

# IAR Embedded Workbench 用户指南

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 4.30，并提供一个 32k 代码限制、但没有时间限制的免费评估版。有兴趣的朋友可以到 IAR 公司的网站 [www.iar.com/ewarm](http://www.iar.com/ewarm) 或南京万利电子的网站 [www.manley.com.cn](http://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。如果用户已经购买了 IAR 的 JTAG 仿真器 J-Link，则可以在真实的目标板上运行。

IAR EWARM 的主要特点如下：

- 高度优化的 IAR ARM C/C++ Compiler
- IAR ARM Assembler
- 一个通用的 IAR XLINK Linker
- IAR XAR 和 XLIB 建库程序和 IAR DLIB C/C++ 运行库
- 功能强大的编辑器
- 项目管理器
- 命令行实用程序
- IAR C-SPY 调试器（先进的高级语言调试器）

下面我们分步介绍如何使用 IAR EWARM

## 一. 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。

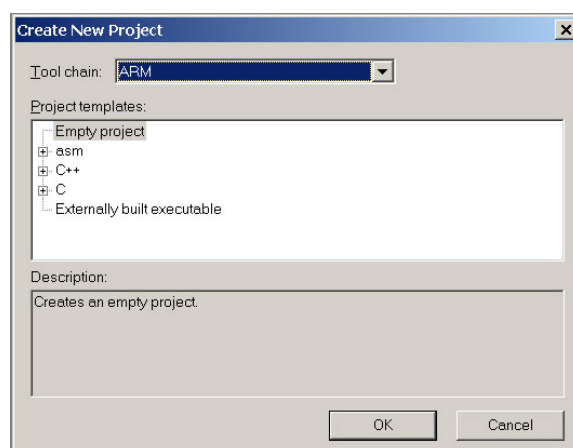


图 1. 生成新项目窗口

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

② 在 Tool chain 栏中选择 ARM，然后单击 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。

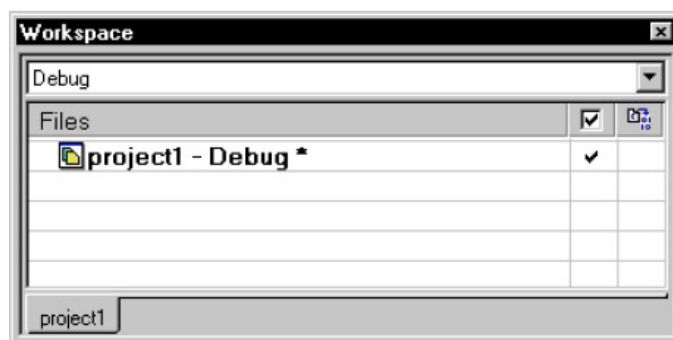


图 2. Workspace 窗口

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。然后将工作区取名为 **tutorials** 输入 **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。

