2006年2月

IAR EWARM 快速入门

(V1.0)

万利电子有限公司

www.manley.com.cn

万利电子有限公司
南京市新模范马路17号02幢二层(210003)
电话：025-83235502
传真：025-83235501

目　　录

编者序

IAR Embedded Workbench for ARM 是 IAR Systems 公司为 ARM 微处理器开发的一个集成开发环境（下面简称 IAR EWARM）。比较其他的 ARM 开发环境，IAR EWARM 具有入门容易、使用方便和代码紧凑等特点。故在这里介绍给打算学习使用或正在使用 ARM 芯片的朋友们共同探讨。

IAR Systems 公司目前推出的最新版本是 IAR Embedded Workbench for ARM version 4.31，并提供一个 32k 代码限制、但时间限制长达 25 年的免费评估版。有兴趣的朋友可以到 IAR 公司的网站 www.iar.com/ewarm 或万利电子有限公司的网站 www.manley.com.cn (本地网站)去寻找和下载。

IAR EWARM 中包含一个全软件的模拟程序（simulator）。用户不需要任何硬件支持就可以模拟各种 ARM 内核、外部设备甚至中断的软件运行环境。从中可以了解和评估 IAR EWARM 的功能和使用方法。

我们编译整理的这本快速用户指南采用评估版软件安装目录 C:\Program files\IAR System\Embedded workbench 4.0\ARM\tutor 下的教程为例，一步一步介绍 IAR EWARM 的使用方法。该教程采用了两个 C 语言程序，tutor.c 和 utilities.c。它们不和任何特定的硬件关联，所以介绍中的全部操作都是用模拟程序完成的。在以后的章节里，我们将具体介绍 EWARM 软件及配套硬件工具、如何使用 EWARM 集成开发环境以及在 EWARM 下烧写 Flash 的方法。

如果用户希望在真实的目标板上进行代码运行和调试，请到万利电子有限公司全国各直销点购买 IAR 的 JTAG 仿真器 J-Link。

宋祈真

2006 年于南京

附：EWARM 的学习步骤

① 下载安装 EWARM 32K 学习版软件；

② 进入 www.iar.com/ewarm -> Online Demos，下载 Flash 格式的软件使用动画演示；

③ 以本入门手册结合软件使用的动画演示，进行软件使用的入门学习；

④ 在软件安装目录...\arm\src\examples 下，寻找感兴趣的芯片例程学习；

⑤ 可选项 1：购买 J-Link 仿真器和开发板，实现在硬件上的代码运行和调试；

⑥ 可选项 2：购买由北航出版社的《IAR EWARM 嵌入式系统编程与实践》一书，深入学习；

⑦ 学习“可选项 1 或 2”的随附光盘中《Converting ADS Projects to EWARM Projects》白皮书，实践如何移植一个 ADS 工程到 EWARM 格式的工程。

第一章 EWARM 集成开发环境及配套仿真器

IAR Embedded Workbench for ARM version 4.31 是一个针对 ARM 处理器的集成开发环境，包含项目管理器、编辑器、编译连接工具和支持 RTOS 的调试工具，在该环境下可以使用 C/C++和汇编语言方便地开发嵌入式应用程序。IAR EWARM 的主要模块如下：

- 项目管理器
- 功能强大的编辑器
- 高度优化的 IAR ARM C/C++ Compiler
- IAR ARM Assembler
- 1 个通用的 IAR XLINK Linker
- IAR XAR 和 XLIB 建库程序和 IAR DLIB C/C++运行库
- IAR C-SPY 调试器（先进的高级语言调试器）
- 命令行实用程序

以下介绍一下 EWARM 4.31 版本及其相关配套硬件的一些特点：

1. IAR EWAM 软件的特点
   ① EWARM 4.31 版基本特点
   - 完善的 ARM 内核支持
     - 最新支持到 ARM11 内核（ARM1136J，ARM1136J-S，ARM1136JF，ARM1136JF-S）
     - 早已支持的其他 ARM 内核
       - ✓ ARM7（ARM7TDMI，ARM7TDMI-S，ARM720T）
       - ✓ ARM9（ARM9TDMI，ARM920T，ARM922T，ARM940T，ARM9E，ARM9E-S，ARM926EJ-S，ARM946E-S，ARM966E-S，ARM968E-S）
       - ✓ ARM10（ARM10E，ARM1020E，ARM1022E，ARM1026EJ-S）
       - ✓ XScale（XScale，XScale-IR7）
   - 更加客户化地提供芯片级的支持
     - 完备的各厂商 ARM 处理器的 C/C++和汇编语言外设寄存器定义文件
       支持的芯片厂商有 Analog Devices、ARM、Atmel、Cirrus Logic、Freescale、Intel、NetSilicon、OKI、Philips、Samsung、Sharp、ST 和 TI
     - 支持 Analog Devices、Atmel、Freescale、OKI、Philips、ST 和 TI 等厂商的 ARM 处理器的 Flash Loader 程序
     - 软件集成了 200 余个代码例程，对应于各种不同的芯片，位于...\arm\src\examples 目录下
   - 进一步改进了编译器速度优化，重写了的浮点运算库

- 对更多嵌入式操作系统的支持
  - 新增支持 OSEK 类操作系统的 OSEK Run-Time Interface (ORTI)
  - 新增支持 OSE Epsilon RTOS 的 Kernel Awareness 调试
  - 新增支持 embOS、SMX、NORTi 等的支持
- 调试器的增强功能
  - 对堆栈运行的监测功能
  - 配合 IAR J-Link 仿真器的新增功能
    - ✓ J-Link TCP/IP 服务器
    - ✓ 调试器和 IAR J-Link 仿真器协同配合，实现对 ARM 处理器的多核调试
  - 对 IAR J-Trace 仿真器提供全面的支持
  - 在 C-SPY 模拟器中可执行 Trace 的模拟
  - 支持同一芯片上多颗 Flash 的 Flash Loader 程序，以及通用的 Flash Loader 开发指南

② EWARM 软件在芯片级支持方面的特色

- 完备的各厂商 ARM 处理器的 C/C++和汇编语言外设寄存器定义文件
- 大量适合于嵌入式代码的编程语言扩展特性，包括存储器关键字，本征函数，中断函数，存储器映射 I/O 等
- 针对评估板的例程，包含 IAR、Analog Devices、Aiji System、ARM、Atmel、Cirrus Logic、Freescale、Keil、OKI、Olimex、Pasat、Philips、Phytec、ST 和 TI 等厂家的开发板
- 支持 ARM 或 Thumb 模式下大至 4G 字节的应用程序
- 每个函数都能选择在 ARM 或 Thumb 模式下编译
- 可生成 VFP 向量浮点协处理器代码
- 支持 Analog Devices、Atmel、Freescale、OKI、Philips、ST 和 TI 等厂商的 ARM 处理器的 Flash Loader 程序
- 支持 ARM Angel Debug monitor

③ EWARM 编译器的软件特色

- 先进的通用编译器优化和针对特定处理器的速度优化及存储器优化功能
- 轻量运行库，用户可以根据需要自行配置，提供全部源代码
- 灵活的存储器控制，允许详细地为代码和数据分配地址
- 去除不需要的函数和变量
- C/C++变量和函数连接时全局类型检查
- 可选的校验和生成功能，用于运行时映象校验
- 自动将代码和数据放置到非连续的存储器区域
- 强大的可重定位宏汇编器，支持丰富的命令集和操作符

④ EWARM 调试器的软件特色

- 完全集成的源代码和反汇编程序调试器
- 非常细化的执行控制（函数调用级步进）
- 复杂的代码和数据断点
- 丰富的数据监视功能
- Locals，Watch，Auto，Live Watch 和 Quick Watch 等变量查看窗口
- 寄存器和存储器查看窗口
- 支持 STL 容器
- C/C++调用栈窗口，同时还可以显示将要进入的函数
- 双击调用链上的任何函数将更新编辑器、局部变量、寄存器、变量查看和反汇编窗口，以显示在该函数调用时的状态
- 跟踪功能，可以检查执行的历史记录。在跟踪窗口中移动时将更新编辑器和反汇编窗口以显示合适的位置
- 控制台 I/O 仿真
- 中断和 I/O 模拟仿真
- 类似 C 语言的宏系统，可扩充调试器的功能
- 由主机执行的应用程序系统调用仿真
- 代码覆盖率和执行时间分析工具
- 通用的 Flash Loader 程序及开发指南
- 同时支持多颗 Flash 的 Flash Loader 程序
- 支持 OSEK Run-Time Interface (ORTI)
- 提供为调试器扩充第三方功能的软件开发包，如 RTOS 调试扩充和仿真器驱动扩充
- 命令行调试工具

⑤ IAR C-SPY 支持的调试方法

- IAR J-Link JTAG 接口（支持所有 ARM7 和 ARM9 核，通过 USB 或 TCP/IP 连接）
- IAR J-Trace JTAG 接口（支持所有 ARM7 和 ARM9 核，通过 USB 或 TCP/IP 连接）
- RDI 接口类的第三方仿真器（Abatron BDI1000 & BDI2000, EPI Majic, Ashling Opella, Aiji OpenICE, Signum JTAGjet, ARM Multi-ICE 等）
- Macraigor Wiggler, Raven, mpDemon 和 USBdemon 等调试接口
- EPI Jeeni 仿真器支持
- IAR 的 ROM-Monitor
- ARM 公司的 Angel ROM-Monitor（用于 Atmel 和 Cirrus Logic 的评估板）

⑥ IAR 对嵌入式实时操作系统的 Kernel Awareness 调试支持

操作系统	IAR EWARM 内置的插件	由第三方 RTOS 厂商提供的插件
CMA-RX	X	
CMX-Tiny+	X	
uC/OS-II		X
ThreadX	X	
RTXC Quadros		X
Fusion RTOS		X
OSEK(ORTI)	X	
ENEA OSE Epsilon	X	
MiSPO NORTi		X
Micro Digital SMX		X
Segger embOS	X	

每种 RTOS 插件都会在 C-SPY 中安装一批新的窗口，其中最重要的是任务或线程列表窗口，在此窗口中可以在指定的任务上设置断点和执行程序。其它不同的监测窗口可以显示 RTOS 内部数据结构的内容，例如定时器、队列、信号量、资源和邮箱等。

⑦ EWARM 图形化的集成开发环境的界面特色

- 分层次的工程组织
- 同一工作空间中允许存放多个工程
- 可停靠的窗口和多视图
- 源代码浏览
- 创建和维护库的工具
- 可以和源代码控制系统相集成
- 文本编辑器
  - 支持多字节字符（汉字）
  - 上下文相关的帮助系统
  - 根据句法着色
  - 无限制的 undo/redo
  - 搜寻、替换和增量搜寻
  - Go to
  - 书签
  - 错误标签：查阅前一个/下一个

- 自动括号配对
- 智能缩排
- 类似网页浏览器的前向/后向源码查阅
- 代码断点的设置/清除/使能/禁止

➢ 命令行编译连接工具

⑧ EWARM 的编程语言和标准

➢ 遵循 ISO/ANSI C94（带有一些从 C99 标准中挑选的特性）标准的 C 编程语言
➢ 嵌入式 C++扩展，支持模板、多重继承和虚拟继承、名字空间以及其它不增加执行时间或存储器开销的 C++特性。完整的嵌入式 C++库还包含字符串、流等特性，以及标准模板库（STL）
➢ IEEE-754 浮点运算规则
➢ MISRA C 检查器
➢ 支持大量工业标准的调试和映象文件格式（如 ELF/DWARF），与大多数常见的调试器和仿真器兼容

⑨ 用户帮助

➢ 完备的例程和工程模板。
➢ 上下文相关的联机帮助系统，带有库函数查阅功能
➢ 印刷好的用户指南，带有详细的 step-by-step 教程
➢ 友好、详尽和精确的错误信息和警告信息

