

AVR® IAR Embedded Workbench® IDE

用 户 手 册

基于Atmel® 公司AVR® 微处理器

目 录

| | |
|---|----|
| 第一部分 产品介绍..... | 1 |
| 1.1 产品介绍..... | 1 |
| 1.1.1 嵌入式IAR Embedded Workbench IDE | 1 |
| 1.1.2 IAR C-SPY 调试器..... | 3 |
| 1.1.3 IAR C-SPY 调试器系统..... | 5 |
| 1.1.4 IAR C/C++编译器..... | 8 |
| 1.1.5 IAR汇编器..... | 9 |
| 1.1.6 IAR XLINK连接器..... | 9 |
| 1.1.7 IAR XAR Library Builder库创建器和IAR XLIB Librarian库管理器..... | 10 |
| 1.2 已安装文件..... | 11 |
| 1.2.1 目录结构..... | 11 |
| 1.2.2 文件类型..... | 14 |
| 1.2.3 文档..... | 16 |
| 第二部分 教程..... | 18 |
| 2.1 创建一个应用工程..... | 18 |
| 2.1.1 创建一个新工程..... | 18 |
| 2.1.2 应用程序编译和连接..... | 23 |
| 2.2 使用IAR C-SPYDebugger进行调试..... | 28 |
| 程序调试..... | 28 |
| 2.3 C与汇编混合模式 | 36 |
| 2.3.1 检查调用规则..... | 36 |
| 2.3.2 在工程中添加一个汇编模块..... | 37 |
| 2.4 使用C++ | 38 |
| 创建一个C++应用程序..... | 39 |
| 2.5 模拟一个中断..... | 41 |
| 2.5.1 加入一个中断句柄..... | 42 |
| 2.5.2 创建仿真环境..... | 43 |
| 2.5.3 中断仿真..... | 48 |
| 2.5.4 中断和断点中宏的使用 | 49 |
| 2.6 使用库模块..... | 50 |
| 使用库..... | 50 |

第一部分 产品介绍

AVR® IAR Embedded Workbench® IDE用户手册的这部分包括以下章节：

- 产品介绍
- 已安装文件

1.1 产品介绍

嵌入式 IAR Embedded Workbench®是一个非常有效的集成开发环境（IDE），它使用户充分地开发并管理嵌入式应用工程。作为一个开发平台，它具备任何在用户每天的工作地方所想要的特性。

本章介绍了嵌入式 IAR Embedded Workbench IDE，旨在使用户获得对本产品的所有集成工具的总体了解。

1.1.1 嵌入式 IAR Embedded Workbench IDE

嵌入式 IAR Embedded Workbench IDE 提供一个框架，任何可用的工具都可以完整地嵌入其中，这些工具包括：

- 高度优化的 IAR AVR C/C++编译器；
- AVR IAR 汇编器；
- 通用 IAR XLINK Linker；
- IAR XAR 库创建器和 IAR XLIB Librarian；
- 一个强大的编辑器；
- 一个工程管理器；
- IAR C-SPY™调试器，一个具有世界先进水平的高级语言调试器。

嵌入式 IAR Embedded Workbench 适用于大量 8 位、16 位以及 32 位的微处理器和微控制器，使用户在开发新的项目时也能在所熟悉的开发环境中进行。它为用户提供一个易学和具有最大量代码继承能力的开发环境，以及对大多数和特殊目标的支持。嵌入式 IAR Embedded Workbench 有效提高用户的工作效率，通过 IAR 工具，用户可以大大节省工作时间。我们称这个理念为：“不同架构，同一解决方案”。

如果用户想获得关于所支持的目标处理器的更详细的信息，请与用户的软件提供商或者与用户的IAR代理联系，或者登陆IAR网站 www.iar.com 以查询最新的产品信息。

一个可扩展的模块化的环境

尽管嵌入式 IAR Embedded Workbench IDE 可以提供完成一个成功工程所需的所有工具，但

我们也认识到集成其他工具的必要性。因此，IAR Embedded Workbench IDE 容易适应于用户喜欢的编辑器和源代码控制系统。IAR XLINK Linker 可以输出多种格式，使用户可在第三方的软件上进行调试。实时操作系统（RTOS）支持也可加载到产品中。

编译器，汇编器和连接器也可在命令行环境中运行，用户可以在一个已建好的工程环境中把它们作为外部工具使用。

特性

嵌入式 IAR Embedded Workbench 是一个灵活的集成开发环境，使用户可以针对多种不同的目标处理器开发应用程序。并为快速开发和调试提供便捷的 Windows 截面。

项目管理

嵌入式 IAR Embedded Workbench IDE 能帮助用户控制所有的工程模块，例如，C 或者 C++ 源代码文件、汇编文件、“引用”文件、以及其他相关模块。用户创建一个工作区，可以在此开发一个或多个工程。文件可以组合，并且可以为各级设置选项—工程、组、或者文件。任何修改都被记录，从而保证重新设计时可以获得所有所需的模块，而可执行文件中不会包含已过期的模块。下表指出另外的一些特性：

- 通过工程模板可以创建独立的可编辑和可运行的工程文件，使开发平稳启动；
- 分级的工程表述；
- 具有分级图标的源代码浏览器；
- 可以为全球化、组和个人源代码文件设置选项；
- “Make”功能只在必要时才实行再编译、再汇编和再连接文件；
- 基于文本的工程文件；
- 自定义功能使用户轻松的扩展标准工具栏；
- 工程文件输入时可使用命令行模式。

源代码控制

源代码控制（Source Code Control,SCC），作为修订控制，可用于跟踪用户的源代码的不同版本。IAR Embedded Workbench 可以识别和接受基于 Microsoft 发布的 SCC 接口规范的任何第三方源代码控制系统。

窗口管理

为使用户充分而方便地控制窗口的位置，每个窗口都可停靠，用户就可以有选择地给窗口做上标记。可停靠的窗口系统还通过一种节省空间方式使多个窗口可同时打开。另外，重新分配窗口大小也很方便。

文本编辑器

集成化的文本编辑器可以并行编辑多个文件，并具有时兴编辑器所期望的所有编辑特性，包括无限次的撤销/重做和自动完成。另外它还包含针对软件开发的特殊功能，比如关键字的着色（C/C++，汇编和用户定义等）、段缩进、以及对源文件的导航功能。还可识别C语言元素（例如括号的匹配问题）。下表指出另外的一些特性：

- 上下文智能帮助系统可以显示 **DLIB** 库的参考信息；
- 使用文本风格和色条指出 C、C++ 和汇编程序的语法；
- 强大的搜索和置换功能，包括多文件搜索；
- 从错误列表直接跳转到程序行；
- 支持多字节字符；
- 圆括号匹配
- 自动缩排；
- 书签功能；
- 每个窗口均可无限次撤销和重做。

文档

AVR 嵌入式 IAR Workbench IDE 在本文档中有详细讲解。另外还有在线的帮助文件以及超文本格式的 PDF 用户文档。

1.1.2 IAR C-SPY 调试器

IAR C-SPY 调试器是为嵌入式应用程序开发的高级语言调试器。在设计上，它与 IAR 编译器和汇编器一起工作，并且与嵌入式 IAR Embedded Workbench IDE 完全集成，可在开发与调试间自由切换。因此，它使用户可做到：

- 在调试时进行编辑。在调试过程中，源代码的修正可以直接写入用来控制调试过程的同一窗口中。其修改将在项目重启后生效；
- 在启动调试器之前可设置源代码断点。源代码中的断点可与同一段源代码相关联，即使中间插入了新的代码。

IAR C-SPY 调试器由一个具备基本的 C-SPY 系列特点的主要部分和驱动部分组成。C-SPY 驱动确保与目标系统的通信和控制。并提供一个用户接口—特殊菜单，窗口和对话框—以连接到目标系统的功能上，比如，特殊断点。在下面内容中，将概述通用 C-SPY 调试器的特性。对于可用的 C-SPY 驱动的概述，请参见第 8 页，IAR C-SPY 调试器系统。

C-SPY 调试器的总体特性

因为 IAR 系统提供的是一个整体工具链，编译器和连接器的输出结果包含调试器的扩展调试信息，从而使用户获得最佳的调试效果。IAR C-SPY 调试器具备本节中所介绍的总体特性。

源代码和反汇编调试

IAR C-SPY 调试器使用户能按要求在源代码和反汇编调试间切换，适用于 C/C++ 和汇编语言源代码。

调试 C 或 C++ 源代码是验证用户的应用程序的逻辑性最快捷、最便利的方式，然而，反汇编调试则针对应用程序的错误段，并对硬件进行精确控制。在混合显示模式中，调试器显示 C/C++ 源代码及其对应的反汇编代码清单。

程序调用级的单步调试

传统的调试器设置，认为最佳的源代码调试间隔是“行到行”，与之相比，C-SPY 则更细化，将每个语句和调用函数称为“步点”，并加以控制。这就意味着在每个表达式里的函数调用，以及函数调用作为参数甚至到其他类型的函数调用都可以进行“单步”调试。后者在调试 C++ 源代码时特别有效，主要针对大量的外部函数调用，比如对象构造器。

调试信息提供了内嵌函数，如果执行了这类函数的调用，也可进行源码级调试。

代码和数据断点

C-SPY 断点系统允许用户在调试程序过程中设置多种断点，并按照特定需要在某一位置停止。用户可以设置代码断点来验证程序的逻辑性是否正确。也可以设置数据断点来检验数据如何以及何时改变。最后，用户还可以添加条件至断点处。

变量和表达式监控

当用户监控变量和表达式时，用户可以选择很多工具。任何变量和表达式都可通过一次扫描来求值。用户可以很轻松地在一较长的时间内对已定义的表达式进行监控和记录其值。对局部变量用户可以直接控制，同时可以无干扰地显示即时数据。最终将自动显示最后指定的变量。

Container 响应

当用户在 IAR C-SPY 调试器中运行程序时，可以查看诸如 STL 列表和向量地址等库内数据类型。因此，用户在运行 C++ STL containers 时，可以对程序进行总体浏览，以及良好的调试。

