16M×8)，4Banks，Row length=12，Column length=10
0	00	010	01	128Mbit (8M×16)，4Banks，Row length=12，Column length=9
0	00	011	00	256Mbit (32M×8)，4Banks，Row length=13，Column length=10
0	00	011	01	256Mbit (16M×16)，4Banks，Row length=13，Column length=9
0	00	100	00	512Mbit (64M×8)，4Banks，Row length=13，Column length=11
0	00	100	01	512Mbit (32M×16)，4Banks，Row length=13，Column length=10

2. 配置示例

Hi3510 VSEVB 评估板中 DDR SDRAM 大小为 64MB，根据表 3-12 中“512Mbit(32M×16)，4Banks，Row length=13，Column length=10”得知，bit[14:7] = 00010001，其配置值为 0x884。

设置 DDR_EXTEND_MODE

1. DDR SDRAM 扩展模式寄存器定义

Hi3510 VSEVB 评估板中采用 Hynix 公司的 HY5DU121622B(L)T 型号的 DDR SDRAM，关于 HY5DU121622B(L)T 详细内容请参考该芯片的相关资料。如果采用其他型号的 DDR SDRAM 存储芯片，请参考对应型号的 DDR SDRAM 相关资料。

DDR_EXTEND_MODE 定义 DDR SDRAM 扩展模式的配置。

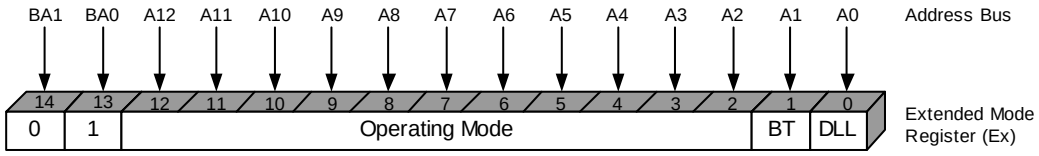


表3-13 DLL 的定义

E0	DLL
0	Enable
1	Disable

表3-14 DS 的定义

E1	DS
0	Normal
1	Reduced

表3-15 Operating Mode 的定义

E12	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1,E0	Operating Mode
0	0	0	0	0	0	0	0	0	0	0	Valid	Reserved
-	-	-	-	-	-	-	-	-	-	-	-	Reserved

2. 配置示例

Hi3510 VSEVB 评估板中，DDR 芯片通过 b0(Bank0)和 b1(Bank1)的值来选择配置扩展模式寄存器还是模式寄存器。

b0 为 0 且 b1 为 0：选择配置模式寄存器。

b0 为 1 且 b1 为 0：选择配置扩展模式寄存器。

发出的地址还是按照 Mode、Bank、Column 的顺序。由于 DDR 是 16 位的，因此最后还要加上 1 位 0。所以，则 Hi3510 VSEVB 评估板配置为 0xF0000800。

设置 DDR SDRAM 模式寄存器

1. DDR SDRAM 模式寄存器定义

DDR_MODE_REGISTER1 和 DDR_MODE_REGISTER2 定义 2 次模式寄存器所需的配置。

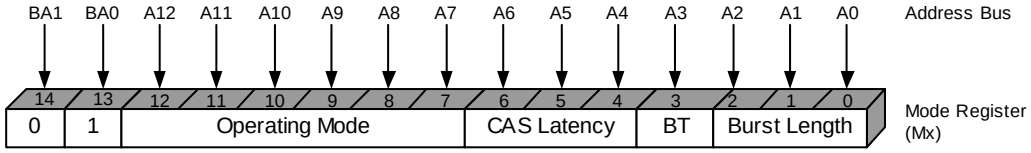


表3-16 Burst Length 的定义

M2 M1 M0	CAS Latency
0 0 0	Reserved
0 0 1	2
0 1 0	4
0 1 1	8
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

表3-17 BT 的定义

M3	Burst Type
0	Sequential
1	Interleaved

表3-18 CAS Latency 的定义

M6 M5 M4	CAS Latency
0 0 0	Reserved
0 0 1	Reserved
0 1 0	2
0 1 1	3 (DDR400 only)
1 0 0	Reserved
1 0 1	Reserved
1 1 0	2.5
1 1 1	Reserved

表3-19 Operating Mode 的定义

M12	M11	M10	M9	M8	M7	M6～M0	Operating Mode
0	0	0	0	0	0	Valid	Normal operation
0	0	0	0	1	0	Valid	Normal operation / reset DLL
-	-	-	-	-	-	-	All others states reserved

2. 配置示例

Hi3510 VSEVB 评估板中，模式寄存器配置同 SDRAM，设定 CAS=2，sequential 模式，burst length=2，发出的地址还是按照 Mode、Bank、Column 的顺序。

注意

如果不进行复位操作，可能导致存储器工作异常。

Hi3510 VSEVB 评估板先读取 0xF0242000 地址，用来复位内部 DLL，然后再次读取 0xF0042000 地址进入 Normal Operation 即可。

3.3 修改 Flash 驱动

Flash 驱动的修改包括 board/hisilicon/myboard/flash.c、include/flash.h 和 include/configs/myboard.h 三个文件。

其中：

- flash.c 文件为 Flash 驱动的主体文件
- flash.h 文件为 Flash 生产商和型号 ID 号的定义
- myboard.h 文件为单板所选用的 Flash 的相关配置信息

HiBoot 中的 Flash 驱动代码支持 Intel Flash 和 AMD Flash 芯片。Flash 驱动主要实现初始化、擦除、读写等操作。由于擦除和读写操作代码为通用代码，因此当选用不同于 Hi3510 VSEVB 评估板的 Flash 芯片时，只需要修改驱动中相关初始化的代码即可。Flash 驱动的初始化代码记录了所选用芯片的型号 ID、大小、块信息等。初始化函数 flash_init 位于 flash.c 文件中，调用函数是 flash_get_size 和 flash_get_offset。

修改 flash_get_size 函数

该函数通过 ID 匹配，增加 Flash 对应型号的相关信息，例如以 AMD S29GL256N 芯片为例，则修改如下：

```
case (FPW)AMD_ID_S29GL256N:
    info->flash_id += FLASH_AMDS29GL256N;
    info->sector_count = 256;
    info->size = 0x02000000;
    break;
```

其中，info 的数据结构为：

```
typedef struct {
    ulong  size;            /* total bank size in bytes        */
    ushort sector_count;    /* number of erase units           */
    ulong  flash_id;        /* combined device & manufacturer code */
    ulong   start[CFG_MAX_FLASH_SECT];  /* physical sector start addresses
*/
    uchar  protect[CFG_MAX_FLASH_SECT]; /* sector protection status    */
} flash_info_t;
```

AMD_ID_S29GL256N 和 FLASH_AMDS29GL256N 为该型号芯片对应的 ID 信息，通常定义在 flash.h 文件中。如果选用芯片型号已经定义在 flash.h 文件中，则可以直接使用。

修改 flash_get_offsets 函数

该函数通过 ID 匹配，记录 Flash 每一块的起始地址。例如以 AMD S29GL256N 芯片为例，其所有擦除块为统一大小，则修改代码如下：

```
else if ((info->flash_id & FLASH_VENDMASK) == FLASH_MAN_AMD&& (info-
```

```
>flash_id & FLASH_TYPEMASK) == FLASH_AMDS29GL256N)
{
    int sect_size;
    sect_size = 0x00020000;
    /* set up sector start address table (uniform sector type) */
    for( i = 0; i < info->sector_count; i++ )
    info->start[i] = base + (i * sect_size);
}
```

对于存在不同的块大小的 Flash，则可以修改代码如下：

```
else if ((info->flash_id & FLASH_VENDMASK) == FLASH_MAN_INTEL && (info-
>flash_id & FLASH_BTYPE) == YOUR_FLASH_ID)
{
    int bootsect_size;      /* number of bytes/boot sector  */
    int sect_size;          /* number of bytes/regular sector */

    bootsect_size = 0x00002000;
    sect_size     = 0x00010000;

    /* set sector offsets for bottom boot block type       */
    for (i = 0; i < 8; ++i)
    {
        info->start[i] = base + (i * bootsect_size);
    }

    for (i = 8; i < info->sector_count; i++)
    {
        info->start[i] = base + ((i-7) * sect_size);
    }
}
```

修改 flash_print_info 函数

修改函数 flash_print_info，以便打印出正确的 Flash 信息。修改如下：

```
case FLASH_AMDS29GL256N:
    fmt = "S29GL256N(32 Mbit, uniform sectors)\n";
    break;
```

Flash 参数配置宏定义

在 myboard.h 文件中定义了 Flash 及 HiBoot 环境变量存储的参数设置，需要做进一步的修改，其定义如下：

```
/*-----------------------------------------------------------------------
 * FLASH and Environment Configuration
```

```
 */

#define CFG_FLASH_BASE          0x34000000
#define CFG_FLASH_BLOCK_SIZE    0x20000

#define CFG_FLASH_ERASE_TOUT    (20*CFG_HZ)
#define CFG_FLASH_WRITE_TOUT    (20*CFG_HZ)

#define CFG_ENV_IS_IN_FLASH     1
#define CFG_ENV_SECT_SIZE       0x20000
#define CFG_ENV_SIZE            0x20000
#define CFG_ENV_OFFSET          0x40000
#define CFG_ENV_ADDR            (CFG_FLASH_BASE + CFG_ENV_OFFSET)
#define CONFIG_ENV_OVERWRITE

#define CFG_MAX_FLASH_BANKS     1
#define CFG_MAX_FLASH_SECT      256

#define FLASH_PORT_WIDTH16
```