2．IAR J-Link 仿真器简介

IAR J-Link 是 IAR 为支持仿真 ARM 内核芯片推出的 JTAG 方式仿真器。配合 IAR EWARM 集成开发环境支持所有 ARM7/ARM9 内核芯片的仿真，无需安装任何驱动程序与 EWARM 集成开发环境无缝连接，操作方便、连接方便、简单易学是学习开发 ARM 最好最实用的开发工具。

同时，最近的有关权威测试显示，J-Link 目前是同类产品中下载调试速度最快的 J-Tag 仿真器：

公司	产品	通讯接口	支持内核	下载速度	对开发板供电功能	备注
Macraigor	Wiggler	LPT	ARM7/9	16 KB/秒	无	即并口仿真头
Keil	U-Link	USB	ARM7	28 KB/秒	无	
IAR	J-Link	USB 2.0	ARM7/9	600 KB/秒	有	

① J-Link ARM 主要特点

➢ IAR EWARM 集成开发环境无缝连接的 JTAG 仿真器
➢ 支持所有 ARM7/ARM9 内核的芯片，包括 Thumb 模式

- 下载速度高达 600 kB/s
- 最高 JTAG 速度 12 MHz
- 目标板电压范围 1.2V – 3.3V
- 自动速度识别功能
- 监测所有 JTAG 信号和目标板电压
- 完全即插即用
- 使用 USB 电源
- 带 USB 连接线和 20 芯扁平电缆
- 支持多 JTAG 器件串行连接
- 标准 20 芯 JTAG 仿真插头
- 选配 14 芯 JTAG 仿真插头
- 选配用于 5V 目标板的适配器
- 带 J-Link TCP/IP server，允许通过 TCP/ IP 网络使用 J-Link

② IAR J-Link 的物理连接

J-LINK 一端通过 USB 口与 PC 连接，另一端通过标准 20 芯 JTAG 插头与目标板连接。建议首先连接 J-LINK 到 PC,再连接 J-LINK 到目标系统，最后给目标系统供电（如果目标系统为独立供电、而非由 J-TAG 口供电的情况)。

③ IAR J-Link 主要技术指标

功耗	吸取 USB 供电电力< 50 mA
通讯方式	USB 2.0 全速
目标板接口	20 芯 JTAG 口（14 芯 JTAG 口选件）
J-Link 和 ARM 间串行传输速率	最高 12 MHz
支持目标电压	1.2 – 3.3 V （5V 适配头选件）
工作温度	+5 C - +60 C
储存温度	-20 C - +65 C
相对湿度（无冷凝水）	< 90% RH
体积	100mm x 53mm x 27mm
重量（不含电缆）	70 克
电磁兼容性（EMC）	EN 55022，EN55024

④ 目标板5V电源适配器选件

当目标系统为 5V 电源系统时，必须使用 J-LINK 提供的 5V 电源适配器选件。对于 1.2V~3.3V 电源系统，可以直接使用 J-Link。使用时将适配器的 20 芯 IDC 插头插进 J-Link 的 20 芯插座，再将连接目标的 20 芯扁平电缆插进适配器的插座。

5V 适配器选件由目标供电(3.3V~ 5V)，电流<20mA，有一个 LED 指示电源状态。

⑤ JTAG插头定义

J-Link 的 JTAG 20 芯的 IDC 插头与 ARM 公司的仿真器插头定义兼容，有关定义如下：

引脚	名称	方向	功 能 描 述
1	VTref	Input	目标系统参考电压。 用于检查目标系统是否供电，并产生一个逻辑电平送给 J-Link 内部比较器。检测结果用来控制输出给目标的逻辑电平幅度。此引脚通常与目标的 Vdd 联，中间不允许串接电阻。
2	Vsupply	NC	J-Link 不用此引脚，在目标系统中连接到 Vdd 或开路。
3	nTRST	Output	JTAG 复位，J-Link 输出给目标的 Reset 信号。 通常连接到目标 CPU 的 nTRST 引脚。目标板上应将此脚上拉到高电位，避免意外复位。
4	GND	-	公共地。
5	TDI	Output	J-Link 输出给目标 CPU 的 JTAG 数据。 通常与目标 CPU 的 TDI 引脚相连。建议在目标板上将此脚上拉到 Vdd。
6	GND	-	公共地。
7	TMS	Output	J-Link 输出给目标 CPU 的 JTAG 模式设置信号。 通常与目标 CPU 的 TMS 引脚相连。建议在目标板上将此脚上拉。
8	GND	-	公共地。
9	TCK	Output	J-Link 输出给目标 CPU 的 JTAG 时钟信号。 通常与目标 CPU 的 TCK 引脚相连。建议在目标板上将此脚上拉。
10	GND	-	公共地。
11	RTCK	Intput	目标 CPU 提供给 J-Link 的测试时钟信号。 有些目标要求 JTAG 的输入与其内部时钟同步。J-Link 利用此引脚的输入可动态地控制自己的 TCK 速率。 若不使用此功能，在目标板上将此脚接地。
12	GND	-	公共地。
13	TDO	Intput	目标 CPU 返回给 J-Link 的数据信号。 通常与目标 CPU 的 TDO 引脚相连。
14	GND	-	公共地。
15	RESET	I/O	目标 CPU Reset 信号
16	GND	-	公共地。
17	DBGRQ	NC	J-Link 不用此引脚，在目标系统中将此引脚开路。
18	GND		公共地。
19	Vdd	Output	+3.3V 电源输出。
20	GND		公共地。

⑥ JTAG速度

J-Link 有两种速度设定，即固定 JTAG 速度、自动 JTAG 速度，该功能选项位于 Project Options -> Debugger -> J-Link 设置页面中。

❖ 固定 JTAG 速度

目标被锁定在固定时钟速度。目标能执行的最大 JTAG 速度取决于目标自身。一般来讲，不带 JTAG 同步逻辑的 ARM 内核（如 ARM7-TDMI）能执行与 CPU 速度相当的 JTAG 速度。而带 JTAG 同步逻辑的 ARM 内核（例如 ARM7-TDMI-S，ARM946E-S，ARM966EJ-S）能执行相当 CPU 速度 1/6 的 JTAG 速度。JTAG 速度不应超过 10 MHz。

❖ 自动 JTAG 速度

由 TAP 控制器选择最大的 JTAG 速度。要注意，不带同步逻辑的 ARM 内核可能会工作不稳定。因为 CPU 内核时钟可能慢于最大 JTAG 速度。

附：J-Link 全国分销点

地区	销售点	地 址	电 话
华北	北京万利	北京海淀区知春路 118 号知春电子城 B193 柜	010-62562744 / 62526647
	天津万利	天津南开区鞍山西道 323 号增 1 号	022-27376292 / 27471810
华南	深圳万利	深圳深南中路赛格电子市场 3 楼 3B35 柜	0755-83681644 / 83681644
		深圳深南中路华强电子世界一楼 20A257 柜	0755-83687350 / 83665281
	广州万利	广州天河区天河路龙苑大厦 3 栋 506 室	020-87588300 / 87543761
华东	上海万利	上海北京东路赛格电子市场 2A19-2A20 柜	021-53081472 / 53082644
		上海太平洋电脑城三楼 347 室	021-54904533 / 54901862
	南京万利	南京珠江路雄狮电子商城 A529 柜	025-83615784 / 83675529
		南京中山东路 110 号华龙电子商城二楼 96＃柜	025-84412638 / 84412638
	杭州万利	杭州登云路 639 号杭州电子市场 1C205 柜	0571-89901205
西北	西安万利	西安西部电子商城 2 楼 C 区 2C033-035	029-88221873 / 88270877
东北	沈阳万利	沈阳三好街 90 号甲百脑汇科技广场 B 区-W35	024-83991288 / 83990602
华中	武汉万利	武汉武昌珞瑜路 158 号华中数码城 3098 室	027-87654225
	长沙万利	长沙人民路 9 号百脑汇商城二楼 H23-25	0731-4175141 / 4175141
香港	香港万利	香港九龙上海街 67 号 10 楼	00852-27303434

第二章 在 EWARM 中生成一个新项目

EWARM 是按项目进行管理的，它提供了应用程序和库程序的项目模板。项目下面可以分级或分类管理源文件。允许为每个项目定义一个或多个编译连接（build）配置。在生成新项目之前，必须建立一个新的工作区（Workspace）。一个工作区中允许存放一个或多个项目。

另外用户最好建立一个专用的目录存放自己的项目文件。例如在本指南中我们生成一个 C:\Program files\IAR System\My project 目录。现在双击桌面上的 IAR Embedded Workbench 图标，出现 IAR EWARM 开发环境窗口。

1. 生成新的工作区（Workspace）

   选择主菜单 File > New > Workspace 生成新工作区。

2. 生成新项目

   ① 选择主菜单 Project > Create New Project，弹出生成新项目窗口，见图 1。

   本例选择项目模板（Project template）中的 Empty project。

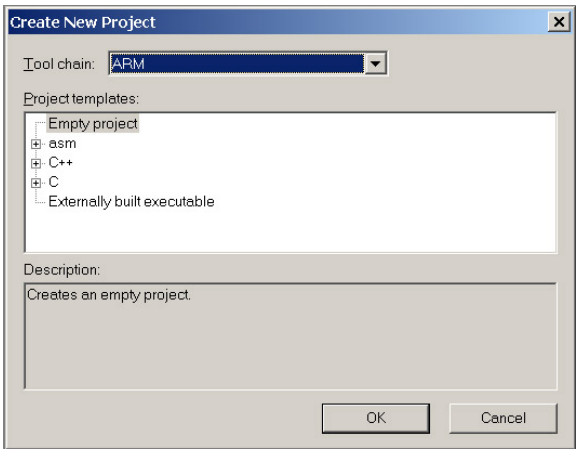


图 1. 生成新项目窗口

   ② 在 Tool chain 栏中选择 ARM，然后点击 OK 按钮。

   ③ 在弹出的另存为窗口中浏览和选择新建的 My projects 目录，输入文件名 project1，然后保存。这时在屏幕左边的 Workspace 窗口中将显示新建的项目名。见图 2 所示：

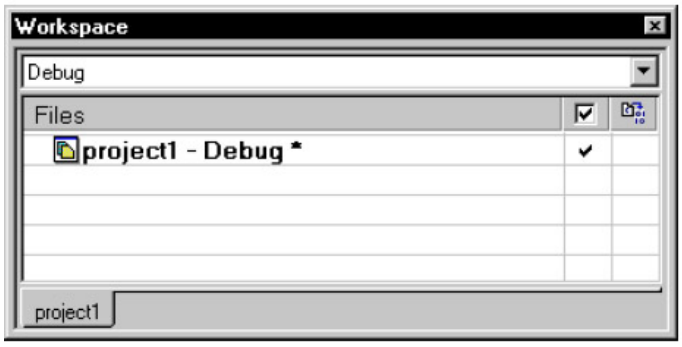


图 2. Workspace 窗口

IAR EWARM 提供两种缺省的项目生成配置，即 Debug 和 Release。本例在 Workspace 窗口顶部的下拉菜单中选取 Debug。现在 My projects 目录下已生成一个 project1.ewp 文件。该文件中包含与 project1 项目设置有关的信息，如 build 选件等。项目名后缀上的＊号表示该工作区有改变但还没有被保存。

本例调用 printf 库函数，这是在 C-SPY 模拟器中的一个低级 write 函数。如果用户希望在真实硬件上以 release 配置运行例子，就必须提供与硬件相适配的 write 函数。

④ 保存工作区

先选择主菜单 File＞Save Workspace，浏览并选择 My projects 目录。然将工作区取名为 tutorials 输进 File name 输入框，按保存按钮退出。这时在 My projects 目录下将生成一个 tutorials.eww 文件，该文件中保存了用户添加到 tutorials 工作区中的所有项目。窗口和断点放置等与当前操作有关的其他信息则被存储在 My projects\ settings 目录下的文件中。