调用栈信息

AVR IAR C/C++ 编译器产生扩展的函数调用信息。在不影响整个运行环境的情况下，无论程序计数器指在哪里 C-SPY 都能显示整个函数调用栈信息。用户可以在调用栈中选择任何函数，并且可以获得相关的局部变量和寄存器的可用信息。

强大的宏系统

IAR C-SPY 调试器包含了一个强大的内部宏系统，能使用户定义复杂的动作并得以实施。C-SPY 宏可单独使用也可以同复杂的断点联合使用—如果用户是在使用仿真器的话—中断仿真系统需要进行一系列复杂的动作。

C-SPY 调试器的其他特性

下表指出了一些其他的特性：

- 模块化和可扩展化的结构设计允许在调试器中加入第三方设备，比如，实时操作系统，外围仿真模块和驱动；
- 线程运行保证在运行目标应用程序时 IDE 仍处于响应状态；
- 自动步进；
- 源代码浏览器可以方便查看函数，类型以及变量；
- 变量的扩展类型识别；
- 可配置化的寄存器（CPU 和外围设备）以及存储器窗口；
- 支持代码覆盖和函数级模块化；
- 终端 I/O 模拟；
- 支持 UBROF，Intel 扩展和 Motorola 输入格式。

RTOS 响应

IAR C-SPY 调试器支持实时 OS 响应调试。

RTOS 插件模块由 IAR 以及一些第三方的供货商提供。如要了解支持 RTOS 模块信息，请联系用户的软件供应商或 IAR 代理，或者访问 IAR 网站。

文档

IAR C-SPY 调试器在本文档中有详细介绍。调试器的一般特点在第四部分中有介绍，每个调试器驱动特殊性质在第五部分 IAR C-SPY 仿真器和第六部分 C-SPY 硬件调试系统中有介绍。网上还有帮助信息和超文本格式（PDF）文档。

1.1.3 IAR C-SPY 调试器系统

至本手册撰写之时，AVR 公司的 IAR C-SPY 调试器针对下列目标系统发布了驱动：

- 仿真器；
- AVR® ICE200
- AVR® JTAGICE
- AVR® JTAGICE
- AVR® Crypto Controller ROM-monitor for Atmel Smart Card Development Board (SCDB) 和 Voyager 开发系统

需了解C-SPY驱动的相关信息，请联系用户的软件供应商或IAR代理。也可登陆IAR网站，www.iar.com。

如需了解更多关于 IAR C-SPY 调试器的信息，请参看调试器概念，第 107 页。在下面章节，将描述各种驱动。

C-SPY 仿真器驱动

C-SPY 软仿真器驱动在软件上完全模拟了目标处理器的功能。通过这个驱动，在获得相关硬件之前就可对程序的逻辑性进行调试。因为不需要硬件，它同时也是很多应用程序最有效的解决方案。

特性

除具备 C-SPY 调试器的基本特点外，软仿真器驱动还具备：

- 指令级仿真；
- 中断模拟；
- 外围设备仿真，使用 C-SPY 宏系统与直接断点并行。

关于 IAR C-SPY 软仿真器更详细的信息，请参看第五部分 IAR C-SPY 软仿真器一节。

C-SPY ICE200 驱动

C-SPY ICE200 驱动允许连接 AVR® ICE200。C-SPY ICE200 驱动提供了低成本的实时调试功能。

特性

除了具有 IAR C-SPY 调试器的基本特性外，C-SPY ICE200 驱动还具备：

- 全部面向微控制器的实时运行。
- 代码断点数量不限。
- 不占用目标系统的存储器；

注意：C-SPY ICE200 驱动不支持代码和数据覆盖、规格化。

关于 C-SPY ICE200 驱动更详细的信息，请参看本手册第六部分 C-SPY 硬件调试器系统。

C-SPY JTAGICE 驱动

C-SPY JTAGICE 驱动允许连接 AVR® JTAGICE。它提供了自动的 flash 下载，并利用了片上调试的性能。

C-SPY JTAGICE 驱动提供了低成本的实时调试功能。

特性

除了具有 IAR C-SPY 调试器的基本特性外，C-SPY JTAGICE 驱动还具备：

- 全部面向微控制器的实时运行。
- 使用目标设备上的硬件断点。
- 不占用目标系统的存储器；
- 内置的 flash 下载器。
- 串口通信。

注意： C-SPY JTAGICE 驱动不支持代码和数据覆盖、规格化。

关于 IAR C-SPY 软仿真器更详细的信息，请参看本手册第六部分 C-SPY 硬件调试器系统。

C-SPY JTAGICE MKII 驱动

C-SPY JTAGICE MKII 驱动允许连接 AVR® JTAGICE MKII。 它提供了自动的 flash 下载，并利用了片上调试的性能。

C-SPY JTAGICE MKII 驱动提供了低成本的实时调试功能。

特性

除了具有 IAR C-SPY 调试器的基本特性外，C-SPY JTAGICE MKII 驱动还具备：

- 全部面向微控制器的实时运行。
- 使用目标设备上的 4 个硬件断点，以及不限数量的软件断点。
- 不占用目标系统的存储器；
- 内置的 flash 下载器。
- 通过串口或 USB 连接实现通信；

注意： C-SPY JTAGICE MKII 驱动不支持代码和数据覆盖、规格化。

关于 C-SPY JTAGICE MKII 驱动更详细的信息，请参看本手册第六部分 C-SPY 硬件调试器系统。

C-SPY CRYPTO 控制器 ROM 监视器驱动

C-SPY CRYPTO 控制器 ROM 监视器驱动（CCR 驱动）允许连接到 Atmel Smart Card 开发板（SCDB）和 Voyager 开发系统。

特性

除了具有 IAR C-SPY 调试器的基本特性外，CCR 驱动还具备：

- 实时运行；
- RS-232 串口通信；

- 支持实时中断；

注意： C-SPY ROM 监视器不支持代码覆盖。

关于 IAR C-SPY ROM 监视器更详细的信息，请参看本手册第六部分 C-SPY 硬件调试器系统。

1.1.4 IAR C/C++编译器

AVR IAR C/C++编译器是一个具有世界先进水平的具备标准 C/C++特性的编译器，众多的扩展插件让用户可以更好地使用 AVR 的特定功能。

编译器已经和其他的 AVR 的 IAR 系统集成。

特性

AVR IAR C/C++编译器具备以下特性：

代码生成

- 普通或特定的 AVR 的最优化技术可以产生出高效的机器代码；
- 全面的输出选择，包括可重定位的目标代码、汇编源代码和可选的汇编器列表文件；
- 目标代码可与汇编器连接；
- 生成扩展的调试信息。

语言工具：

- 支持 C 或 C++编程语言；
- 具有支持 IAR 扩展的嵌入式 C++的特性：模板、名称空间、多重的虚拟外设、固定操作符（`static_cast`, `const_cast`, 和 `reinterpret_cast`），以及标准的模板库（STL）；
- 在不同的存储器中放置类；
- 作为一个独立自主的环境，与 ISO/ANSI 标准相一致；
- 有特殊目标语言的扩展，比如特殊函数的输入，扩展的关键字，`#pragma` 指示，预设标志，内部函数，完全分配和行内汇编器；
- 针对嵌入系统的应用函数的标准库；
- 与 IEEE 标准兼容的浮点算法；
- 可在 C 或者 C++中应用的中断函数。

类型检查

- 在编译时进行扩展类型检查；
- 在连接时进行外部调用类型检查；
- 连接时检查应用程序的内部模块移植性。

运行环境

AVR IAR Embedded Workbench 提供了两套运行库：

- IAR DLIB 库，支持 ISO/ANSI C 和 C++。这个库还支持 IEEE 754 格式的浮点数，多字节参数和局部参数。
- IAR CLIB 库是一种轻型库，并不完全与 ISO/ANSI C 兼容。同时，它也不支持 IEEE 754 格式或者 C++ 格式。

现在有几种模式来定制运行环境和运行库。就这两种运行库而言，库的源代码已经包含在其中了。

文档

AVR IAR C/C++ 编译器在 AVR® IAR C/C++ 编译器参考手册里有详细介绍。

1.1.5 IAR 汇编器

AVR IAR 汇编器同其他的 IAR 系统软件集成。它是一个强大的重定位宏汇编器（支持 Intel/Motorola 格式），并且含有多种指示符和表达式。它具备一个内部 C 语言预处理器，因而支持条件汇编。

AVR IAR 汇编器使用与 Atmel® 公司 AVR 汇编器相同的存储机制和操作语法，从而简化了对已有代码的移植过程。关于详细信息，请参见汇编器 AVR® IAR 汇编器参考手册。

特性

AVR IAR 汇编器具备以下特性：

- C 预处理器；
- 扩展的交叉调用输出的列表文件；
- 由可用存储器大小决定参数个数和程序大小；
- 支持外部调用的复杂表达式；
- 每个模块有多达 65536 个可重定位段；
- 在参数表中有 255 个重要参数。

文档

AVR IAR 汇编器在 AVR® IAR 汇编器参考手册里有详细介绍。

1.1.6 IAR XLINK 连接器

IAR XLINK 连接器连接一个或多个由 AVR IAR 汇编器或者 AVR IAR C/C++ 编译器产生

的可重定位的目标文件，并生成 AVR 处理器所需的机器代码。它在连接小的单个文件、完全汇编程序时同连接大的、可重定位的、多模块的 C/C++ 或混合 C/C++ 以及汇编程序时一样快捷便利。

它可以识别超过 30 种的工业标准的代码，另外还包括 IAR C-SPY 调试器所使用的 IAR 系统调试格式——UBROF（通用的二进制可重定位目标文件格式）。一个应用程序可由任意多个 UBROF 可重定位文件构成，并且可以和汇编器以及 C 或 C++ 程序合成。

IAR XLINK 连接器最终输出结果是一个完整的、可执行的目标文件，并可以下载到 AVR 的处理器中或到一个硬件仿真器中。当然，输出文件是否包含调试信息取决于用户所选择的输出格式。