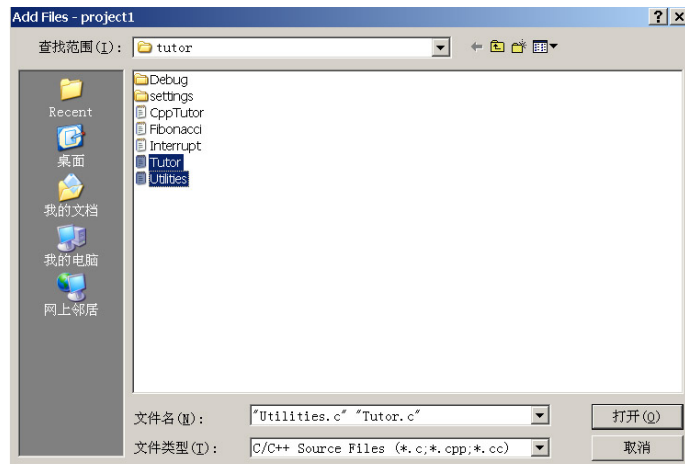


图 3. 添加文件窗口

### 4. 设置项目选项

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

- ① 选择通用选项

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**。

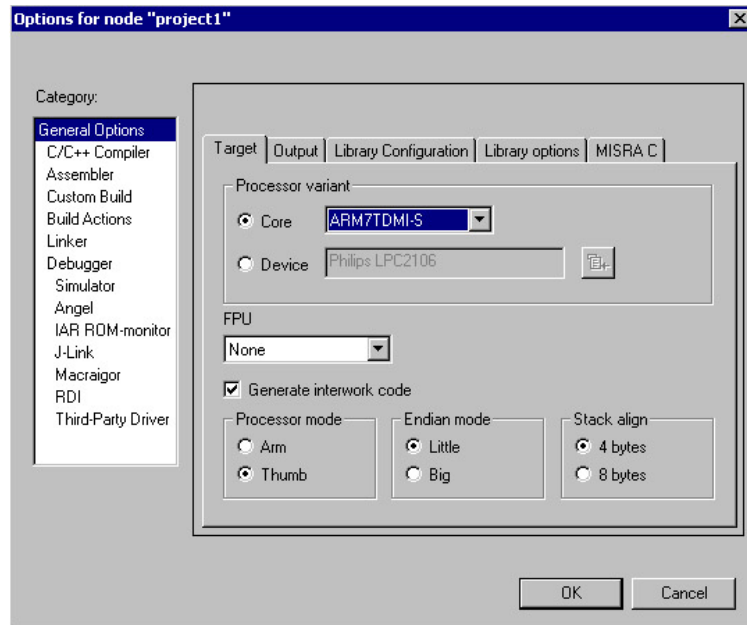


图 4. 项目通用选项窗口

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

Target 页面中，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 选择编译器选项

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

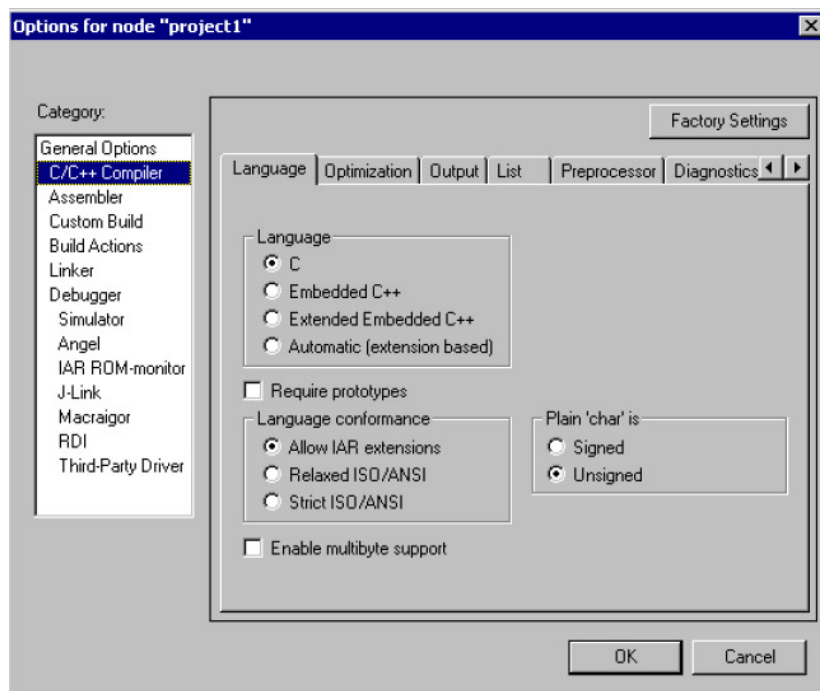


图 5. C/C++ Compiler 选件窗口

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。

## 二. 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。

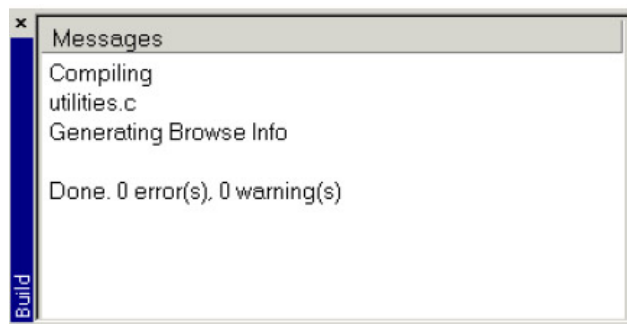