3. 给项目添加文件

本例我们将采用 arm\tutor 目录下的两个源文件，*Tutor.c* 和 *Utilities.c*。

Tutor.c 是一个只用到标准 C 语言的简单程序。它用 Fibonacci 数列的前十个数初始化一个数组，并把结果打印到 stdout；*Utilities.c* 包含计算 Fibonacci 数列的实用程序。

IAR EWARM 允许生成若干个源文件组。用户可以根据项目需要来组织自己的源文件。但在本例中没有必要。

① 在 Workspace 中选择希望添加文件的目的地，可以是项目或源文件组。本例直接选 project1。

② 选择主菜单 Project > Add Files 打开标准浏览窗口，见图 3。选择安装目录 ARM\tutor 下的上述 2 个文件，点击打开按钮，把它们添加到 Project1 目录下。

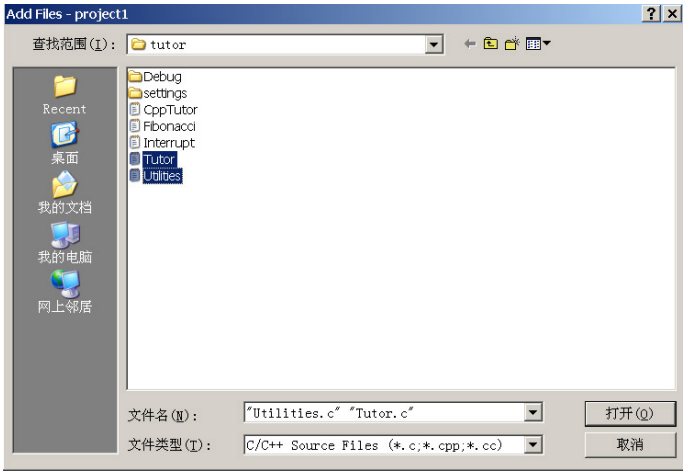


图 3. 添加文件窗口

4. 设置项目选件

生成新项目和添加文件后就应该为项目设置选件。IAR EWARM 允许为任何一级目录和文件单独设置选件，但是用户必须为整个项目设置通用的编译连接（build）选件。

① 选择通用选件

选中 Workspace 中的 project1 – Debug，然后选择主菜单 Project > Options。也可以先选择 project1 – Debug，然后选择鼠标右键命令中的 Options。

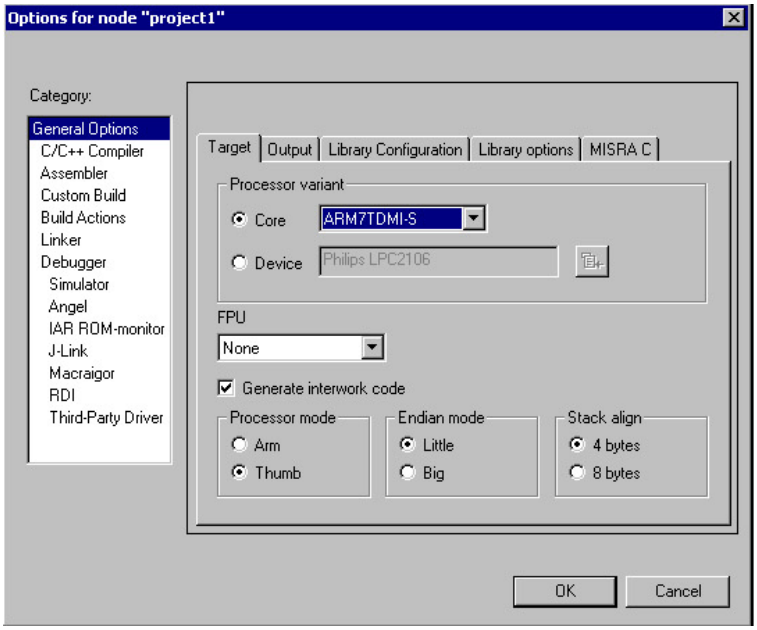


图 4. 项目通用选件窗口

在打开的 Options 窗口左边的 Category 中选择 General Options。然后分别在：

- Target 页面/Core 条目下选择 ARM7TDMI-S
- Output 页面中，Output file 条目下选择 Executable
- Library Configuration 页面中，Library 条目下选择 Normal

② 选择编译器选件

在 Options 窗口的 Category 中选择 C/C++ Compiler，见图 5。

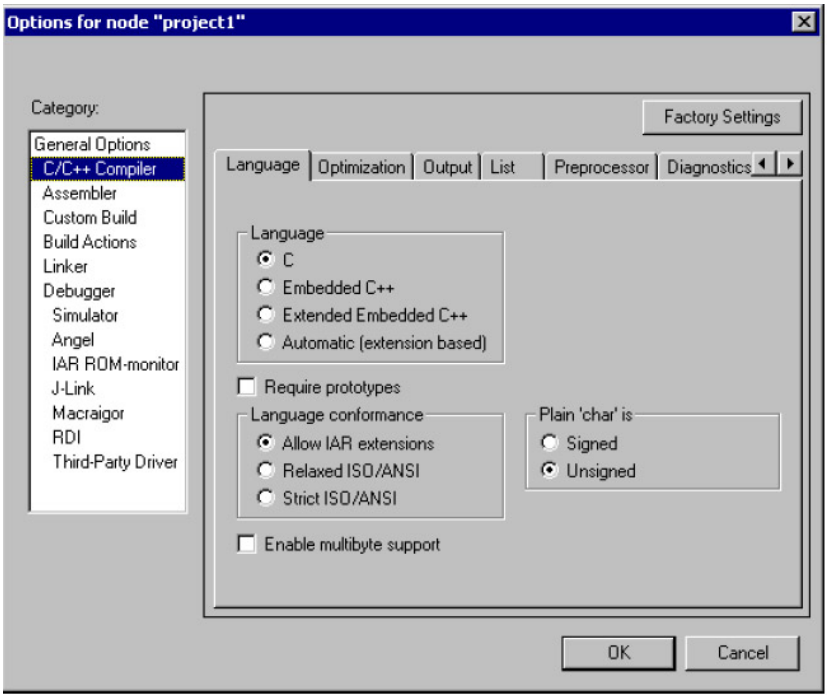


图 5. C/C++ Compiler 选件窗口

然后在：

- Language 页面中，选择 C，Allow IAR extensions 等
- Optimization 页面中，选择 Generate debug information
- Output 页面中，选择 Output list file 和 Assemble mnemonics
- List 页面中，选择 Output list file。并选择 Assembler mnemonics 和 Diagnostics
- 点击 OK 按钮，确认选择的选件

在设置项目选件窗口中有许多其他信息。由于本例比较简单，所以不涉及这些内容。

第三章 编译和连接应用程序

这一步编译和连接（build）项目程序，同时生成一个编译器列表文件（compiler list file）和一个连接器存储器分配文件（linker map file）。

1．编译源文件

① 选中 workspace 中 *utilities.c* 文件。

② 选择主菜单 Project > Compile，或工具条中的 Compile 按钮，或按右键后选择 Compile 命令。编译结束后在消息窗口中出现如图 6 中的信息。

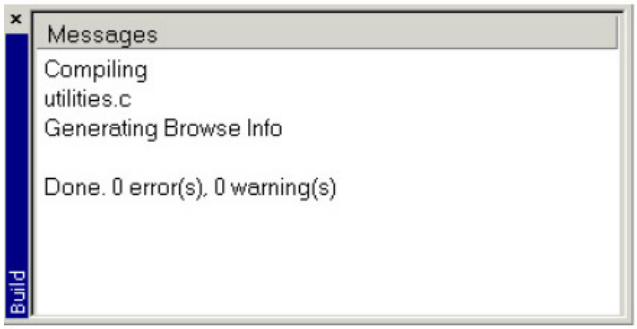


图 6. Build 窗口中的编译处理消息

③ 用同样的方法编译 tutor.c。

编译完成后在 My projects 目录下将生成一批新子目录。因为我们在建立新项目时选择 Debug 配置，所以在 My projects 目录下自动生成一个 Debug 子目录。Debug 子目录下又包含另 3 个子目录，名字分别为 List、Obj、Exe。它们的用途如下：

- List 目录存下放列表文件，列表文件的后缀是 lst；
- Obj 目录下存放 Compiler 和 Assembler 生成的目标文件，这些文件的后缀为 r79，可以用作 IAR XLINK 连接器的输入文件；
- Exe 目录下存放可执行文件，这些文件的后缀为 d79，可以用作 IAR C-SPY 调试器的输入文件，注意在执行连接处理之前这个目录是空的。

点击 project1 – Debug 前面的+号将目录展开。你可以从自动生成的 Output 目录中看到所有生成的输出文件名以及反映相互依赖关系的的头文件名。

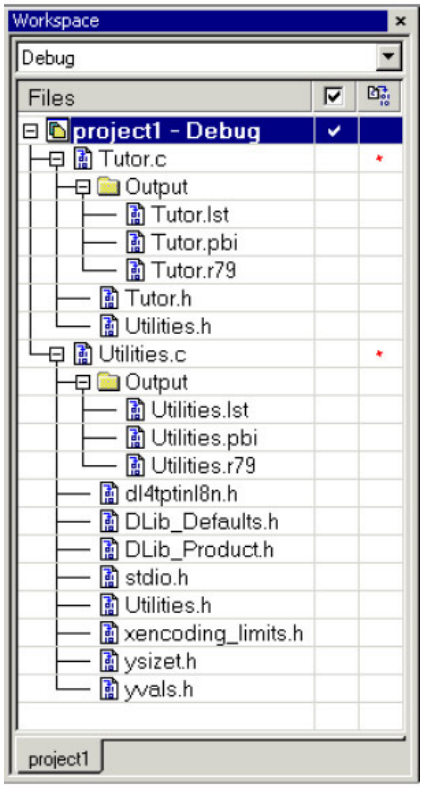


图 7. 编译处理后的文件结构

2. 查看编译器列表文件

现在我们通过改变编译器选件中的优化级别（Optimization）来观察 list 文件是如何自动更新生成的代码量的。

① list 文件的结构

双击 Workspace 窗口中的 *Utilities.lst*，打开 list 文件，它包含以下信息：

- 文件头 — 显示编译器的版本信息，列表文件生成时间，source 文件、list 文件和 object 文件的名字和路径，编译命令行及选件等信息。
- 文件体 — 显示为每条源语句生成的汇编代码和二进制代码，以及变量如何被分配到不同的段。
- 文件尾 — 显示所需的堆栈、程序代码以及数据存储器的总量，同时报告错误和警告信息。

② 选择主菜单 Tools > Options 弹出 IDE Options 对话窗口，选择 Editor 页面。选择 Scan for Change Files 选件。此选件将自动打开编辑窗口中的文件，目前是 *Utilities.lst* 文件。按 OK 按钮。