Flash 参数配置宏定义详细描述如下：

1. CFG_FLASH_BASE

定义 Flash 的起始物理地址，不用修改。

2. CFG_FLASH_BLOCK_SIZE

定义 Flash 的块大小，根据具体 Flash 芯片需修改。当 Flash 存在大小不一的块时，配置为多数块的大小。

3. CFG_FLASH_ERASE_TOUT 和 CFG_FLASH_WRITE_TOUT

CFG_FLASH_ERASE_TOUT 和 CFG_FLASH_WRITE_TOUT 定义了 Flash 擦除和写的超时时间，不用修改。

4. CFG_ENV_IS_IN_FLASH

表示 HiBoot 的环境变量存储在 Flash 中，不用修改。

5. CFG_ENV_SECT_SIZE

定义存储 HiBoot 环境变量所在块的大小，根据具体芯片和存储位置需做调整。

6. CFG_ENV_SIZE

定义存储 HiBoot 环境变量的 Flash 大小，通常设定为 Flash 擦除块的大小，也可根据实际情况调整。

7. CFG_ENV_OFFSET

定义存储 HiBoot 环境变量的 Flash 偏移地址，根据具体情况需做调整。

8. CFG_ENV_ADDR

定义存储 HiBoot 环境变量的起始地址，为 CFG_FLASH_BASE 与 CFG_ENV_OFFSET 之和，因此不用修改。

9. CONFIG_ENV_OVERWRITE

定义部分特殊环境变量允许覆盖的标志，不用修改。

10. CFG_MAX_FLASH_BANKS

定义 Flash 的 Bank 数，通常为 1，不用修改。

11. CFG_MAX_FLASH_SECT

定义 Flash 的块数目，根据具体芯片设定。

12. FLASH_PORT_WIDTH16

定义 Flash 的地址总线宽度，当前配置为 16 位宽度。

如果是 Flash 的地址总线宽度是 8 位宽，则宏定义为 FLASH_PORT_WIDTH8。

如果是 Flash 的地址总线宽度是 32 位宽，则宏定义为 FLASH_PORT_WIDTH32。

3.4 修改 PHY 驱动

HiBoot 中支持 Intel、Realtek、Micrel 三种类型的 PHY 芯片，通过配置就可以选择所使用的芯片类型。

在文件 myboard.h 中的“Net Driver Configuration”区包括以下内容：

```
#define CONFIG_HISILICON_SF_PHYCHIP_REALTEK
#undef CONFIG_HISILICON_SF_PHYCHIP_INTEL
#undef CONFIG_HISILICON_SF_PHYCHIP_MICREL

#define CONFIG_HISILICON_SF_UPP_PHYNAME    "realtek"
#define CONFIG_HISILICON_SF_UPP_PHYID      0
#define CONFIG_HISILICON_SF_UPP_INITSTAT   "00000"

#define CONFIG_HISILICON_SF_DNP_PHYNAME    "realtek"
#define CONFIG_HISILICON_SF_DNP_PHYID      0
#define CONFIG_HISILICON_SF_DNP_INITSTAT   "00000"

#define CONFIG_HISILICON_PHYCHIP_EXTADDR_S "0x01,0x02"
```

步骤如下：

1. 选定 PHY 芯片类型

```
#define CONFIG_HISILICON_SF_PHYCHIP_REALTEK
#undef CONFIG_HISILICON_SF_PHYCHIP_INTEL
#undef CONFIG_HISILICON_SF_PHYCHIP_MICREL
```

以上宏定义设定系统使用的 PHY 芯片类型。如果待移植单板中仅使用了一种芯片，则定义其中一个 PHY 芯片类型即可。假设选用了两种芯片，比如 Realtek 和 Intel 的则如下定义：

```
#define CONFIG_HISILICON_SF_PHYCHIP_REALTEK
#define CONFIG_HISILICON_SF_PHYCHIP_INTEL
#undef CONFIG_HISILICON_SF_PHYCHIP_MICREL
```

2. 设定上/下行口 PHY 芯片类型

```
#define CONFIG_HISILICON_SF_UPP_PHYNAME "realtek"
#define CONFIG_HISILICON_SF_UPP_PHYID 0
#define CONFIG_HISILICON_SF_UPP_INITSTAT "00000"
```

以上宏定义用于配置上行口所使用的 PHY 芯片，如果选用 Intel 的 PHY 芯片则将宏 CONFIG_HISILICON_SF_UPP_PHYNAME 定义为 Intel 即可。另外两个宏定义不用修改。

```
#define CONFIG_HISILICON_SF_DNP_PHYNAME "realtek"
#define CONFIG_HISILICON_SF_DNP_PHYID 0
#define CONFIG_HISILICON_SF_DNP_INITSTAT "00000"
```

以上宏定义配置下行口所使用的 PHY 芯片。配置方法与上行口一致。

3. 配置 PHY 芯片的扩展地址

```
#define CONFIG_HISILICON_PHYCHIP_EXTADDR_S "0x01,0x02"
```

配置 PHY 芯片的扩展地址，依次设定上下行口的地址，中间使用逗号分隔。

3.5 编译 HiBoot

完成以上 4 个移植步骤后，编译 HiBoot，操作步骤如下：

```
make mrproper
make myboard_config
make
```

编译成功后，将在 HiBoot 目录下生成 hiboot.bin。

4 烧写 HiBoot

如果待移植单板中已有 HiBoot 运行，则可以通过串口或网口直接更新 HiBoot，具体操作请参见《Hi3510 Linux 开发环境用户指南》。

如果是第一次烧写，则需要通过 JTAG 接口使用 ADS 工具进行烧写。因此，在 ADS 开发环境中必须有应用程序对 Flash 进行读写操作，而此应用程序又必须在单板的内存中运行，则首先需要完成系统状态和内存等配置。在 Hi3510_VSSDK 中提供了相应的初始化脚本，当选用了不同的外围芯片，则需要对以上应用程序做适当修改方可使用。

4.1 初始化脚本

初始化脚本与 HiBoot 中 lowlevel_init.S 文件完成相同的功能，但实现的方法有所不同，在 HiBoot 中使用汇编代码进行编写，而在 ADS 环境中则使用 AXD Debugger 提供的命令进行操作，有以下三条命令：

- com 命令：注释命令，直接在要注释的行前面加 com 即可。
  如：com smem 0x10110224 0x0 32，则此语句就会被注释掉，初始化时不会被执行。
- smem 命令：往寄存器的某个地址中写入某个值。
  如：smem 0x10150030 0x1 32，表示往 0x10150030 的地址中写入一个 32 位宽的值 0x1。
- mem 命令：读出某个地址。
  如：mem 0xF0000400, +0x4, 32，表示从地址 0xF0000400 读取 4 个字节的数据，其中位宽为 32。

初始化脚本依次完成系统控制器的配置、SDRAM 和 MEMC 的配置、DDR SDRAM 和 DDRC 的配置。详细脚本请阅读 Hi3510_VSSDK 中对应的文件。

当选用不同的存储器芯片时，需要修改的内容与 HiBoot 中存储器配置一节修改的内容一致，具体请参见“3.2 配置存储器”。对应的配置值可以直接使用 HiBoot 中已经移植的数据。可参考\tools\flash_easy_Hi3510DVS 目录下的 Hi3510_DVS_MEMC_DDRC_100MHz.li 文件，具体修改内容如下：

```
smem 0x10110100 0x000004680 32     //配置MEMC Dynamic Config0寄存器
```

```
smem 0x10110020 0x0000C083 32
mem 0x60040000, +0x4, 32            //配置SDRAM模式寄存器
smem 0x10150100 0x00000484 32       //配置DDRC Dynamic Config0寄存器

smem 0x10150020 0x0000C083 32
mem 0xf0000400, +0x4, 32            //配置DDR SDRAM扩展模式寄存器

mem 0xf0121000, +0x4, 32            //第一次配置DDR SDRAM模式寄存器

smem 0x10150020 0x0000C083 32
mem 0xf0021000, +0x4, 32            //第二次配置DDR SDRAM模式寄存器

smem 0x10110220 0x81 32             //配置MEMC Static Config0寄存器
```

4.2 烧写 HiBoot 的 Flash 应用程序

在 Hi3510_VSSDK 中提供了 ADS 环境下 HiBoot 的 Flash 应用程序，其核心代码为 Flash 的驱动程序。当选用不同的 Flash 芯片时，需要对驱动作相应修改。修改内容主要是 Flash 初始化函数。需要修改文件是 amdflash.c，需要修改函数是 amd_flash_init。

修改 amd_flash_init 函数

amd_flash_init 函数定义了 Flash 芯片 ID，根据 ID 完成芯片信息的记录。

本例是选择 Flash 芯片为 AMD S29GL256N，块大小为 128kB，Flash 大小为 32MB。修改代码如下：

```
else if(device_id == 0x227e)
{
    /*size:32MB, blocksize:128kB;blocknum:256*/
    flash_handle.flashsize = 0x2000000;
    flash_handle.blocksize = 0x20000;
    flash_handle.blocknum  = 256;
    flash_handle.bottom_or_top = 0;
}
```

编译 Flash 应用程序

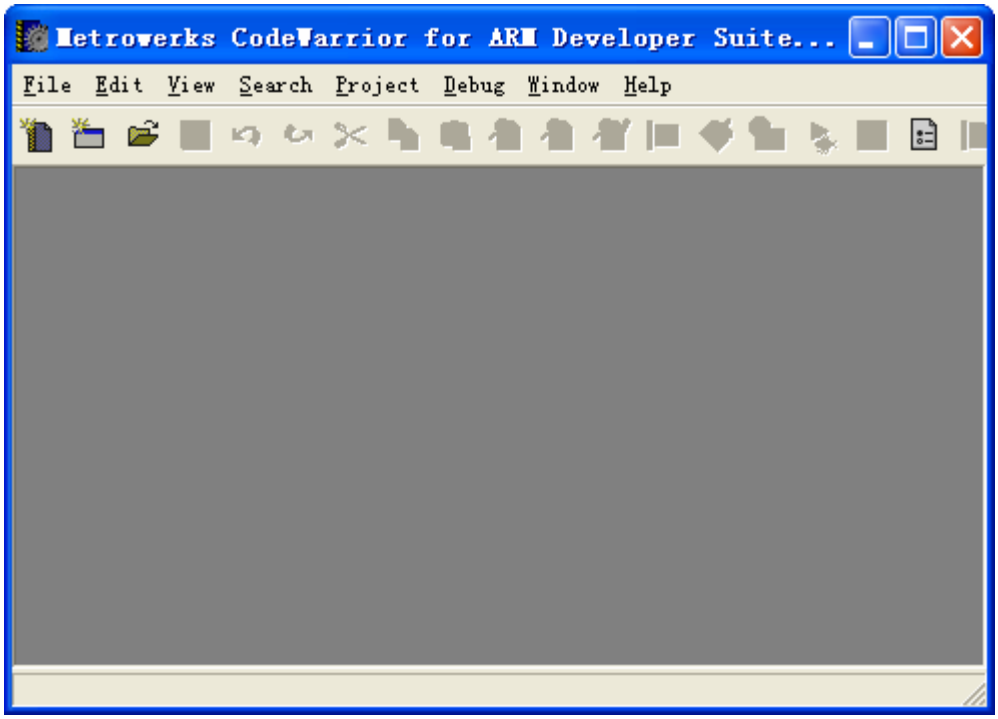
在完成修改 Flash 应用程序后，使用 ARM 提供的开发工具 CodeWarrior for ARM Developer Suite（以下简称 CodeWarrior，安装程序是 ARM developer Suite v1.2）进行应用程序的编译。