图 6. Build 窗口中的编译处理消息

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。

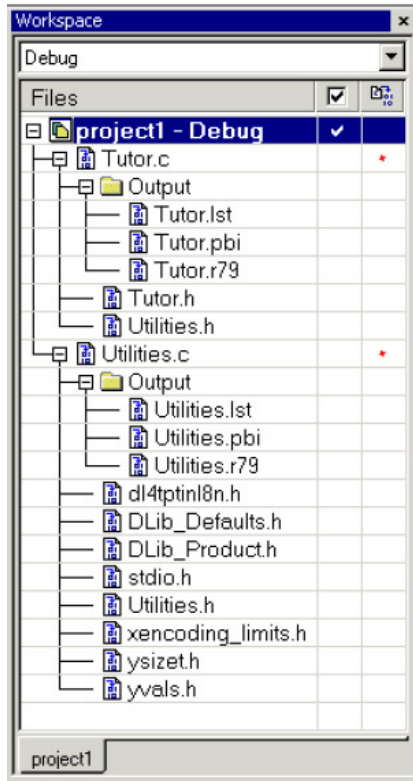


图 7. 编译处理后的文件结构

## 2. 查看编译器列表文件

现在我们通过改变编译器选项中的优化级别（**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` 文件的名称和路径，编译命令行及选项等信息。
- 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** 对话框，选择 **Editor** 页面。选择 **Scan for Change Files** 选项。此选项将自动打开编辑窗口中的文件，目前是 `Utilities.lst` 文件。按 **OK** 按钮。

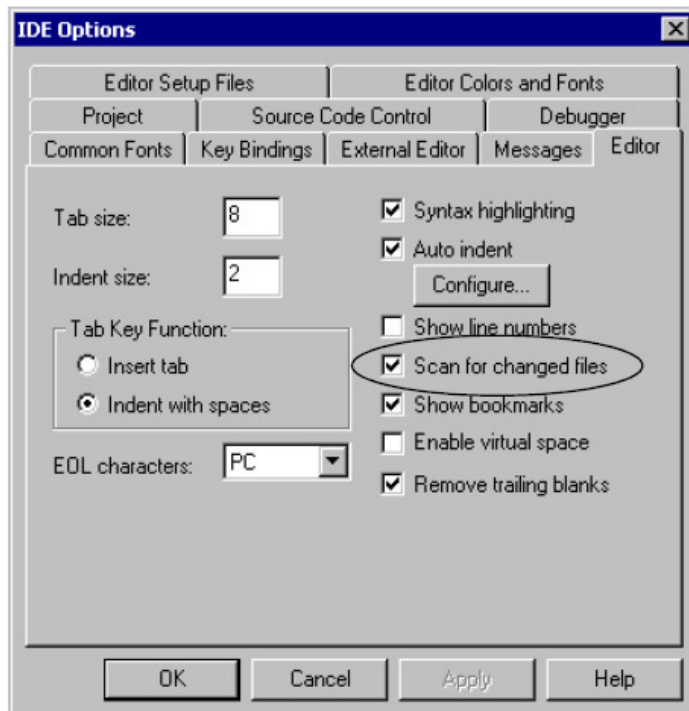


图 8. IDE Option 窗口

- 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 对话框，见图 9。在左边的 Category 中选择 Linker，显示 IAR XLINK 的各选项页面。