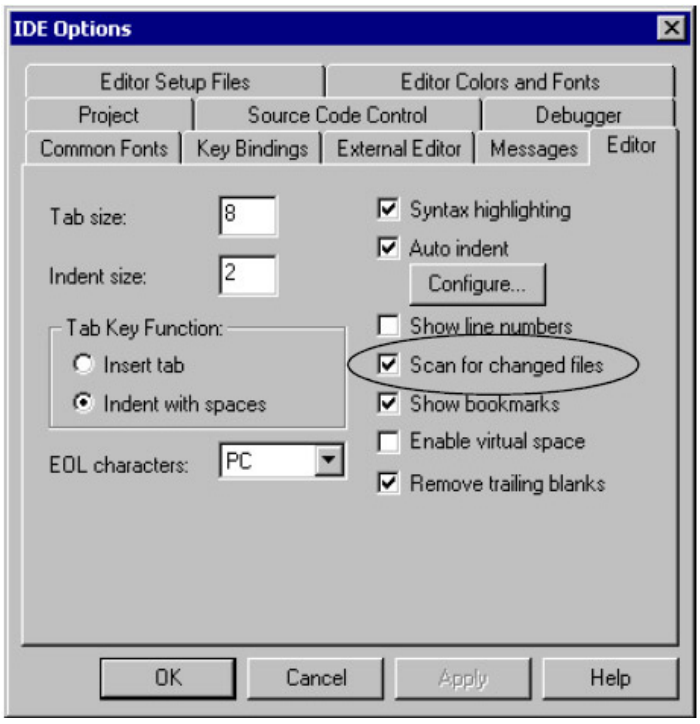


图 8. IDE Option 窗口

③ 选中 Workspace 窗口中的 Utilities.c，按鼠标右键选择弹出框中的 Options…。从弹出的对话框左边的 Category 中选择 C/C++ Compiler 并确定 Override inherited settings。打开 Optimization 页面，把优化级别从 None 改定为 High。然后按 OK 按钮。

④ 重新编译 *Utilities.c*，请注意这时编辑窗口中的 *Utilities.lst* 文件已经自动被刷新。文件尾显示的代码大小已经因优化级别的升高而减小。

⑤ 对本例而言，Optimization 应选择 None。所以在连接处理前应该将优化级别恢复到原来的设置。这时应选中 *Utilities.c*，按鼠标右键选择弹出框中的 Options…。选择 C/C++ Compiler 并取消 Override inherited settings。然后重新编译 *Utilities.c*。

3．连接应用程序

① 先选中 Workspace 窗口中的 Project1 – Debug，然后选择主菜单 Project > Options，弹出 Options 对话窗口，见图 9。在左边的 Category 中选择 Linker，显示 IAR XLINK 的各选件页面。

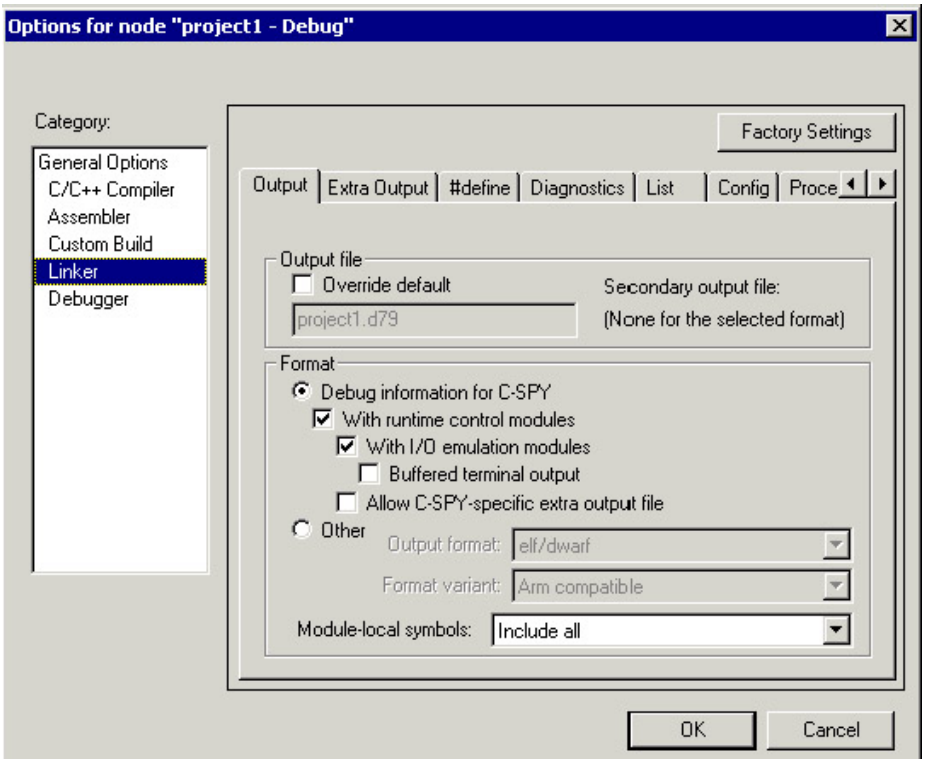


图 9. XLINK 参数选件窗口

本例全部采用缺省的连接处理选件。但是仍需要强调一下输出文件格式和 Linker 命令行文件的选择方法：

❖ 输出格式

选择合适的输出格式十分重要。你可能需要将输出文件送给一个调试器进行调试，这时就要求输出格式带有调试信息。本例采用适合 C-SPY 调试器的缺省输出选件，它们是 Debug information for C-SPY、With runtime control modules 和 With I/O emulation modules。指示需要连接将 stdin 和 stdout 指向 C-SPY 的 I/O 窗口的低级例程。

如果用户希望把应用下载到一个 PROM 编程器时，则其输出格式不需要带调试信息，如 Intel-hex 或 Motorola S-records。

在 list 页面中选择 Generate Linker listing 和 Segment map（见图 10）。允许生成存储器分配 MAP 文件。

注）本例连接器命令文件中的定义不与任何特定的硬件相关联。EWARM 提供的连接器命令文件模板都可以在模拟器（simulator）中使用。但是如果要把它们用于目标系统时必须与实际的硬件存储器分布相适配。用户可以从…src\ examples 目录中找到与评估板相关的连接器命令

文件。

❖ 连接器命令文件

在连接器命令文件中，用于段（segment）控制的 XLINK 命令行是用来放置段的。熟悉连接器命令文件和段的放置十分重要。用户可以从 ARM IAR C/C++ Compiler Reference Guide 中了解更多信息。

本例使用缺省的连接器命令文件，请见图 9 或图 10 中的 Config 页面。

用户如果要检查连接器命令文件，需用合适的文本编辑器，例如 IAR EWARM 的编辑器。也可以打印出来，检查各项定义是否符合要求。

② 点击 OK 按钮保存 IAR XLINK 选件

③ 选择主菜单 Project > Make 或鼠标右键 Make 命令，连接目标文件，生成可执行代码。

Build 消息窗口中将显示连接处理的消息。连接的结果将生成一个带调试信息的代码文件 *project1.d79* 和一个存储器分配（MAP）文件 *project1.map*。

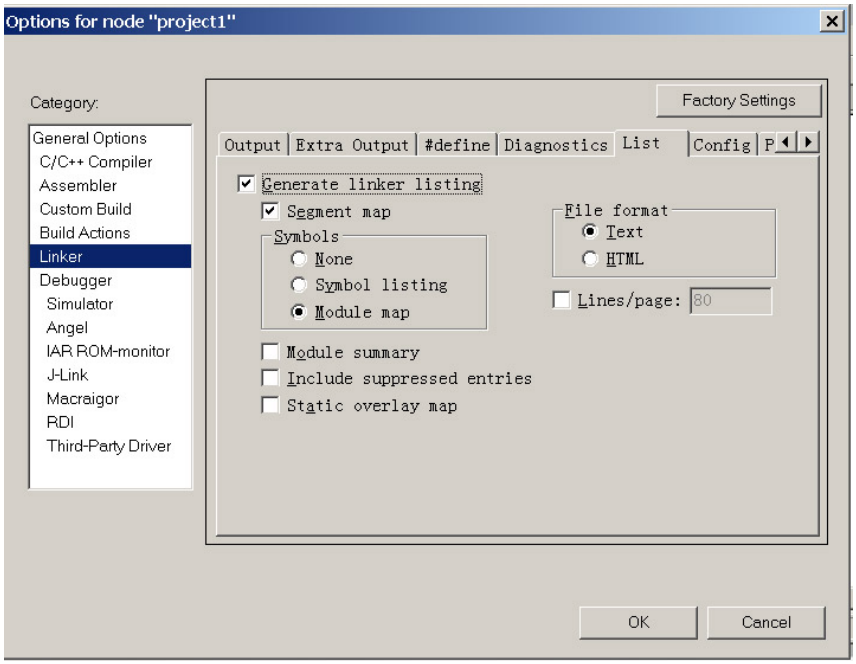


图 10. XLINK 选件中的 list 页面

4．查看MAP文件

双击 Workspace 中的 *project1.map* 文件名，编辑器窗口中将显示该 MAP 文件。从 MAP 文件中我们可以了解以下内容：

- 文件头中显示连接器版本，输出文件名以及连接命令使用的选件。

- CROSS REFERENCE 段显示程序入口地址。
- RUNTIME MODEL 段显示使用的运行时模块的属性。
- MODULE MAP 段显示所有被连接的文件。每个文件中，作为应用程序一部分加载的有关模块的信息，包括各段和每个段中声明的全局符号都列出来。
- SEGMENTS IN ADDRESS ORDER 段列出了组成应用程序的所有段的起始地址和结束地址，字节数，类型和对齐标准等。
- END OF CROSS REFERENCE 段落显示总的代码和数据字节数。

到此为止，已经生成 project1.d79 应用程序并可以用于在 IAR C-SPY 中调试。

新闻：万利电子有限公司成为 IAR 公司 8051 开发工具代理商

2 月 23 日，中国上海

IAR 公司今天宣布万利电子成为其 8051 开发工具代理商，并在其最新发布的 7.20c 版的 EW8051 集成开发环境中，无缝集成了万利 Insight 仿真器的系列驱动。双方在软硬件技术上的互补能力，使得该工具包比同类竞争产品**代码效率提高 15%-20%**，并实现**完全实时**的在线调试。

双方同时联合宣布在中国发布一套市场售价为人民币 **2,900 元**的 8051 开发工具包，其中包含：

- 1 套 8k 代码限制版的 IAR EW8051 集成开发软件
- 1 个 Insight 8051 仿真器（用户可从万利 8051 系列仿真器中选配 1 款，以适用不同的 8051 芯片）

该工具包即日起于万利电子全国 13 个直销点上柜销售。

双方在技术上将紧密配合，进一步为国内外 8051 芯片厂商提供**工具链定制**的服务。

新闻：IAR 公司宣布支持 ARM Cortex M3

2 月 28 日，瑞典乌普萨拉

IAR 公司今天宣布了在集成开发环境 EWARM 中支持新的 ARM® Cortex® M3(CM3)架构，这在 ARM 第三方工具合作伙伴中，首个宣布支持该架构。新的 EWARM 集成开发环境中将会包含一个高度优化 Thumb2 指令的 C/C++编译器。4 月份，IAR 将在一些选定的合作伙伴中进行该编译器的β版测试，同时还将发布新版的 USB 接口的 J-Link JTAG 仿真器，无缝支持 Cortex M3 新处理器的调试和 flash 烧写功能。

第四章 用 C-SPY 调试应用程序

本例使用C-SPY 的模拟器(Simulator)来展现IAR C-SPY调试器的基本特点。前面各节生成的*project1.d79*应用程序已经可以用 C-SPY 调试器进行调试。用户利用调试器可以查看变量、设置断点、观察反汇编代码、监视寄存器和存储器、在 Terminal I/O 窗口打印输出。

1．开始调试

在开始调试之前必须设置几个 C-SPY 选件。具体操作如下：

① 选择主菜单 Project > Option，选择 Category 中的 Debugger。在 Setup 页面，在 Driver 的下拉菜单中选择 Simulator，同时选择 Run to main，点击 OK。
如果用户已经购买了 IAR 的 JTAG 仿真器，请选择 J-Link。

② 选择主菜单 Project > Debug 或工具条上的 Debugger 按钮。IAR C-SPY 将开始装载 *project1.d79*。除了已经打开的窗口外，将显示一组 C-SPY 专用窗口。