本例以 Flash_easy_prj 工程对编译 Flash 应用程序进行详细描述。

步骤 1　运行 CodeWarrior

运行 CodeWarrior，进入 CodeWarrior 环境，如图 4-1 所示。

图4-1 CodeWarrior 窗口

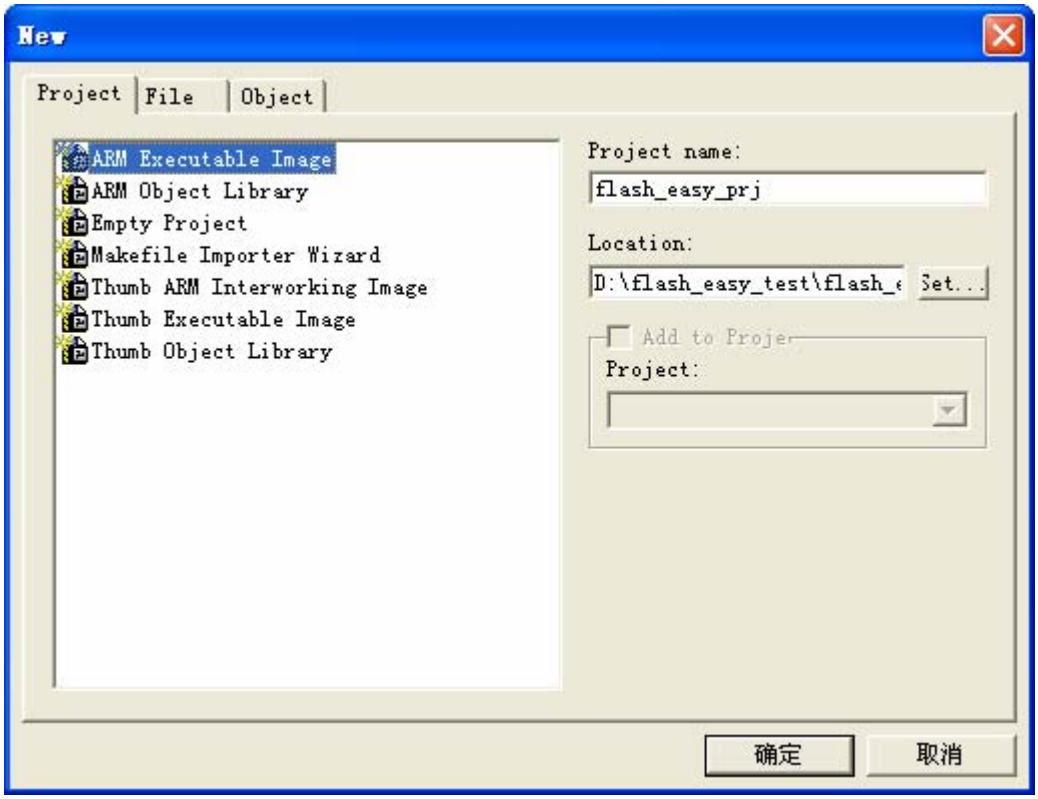


步骤 2 创建新项目

创建一个新工程 Flash_easy_prj。选择“File > New”菜单项，进入“New”窗口，如图 4-2 所示。

在“Project”页签的列表框中选择“ARM Executable Image”选项，在“Project name”的文本框中输入文件名 flash_easy_prj，在“Location”文本框中输入路径，单击“确定”按钮。

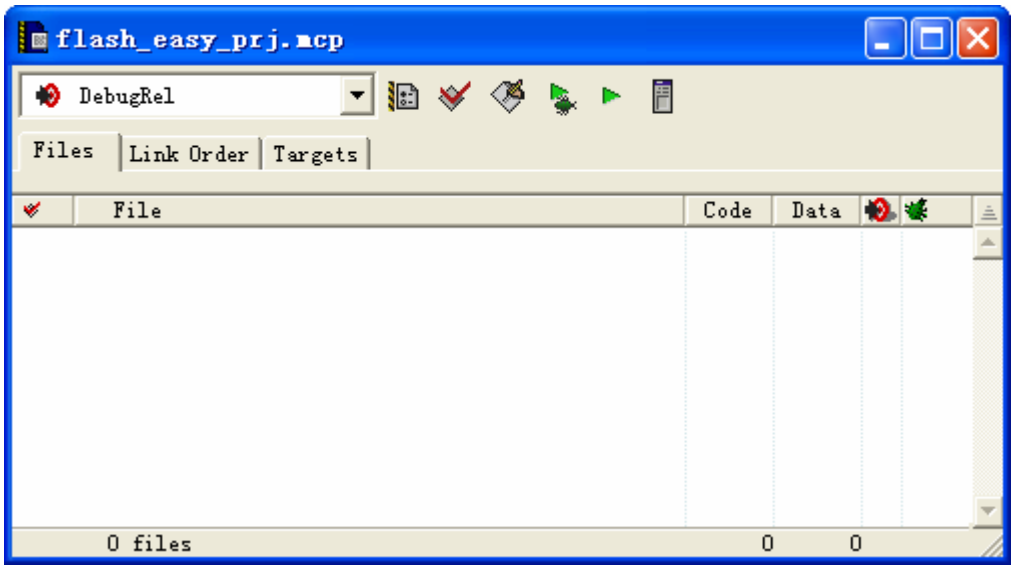
图4-2 New 窗口



步骤 3 添加文件

进入 flash_easy_prj 项目的界面，如图 4-3 所示。

图4-3 flash_easy_prj 工程窗口

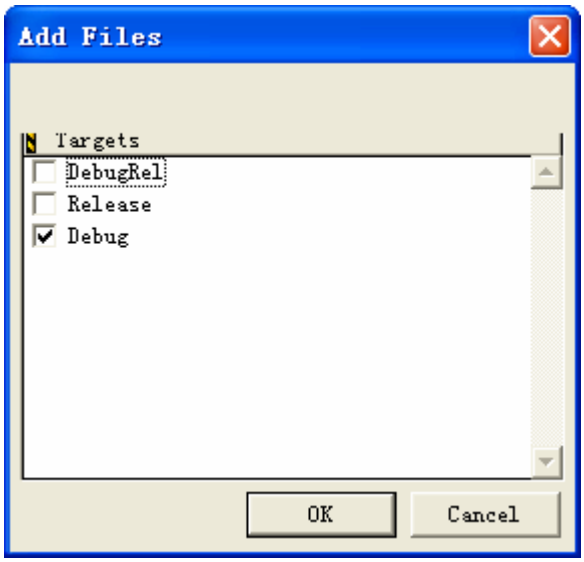


在新建 flash_easy_prj 工程的目录下，编辑 HiBoot 所用的 Flash 源程序及头文件，或者把已经编辑好的 HiBoot 所用的 flash 源程序及头文件拷贝到此目录下，在本例中包含了 4 个文件：main.c、amdflash.c、amdflash.h 和 hi_amdflash.h（在发布包中这 4 个文件在\tools\flash_easy_Hi3510DVS 目录下）。

选择“Project > Add Files…”菜单，选中 main.c 和 amdflash.c 源文件，单击“打开”按钮。

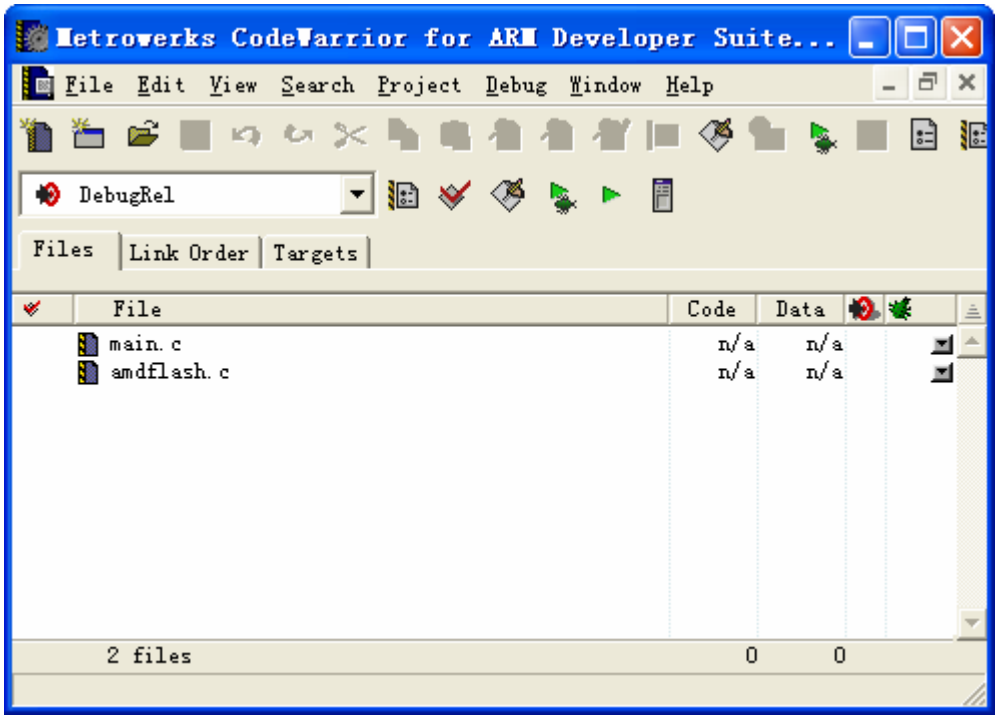
弹出“Add Files”对话框，选中“Debug”前的复选框（打上钩后表示选中），单击“OK”按钮，表示只选择 Debug 调试模式。