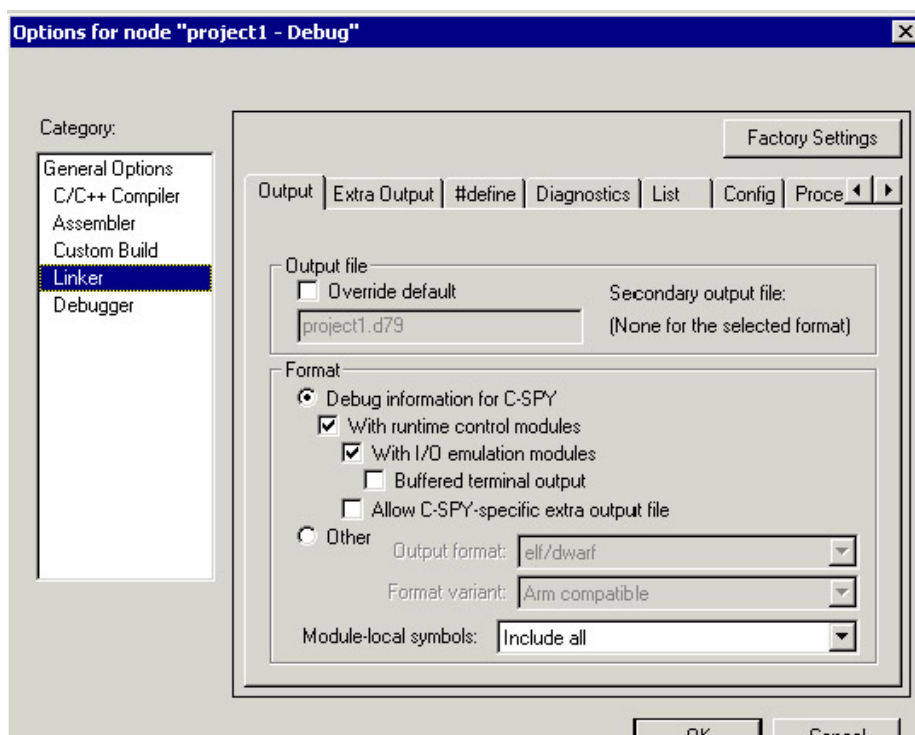


图 9. XLINK 参数选件窗口

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。

注意：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 选项

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*。

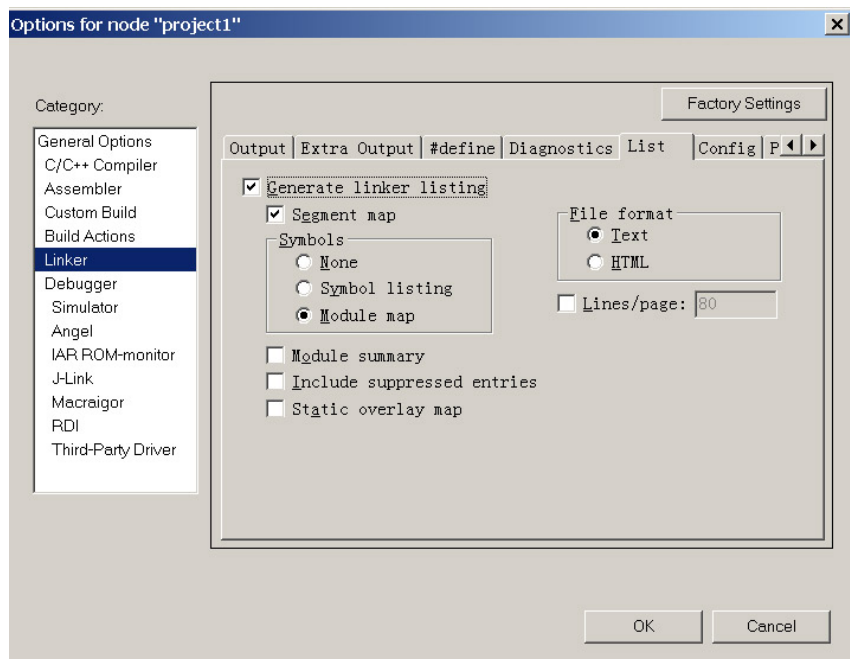


图 10. XLINK 选项中的 list 页面

#### 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项。
- 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。

### 三. 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 选项。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`。

(注意：**Step Over** 命令用来执行源程序中的一条语句或一条指令，即使这条语句是一条函数调用语句。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。

(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。)