2．组织窗口

在 EWARM 中可以固定窗口（所谓 dock），也可以组织成书签形式，也可以让它们浮动。改变浮动窗口的大小时其他窗口不受影响。

注意 EWARM IDE 窗口最底部的状态条中包含如何安排窗口的有用信息。详细信息请参见 77 页 *Organizing the windows on screen*。

在开始调试前请确认如图 11 所示的各窗口和内容已经显示在屏幕上。在编辑器窗口应能看到源文件 *Tutor.c* 和 *Utilities.c* 以及 Debug Log 消息窗口。

3．检查源语句

① 检查源语句，双击 Workspace 中的 *Tutor.c*；

② 在编辑器显示文件 *Tutor.c* 后，用 Debug > Step Over 命令（或 F10），步进到 init_fib 函数调用语句；

③ 用 Debug > Step Into 命令（或 F11）进入函数 init_fib；
注）Step Over 命令用来执行源程序中的一条语句或一条指令，即使这条语句是一函数调用语句。

而 Step Into 命令则进入到函数或子程序调用内部

当执行 Step Into 后，活跃窗口已经切换到 Utilities.c，因为 init_fib 在这个文件里。

④ 继续用 Step Into 命令直到 for 循环语句；

⑤ 再用 Step Over 命令回到 for 循环的头。请注意，现在是在函数调用级上而不是语句级步进。

注）还有一种语句级步进的命令，Debug > Next statement 或工具条上的 Next statement 按钮。这条命令与 Step Into 和 Step over 不同。

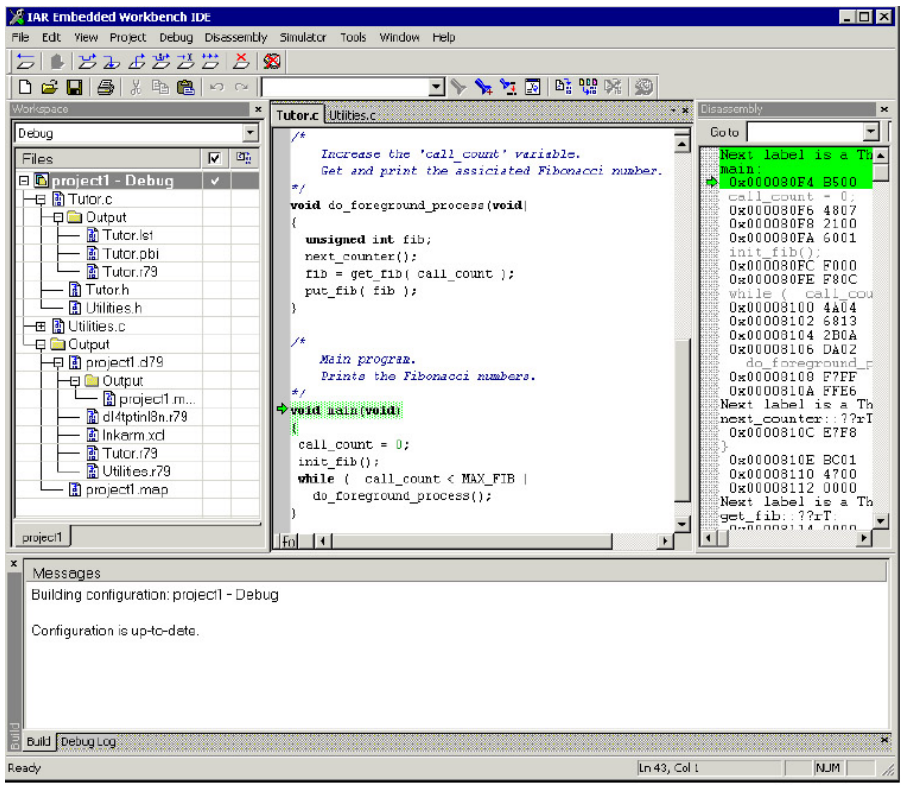


图 11. C-SPY 调试窗口

4．检查变量

C-SPY 允许在源程序上查看变量或表达式，所以可以在执行程序过程中跟踪它们的值的变化。查看变量的方法有几种，在源码窗口用鼠标双击变量名、然后打开 Locals、Live Watch 或 Auto 窗口。如何检查变量的更详细信息请看章节 *Working with variables and expressions*。

注） 当采用 None 优化级时，所有的非静态变量在它们的活动范围内都是活跃的，所以这些变量是完全能够调试的。但如果使用更高级别的优化，变量可能不能完全调试。

① 利用 Auto 窗口查看变量

选择 View > Auto 打开 Auto 窗口。 Auto 窗口显示最近修改过的表达式的当前值，单步执行程序观察变量如何变化。

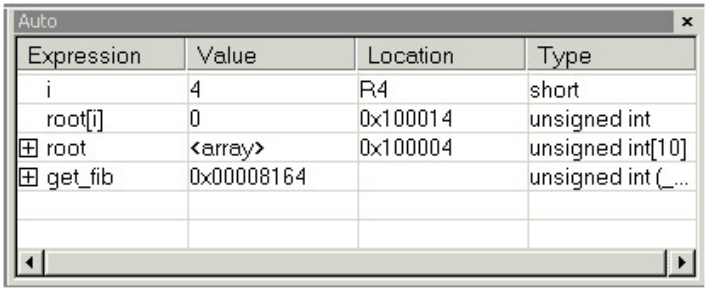


图 12. Auto 窗口中检查变量

② 设置一个 Watchpoint，利用 Watch 窗口查看变量

选择 View > Watch 打开 Watch 窗口。 请注意 Watch 窗口和 Auto 窗口按书签形式显示。按以下步骤在变量 i 上设置一个 Watchpoint。

- 点击 Watch 窗口中的虚线框，当输入区出现时输入 i，然后按 Enter 键。也可以从编辑器窗口拖一个变量到 Watch 窗口。
- 双击 init_fib 函数中的 root 数组名，将其拖到 Watch 窗口。

Watch 窗口将显示 i 和 root 的值。将 root 展开观察每个元素的值。

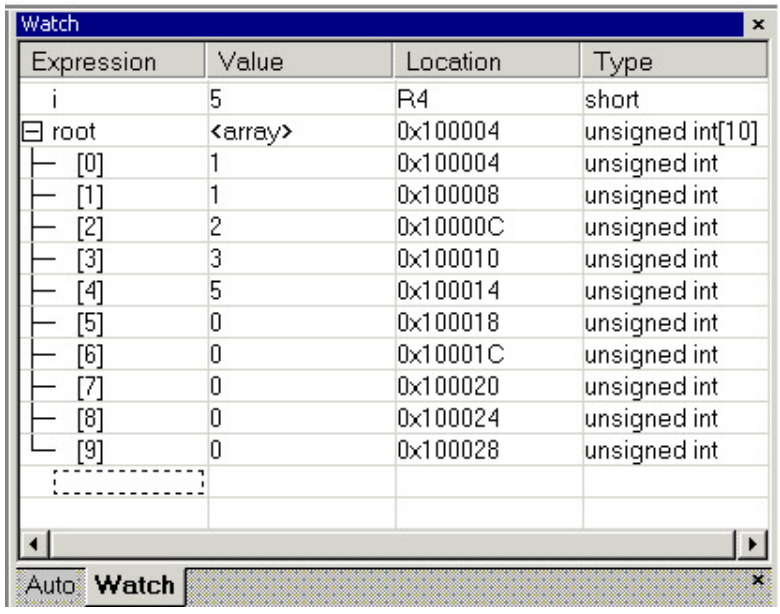


图 13. Watch 窗口

- 继续执行单步，观察 i 和 root 值的变化。
- 从 Watch 窗口中除去一个变量时，只需选择它然后删除。

5. 设置和监视断点

IAR C-SPY 具有强大的断点功能。详细请见手册 131 页 *The breakpoint system*。

设置断点最简单的方法是将光标定位到某条语句，然后按鼠标右键选择 Toggle Breakpoint 命令。实验

方法如下：

① 设置断点

用下面方法在 get_fib(i)语句上设置断点。在编辑器窗口显示 utilities.c。点击要设置断点的语句，选择主菜单 Edit > Toggle Breakpoint。也可以按工具条上的 Toggle Breakpoint 按钮。这时该语句上将出现断点标记。

如果要查看刚定义的断点，选择主菜单 View > Breakpoint 打开 Breakpoint 窗口。在 Debug Log 窗口也显示有关断点执行的信息。

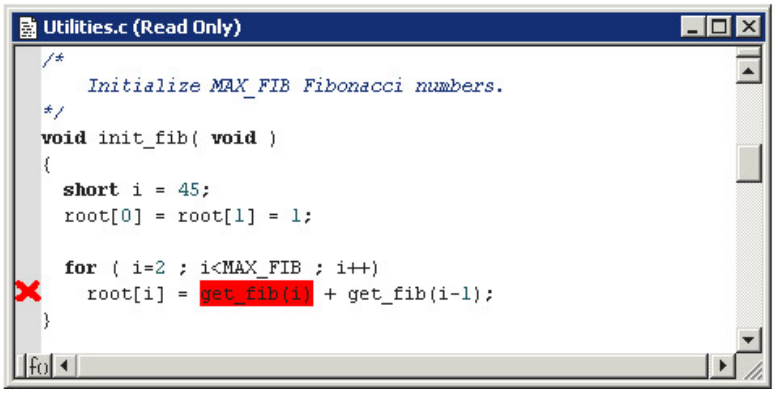


图 14. 设置断点

② 执行到断点

选择主菜单 Debug > Go 或者工具条上的 Go 按钮都可以让程序执行到断点。Watch 窗口将显示 root 表达式的值。Debug Log 窗口将显示关于断点的信息。

③ 消除断点

可用主菜单 Edit > Toggle Breakpoint 或按鼠标右键选择 Toggle Breakpoint。

6. 在反汇编窗口上调试

通常，在 C\C++程序上调试应该更快速和更直接。但是如果用户希望在反汇编程序上调试，C-SPY 也提供了这种功能，而且 C-SPY 允许方便地在两种方式上切换。反汇编程序的调试方法如下：