图4-4 Add Files 窗口



添加 main.c 和 amdflash.c 源文件后，如图 4-5 所示。

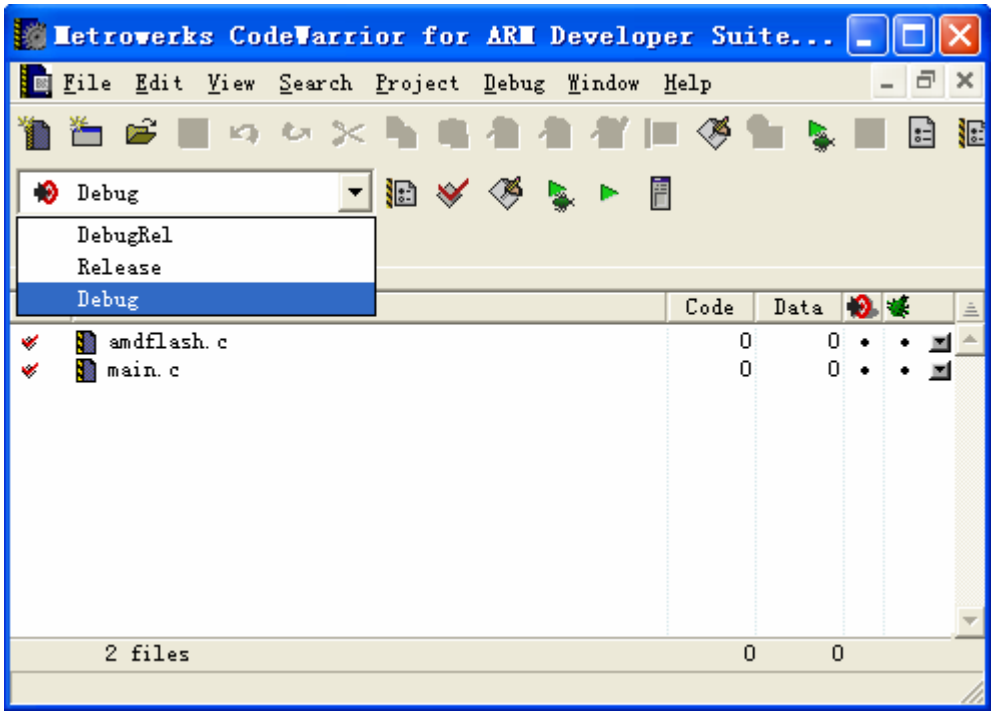
图4-5 已添加文件的 flash_easy_prj 工程窗口



步骤 4 编译设置

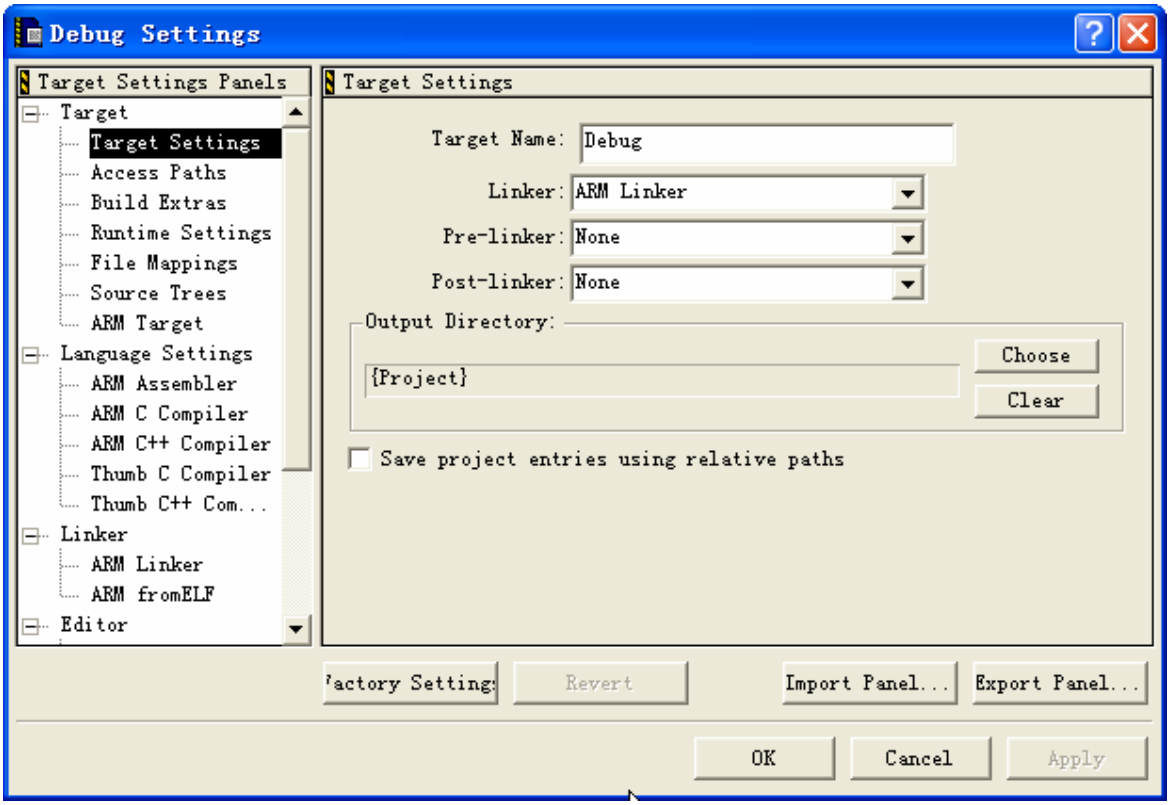
在“Debug”的下拉组合框中选择“Debug”选项。

图4-6 Debug 下拉组合框



单击编译设置按钮 “  ”，进入 Debug 设置窗口，如图 4-7 所示。

图4-7 Debug 设置窗口



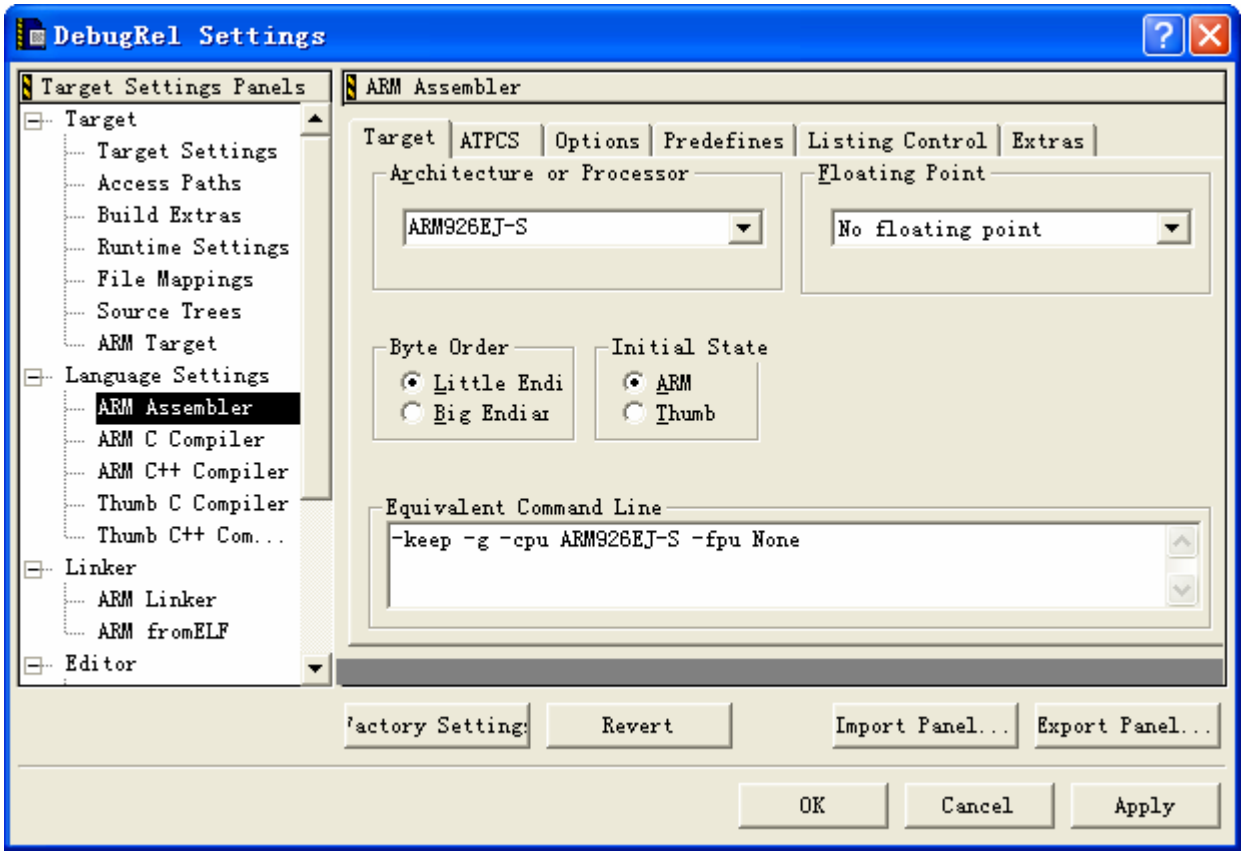
选择“Language Settings”下的“ARM Assembler”选项，进入“Target”页签，设置编译器语言。

设置 CPU 类型：在“Architecture or Processor”区域框中，通过下拉列表框中选择“ARM926EJ-S”。

设置存储器大小端模式：在“Byte Order”区域框中，通过“Little Endian”和“Big Endian”前的单选按钮选择，本例是选择“Little Endian”。

其他选项采用默认设置即可，单击“Apply”按钮，如图 4-8 所示。

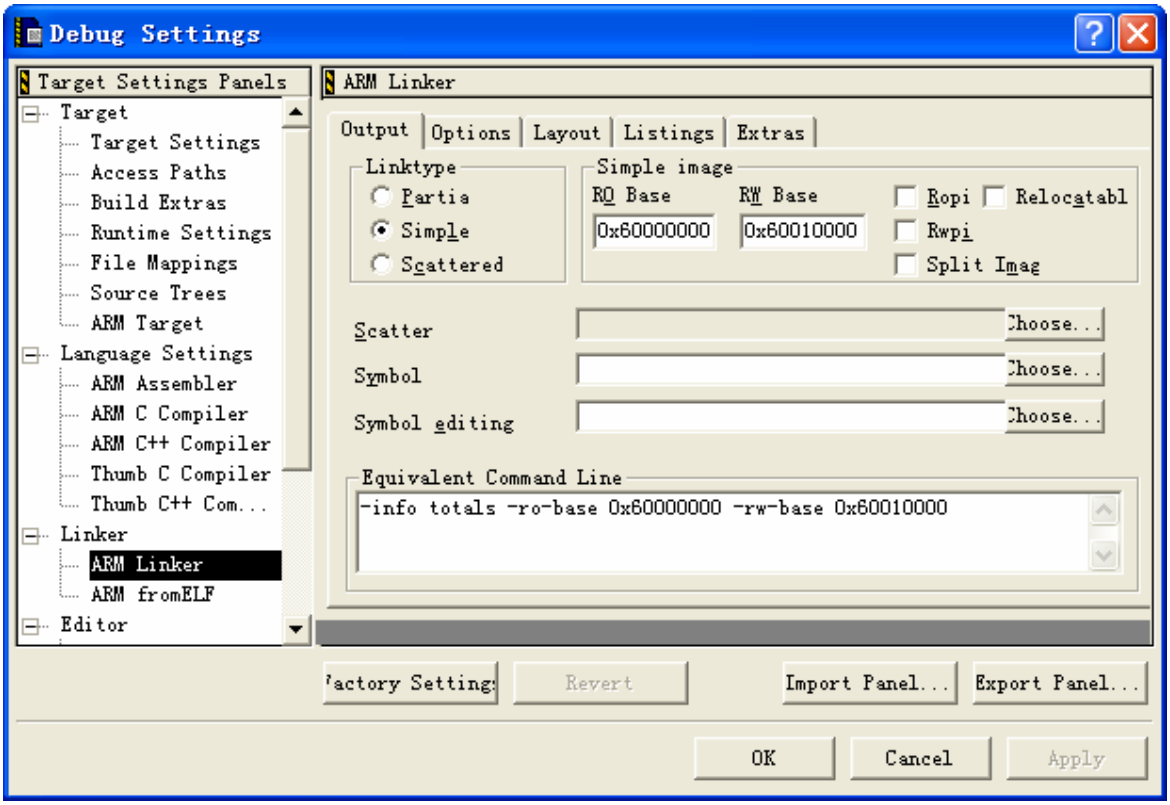
图4-8 各种语言编译器设置



选择“Linker”下的“ARM Linker”选项。在“Output”页签的“Simple image”区域框中，修改“RO Base”文本框为 0x60000000，修改“RW Base”文本框为 0x60010000。