IAR XLINK 连接器支持用户定义库，并只下载那些在连接应用程序时所需的模块。在连接前，IAR XLINK 连接器将对所有的模块进行 C 语言级的类型检查，并对所有输入文件中的所有参数进行完全的可靠性检查。它还对所有的模块进行统一的编译器设置检查，从而确保使用 C 或 C++ 运行库的正确类型和参数。

特性

- 完全的内部模块类型检查；
- 简易的库模块的覆盖；
- 灵活的段命令可以更细致地掌控代码和数据的定位；
- 连接符的定义使对配置的控制更加自如；
- 可选的代码检测功能对运行监测；
- 去除无用代码和数据。

文档

IAR XLINK 连接器在 IAR Linker 和库工具参考手册中有详细介绍。

1.1.7 IAR XAR Library Builder 库创建器和 IAR XLIB

Librarian 库管理器

一个库是包含一系列可重定位目标模块的单个文件，每个模块都按需要加载到文件中，互相独立。IAR XAR Library Builder 库创建器帮助用户轻松地创建库。此外，IAR XLIB Librarian 库管理器使用户可以对 IAR 系统汇编器和编译器产生的可重定位目标库文件进行操作。

一个库文件与任何由汇编器或编译器生成的可重定位目标文件并没有区别，除非它包含了“LIBRARY”类型的模块。所有的 C 或 C++ 程序都使用库，同时 AVR IARC/C++ 编译器则使用一些标准的库文件。

特性

IAR XAR Library Builder 库创建器和 IAR XLIB Librarian 库管理器均具备以下特性：

-
- 其模块可以嵌入一个库文件中；
 - 交互式或者批量式操作。

IAR XLIB Librarian 库管理器还具备以下特点：

- 其模块可以列出、添加、插入、更换或移除；
- 其模块可以在程序和库类型间变更；
- 字段可以列出；
- 字符可以列出。

文档

IAR XAR Library Builder 库创建器和 IAR XLIB Librarian 库管理器在 IAR 连接器和库工具参考手册中有详细介绍。另外，在嵌入式 IAR Embedded Workbench 的帮助菜单中还有一份 PDF 格式的文档供查询。

1.2 已安装文件

本章介绍了安装过程中创建了哪些目录以及使用了哪些文件类型。在本章末尾，介绍了在不同版本的手册和在线文档中可以找到的信息。

参考产品附带的“快速入门”和“安装和注册手册”，可以查到关于系统要求以及如何安装及注册 IAR 系统产品的信息。

1.2.1 目录结构

安装过程中创建了几个目录，用来放置 IAR 系统开发工具所使用的不同类型的文件。下面就详细讲解每个目录下默认的文件。

根目录

在安装过程中创建的默认根目录为：**x: \Program Files\IAR Systems\Embedded Workbench 4.n**，**x** 是指 Microsoft Windows 的安装目录，而 **4.n** 是嵌入式 IAR Embedded Workbench IDE 的版本号。

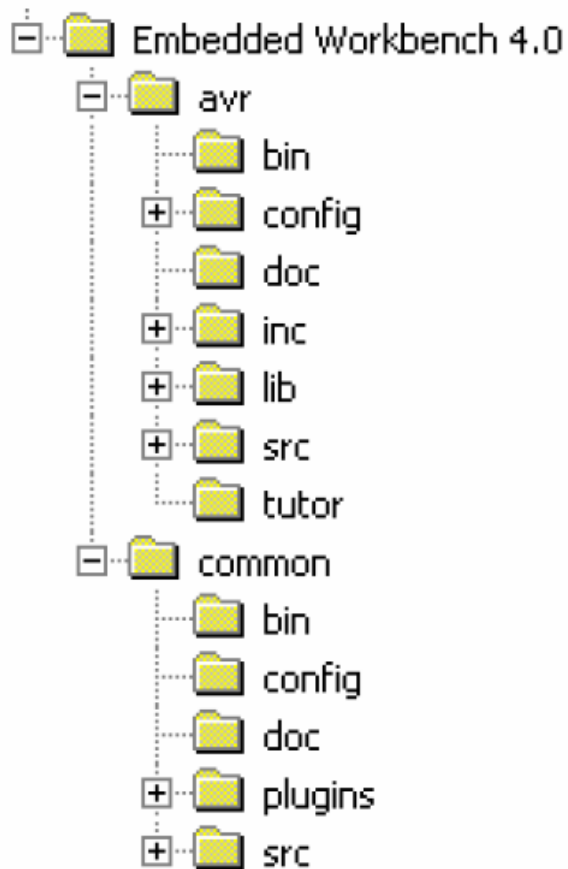


图 1 目录结构

注意：安装路径可与上图设置不同，这取决于先前安装的 IAR 产品以及用户个人的喜好。

AVR 目录

Avr 目录包含所有特定产品的相关子目录。

Avr\bin 目录

Avr\bin 子目录包含特殊 AVR 插件的可执行文件，比如 AVR IAR C/C++编译器，AVR IAR 汇编器和 AVR IARC-SPY 驱动。

Avr\config 目录

Avr\config 子目录包含用于配置开发环境和工程的文件，比如：

- 连接器命令模板文件 (*.xcl)
- 特殊函数注册描述文件 (*.sfr)
- C-SPY 设备描述文件 (*.ddf)

-
- 语法着色配置文件 (*.cfg)
 - 应用工程和库工程文件的模板文件 (*.ewp) 以及它们相应的库配置文件。

Avr\doc 目录

Avr\doc 目录包含 AVR 工具的最新信息的帮助文档。我们建议用户先读一下这些文档。该目录下也包含了本手册和 AVR 参考手册的在线超文本 (PDF 格式) 文件, 还有在线帮助文件 (CHM 格式)。

Avr\inc 目录

Avr\inc 子目录包含内部文件, 比如标准 C 或 C++ 库的头文件。同样, 还有定义特定功能寄存器的特殊头文件, 而这些文件主要由编译器和汇编器来使用。

Avr \lib 目录

Avr \lib 子目录包含编译器使用的预先创建的库以及相应的库配置文件。

Avr \src 目录

Avr \src 子目录包含一些可配置库功能的源文件以及一些应用程序代码示例。此外还包含库的源代码。

Avr\tutor 目录

Avr\tutor 子目录包含本文档中的教程的相应文件。

公共目录

公共目录包含所有嵌入式 IAR Embedded Workbench 产品共享的插件所在的子目录。

Common\bin 目录

Common\bin 子目录包含所有嵌入式 IAR Embedded Workbench 产品共享插件的可执行文件, 例如 IAR XLINK Linker, IAR XLIB Librarian, IAR XAR Library Builder 以及编辑器和图形用户接口插件。IAR Embedded Workbench 的可执行文件也放置在这里。

Common\config 目录

Common\config 子目录包含嵌入式 IAR Embedded Workbench 在开发环境中所保持的设置。

Common\doc 目录

Common\doc 子目录包含了所有嵌入式 IAR Embedded Workbench 产品的共享插件的最新信息的帮助文档，例如连接器和库工具。我们建议用户先读一下这些文档。这个目录还包括“**IAR 连接器和库工具参考手册**”的 PDF 在线版文档。

Common\plugin 目录

Common\plugin 子目录包含可作为载入式插件模块的插件的执行文件与描述文件。

Common\src 目录

Common\src 子目录包含所有嵌入式 IAR Embedded Workbench 产品的共享插件的源文件，比如一个简单的 IAR XLINK 连接器的输出格式文件“**SIMPLE**”。

1.2.2 文件类型

IAR 系统的开发工具的 AVR 版中使用下列默认的文件扩展名来确认 IAR 特定文件类型：

| Ext. | Type of file | Output from | Input to |
|------|---|------------------------|------------------------------------|
| a90 | Target application | XLINK | EPROM, C-SPY, etc. |
| asm | Assembler source code | Text editor | Assembler |
| c | C source code | Text editor | Compiler |
| cfg | Syntax coloring configuration | Text editor | IAR Embedded Workbench |
| cpp | Embedded C++ source code | Text editor | Compiler |
| d90 | Target application with debug information | XLINK | C-SPY and other symbolic debuggers |
| dbg | Target application with debug information | XLINK | C-SPY and other symbolic debuggers |
| dbgt | Debugger desktop settings | C-SPY | C-SPY |
| ddf | Device description file | Text editor | C-SPY |
| dep | Dependency information | IAR Embedded Workbench | IAR Embedded Workbench |
| dni | Debugger initialization file | C-SPY | C-SPY |
| ewd | Project settings for C-SPY | IAR Embedded Workbench | IAR Embedded Workbench |
| ewp | IAR Embedded Workbench project (current version) | IAR Embedded Workbench | IAR Embedded Workbench |
| eww | Workspace file | IAR Embedded Workbench | IAR Embedded Workbench |
| fnt | Formatting information for the Locals and Watch windows | IAR Embedded Workbench | IAR Embedded Workbench |
| h | C/C++ or assembler header source | Text editor | Compiler or assembler #include |
| i | Preprocessed source | Compiler | Compiler |
| inc | Assembler header source | Text editor | Assembler #include |
| lst | List output | Compiler and assembler | – |
| mac | C-SPY macro definition | Text editor | C-SPY |
| map | List output | XLINK | – |
| pbd | Source browse information | IAR Embedded Workbench | IAR Embedded Workbench |
| pbi | Source browse information | IAR Embedded Workbench | IAR Embedded Workbench |
| pew | IAR Embedded Workbench project (old project format) | IAR Embedded Workbench | IAR Embedded Workbench |
| prj | IAR Embedded Workbench project (old project format) | IAR Embedded Workbench | IAR Embedded Workbench |
| r90 | Object module | Compiler and assembler | XLINK, XAR, and XLIB |
| s90 | AVR assembler source code | Text editor | AVR IAR Assembler |
| sfr | Special function register definitions | Text editor | C-SPY |
| wadt | Workspace desktop settings | IAR Embedded Workbench | IAR Embedded Workbench |
| xc1 | Extended command line | Text editor | Assembler, compiler, XLINK |
| xlb | Extended librarian batch command | Text editor | XLIB |