① 按 Reset 按钮复位应用程序。

② 调试时反汇编窗口通常是打开的。如果还没打开可以选择主菜单 View > Disassembly 打开反汇编窗口。

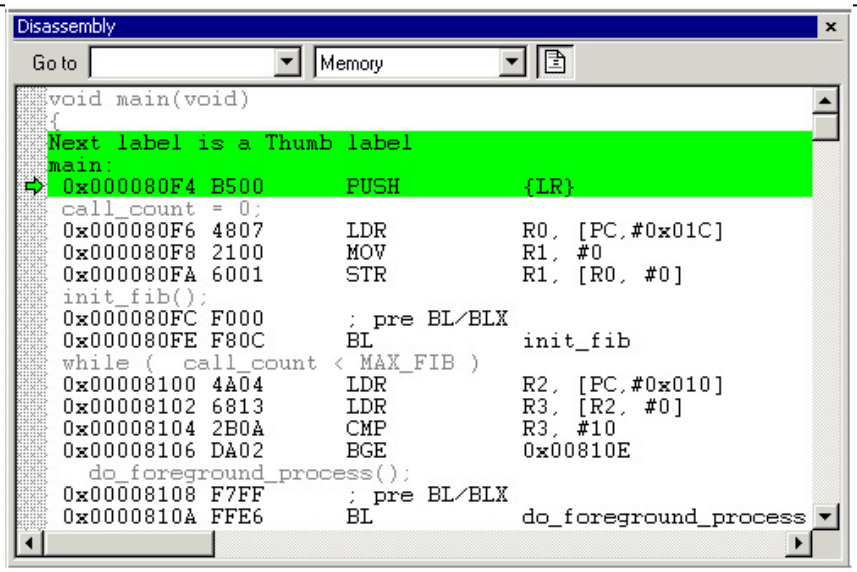


图 15. 反汇编窗口

反汇编窗口如图 15 所示。可以看到汇编代码与 C 语句一一对应。用上面介绍的几种单步命令执行程序观察结果。

7．监视寄存器

寄存器窗口允许用户监视和修改 CPU 寄存器的内容。具体方法如下：

① 选择主菜单 View > Register 打开寄存器窗口，见图 16。

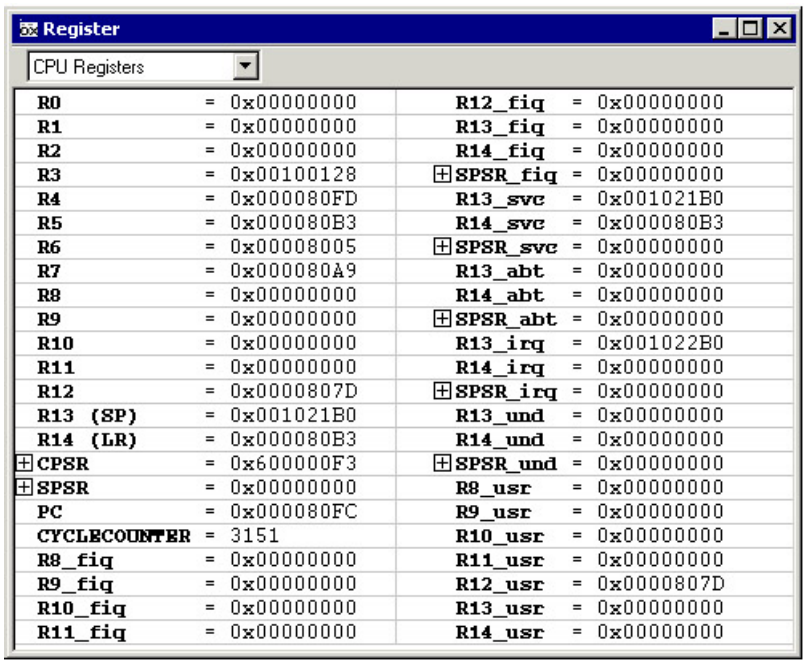


图 16. 寄存器窗口

② 用 Step Over 命令执行下一条指令，观察寄存器窗口中的数据如何变化。

③ 关闭寄存器窗口。

8．查看存储器

用户可以在存储器窗口监视所选择的存储器区域。下面是检查与变量 root 有关的存储器内容。

① 选择主菜单 View >Memory 打开存储器窗口，见图 17（用 8-bit 显示数据）。

② 激活 *Utilities.c* 窗口并双击变量 root。用鼠标将其拖到存储器窗口。

③ 如果希望以 16-bit 显示数据，在存储器窗口定部的下拉菜单中选择 2x Units 命令。
如果 C 应用程序的 init_fib 函数没有初始化所有的存储器单元，继续执行单步，同时观察存储器的内容是如何修改的。用户可以在存储器窗口修改存储单元的内容。只需把插入点放在希望修改的地方，然后输入新值就可以了。

④ 关闭存储器窗口。

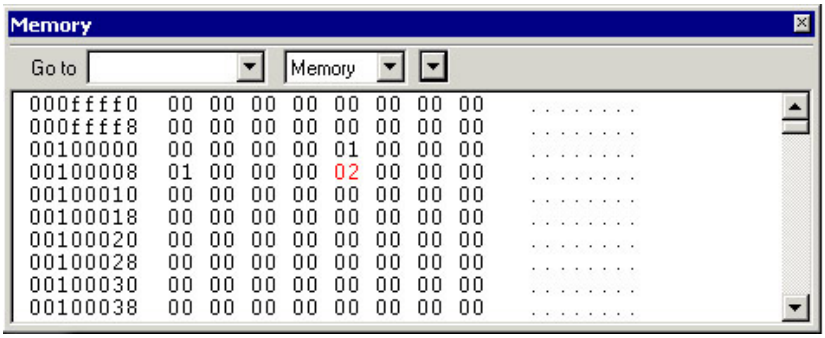


图 17. 8-bit 模式显示存储器窗口

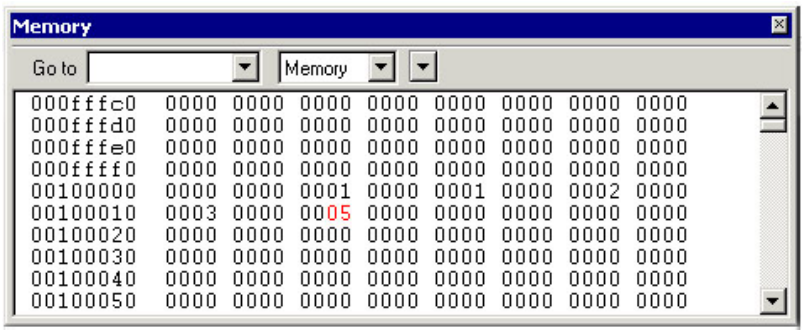


图 18. 16-bit 模式显示存储器窗口

9．观察Terminal I/O

用户有时可能希望调试应用程序中的 stdin 和 stdout 结构，但是又没有实际的硬件支持，C-SPY 允许用户使用 Terminal I/O 模拟 stdin 和 stdout。

注）Terminal I/O 只有在使用了连接输出文件选件 With I/O emulation module 时才可用。也就是说，某些把 stdin 和 stdout 指向 Terminal I/O 的低级例程将被连接进应用程序。

① 选择主菜单 View > Terminal I/O 显示 I/O 操作的输出，见图 19。

Terminal I/O 窗口显示的内容取决于应用程序执行了多远。

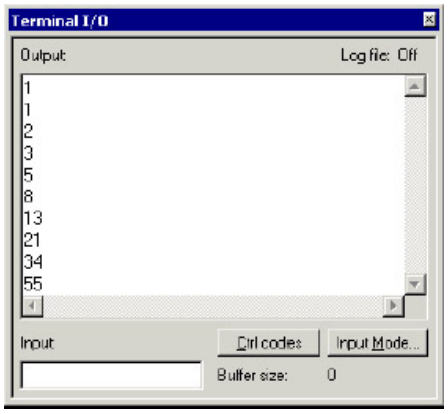


图 19. Terminal I/O 窗口

10．执行程序到结束

① 选择主菜单 Debug > Go 或工具条上的 Go 按钮。因为只有一个断点，所以程序一直执行到结束。同时在 Debug Log 窗口显示已经到达程序 exit 的消息，见图 20。

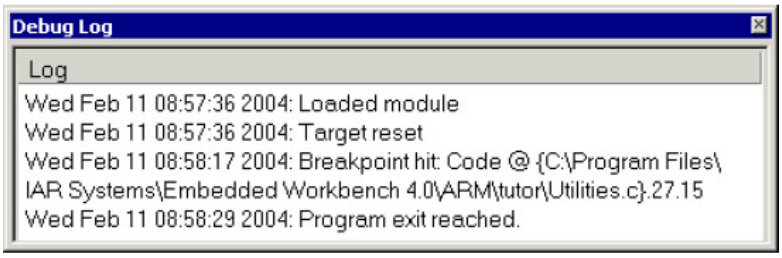


图 20. Debug Log 窗口

② 如果要求复位应用程序，选择主菜单 Debug > Reset 或工具条上的 Reset 按钮。

③ 如果要退出 C-SPY，选择 Debug > Stop Debugging，或工具条上的 Stop Debugging 按钮。

C-SPY 还提供许多其他的调试功能，如宏和中断模拟等，将在指南的其他章节讨论。有关如何使用 Debug 功能的详细介绍请见手册 Part 4。C-SPY 的特点介绍请见手册 Part 7 以及联机帮助信息。

新闻：IAR 公司发布首款 ARM Trace 仿真器

2 月 14 日，德国纽伦堡

IAR 公司今天发布了其 ARM 系列调试工具中的新一代产品：硬件调试设备 IAR J-Trace™。它支持所有带 ETM (Embedded Trace Macrocell) 的 ARM7 和 ARM9 处理器，通过全速 USB 2.0 接口与主机连接，并提供 2M 字节的 Trace 缓存，IAR 公司新版 EWARM 能够完全支持 J-Trace。该 Trace 仿真器零售价仅为 1195 美元，由于还带有标准 JTAG 接口，IAR J-Trace 对于绝大多数开发者而言都可以是唯一需要的 ARM 硬件调试设备。

技术细节

- USB 驱动的 JTAG 和 Trace (38-pin Mictor) 接口；
- 支持所有 ARM7 和 ARM9 处理器；
- 无需电源，通过 USB 接口供电；也带有外接电源插口；
- 集成在 IAR Embedded Workbench for ARM 中，易于设置和使用。

基本性能

- JTAG 速率 12 MHz
- 自动速率识别
- USB 2.0 全速（12Mbit/sec）
- 基于 RTCK JTAG 信号的自适应时钟
- 所有信号均可监控；目标电压可测量
- 20-pin 标准 JTAG 接口，38-pin 标准 Trace Mictor 接口
- 含 USB 电缆，20-pin 扁平电缆和 38-pin Trace 电缆
- 支持的主机操作系统：Windows 2000 和 Windows XP
- 与 C-SPY 调试器全面集成；进阶的调试功能都可以在 C-SPY 中使用

Trace 规格

- Trace 支持最高 200 MHz 全速时钟和 100 MHz 半速时钟
- Trace 功能基于 ARM ETM (Embedded Trace Macrocell)
- Trace 支持带有内置 Trace 端口的设备
- 2M 字节 Trace 缓存
- 体积紧凑，易于放置在桌面
- 安静，无风扇设计
- 支持 cycle accurate 和 compressed tracing
- 支持 4 位 / 8 位 / 16 位的 Trace 端口，每种端口均支持全速时钟和半速时钟
- 集成在 IAR Embedded Workbench for ARM 中，易于设置和使用
- Trace 窗口与 Source 和 Disassembly 窗口同步
- Function Trace 能够分层次地显示被调用的函数

“IAR J-Trace 使得 IAR Systems 为 ARM 开发者提供的工具链更为完整，IAR 公司如今能够为我们所有不同种类的 STR ARM 微控制器用户提供完美的服务。”

Mr.Dominique Jugnon，意法半导体微控制器开发工具经理

“IAR J-Trace 对于我们的汽车客户来说将会极有价值。在这一价位提供的功能允许大多数开发者方便地使用先进的 Trace 调试方案，提高开发质量。” Mr. Gualtiero Bagnuoli，Micronas 的汽车产品应用经理

“毫无疑问，该产品的发布将给 ARM 硬件调试工具市场带来改变”。他总结道：*“内嵌一定容量 Flash Memory 的 ARM 处理器的使用者现在有了一个完整的高端开发工具解决方案：IAR Embedded Workbench for ARM BaseLine 版本，一块开发板以及 IAR J-Trace 调试工具，总价低于 4000 美元！”*