其他选项采用默认设置即可，设置完毕，单击“OK”按钮，如图 4-9 所示。

图4-9 程序代码和数据代码地址设置窗口



设置完毕后，在如图 4-5 所示的窗口中，单击编译按钮“ ”进行编译。

编译完成后，出现如下面所示的编译信息，表示编译成功：

```
==========================================================
Image component sizes
  code   RO Data     RW Data   ZI Data    Debug
  2392     60         0        20      13252     Object Totals
  14564    586        0        300     6332      Library Totals
==========================================================
  code   RO Data     RW Data   ZI Data    Debug
  16956    646          0       320     19584     Grand Totals
==========================================================
  Total RO  Size ( Code + RO Data )           17602  ( 17.19kB)
  Total RW  Size ( RW Data + ZI Data )          320  (  0.3kB)
  Total ROM Size ( Code + RO Data + RW Data)   17602  ( 17.19kB)
```

编译生成 flash_easy_prj.axf 调试文件。

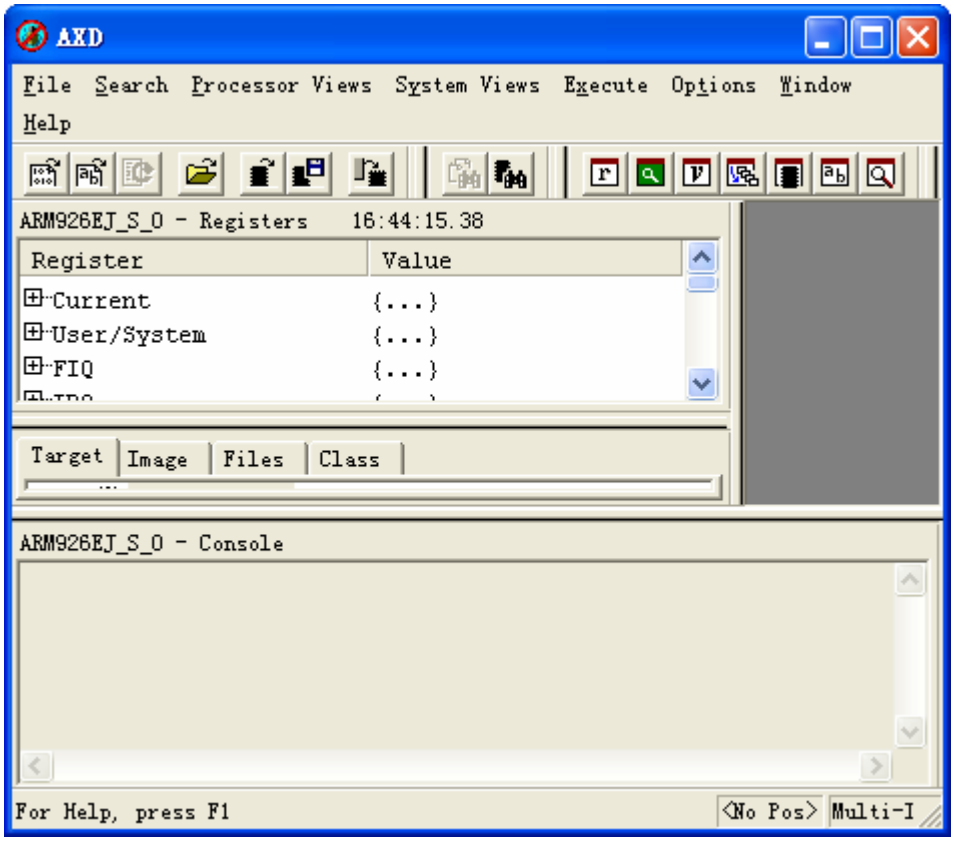
----结束

4.3 烧写 HiBoot

启动 AXD Debugger

首先运行 Multi ICE Server 应用程序，查找所连接目标板的 ARM 芯片。然后启动 AXD Debugger 连接到 Multi-ICE。关于使用 Multi ICE 和 AXD Debugger 工具请参见“5 如何使用 ARM 调试工具”。

图4-10 AXD Debugger 窗口



加载系统初始化脚本程序

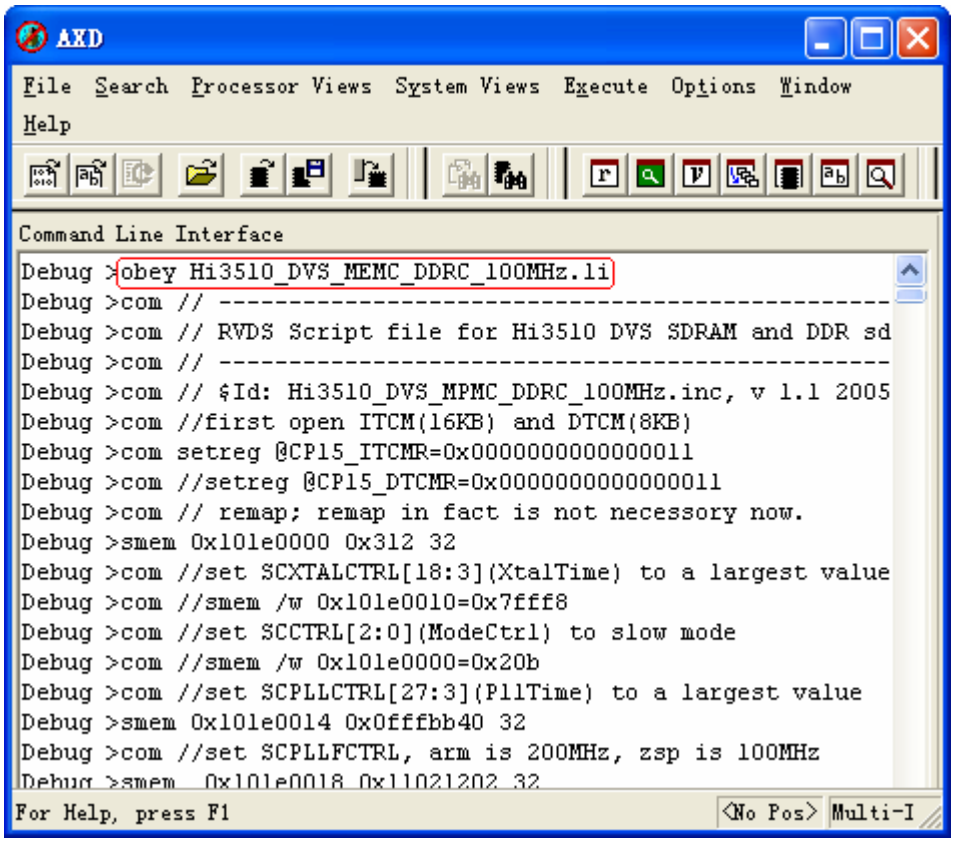
选择“System Views > Command Line Interface”菜单项。烧写 Flash 的程序需要在 SDRAM 中运行，而此时系统和 SDRAM 还没有初始化，因此需要在命令行窗口中载入系统初始化的脚本程序。

注意

此脚本程序必须要放置在 AXD Debugger 软件的安装目录下..\ARM\ADSv1_2\Bin，才能正确的执行此脚本。例如安装目录：C:\Program Files\ARM\ADSv1_2\Bin。

例如在命令行窗口中输入：**obey　Hi3510_DVS_MEMC_DDRC_100MHz.li**，回车后，在 Command Line Interface 窗口中出现命令行如图 4-11 所示，表示已成功载入。

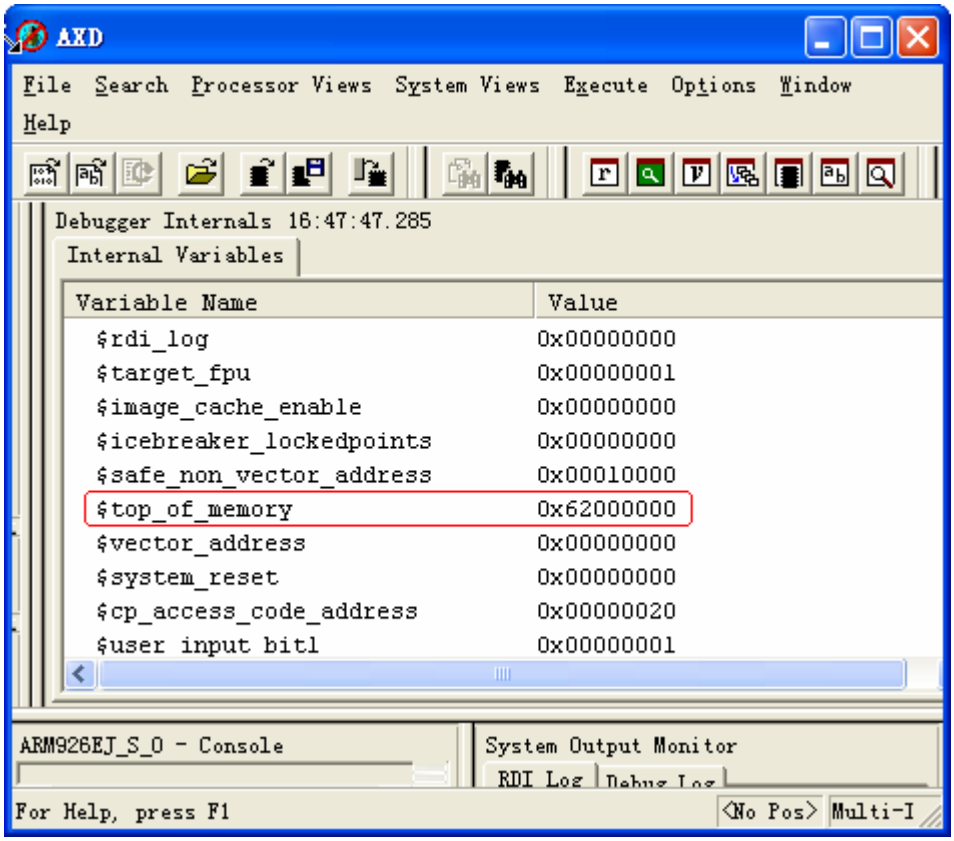
图4-11 Command Line Interface 窗口



设置 Top 堆栈地址

选择“System Views > Debugger Internals”菜单项，在“Internal Variables”页签中，双击“$top_of_memory”变量值，修改值为：0x62000000，用来设置程序运行的 Top 堆栈地址，如图 4-12 所示。如果单板只有 32M 内存，可设置为 0x61000000，。

图4-12 Top 堆栈地址设置窗口



打开映象文件并运行

步骤 1 选择“File > Load image…”菜单项。

步骤 2 打开“Load Image”对话框，找到已经编译好的.axf（如：flash_easy_prj.axf）映象文件，单击“打开”按钮，把映象文件加载到内存中。