表 2 文件格式

当用户需要确定一个文件名时，可以引用一个清楚的扩展名来覆盖默认的文件扩展名。

扩展名为 `ini` 和 `dni` 的文件是在运行嵌入式 IAR Embedded Workbench 工具时同步生成的。这些文件包含关于用户的工程配置及其他设置的信息，然后被放到工程目录下的 `settings` 子目录里。

注意：如果用户是从命令行来运行工具，**XLINK** 列表文件（映象）会采用默认的扩展名 `lst`，这样可能会覆盖由编译器所生成的扩展名。因此我们简易用户将 **XLINK** 的映象文件完整地定义出来，比如 `project1.map`。

1.2.3 文档

这部分简要介绍了在 AVR 用户手册和参考手册中地信息，以及一些在线帮助。

用户可以在嵌入式 IAR Embedded Workbench 中的 **help** 菜单里访问 AVR 的在线文档。或是按 **F1**，在 IAR Embedded Workbench IDE 中获取帮助。

我们建议用户读一下 `readme.htm` 文件以获得最新的信息，而用户手册里有可能没有这些信息。它放在 `AVR\doc` 目录下。

用户和参考手册

嵌入式 IAR Embedded Workbench 提供以下的用户和参考手册：

AVR® IAR Embedded Workbench® IDE 用户手册

即本文档。

AVR® IAR C/C++ 编译器参考手册

这本手册提供关于 AVR IAR C/C++ 编译器的相关信息。用户可以通过它查看到：

- 如何配置编译器以适应用户的目标处理器，并达到应用程序要求；
- 如何为用户的目标处理器写高效代码；
- 汇编语言接口和调用规则；
- 可用数据类型；
- 运行库；
- IAR 语言扩展。

AVR® IAR 的汇编器参考手册

该手册介绍了 AVR IAR 汇编器的相关信息，包括汇编器源代码格式的细节信息以及操作、指示符、存贮和诊断。

IAR 连接器和库工具参考手册

这篇在线的 PDF 文档提供关于 IAR 连接器和库工具的相关信息：

- IAR XLINK 连接器部分介绍了 XLINK 选项，输出格式，环境变量和诊断工具；
- IAR XAR Library Builder 库创建器部分则介绍了 XAR 选项和输出；
- IAR XLIB Librarian 库管理器部分介绍了关于 XLIB 命令，环境变量和诊断工具。

DLIB 库参考信息

该在线文档，格式为 HTML，提供了 IAR DLIB 库函数的参考信息。在 AVR® IAR Embedded Workbench® IDE 在线帮助系统中，含有该文档。

CLIB 库参考手册

PDF 格式的在线手册包含 IAR CLIB 库的参考信息。在 AVR® IAR Embedded Workbench® IDE 在线帮助系统中，含有该文档。

在线帮助

互动的在线帮助提供关于嵌入式的 IAR Workbench IDE 的菜单和对话框的参考信息。还有关于 DLIB 函数的关键信息。如果需要查看关于某一函数的相关信息，在编辑器窗口中选择该函数，然后按 F1 即可。

注意：使用 CLIB 库的过程中，如果用户在编辑器窗口中选择一个函数名称，然后按下 F1 键，将会获取 CLIB 库的参考信息。

IAR 网站

在 www.iar.com 上能找到最新的关于 IAR 系统的信息，在嵌入式的 IAR Workbench IDE 的 Help 菜单中也能访问到。浏览该网站，可以看到：

- 产品发布信息；
- 当前版本的最新更新；
- 特殊服务；
- IAR 产品的评估版拷贝；
- 技术支持，包括技术文档；
- 应用要点说明；
- 链接到芯片供应商和其他相关站点；
- 各国供应商的姓名，地址信息。

第二部分 教程

AVR® IAR Embedded Workbench® IDE用户手册的这部分包括以下章节：

- 创建一个应用工程；
- 使用 IAR C-SPY 调试器进行调试；
- C 与汇编语言模块混合；
- 使用 C++；
- 模拟一个中断；
- 使用库模块；

2.1 创建一个应用工程

这一章将向用户介绍 IAR Embedded Workbench 的集成开发环境。

这个教程展示了一个典型的开发流程，并且教用户如何使用编译器和连接器来创建一个适用于 AVR 芯片的小型应用程序。比如，创建一个工作区，以 C 语言代码创建一个工程，并编译、连接这个应用程序。

开发过程在下章中继续讲解，请参见第 37 页，“使用 IAR C-SPY 调试器进行调试”。

2.1.1 创建一个新工程

使用 IAR Embedded Workbench IDE，用户可以设计高级的工程模型。用户可以建立一个工作区，以创建一个或多个工程。并且已经有现成的工程模板用以开发应用工程和库。每个工程都可以建立以组为级别的结构，而在其中用户可以合理放置用户的源文件。每个工程用户都可以定义一个或多个 build 配置。如想了解更多关于工程模块的信息，请参看“工程管理”一章。

因为本教程中的程序都是含有很少文件的简单程序，所以没有使用高级工程模型。

我们建议用户创建一个特定目录，用来存放工程文件。在教程中，我们称这个目录为“projects”。用户可以在 AVR\tutor 目录下找到在教程中所需的文件。

在创建工程前，应先创建一个工作区。

创建一个工作区窗口

在教程程序中，第一步应先创建一个新的工作区。当用户第一次打开 IAR Embedded Workbench 时，应该已经有一个建好的工作区，在其中可以使用教程程序。如果用户是使用那个工作区的话，就可以忽略第一步。

选择 **File>New>Workspace**。现在用户已经做好了准备，来创建一个工程，并且将它放入工作区。

创建新工程

1. 创建一个新工程，选择 **Project->Create New Project**。弹出 **Create New Project** 对话框，可以让用户按照模板创建新工程。

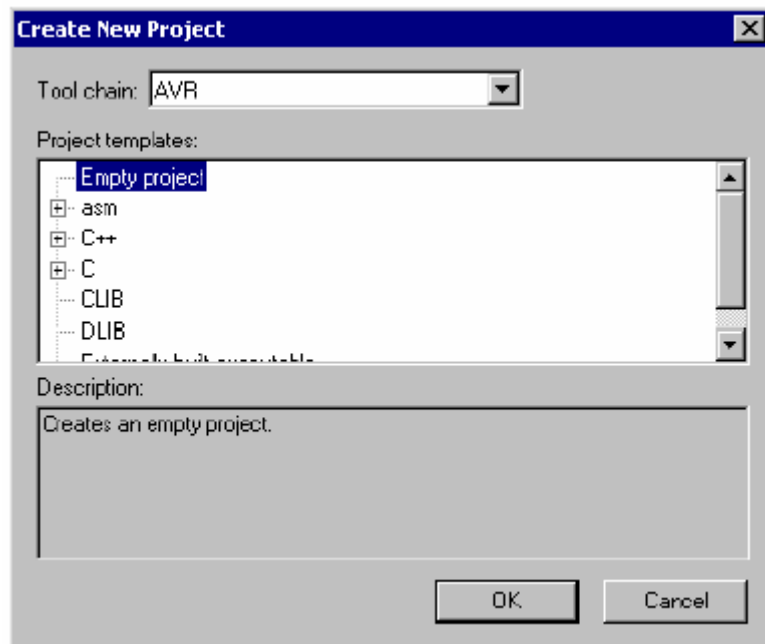


图 2 创建新的工程对话框

在教程中，选择程序模板 **Empty project**，可以快速创建一个采用默认设置的空白工程。

2. 确认 **Tool chain** 选项已经设置为 **AVR**，然后点击 **OK**。

3. 弹出一个标准的 **Save As** 对话框，确认用户想放置工程文件的地方，即新创建的 **projects** 目录。在 **File name** 对话框中键入 **project1**，然后点击 **Save**，从而创建新工程。

这个工程就出现在工作区窗口中了。



图 3 工作区窗口

默认状态下，系统产生两个创建配置：调试和发布。在教程中，只使用 **Debug**。在窗口顶部的下拉菜单中，用户可以选择 **build** 配置选项。项目名称中的星号指的是修改还没有保存。

一个工程文件，其文件扩展名为 **ewp**，已经创建在 **project** 目录下了。这个文件包含用户的工程的特殊设定，例如 **build** 选项。

4. 在用户向工程中添加任何文件时，应该先保存工作区。选择 **File>Save Workspace**，并且说明工作区文件的存放路径。在本教程中，用户还可以将它放到新建立的 **projects** 目录下。

在 File name 对话框中键入 tutorials，点击 Save 来创建新的工作区。

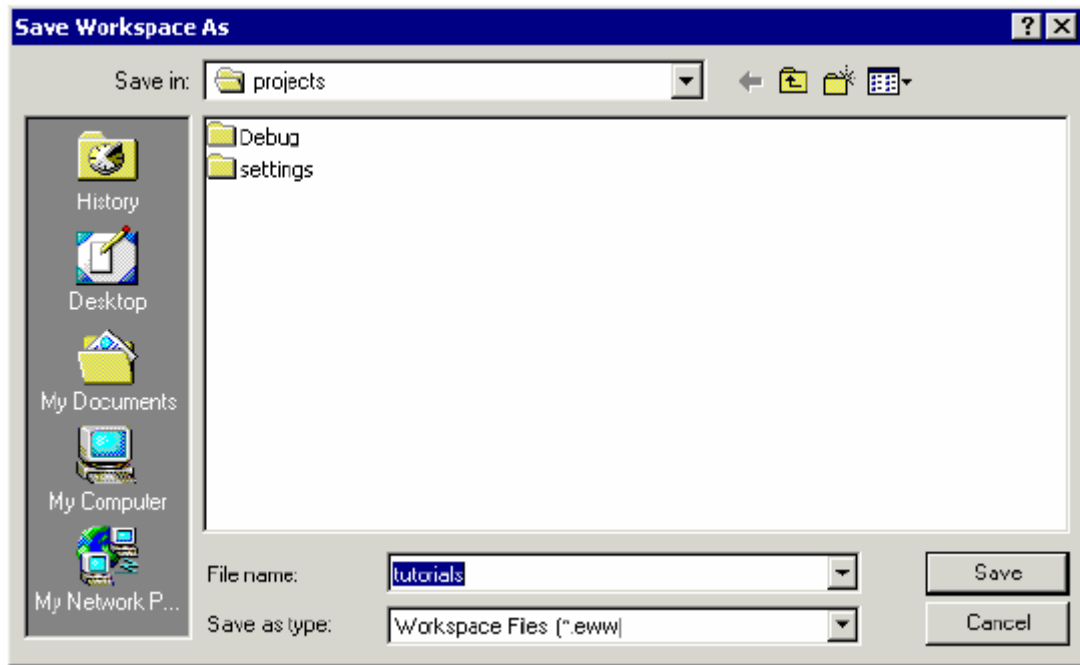


图 4 新的工作区对话框

一个工作区文件—其文件扩展名为 eww—已经创建在 projects 目录下了。这个文件列出了所有用户想加入这个工作区的工程。与之相关的信息，例如窗口放置和断点设置都放在 projects\settings 目录下。