Mr. Mike Skrtic，IAR 公司的开发套件经理

第五章 EWARM Flash Loader 开发指南

本章包含以下内容：

- 如何将应用程序下载到 RAM 中
- 如何将应用程序下载到 Flash 中
- 从框架程序和驱动程序两个部份分别介绍 Flash Loader

本文还介绍了如何编写和调试自己的 Flash Loader，最后，详细描述了 Flash Loader 框架 API 函数。

注）本文中的 xx 表示 2 个数字，用于识别所用的处理器。

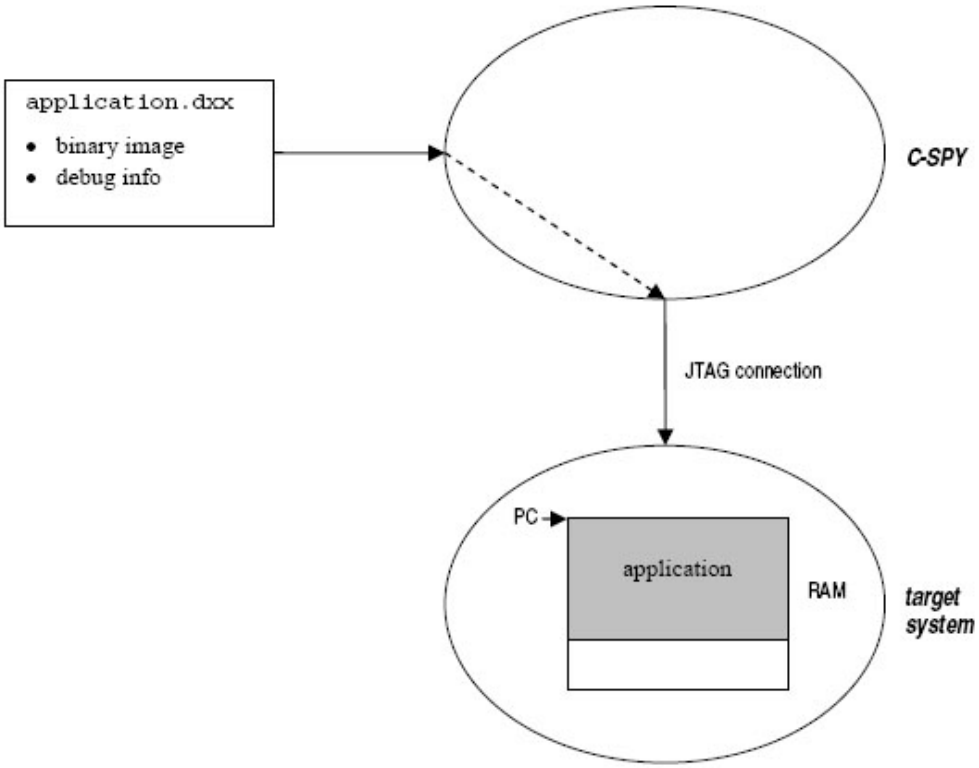
1．将应用程序下载到RAM中

将应用程序下载到 RAM 发生在 C-SPY 启动期间，由 C-SPY 自己控制执行。所谓下载就是通过 JTAG 接口把数据写进目标系统。

当 C-SPY 启动时，它执行以下步骤：

- 从 application.dxx 文件中读取应用程序的二进制映象和调试信息；
- 通过 JTAG 接口将二进制映象传输到目标系统的 RAM 中；
- 将程序计数器（PC）指向 RAM 中的应用程序入口点。

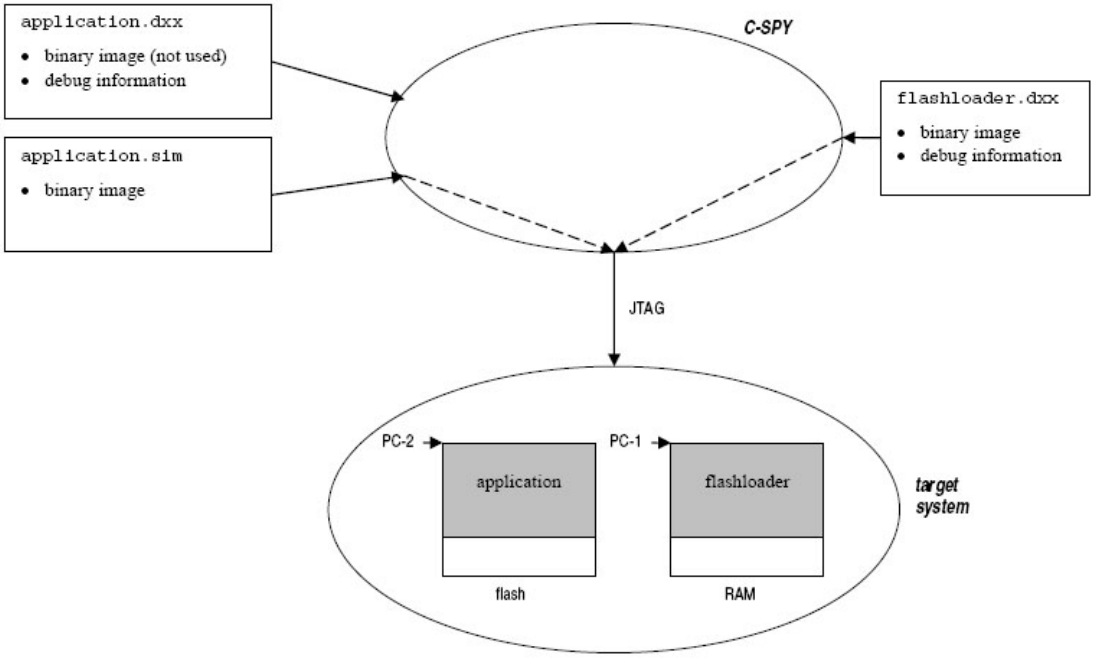
此时 RAM 中的应用程序已经准备好可以运行。



2．将应用程序下载到Flash中

将应用程序下载到 Flash 也发生在 C-SPY 启动期间，但不是由 C-SPY 执行，而是由一个叫做 Flash Loader 的专用程序执行。Flash Loader 先被装入到 RAM 并运行，再将应用程序写进 Flash。链接器（linker）生成两个输出文件，第一个是常规的 UBROF 格式目标文件（扩展名为 dxx），另一个是简单二进制格式目标文件（simple-code，扩展名为 sim）。Simple-code 格式十分简洁而且容易拆包，这是 Flash Loader 得以在目标硬件中执行的重要条件。

Flash Loader 是一个常规的 IAR Embedded Workbench 应用程序，可以在 IAR Embedded Workbench 环境中开发和调试。



当 C-SPY 启动时，它执行以下步骤：

- 从 flashloader.dxx 文件中读取 Flash Loader 的二进制映象；
- 通过 JTAG 接口将该二进制映象写进目标系统的 RAM；
- 将程序计数器（PC-1）指向 Flash Loader 在 RAM 中的入口点，并开始运行；
- 通过文件 I/O，Flash Loader 经由 JTAG 接口把 application.sim 文件中的应用程序二进制映象读入目标系统并写进 Flash 存储器；
- C-SPY 从 application.dxx 文件中读取调试信息，并将程序计数器（PC-2）指向 Flash 中的应用程序入口点；
- 此时 Flash 中的应用程序已经准备好可以运行。

3．Flash Loader介绍

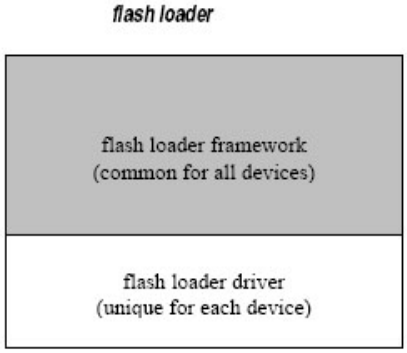
Flash Loader 是用 IAR Embedded Workbench 开发的本地应用程序。其任务是通过文件 I/O 从主机读取应用程序的二进制映象，将映象拆包，并写进 Flash 存储器。

Flash Loader 可以分成两个部分。一是所有 Flash Loader 所共用的框架部分，其源代码由 IAR Systems 提供并包含在 IAR Embedded Workbench 中；二是驱动部份，它是一小段用于实际烧写 Flash 存储器的小程序。在 IAR Embedded Workbench 中已经包含了一组用于各种芯片的 Flash Loader 驱动程序。由于 Flash Loader 驱动程序很简单，所以用户可以自行编写 IAR Systems 尚未支持的芯片驱动程序。

Flash Loader 框架程序实现了所有 Flash Loader 都具备的公共功能，包含从调试器读取二进制映象，把用户变量（选件）传递给 Flash Loader 的机制，以及为了与用户交互而创建 GUI 元素。GUI 元素包括消息窗口、消息记录和进度条等。缺省情况下，进度条由 Flash Loader 框架程序所控制。

Flash Loader 必须遵循 Flash<device>.dxx 的命名约定。例如假设某器件名叫 IAR X99，则它的 Flash

Loader 应命名为 FlashIarX99.dxx。



IAR 公司提供的 Flash Loader 源代码位于下列目录：

arm\src\flashloader\framework	Flash Loader 框架程序源代码，含 API 头文件
arm\src\flashloader\<vendor>\lash<device>	各 Flash Loader 驱动程序源代码，含工程文件

IAR 公司提供的可执行的 Flash Loader 位于下列目录：

arm\config\flashloader\<vendor>\lash<device>.dxx	对应于各驱动程序的 Flash Loader 可执行文件
arm\config\flashloader\<vendor>\lash<device>.mac	可选的C-SPY宏文件。如果宏文件的名字和Flash Loader可执行文件相同，该宏文件将先于同名 Flash Loader 被装进 RAM 并运行。有些芯片需要对一些 I/O 寄存器进行初始化之后 RAM 才能正常工作，这时此项功能就很有用。

4．可选的Flash Loader C-SPY宏文件

在将 Flash Loader 装入 RAM 之前可能需要执行一个 C-SPY 宏来设置目标系统。例如，某些芯片在复位后 RAM 还不能正常工作，就需要用一个宏来初始化必要的寄存器，以便让 RAM 正常工作。

在将 Flash Loader 装入 RAM 之前所执行的宏应满足以下规定：

- 宏文件应存放在同名 Flash Loader 的目录下；
- 宏文件的扩展名应为 mac；
- 宏文件名应与其关联的 Flash Loader 名相同；
- 宏文件中必须定义 **execUserFlashInit()**宏函数。C-SPY 将在把 Flash Loader 装进 RAM 之前调用该宏函数。请注意在调试阶段，当 Flash Loader 作为一个应用程序运行时，必须用 **execUserPreload**()替代 **execUserFlashInit()**。

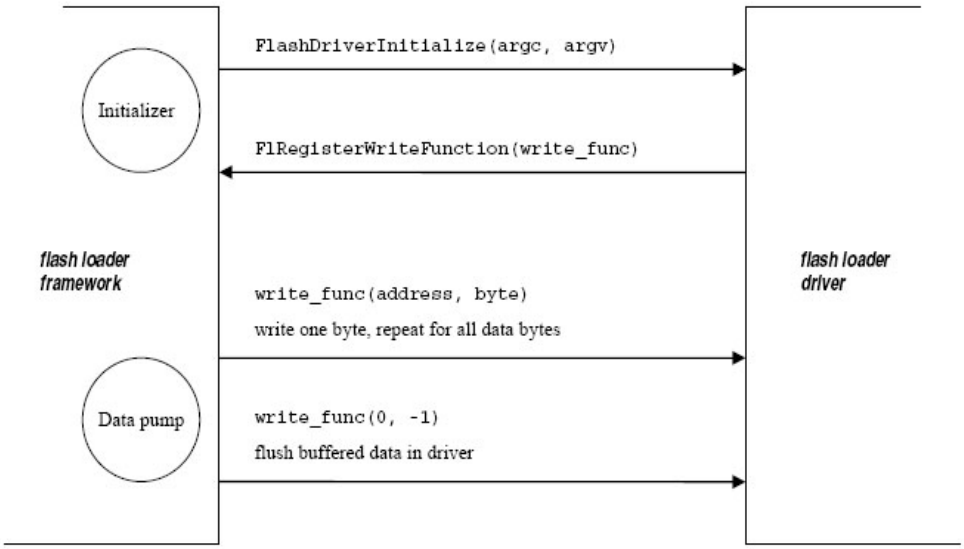
必要的话，在 Flash Loader 运行结束之后，可以用宏函数 execUserFlashExit()来恢复目标系统的初始设置。