步骤 3 然后单击按钮“  ”（**GO**），会出现一个蓝色箭头，指示到当前执行的位置。

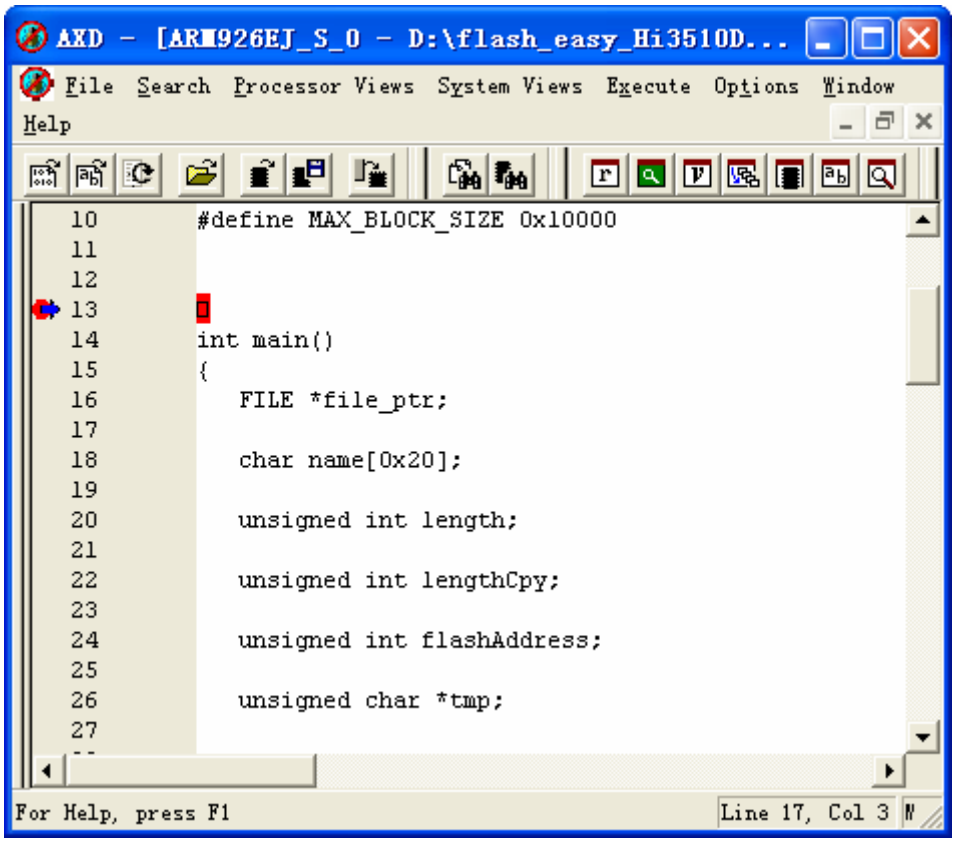
----结束

对于本例，打开映象文件并运行后如图 4-13 所示。

说明

此处的映象文件如果已经编译好了，可以直接加载；如果映象文件还没有编译好，需要使用 CodeWarrior 进行编译生成映象文件，具体操作请参考 ARM 公司提供的相关文档。

图4-13 运行后的映象文件窗口



再次运行映象文件

步骤 1 再次单击按钮“  ”（**GO**），系统显示如图 4-14 所示的提示信息，需要输入要加载的文件路径及文件名。

步骤 2 在命令窗口下按照提示信息先输入要加载的文件路径及文件名（如：d:\HiBoot.bin），回车。系统显示如图 4-15 所示的提示信息，需要输入 Flash 的起始地址。

步骤 3 输入待写入 Flash 的起始地址：0x34000000，回车。运行程序会自动把要烧写的 HiBoot.bin 目标文件烧写到指定的位置。加载完成后，系统显示如图 4-16 所示的提示信息，表示加载成功。

----结束

烧写 HiBoot 完成后，程序自动停止运行，然后关闭 AXD Debugger 软件。这时复位或者重新上电，启动单板，HiBoot 就能自行启动了。

图4-14 再次运行后的命令窗口

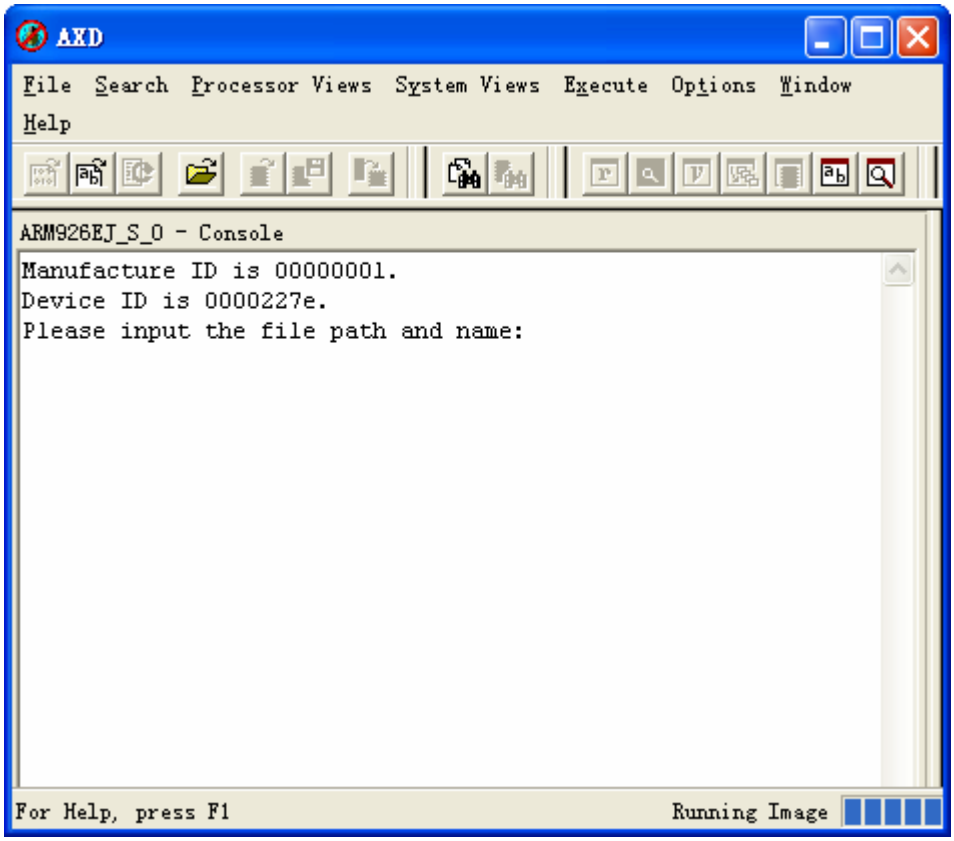


图4-15 已指定 HiBoot 文件和 Flash 起始地址的命令窗口

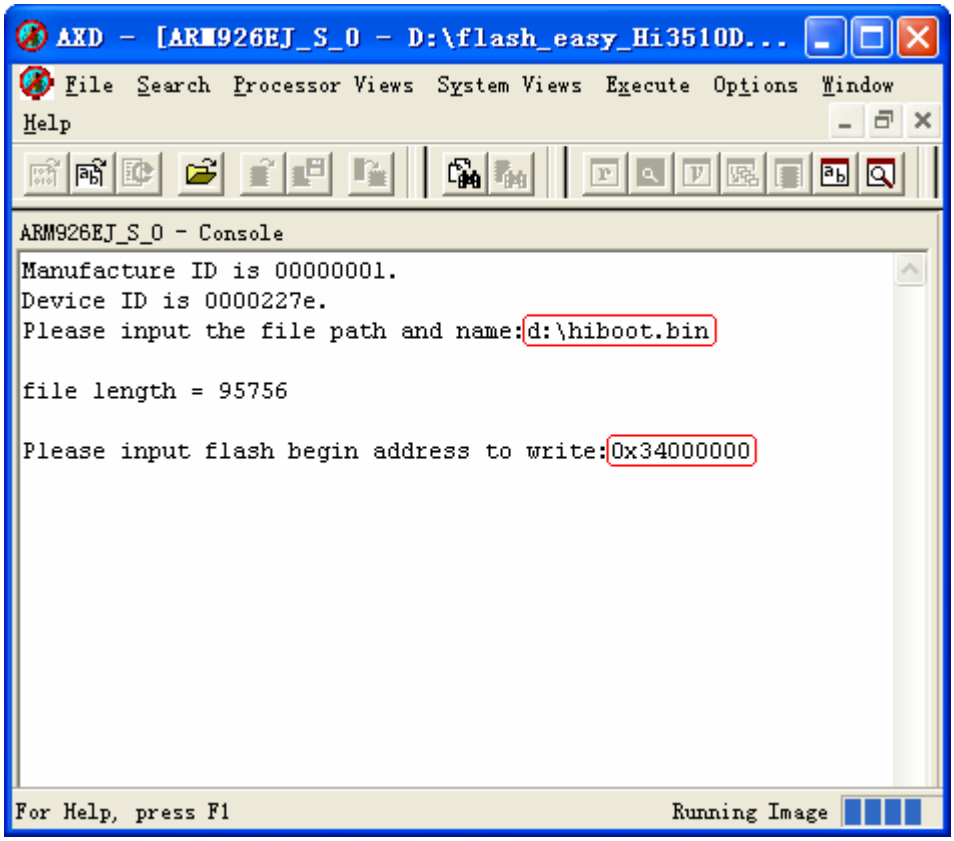
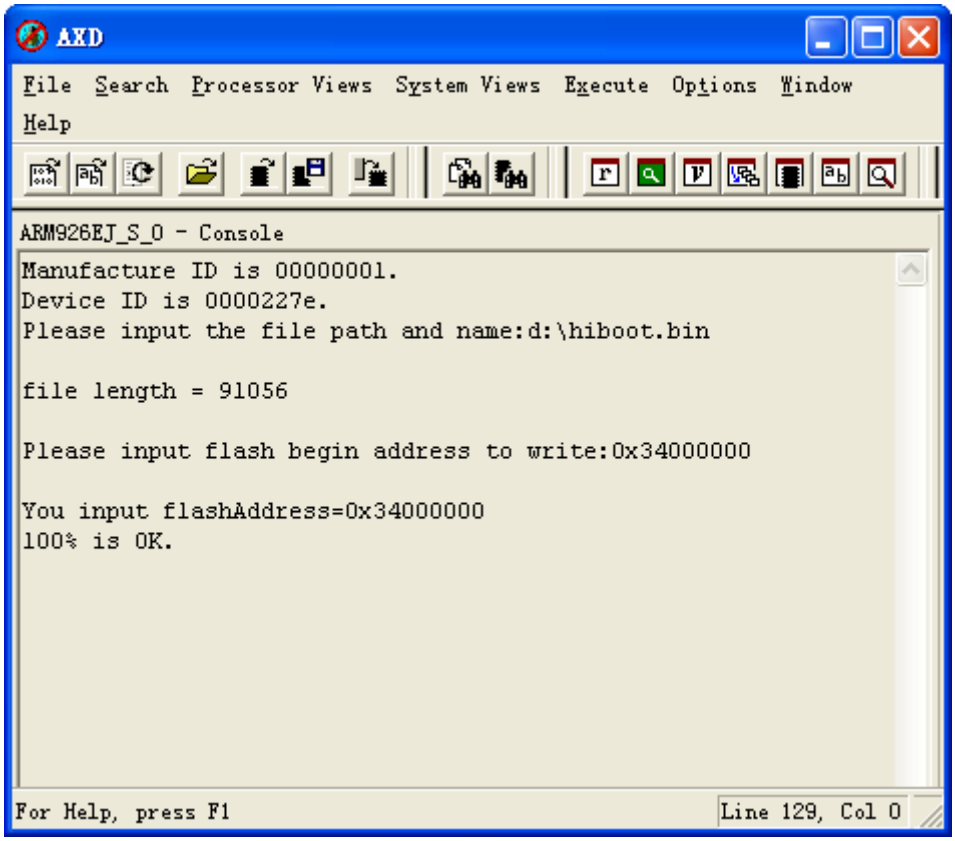