添加文件到工程中

教程中使用源文件 Tutor.c 和 Utilities.c。

- Tutor.c 程序是只用标准 C 语言编写的简单程序。它计算出 Fibonacci 数列的前十个数，并把结果显示在 stdout 上；
- Utilities.c 程序包含了 Fibonacci 数列的相应算法。

创建几个组可以使用户根据工程需要来合理、有效地管理源文件。但是，现在这个工程中只含有两个文件，所以没有必要建一个组。如想了解更多关于创建复杂工程结构的信息，请查阅“工程管理”一章。

1.在工作区窗口中，选择用户想放置源文件的目标地址、或者一个组，在这种情况下，就直接指向工程。

2.选择 Project—>Add Files，打开一个标准的浏览对话框。转到 Tutor.c 和 Utilities.c 所在位置，将它们选中，然后点击 Open 将它们加入到工程 Project1 中。

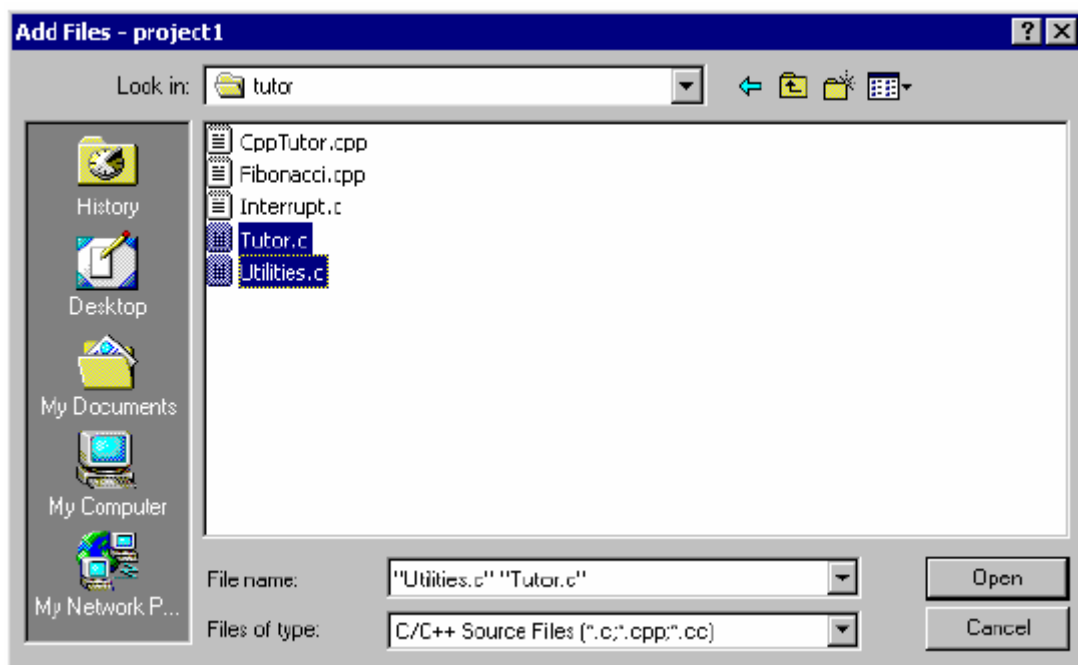


图 5 向工程中添加文件

设置工程选项

接下来要设置工程选项。对于应用工程，可以在所有节点上设置这些选项。在该教程中，首先要设置基本选项来适应处理器的配置。因为这些设置在整个 build 配置中必须一致，它们将被设置到工程节点上。

1. 在工作区窗口中选择工程文件夹 `project1->Debug`，然后选择 `Project->Options` 选项。
`General` 选项中的 `Target` 选项页被显示出来。

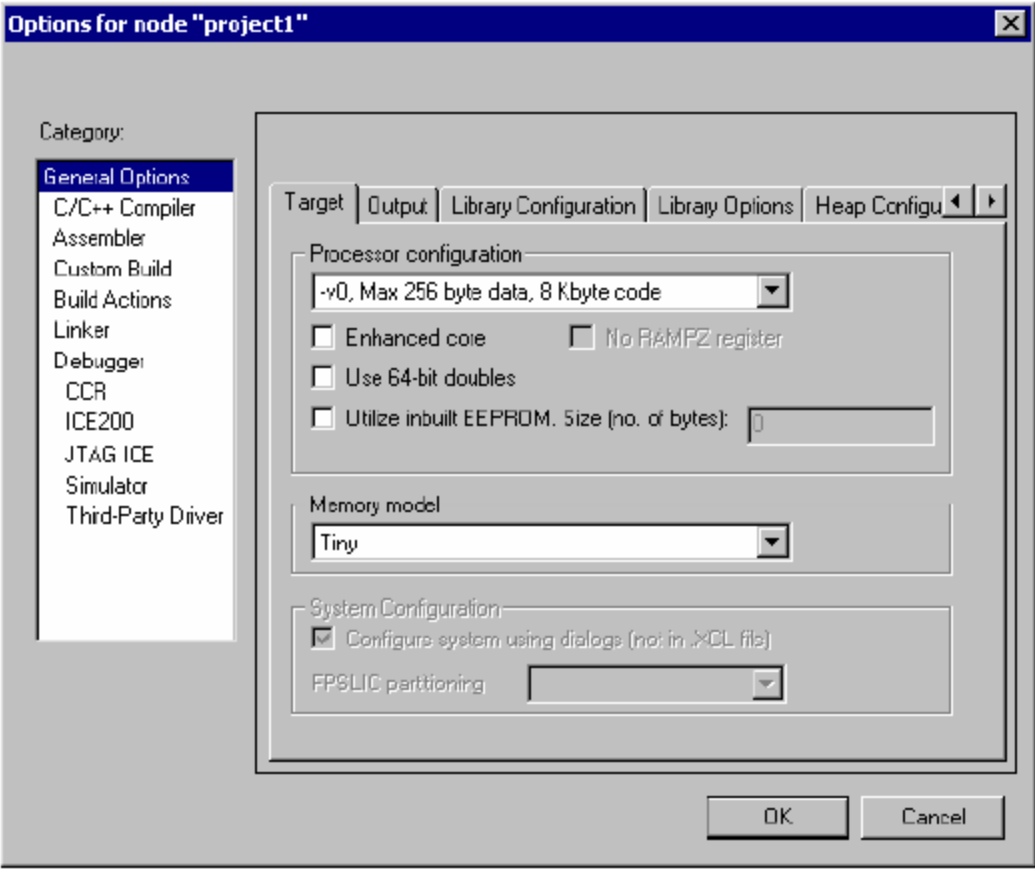


图 6 设置一般选项

确认下列设置：

| Page | Option/Setting |
|-----------------------|------------------------------|
| Target | Processor configuration: -v0 |
| Output | Output file: Executable |
| Library Configuration | Library: CLIB |

表 3 工程 1 的一般设置

接着要设定该工程的编译器选项

2.在 Category 列表中，选择 C/C++ Compiler 显示编译器选项页。

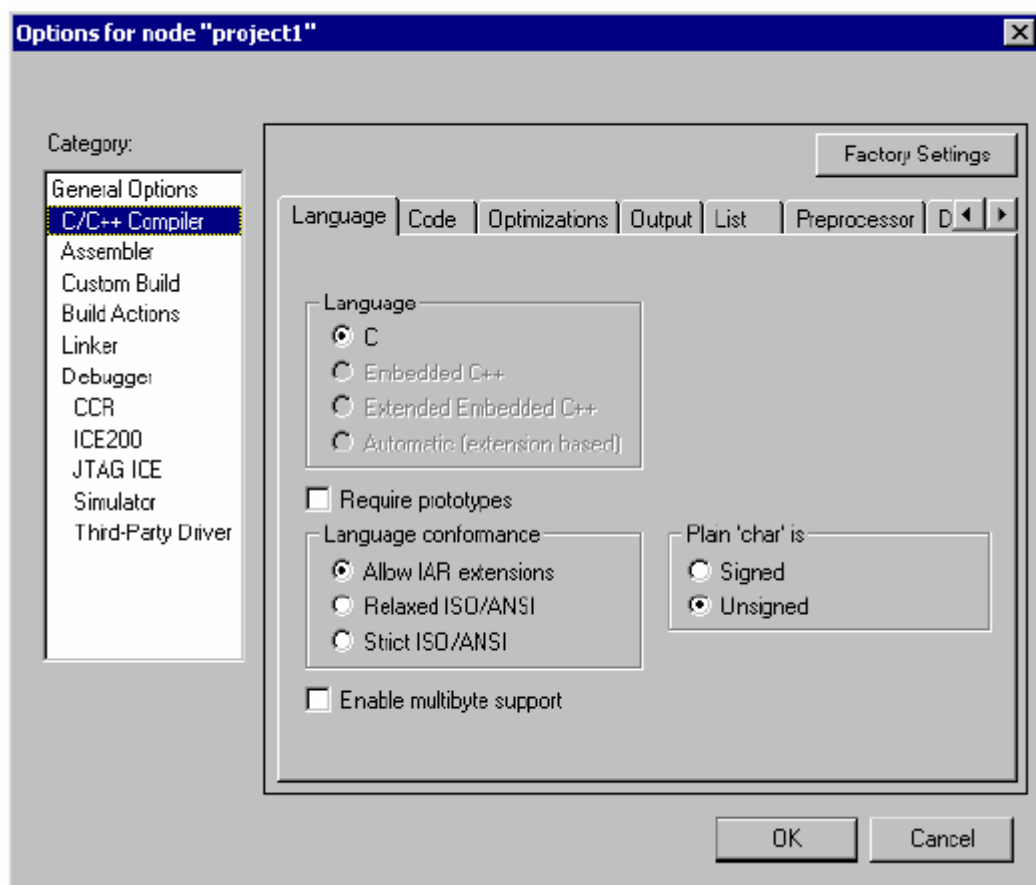


图 7 编译器选项设置

3. 确认下列设定

| Page | Setting |
|---------------|---|
| Optimizations | Size: None (Best debug support) |
| Output | Generate debug information |
| List | Output list file Assembler mnemonics |