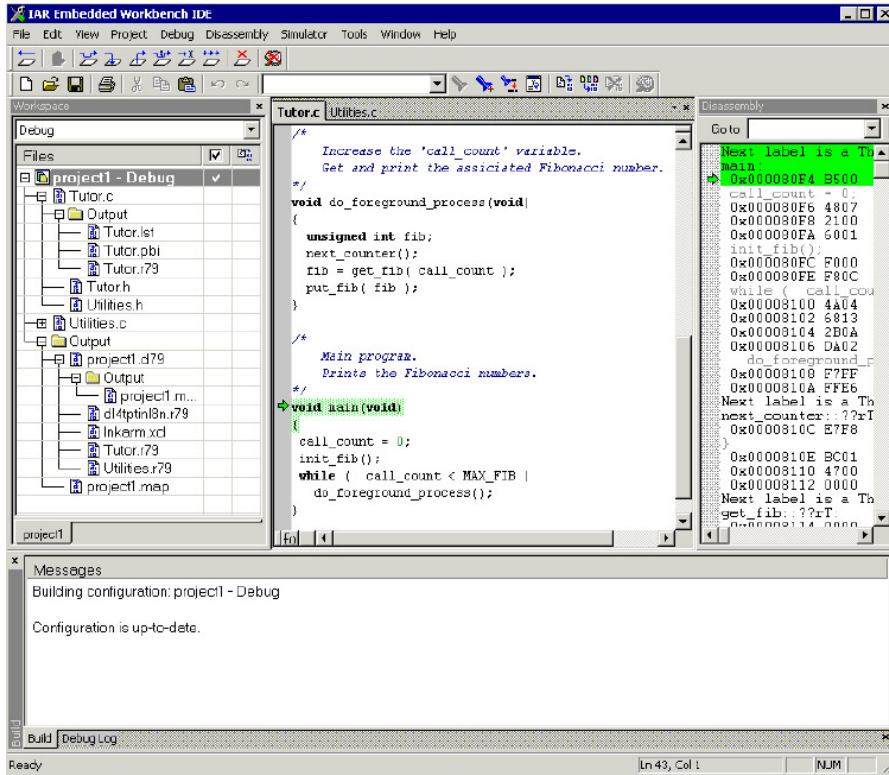


图 11. C-SPY 调试窗口

#### 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*。

注意：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。

| Expression | Value      | Location | Type             |
|------------|------------|----------|------------------|
| i          | 4          | R4       | short            |
| root[i]    | 0          | 0x100014 | unsigned int     |
| root       | <array>    | 0x100004 | unsigned int[10] |
| get_fib    | 0x00008164 |          | unsigned int(... |

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。

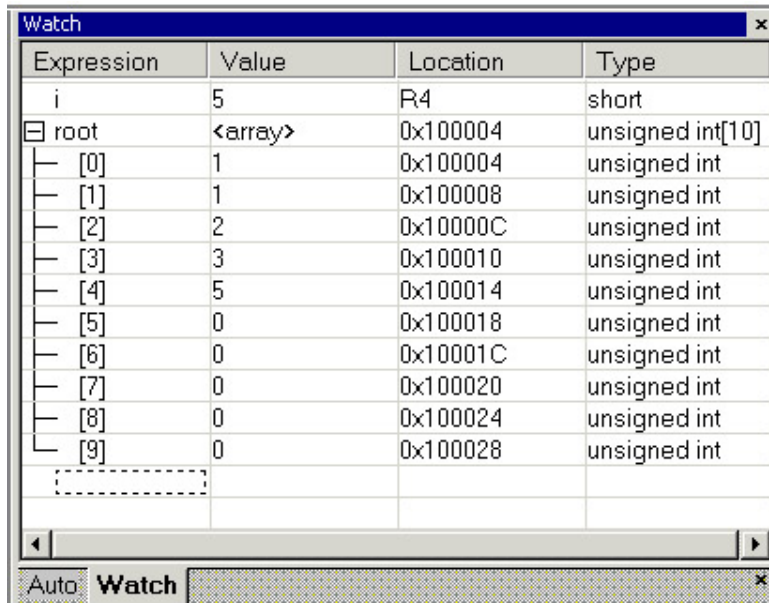


图 13. Watch 窗口

- 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 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。