图4-16 烧写 HiBoot 成功后的窗口



5 如何使用 ARM 调试工具

本章介绍了关于 ARM 处理器调试用到的调试工具的使用方法，调试工具包括：

- Magic-ICE 仿真器
- Multi-ICE Server
- AXD Debugger

5.1 ARM 调试工具简介

5.1.1 Magic-ICE 仿真器

Magic-ICE 仿真器是一款高性能实时仿真器，该款仿真器支持 ARM 内核结构（如 ARM7、ARM9 Xscale）的处理器，是基于 ARM 开发调试的必备工具。

Magic-ICE 仿真器的特点如下：

- 支持内含 Embedded ICETM logic 的 ARM 内核芯片，同时也支持尚在开发中的 ARM 内核芯片，包括 ARM7、ARM710、ARM720、ARM740、ARM9、ARM920、ARM10 等 ARM 内核调试；
- 支持目标系统的电源 1.8V～5V 自适应；
- 支持高速数据下载，下载速度最高可达 1040kbit/s；
- 通过 JTAG 仿真器，用户可以轻松修改寄存器、存储器，设置硬件断点，增加观察窗口等。可以在二进制文件的开始位置或者结束位置保存自定义的数据和调色板。

5.1.2 Multi-ICE Server

Multi-ICE server 是由 ARM 公司提供的应用程序，可以直接在 Windows 平台上运行，实现与 Multi-ICE 接口的连接。

Multi-ICE server 可以在不影响板上其他设备的情况下，单独的与板上的每一个 JTAG 设备地址相对应，建立虚拟连接。调式软件可以仅仅依赖所有虚拟连接中的一条虚拟连接来完成调试操作，而不需要知道板上的其他设备的任何信息。

5.1.3 AXD Debugger

AXD Debugger 是由 ARM 公司提供的软件开发调试软件，为软件开发者开发高质量的 ARM 代码。AXD Debugger 提供了如下便利：

- 支持简单和复杂的断点设置；
- 提供观察窗口，可查看变量的变化情况；
- 支持所有新的和已经存在的 ARM 处理器；
- 增强的 Windows 管理模式；
- 增强的数据显示、修改和编辑功能；
- 提供完备的命令行接口。

5.2 使用 ARM 调试工具

要使用 Magic-ICE 进行程序调试或往评估板烧写 HiBoot 程序，首先必须启用 Multi-ICE server 程序查找到当前硬件相连的 ARM 芯片，然后才能通过 AXD Debugger 调试软件进行程序调试或往开发板烧写 HiBoot 程序。

关于使用 ARM 调试工具的详细描述请参考 ARM 公司提供的文档。

5.2.1 使用 Multi-ICE Server

安装 ARM Multi-ICE v2.2

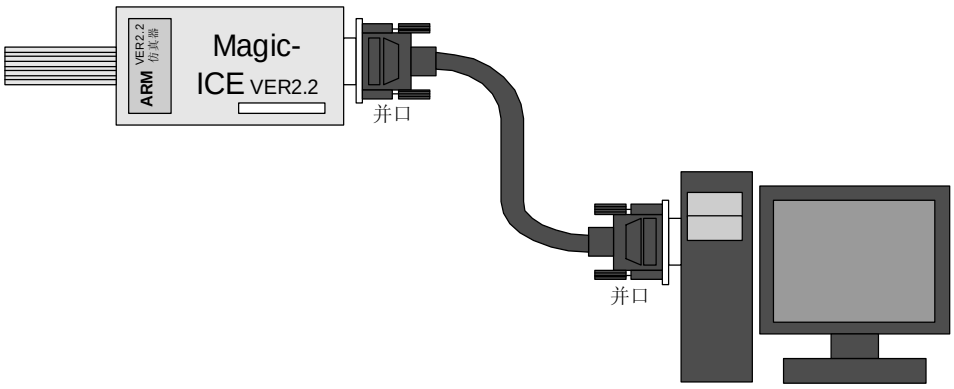
ARM Multi-ICE v2.2 是由 ARM 公司提供的 Multi-ICE Server 安装程序。安装前，请先阅读 ARM 的相关文档，然后安装 ARM Multi-ICE v2.2。

连接 Magic-ICE 仿真器

Magic-ICE 是通过并口和主机相连，如图 5-1 所示，不需要外接电源。

驱动这个 ICE 需要先安装 ARM Multi-ICE v2.2。

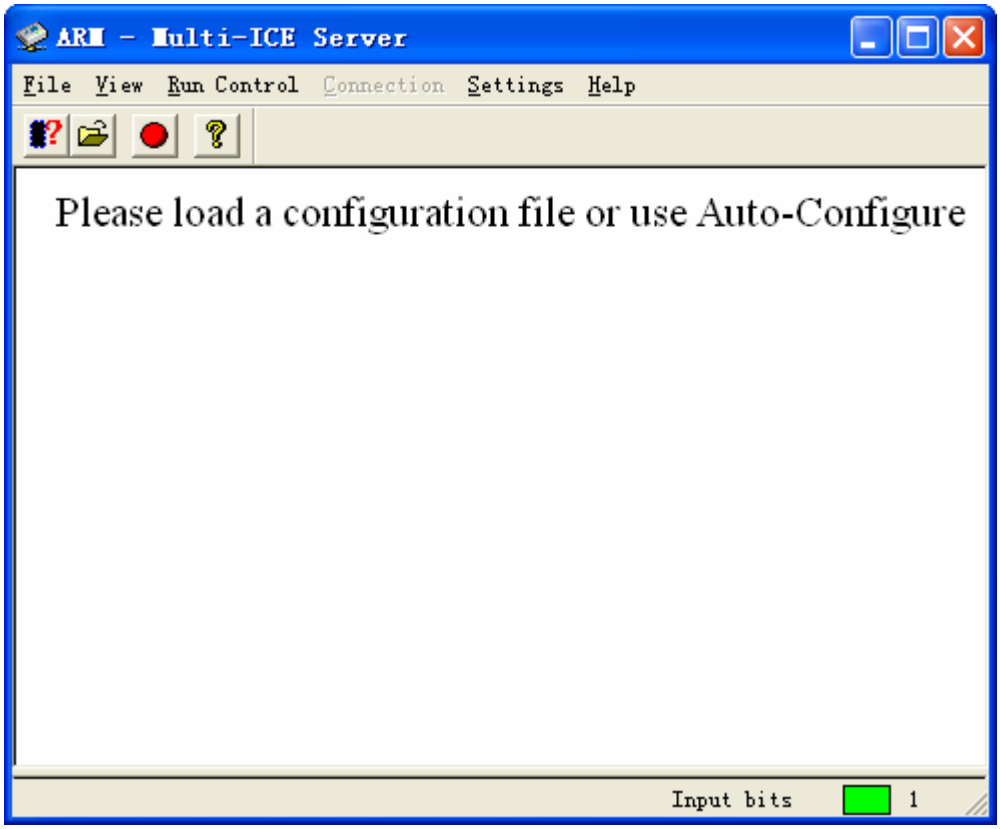
图5-1 Magic-ICE 与主机连接示意图



运行 Multi-ICE Server

成功安装 ARM Multi-ICE v2.2 后，运行 Multi-ICE Server 程序，进入 Multi-ICE Server 窗口，如图 5-2 所示。

图5-2 Multi-ICE Server 窗口



查找 ARM 芯片

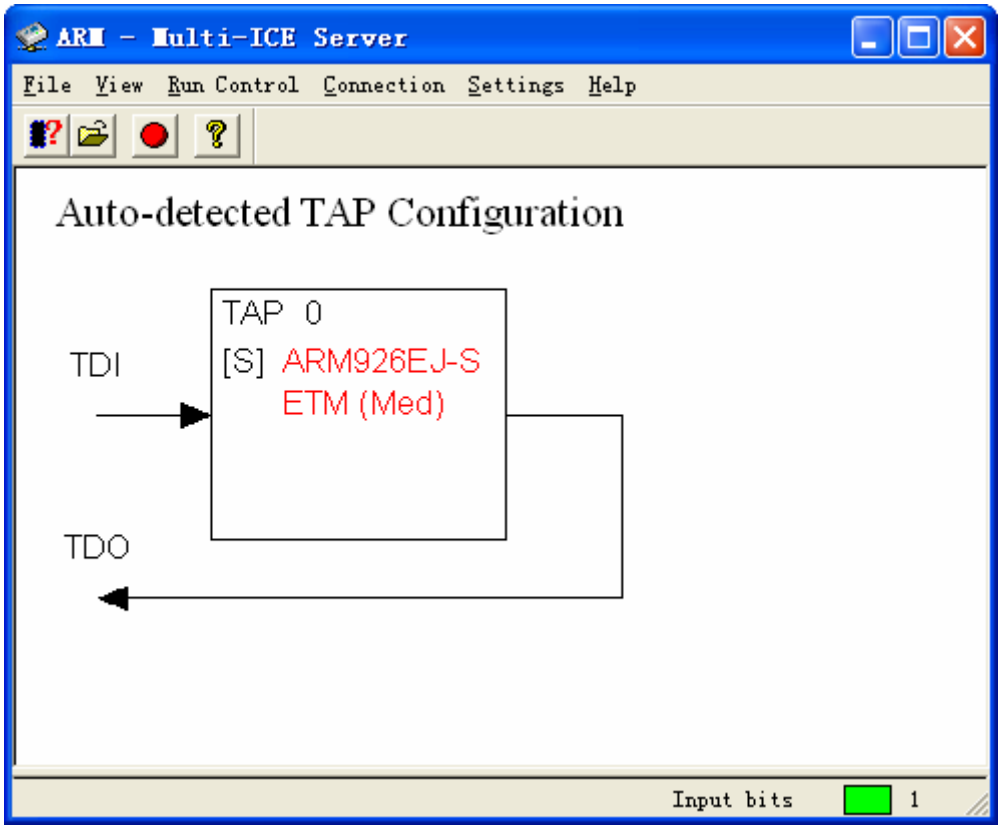
单击“  ”按钮（或者选择“file > auto-configure”菜单项），可以查找到当前硬件连接的 ARM 芯片。如图 5-3 所示，表示已找到一个 ARM926EJ-S 芯片内核。