表 4 工程 1 的编译器设置

4. 击 OK 确认用户的设置

注意：在 Build 消息窗口中显示的信息可以进行自定义设置。在本教程中，没有使用默认设置。因此，Build 消息窗口的内容可能不同于上述屏幕内容。

现在可以开始创建工程了。

2.1.2 应用程序编译和连接

至此，用户可以开始编译和连接应用程序。当然用户应该创建一个编译器列表文件和一个连接库文件，并查看它们的内容。

编译源文件

1. 编译 Utilities.c 文件，在工作区窗口中选中它。

2. 选择 Project->Compile

同样的，用户也可以点击工具栏上的 Compile 按钮或是在工作区的选择文件处点击右键弹出菜单的 Compile 命令。

其进程将显示在 Build 消息框中。

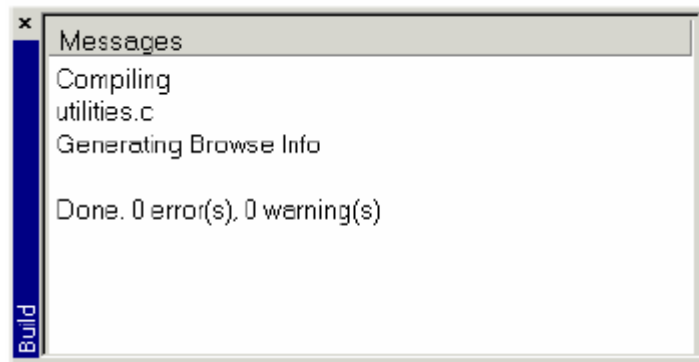


图 8 编译消息

3. 按照同样方式编译 Tutor.c 文件。

IAR Embedded Workbench 在用户的工程目录下已经创建了新的目录。因为用户正使用 build 配置命令 Debug，所以一个 Debug 目录已经创建，并且带有 list，obj 和 exe 目录：

- List 目录用来放置 list 文件。同时 list 文件扩展名为 lst;
- Obj 目录用来放置由编译器和汇编器产生的目标文件。这些文件的扩展名为 r90，并用来作为 IAR XLINK Linker 的输入;
- Exe 目录用来放置可执行文件。其扩展名为 d90，并用来作为 IAR C-SPY 调试器的输入。注意，除非用户已连接了 object 文件，否则此目录为空。

点击工作区窗口中目录树节点上的加号，使视图扩展开。用户可以看到，IAR Embedded Workbench 已经在包含输出文件的工作区窗口中创建了一个输出文件夹。此外，还根据文件间依赖关系显示了内部所有的头文件。

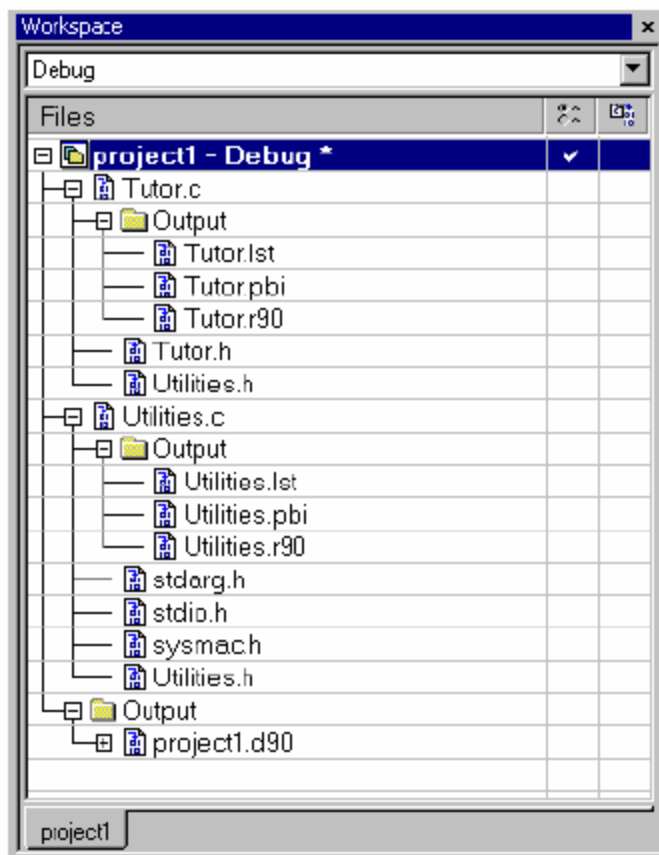


图 9 编译后的工作区窗口

查看列表文件

现在需要检查编译器的列表文件，并请注意当用户检查对生成代码大小的不同优化程度时，它是如何进行自动更新的。

1. 在工作区窗口中双击 **Utilities.lst**，打开文件。检查该文件的下列信息：

- header 部分显示产品版本，文件创建时间以及曾经使用的编译器的命令行版本；
- body 部分显示汇编代码和每个语句的二进制代码。还有分配给不同段的相关变量；
- end 部分显示堆栈大小，代码以及数据所需存储器空间，还有可能产生的错误或警告信息。

注意在文件末端显示的生成代码大小信息，并将文件一直打开。

2. 选择 **Tools->Options**，打开 **IDE Options** 对话框，点击 **Editor** 栏。

然后选择 **Scan for Changed Files**。此项功能将对编辑窗口中的文件实行自动更新，比如一个列表文件。点击 **OK** 按钮。

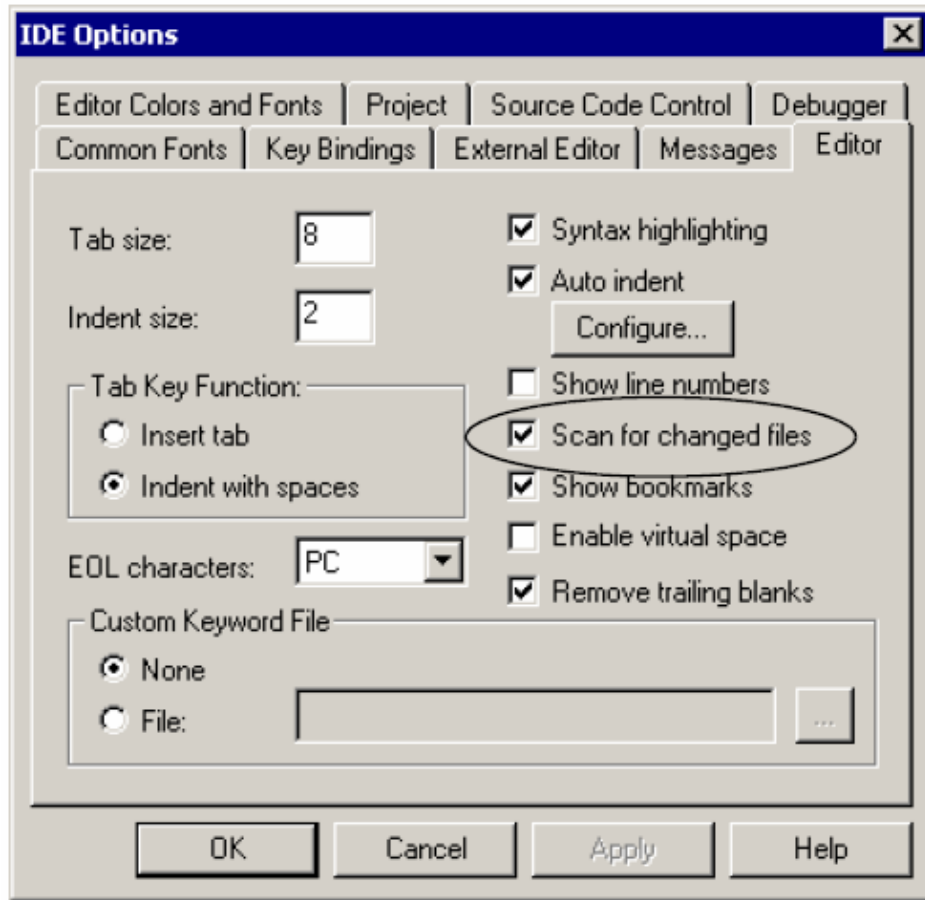


图 10 然后选择 Scan for Changed Files。

3. 在工作区窗口中选择 Utilities.c 文件。在工作区窗口内的所选文件上右击鼠标，打开 C/C++ 编译器选项对话框。点击 Code（代码）栏，选择 Override inherited settings（继承覆盖选项）。接着在 Optimizations（优化）下拉菜单中选择 High 选项。点击 OK。

注意在文件节点处的 override（覆盖）选项是指位于工作区窗口内的。

4. 对 Utilities.c 进行编译。用户需要注意两件事情。第一，打开的列表文件的自动更新取决于 Scan for Changed Files 选项。第二，观察列表文件的末尾，注意代码大小受不断优化的效果。