5．与Flash Loader框架程序的接口

Flash Loader 框架程序将首先初始化 Flash Loader 驱动程序。此时，驱动程序可以执行各种初始化工作，但至少要向框架程序注册它的写函数。

在初始化之后，框架程序将通过驱动程序的写函数，一次传输一个字节给 Flash Loader。根据所用的 Flash 算法，有可能需要在驱动程序中缓冲多个字节，以便在将一个扇区写入 Flash 存储器之前填满整个扇区空间。从框架程序向驱动程序的最后一次写操作将被视为清空请求，允许驱动程序清空扇区缓冲中的任何剩余数据。如果 Flash Loader 驱动程序没有缓冲任何数据，清空请求可以被忽略。

驱动程序不返回任何错误状态给框架程序。一旦驱动程序中发生错误，驱动程序应通过调用 **`FlIMessageBox()`**API 函数向用户报告错误，然后调用 **`FlErrorExit()`**函数退出 Flash Loader。



6．Flash Loader驱动程序实例

本例展示如何为一种芯片编写 Flash Loader 驱动程序。为简单起见，假设该芯片中有一块很容易编程的 Flash；可以用一个简单的 Flash 算法，在一次操作中将一个字节写入 Flash 存储器。本例还展示了如何读取用户指定的选件，该选件说明了芯片运行的时钟频率。

关于怎样实现一个带扇区缓冲的 Flash Loader，请参考 IAR Embedded Workbench 所安装的 Flash Loader 驱动程序源代码。

```
// Flash loader driver example.
#include <stdio>
#include <stdlib.h>
```

```
#include "Interface.h" // The flash loader framework API declaration.

// The CPU clock speed, the default value 4000 kHz is used if no clock // is found.
static int clock = 4000;

// Write one byte to flash at addr.
// If byte == -1 the flash loader framework signals a flush operation
// at the end of the input file.
static void FlashWriteByte(unsigned long addr, int byte)
{
  unsigned char* ptr = (usigned char*)addr;

  if (byte == -1)
    return; // Simple return when the flush operation is requested.

  // Insert device specific instructions here to enable write
  // access to the flash device.
  *ptr = byte; // Write data byte to flash.

  // If some error occurs when writing to flash, this can be
  // communicated to the user by using code like
  // if (ret != STATUS_CMD_SUCCESS)
  //   {
  //     FlMessageBox("CMD_ERASE_SECTORS failed.");
  //     FlERrrorExit();
  //    }
  // A message box will be displayed by C-SPY and downloading
  // will terminate after the user has clicked the OK button.
}

void FlashDriverInitialize(int argc, char const* argv[])
{
  const char* str;

  // Register the flash write function.
  FlRegisterWriteFunction(FlashWriteByte);
```

```
  // See if user has passed a clock speed option.
  // If not, the default CCLK value is used.
  str = FlFindOption("- —clock", 1, argc, argv);
  if (str)
   {
     clock = strtoul(str, 0, 0);
    }
}
```

7．编译链接Flash Loader

① 拷贝一个现有的 Flash Loader，例如 arm\src\flashloader\philips\FlashPhilipsLPC210x；

② 确认编译器所用的文件包含路径包括了 Flash Loader 框架程序目录 arm\src\flashloader\framework 和 Flash Loader 驱动程序目录；

③ 修改 FlashPhilipsLPC210x.c 和 FlashPhilpsLPC210x.h 这两个文件的名称，使之与所用的芯片相符合；

④ 链接器控制文件也应该设置为与所用的芯片相符合。拷贝 FlashPhilipsLPC210x.xcl，并修改其中的地址定义行：

- DMEMSTART=40000000
- DMEMEND=40003FDF

实际使用的地址必须能够映射到目标硬件上。请注意Flash Loader的代码和数据都是下载到RAM中的，这就是为什么 ROM 段和 RAM 段都映射到同一段存储空间。

栈和堆都应该保持在最小。框架程序大约需要 300 字节的栈空间。请注意下面 xcl 文件中的数字都是十六进制的：

- D_CSTAK_SIZE=180
- D_IRQ_STACK_SIZE=40
- D_HEAP_SIZE=0

Flash Loader 框架程序将使用堆（heap）和 RAMEND（在链接器控制文件中声明）之间的内存作为读缓冲区。这就保证了读缓冲区能够利用所有剩余的内存。读缓冲区应当尽可能的大，以提高下载性能。每次 JTAG 传输的数据字节越多，性能就越高。如果剩余的读缓冲区少于 256 字节，框架程序将会报错，因为少于 256 字节将严重影响性能。

在编译链接Flash Loader程序之前，链接选项**With I/O emulation modules**必须被打开。得到的输出文件将以dxx为文件扩展名。

Flash Loader 已经可以用于把应用程序下载到 Flash。在 Embedded Workbench 中，打开应用程序工程，再打开 **Debugger download option** 对话框。使能 Flash download 选项，并选择 **Override default flash loader** 选项，指定你生成的 Flash Loader 输出文件。任何需要传递给 Flash Loader 的参数都可以写进 **Flash Loader arguments** 文本域。

现在启动调试器，就可以用你自己的 Flash Loader 将应用程序下载到 Flash。

8．调试Flash Loader

调试 Flash Loader 的方法和调试普通的应用程序一样。需要指出的是，Flash Loader 程序在作为 Flash Loader 被装入调试器时是不能调试的。只有当 Flash Loader 本身就是 IAR Embedded Workbench 中当前打开的工程时，它才能被调试。

在 Flash Loader 框架程序中有一个调试环境。该环境受头文件 **`DriverConfig.h`** 中定义的 C 预处理器宏变量控制，而 **`DriverConfig.h`** 被包含在框架程序的头文件 **`Congig.h`** 中。在 **`Congig.h`** 文件中，你可以看到哪些变量允许在 **`DriverConfig.h`** 中被覆盖。

在调试器中以一个独立的应用程序运行 Flash Loader 时有几点不同。要启动框架程序的调试环境，必须设置调试宏变量 DEBUG。要写入 Flash 的文件也必须用宏变量 DEBUG_FILE 显式说明。在独立调试时，**`argc/argv`** 参数传递机制是不工作的，参数必须用 C 预处理器宏变量 DEBUG_ARGS 硬性编码。

9．Flash Loader框架程序的API

本节介绍 Flash Loader 框架程序所提供的 API。对于大多数 Flash Loader 来说，很多 API 函数都是用不到的，把它们罗列在此只是为了内容的完整性而已。

所有函数都标出了它们的用途。***Mandatory*** 表示 Flash Loader 驱动程序必须实现此函数。

Optional 表示 Flash Loader 驱动程序可以根据需要选择性地实现此函数。***Framework*** 表示该函数只会被 Flash Loader 框架程序调用，而通常不应该被 Flash Loader 驱动程序调用。

所有 API 函数的原型都定义在头文件 arm\src\flashloader\framework\Interface.h 之中。

初始化函数

```
void FlashDriverInitialize (int argc, char const* argv);
```

用途：***Mandatory***

Flash Loader 驱动程序必须定义此函数。Flash Loader 框架程序用它来初始化 Flash Loader 驱动程序。

argc 传递 Flash 变量的数目；**argv** 传递 Flash 变量。

Flash 变量允许通过 C-SPY 的 Flash 选项对话框将参数传递给 Flash Loader。此函数的一个典型用例是将 CPU 的时钟速率传递给 Flash Loader 驱动程序。

void FlRegisteWriteFunction (WriteFunctionType write_func);

用途：***Mandatory***

Flash Loader 驱动程序调用此函数向 Flash Loader 框架程序注册写函数。变量 **write_func** 是指向写函数的指针。Flash Loader 框架程序将为每个要写进 Flash 的字节调用这个写函数，并将字节和要写的地址作为参数传递。地址的顺序应确保是递增的，但不一定连续（即允许有间隙）。Flash Loader 驱动程序必须从 **FlashDriverInitialize（）**中调用此函数。

typedef void (*WriteFunctionType) (unsigned long address, int byte);

此类型声明定义了写函数的函数指针类型。该写函数必须在 Flash Loader 驱动程序中定义。

unsigned long FlGetBaseAddress ();

这是一个可选函数，用于获得用户在 IDE 中设置的 Flash 基地址。如果用户没有设置 Flash 基地址，该函数返回 0xffffffff。例如，AMD 兼容的 Flash 器件在进行编程和擦除时不依赖于任何控制寄存器，而是对以 Flash 基地址为基准的多个偏移地址进行一系列总线写操作。此时就有必要知道 Flash 的基地址，以便执行可能的 Flash 地址重映射。

变量传递函数

const char* FlFindOption (char* option, int with_value, int argc, char const* argv[]);

用途：***optional***

此函数用于在变量数组 **argv** 中寻找指定的选件。

with_value 参数说明了该函数是用于检 **argv** 中是否存在某个选件，或者是否需要返回该选件的值。该选件的值是在 **argv** 中找到匹配变量后的下一个变量。如果要检查一个 flag 选件，例如--smallram，赋 **with_value** 为 0。如果要检查一个带值的选件，例如--speed 14600，则赋 **with_value** 为 1。

argc 参数是 **argv** 数组中的变量个数，**argv** 参数是字符串指针数组。当调用此函数时，可以直接使用 **FlashDriverInitialize()**函数中的 **argc/argv** 参数。

如果在 **argv** 中没有找到指定的选件，函数返回一个空指针。

如果 **with_value** 设为 1，函数返回指向匹配选件后面的那个变量的指针。

如果 **with_value** 设为 0，函数返回指向匹配选件在 **argv** 中入口的指针。

int FlMakeArgs (char* args, char const* argv[]);

用途：***framework***

取一个用 space/tab 分隔的字符串，并生成一个 **argv** 字符串数组，每个数组元素对应于一个选件。**argv** 字符指针数组必须足够大，以便容纳 **args** 字符串中的所有选件。

函数返回 **argv** 数组中的字符串数目（变量数）。

C-SPY 用户接口函数

void FlMessageBox (char* msg);

用途：***optional***

C-SPY 将显示一个消息窗口，其中显示 msg 参数给出的文本。消息字符串中的文本可以用换行符（\n）分割成多行。Flash Loader 将停止执行，直到按下消息窗口中的 OK 按钮为止。

void FlMesagaLog (char* msg);

用途：***optional***

C-SPY 将在调试器的 **log** 窗口显示一个由 **msg** 给出的 **log** 消息。消息字符串中的文本可以用换行符（\n）分割成多行。

void FlProgressBarCreate (char* title);

用途：***optional***

C-SPY 将生成一个进度条窗口。**Title** 参数字符串显示在进度条窗口的上方。

void FlProgressBarDestroy ();

用途：***optional***

C-SPY 将关闭进度条窗口。

void FlProgressBarUpdate (int progress);

用途：***optional***

C-SPY 将更新进度条，以反映 **progess** 参数的值。**progess** 参数的有效范围是 0 至 100。此函数只有在进度条已经生成的前提下才有效。调用 **FlProgressBarValue()**的次数应尽可能少（<10），目的是为了减少 JTAG 总线上的（慢）传输次数。

void FlOverrideProgressBar ();

用途：***optional***

重载由 Flash Loader 框架程序实现的缺省进度条。大多数情况下，Flash Loader 驱动程序不需要操纵进度条，因为这是由框架程序中的输入文件读取例程来控制的。如果 Flash Loader 驱动程序实现了它自己的进度条，就必须调用此函数来禁止缺省的进度条。

void FlErrorExit ();

用途：***optional***

终止 Flash Loader，并通知 C-SPY 调试器：Flash 下载已经失败。

文件函数

int FlFileOpen (char* name);

用途：***framework***

此函数打开一个文件用于二进制读。如果打开成功返回一个文件句柄；如果打开失败返回-1。

int FlFileReadByte (int fd);

用途：***framework***

从与文件句柄 **fd** 相关联的已打开文件中读取一个字节。该函数返回读到的字节；当已读到文件末尾时返回-1。

void FlFileClose (int fd);

用途：***framework***

关闭与文件句柄 **fd** 相关联的已打开文件。