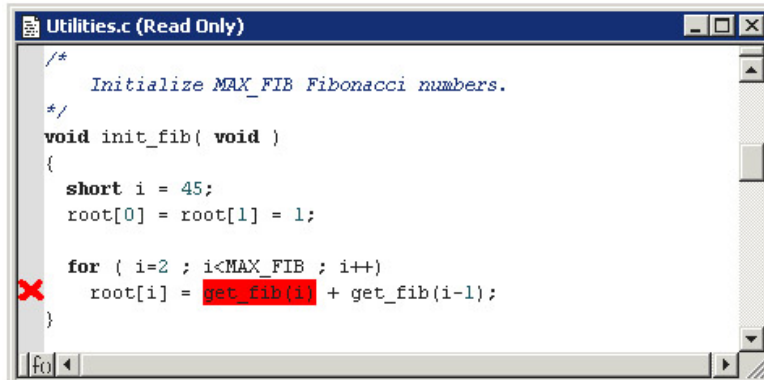


图 14. 设置断点

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。

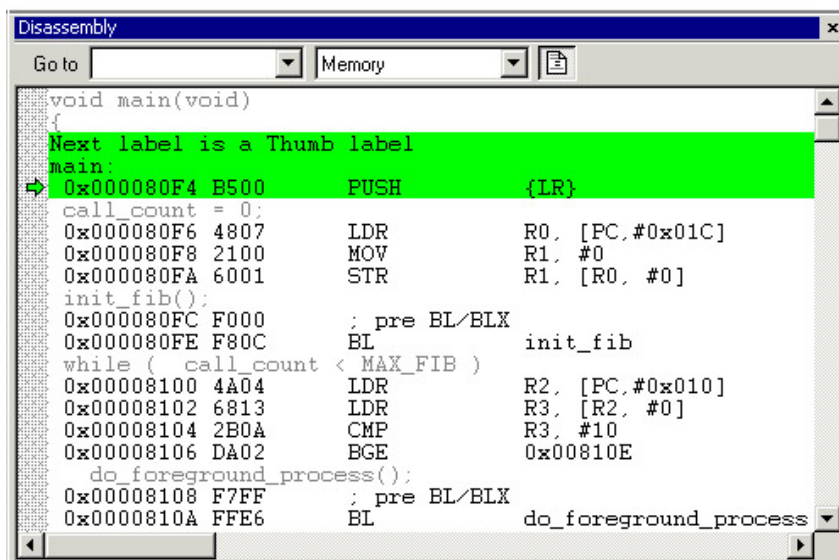


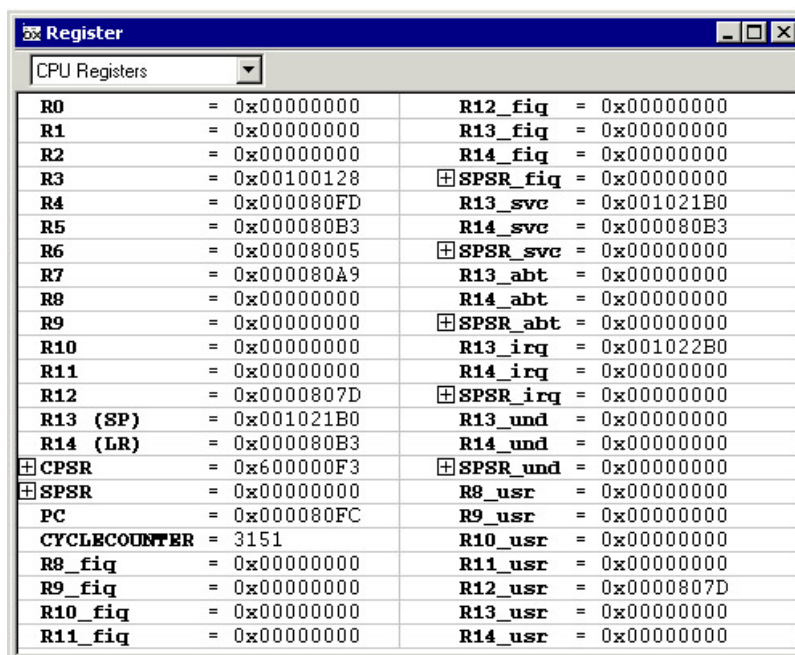
图 15. 反汇编窗口

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。



| CPU Registers |              |
|---------------|--------------|
| R0            | = 0x00000000 |
| R1            | = 0x00000000 |
| R2            | = 0x00000000 |
| R3            | = 0x00100128 |
| R4            | = 0x000080FD |
| R5            | = 0x000080B3 |
| R6            | = 0x00008005 |
| R7            | = 0x000080A9 |
| R8            | = 0x00000000 |
| R9            | = 0x00000000 |
| R10           | = 0x00000000 |
| R11           | = 0x00000000 |
| R12           | = 0x0000807D |
| R13 (SP)      | = 0x001021B0 |
| R14 (LR)      | = 0x000080B3 |
| ⊕ CPSR        | = 0x600000F3 |
| ⊕ SPSR        | = 0x00000000 |
| PC            | = 0x000080FC |
| CYCLECOUNTER  | = 3151       |
| R8_fiq        | = 0x00000000 |
| R9_fiq        | = 0x00000000 |
| R10_fiq       | = 0x00000000 |
| R11_fiq       | = 0x00000000 |
| R12_fiq       | = 0x00000000 |
| R13_fiq       | = 0x00000000 |
| R14_fiq       | = 0x00000000 |
| ⊕ SPSR_fiq    | = 0x00000000 |
| R13_svc       | = 0x001021B0 |
| R14_svc       | = 0x000080B3 |
| ⊕ SPSR_svc    | = 0x00000000 |
| R13_abt       | = 0x00000000 |
| R14_abt       | = 0x00000000 |
| ⊕ SPSR_abt    | = 0x00000000 |
| R13_irq       | = 0x001022B0 |
| R14_irq       | = 0x00000000 |
| ⊕ SPSR_irq    | = 0x00000000 |
| R13_und       | = 0x00000000 |
| R14_und       | = 0x00000000 |
| ⊕ SPSR_und    | = 0x00000000 |
| R8_usr        | = 0x00000000 |
| R9_usr        | = 0x00000000 |
| R10_usr       | = 0x00000000 |
| R11_usr       | = 0x00000000 |
| R12_usr       | = 0x0000807D |
| R13_usr       | = 0x00000000 |
| R14_usr       | = 0x00000000 |