5. 在教程中，使用的优化程度为“None”，因此在连接应用程序之前，要先恢复为默认的优化程度。在工作区窗口内的所选文件上右击鼠标，打开 C/C++ 编译器选项对话框。不选择 Override inherited settings（继承覆盖选项），然后点击 OK。重编译 Utilities.c 文件。

连接应用程序

现在用户应该设置 IAR XLINK Linker™ 的选项。

1. 在工作区窗口中选择工程文件夹 project1->Debug，然后选择 Project->Options 选项。接着在 Category 列表中选择 Linker，打开 XLINK 选项页。

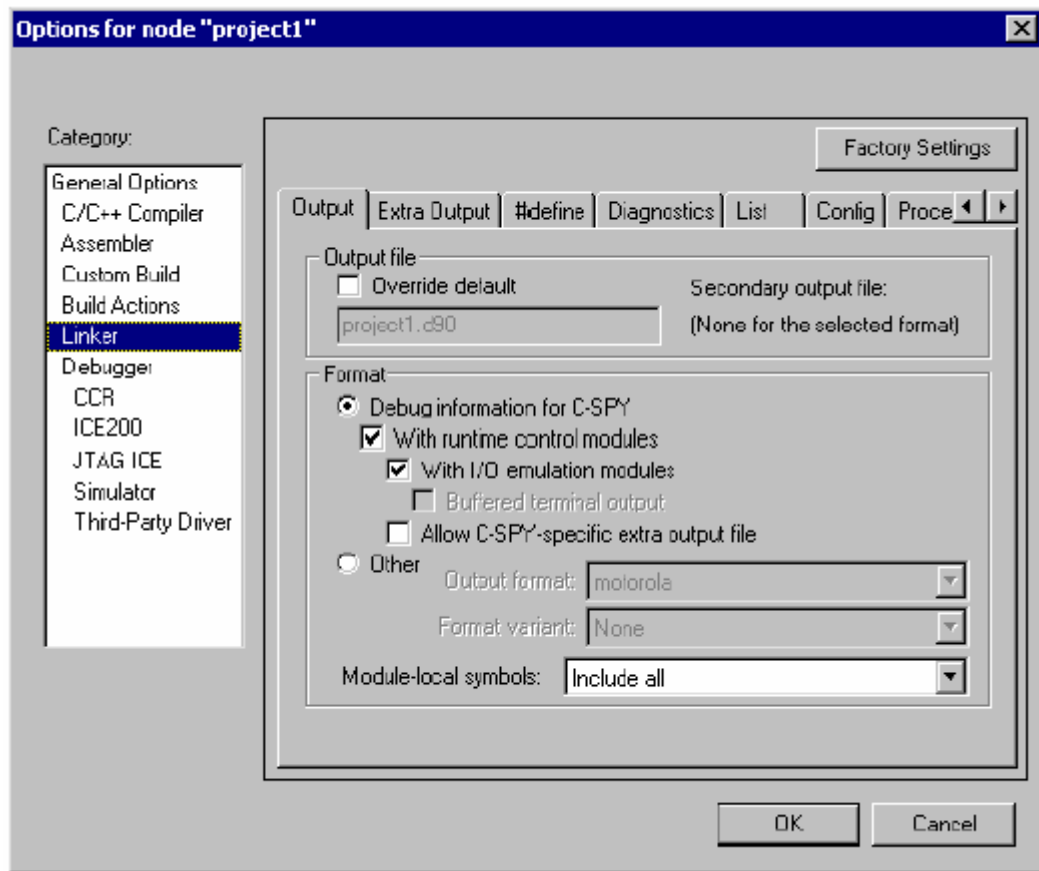


图 11 工程 1 的 XLINK 选项对话框

对于该教程，使用出厂的默认设置。对于连接器指令文件和输出格式的选择是很重要的。

输出格式

选择合适的输出格式非常重要。也许用户想将它加载到调试器中——即需要在输出中显示调试信息。在本教程中，用户会用到 C-SPY Debug information，选择 With I/O 仿真模块选项，意味着一些低级别的进程会被连接，即在 C-SPY 调试器中把 stdin 和 stdout 直接导入到终端 I/O 窗口中。用户可以在 Output 页看到这些选项。

或者，用户想将输出结果加载到一个 PROM 编程器中，此时就不需要在输出中显示调试信息，比如用 Intel-hex 或 Motorola S-records。

连接器命令文件

在连接命令文件中，用 XLINK 命令行选项控制段的放置。熟悉连接器命令文件和段的放置方法是很重要的。在 AVR® IAR C/C++编译器参考手册中，含有更详细的信息。

产品提供的连接命令模板可用在仿真器中，但当用于实际目标系统时，就必须根据实际的硬件存储情况来调整其设置。用户可以在 config 目录下找到系统提供的连接器指令文件。

该教程中，用户可以使用默认的连接命令文件，该文件可以在 Config 页中看到。

如果用户想检验连接命令文件，可以使用一个合适的文本编辑器，比如嵌入式 IAR

Embedded Workbench 编辑器，或者打印一份该文件的拷贝，然后验证其定义是否符合用户的要求。

2. 点击 OK，保存 XLINK 的设置。

现在用户应该连接目标文件了，接着生成可调试的代码。

3. 选择 Project->Make。其进程在 Build 消息窗口中显示。连接的结果是得到一个含有调试信息的代码文件 project1.d90 以及一个映象文件 project1.map。

查看映象文件

检查 project1.map 文件，观察段是如何定义以及代码是怎样放置在存储器中的。一个映象文件主要包含以下几点信息：

- header 包含连接选项；
- CROSS REFERENCE 段显示程序开始段的地址；
- RUNTIME MODEL 段显示运行模式的相关属性；
- MODULE MAP 显示被连接的文件。对每个文件而言，模块信息都作为程序的一部分被加载和显示，包括段标号和各段中定义的全址标号。
- SEGMENTS IN ADDRESS ORDER 部分列出了所有构成用户的程序的段。

至此，project1.d90 程序已经可以在 IAR C-SPY 调试器中运行了。

2.2 使用 IAR C-SPY Debugger 进行调试

这一章继续上章开始的开发过程，并讲解 IAR C-SPY Debugger 的基本特点。

注意，IAR C-SPY Debugger 安装与否，取决于用户所安装的 IAR 产品的版本。本教程假设用户正使用的是 C-SPY 软仿真器。

程序调试

在上章中创建的 project1.d90 程序，现在可以在 IAR C-SPY Debugger 上运行了。用户可以监控变量，设置断点，反汇编查看源代码，控制寄存器和存储器，以及在终端 I/O 窗口中查看程序输出结果。

启动调试器

在启动 IAR C-SPY Debugger 之前，用户必须设定 C-SPY 的几个相关选项。

1. 选择 Project->Options，然后选择 Debugger 列表。在 Setup 页，确认在 Driver 下拉菜单中选择了 Simulator 选项，接着选择 Run to main。点击 OK。

2 选择 Project->Debug。或者点击位于工具栏上的 Debugger 按钮。从而启动 IAR C-SPY Debugger，并加载了 project1.d90 应用程序。除了在嵌入式 Workbench 中已经打开的窗口，还有一系列 C-SPY 的特殊窗口。

窗口管理

在 IAR Embedded Workbench 中,用户可以在特定位置停靠窗口,并利用标签组来管理它们。用户也可以使某个窗口处于“悬浮”状态,即让它始终处于其他窗口的上层。如果用户改变了“悬浮窗口”的大小和位置,其他窗口不受影响。

状态栏,位于嵌入式 Workbench 的主窗口的底部,包含了如何管理窗口的帮助信息。对于更为详细的信息,请参考 P77 页,屏幕中的窗口管理。

确保下列窗口和窗口内容始终开启,并处于屏幕上的视野内:打开了 build 配置文件 tutorials-project1 的工作区窗口,打开了 Tutor.c 和 Utilities.c 源文件的编辑窗口以及调试日志窗口。