注意

所操作的单板必须是在调试状态下：比如 Hi3510 CODB VER B 板的 DIP 3 应该为 OFF 状态，否则 Multi-ICE Server 无法自动检测到单板 ARM 芯片。

同时系统会启动一个 portmap.exe 程序，请不要关闭 portmap.exe 程序。如果关闭 portmap.exe 程序将无法启动 AXD Debuger 程序。portmap.exe 是一个 server 程序，不能做调试使用。

图5-3 查找 ARM 芯片窗口



5.2.2 使用 AXD Debugger

安装 ARM developer Suite v1.2

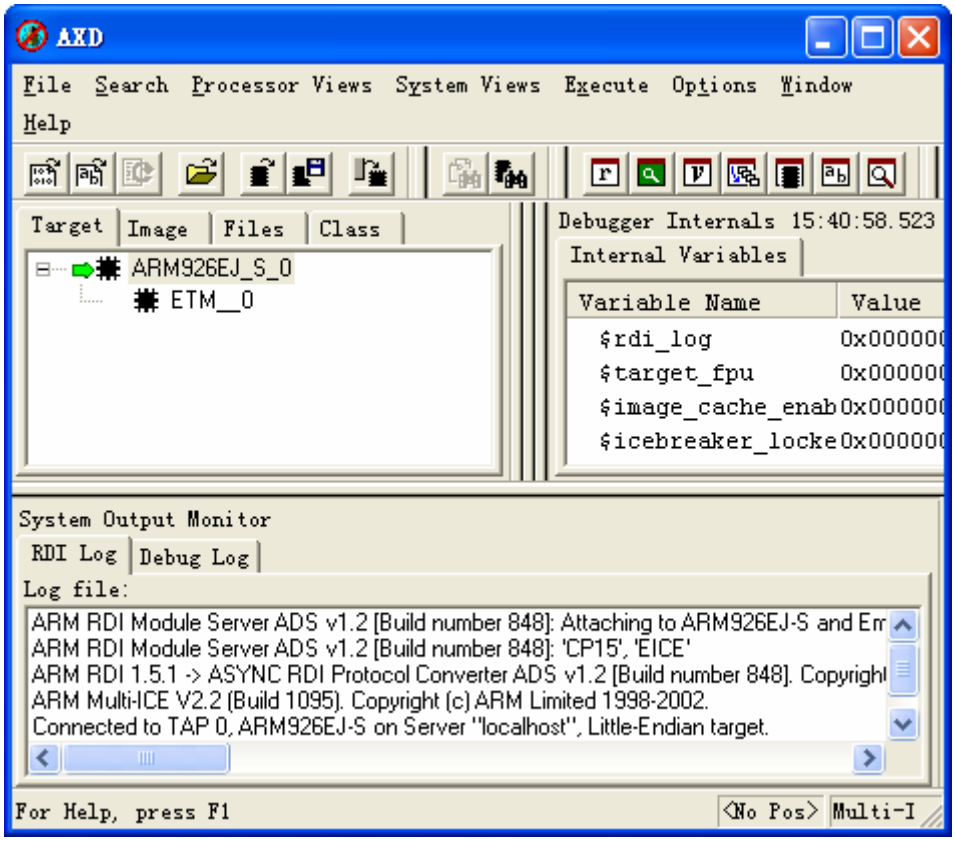
ARM developer Suite v1.2 是由 ARM 公司提供的 AXD Debugger 安装程序。安装前，请先阅读 ARM 的相关文档，然后安装 ARM developer Suite v1.2。

运行 AXD Debuger

运行 AXD Debugger 后，AXD Debugger 启动如图 5-4 所示。

如果是第一次启动 AXD Debugger，此时没有连接任何 ICE 设备，因此需要配置 AXD Debugger 的 target。如果之前已经启动过，并且已连接了 ICE 设备，不需要再配置 AXD Debugger 的 target。target 选择 Multi-ICE。如果要使用硬件调试，需要添加 Multi-ICE target。

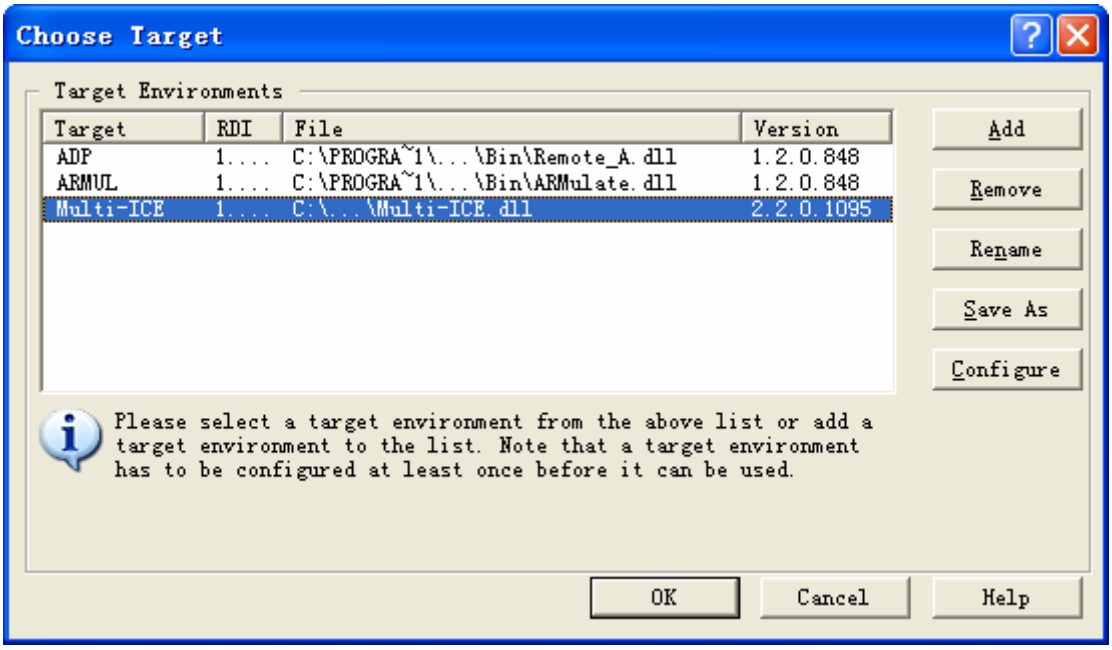
图5-4 AXD Debugger 窗口



选择 Target

选择“options/config target”菜单项，弹出如图 5-5 所示的窗口。在图中已经添加了 Multi-ICE。

图5-5 Choose Target 窗口



添加 Multi-ICE

单击“Add”按钮，在安装路径下找到并添加..\Multi-ICE\Multi-ICE.dll，即如图 5-5 所示。

单击“Configure”按钮，弹出如图 5-6 所示的窗口。

单击“OK”缺省按钮，弹出如图 5-7 所示的窗口，表示配置成功。

单击“确定”按钮，此时 Multi-ICE server 与 AXD Debugger 联系上了，可以使用 Multi-ICE 工具了。

图5-6 Mutil-ICE 配置信息提示框

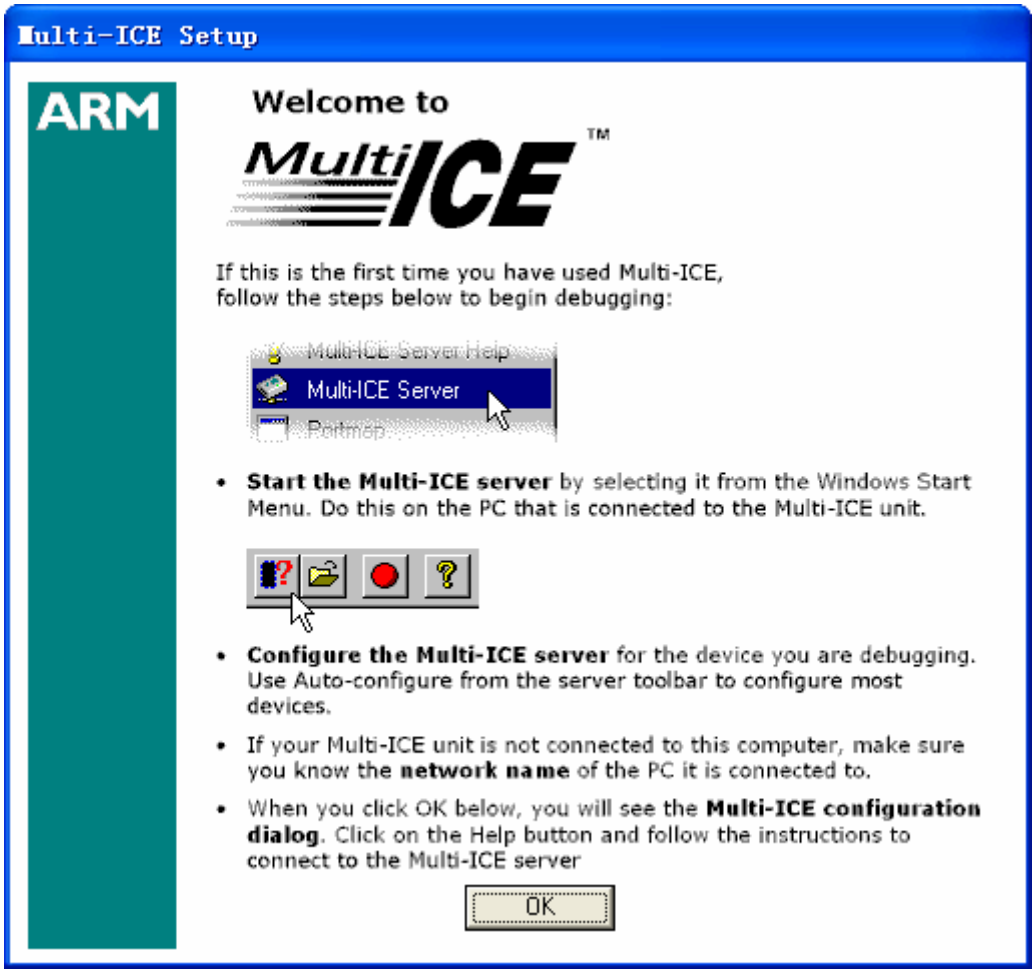


图5-7 Target 配置窗口