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。

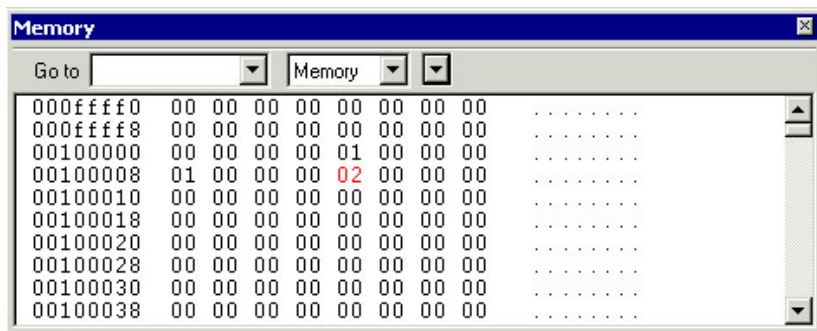


图 17. 8-bit 模式显示存储器窗口

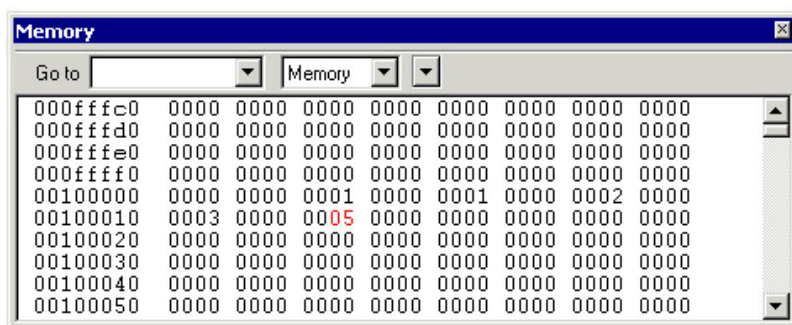


图 18. 16-bit 模式显示存储器窗口

## 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`。

注意: Terminal I/O 只有在使用了连接输出文件选项 `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。

⑤ 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。

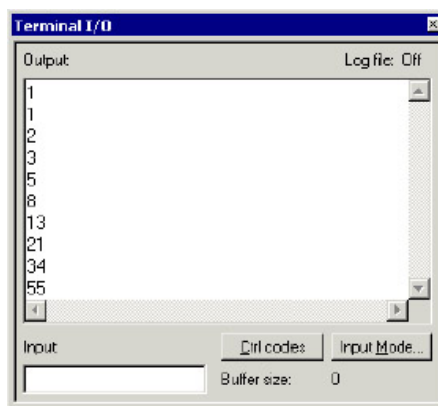


图 19. Terminal I/O 窗口

## 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。

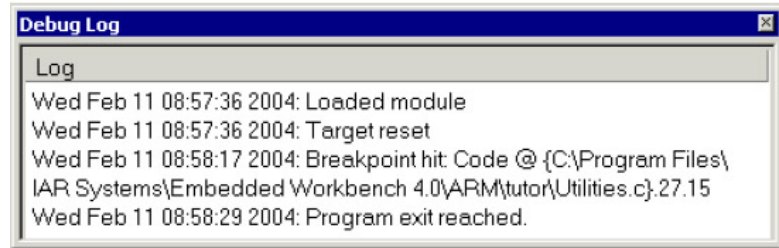


图 20. Debug Log 窗口

- 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。
- ⑥ 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的 **Part 4**。C-SPY 的特点介绍请见 **Part 7** 以及联机帮助信息。

## 附录：IAR Embedded Workbench for ARM version 4.30 简介

IAR Embedded Workbench for ARM version 4.30 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。

### EWARM 4.30版本的最新特点

- 已经支持到 ARM11 内核。
- 进一步改进了编译器速度优化，重写了浮点运算库。
- 支持 OSEK Run-Time Interface (ORTI)。
- 支持 OSE Epsilon RTOS 的 Kernel Awareness Debug。
- 同时支持多颗 Flash 的 Flash Loader 程序，以及通用的 Flash Loader 开发指南。
- 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。
- 大量的厂商评估板例子，包括 IAR、Analog Devices、Aiji System、ARM、Atmel、Cirrus Logic、Freescale、Keil、OKI、Olimex、Pasat、Philips、Phytec、ST 和 TI 等。
- 提供常用的程序架构模板。
- C-SPY 模拟器中可执行跟踪 (Trace)。
- J-Link TCP/IP 服务器。

### 支持的ARM核和芯片

IAR EWARM v4.30 支持 ARM7 (ARM7TDMI, ARM7TDMI-S, ARM720T), ARM9 (ARM9TDMI, ARM920T, ARM922T, ARM9940T), ARM9E (ARM926EJ-S, ARM946E-S, ARM966E-S), ARM10 (ARM1020E, ARM1022E), ARM11 和 XScale 等内核，以及下述厂商生产的含 ARM 内核的芯片：

- Analog Devices
- Atmel
- Cirrus Logic
- Freescale
- Intel
- NetSilicon
- OKI

- Philips
- Samsung
- Sharp
- STMicroelectronics
- Texas Instruments

#### 针对芯片的支持

- 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。
- 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。
- 针对评估板的例子工程, 包含 IAR、Analog Devices、Aiji System、ARM、Atmel、Cirrus Logic、Freescale、Keil、OKI、Olimex、Pusat、Philips、Phytec、ST 和 TI 等厂家。
- 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。

#### 针对嵌入式应用的特点

- 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, 用于运行时映像校验。
- 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。

#### 嵌入式调试

- 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。

#### 支持的硬件调试工具

- IAR J-Link JTAG 接口（支持所有 ARM7 和 ARM9 核，通过 USB 或 TCP/IP 连接）
- RDI（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
- ROM-Monitor
- Angel（用于 Atmel 和 Cirrus Logic 的评估板）

#### 支持ARM的IAR J-Link（选件）

- USB 驱动的 JTAG 接口。
- 支持 ARM7 和 ARM9，ARM 模式和 Thumb 模式。
- 下载速度达 120kb/s。
- 通过 USB 接口供电，不需要另接电源。
- 最大 JTAG 速率 8MHz。
- 自动速率识别。
- 能够监视所有的 JTAG 信号，目标电源可测量。

- J-Link TCP/IP 服务器。
- 20 脚标准 JTAG 插头，可选 14 脚插头。
- 含 USB 电缆和 20 芯扁平电缆。

#### 支持的RTOS插件

| 操作系统                  | IAR EWARM<br>内置的插件 | 由 RTOS 厂商<br>提供的插件 |
|-----------------------|--------------------|--------------------|
| CMA-RX                | X                  |                    |
| CMX-Tiny <sup>+</sup> | X                  |                    |
| uC/OS-II              |                    | X                  |
| ThreadX               | X                  |                    |
| RTXC Quadros          |                    | X                  |
| Fusion RTOS           |                    | X                  |
| OSEK(ORTI)            | X                  |                    |
| OSE Epsilon           | X                  |                    |
| MiSPO NORTi           |                    | 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。

#### 图形化的集成开发环境

- 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

## 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像文件格式（如 ELF/DWARF），与大多数常见的调试器和仿真器兼容。

## 用户帮助

- 完备的例子工程和工程模板。
- 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。

## 系统要求

IAR Embedded Workbench 可以在下列操作系统平台上运行：

- Microsoft® Windows® 98 SE
- Microsoft® Windows® ME
- Microsoft® Windows NT® 4.0
- Microsoft® Windows® 2000
- Microsoft® Windows® XP

建议使用奔腾处理器，至少 128MB RAM 和 100MB 空余硬盘空间。