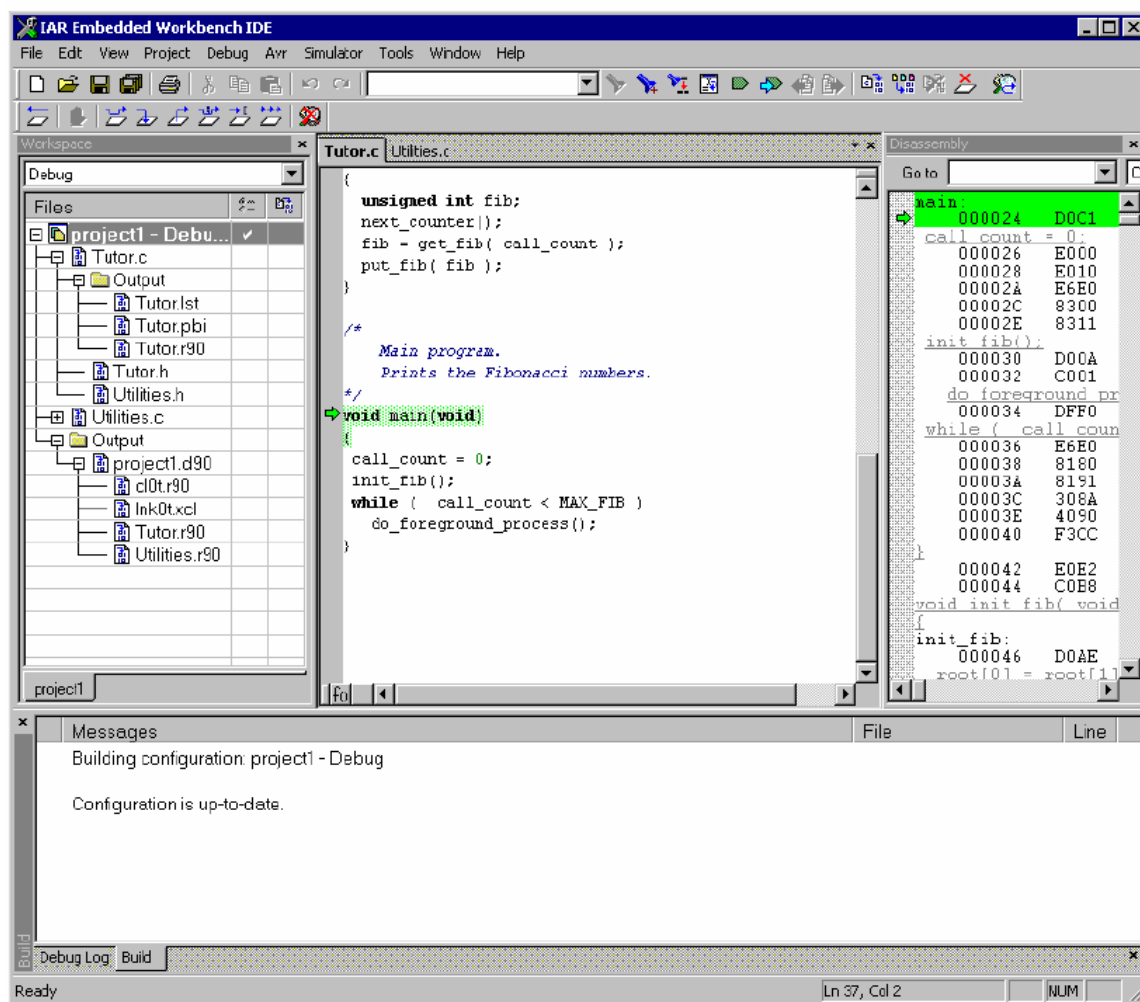


图 12 C-SPY 调试器主窗口

查看源文件语句

1. 要查看源文件语句,在工作区中双击 Tutor.c 文件。
2. 在编辑窗口中显示 Tutor.c 文件,首先执行 Debug->Step Over 命令。或者点击工具栏上的 Step Over 按钮。

当前位置应该是调用 init_fib 函数。

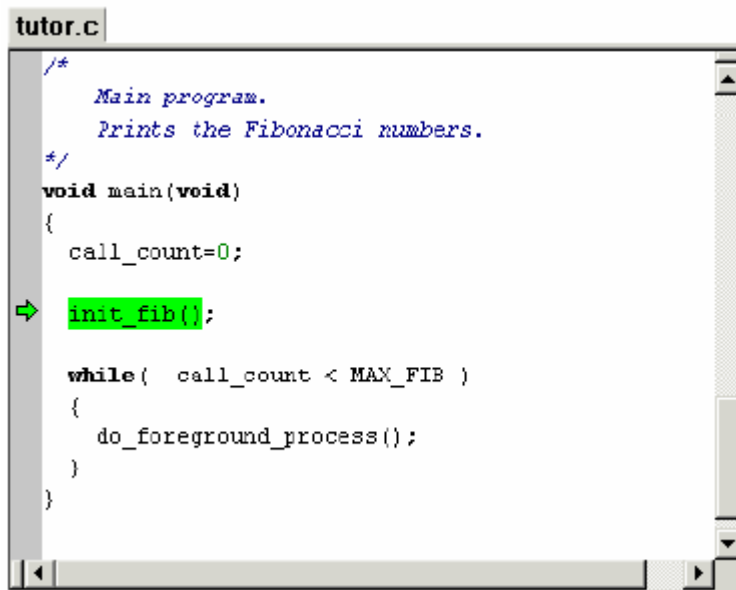


图 13 C-SPY 中的步进调试

3. 选择 Debug-> Step Into ， 程序运行到 init_fib 函数。 也可点击工具栏上的 Step Into 按钮。

在源文件级， Step Over 和 Step Into 命令使用户可以执行应用程序的一个语句或一个函数。 Step Into 命令要执行内部函数或子进程的调用，而 Step Over 每一步只执行每一个函数调用。想了结更多，参看 116 页的“单步”。

当执行 Step Into 后，用户会发现当 init_fib 函数被定位到文件中后引起当前窗口变为“Utilities.c”。

4. 使用 Step Into 命令，直到运行到 for 循环为止。

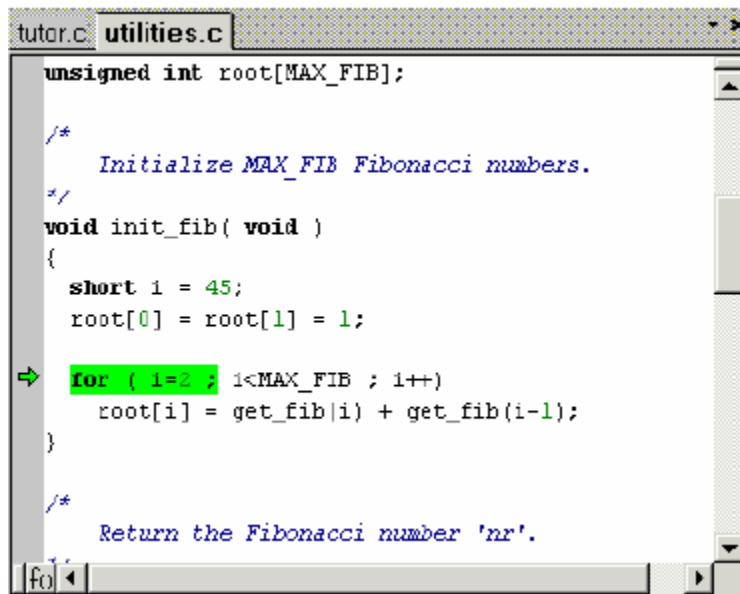


图 14 在 C-SPY 中使用 Step Into

5. 当回到 for 循环的开端，使用 Step Over 命令。用户会发现步进点处于函数调用级，而不是语句级。

当然用户也可以设置为语句级。选择 Debug->Next statement，使程序每次执行一个语句。

或者用户可以点击位于工具栏的 Next statement 按钮。
请注意该命令与 Step Over 和 Step Into 命令间的区别。

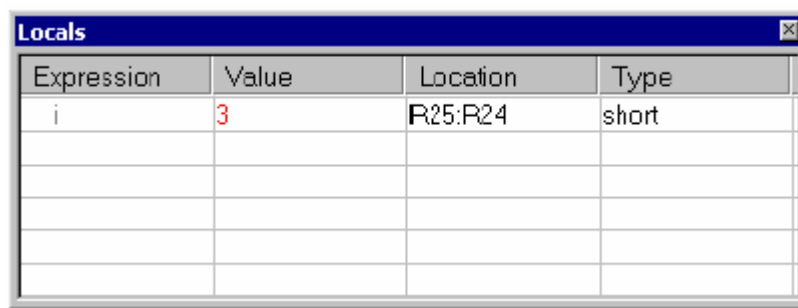
查看变量

C-SPY 允许用户查看在源代码中查看变量或表达式，使用户可以在运行程序时跟踪其值的变化。用户可以通过多种方式查看变量。例如，在源代码窗口，用鼠标点击在所要查看的变量上面，或者通过打开 Locals、Watch、Live Watch、or Auto 窗口。如需了解更多，请参看“变量与表达式”一章。

注意：当优化等级设为“None”时，所有的非静态变量在整个进程中都是可用的，因此这些变量是可完全调试的。一旦升高了优化等级，变量可能就不再是完全可调试的了。

使用自动窗口

1. 选择 View->Auto，开启 Auto 窗口。
自动窗口会显示当前被修改过的表达式。



| Expression | Value | Location | Type |
|------------|-------|----------|-------|
| i | 3 | R25:R24 | short |
| | | | |
| | | | |
| | | | |
| | | | |

图 15 在 Auto 窗口查看变量

2. 连续步进，观察 i 的值的变化情况。

设定监控点

接下来使用 Watch 窗口来查看变量。

3 选择 View->Watch，打开 Watch 窗口。注意：按照默认设置，它与当前打开的 Auto 窗口叠在同一组中。

4. 使用以下进程在变量 i 上设置一个监控点：在 Watch 窗口中点击虚线矩形。出现输入区域时，键入 i 并回车。

用户也可以从编辑窗口中将一个变量拖到 Watch 窗口中。

5. 在 init_fib 函数中选择 root 列，把它拖到 Watch 窗口中。

Watch 窗口中可以显示 i 和 root 的当前值。用户还可以扩展 root 列来监控它的更多信息。

| Expression | Value | Location | Type |
|------------|---------|-----------|------------------|
| 1 | 1 | | int |
| root | <array> | DATA:0x62 | unsigned int[10] |
| [0] | 1 | DATA:0x62 | unsigned int |
| [1] | 1 | DATA:0x64 | unsigned int |
| [2] | 2 | DATA:0x66 | unsigned int |
| [3] | 0 | DATA:0x68 | unsigned int |
| [4] | 0 | DATA:0x6A | unsigned int |
| [5] | 0 | DATA:0x6C | unsigned int |
| [6] | 0 | DATA:0x6E | unsigned int |
| [7] | 0 | DATA:0x70 | unsigned int |
| [8] | 0 | DATA:0x72 | unsigned int |
| [9] | 0 | DATA:0x74 | unsigned int |
| | | | |

图 16 在 Watch 窗口查看变量

- 继续往下执行，看 i 和 root 的值如何变化。
- 如果要在 Watch 窗口中去掉一个变量，选中它，然后点击 Delete（删除）。

设置并监控断点

IAR C-SPY 调试器拥有一个多重特性的强大的断点调试系统。想了解不同的断点的详细信息，请参看 127 页的“断点系统”一章。

使用断点最便捷的方式是将其设置为交互式的，即将插入点的位置指到一个语句里或者靠近一个语句，然后选择“Toggle Breakpoint（触发断点）”命令。

1. 在语句 `get_fib(i)` 上设置断点，其步骤为：首先，在编辑窗口中点击 `Utilities.c`，然后选择需要设定插入点的语句。然后，选择 `Edit-> Toggle Breakpoint` 命令。

或者，在工具栏上点击 `Toggle Breakpoint` 按钮。

此时，在这个语句处已经设置好一个断点。并且还会使用亮条以及在旁边的空白处标注一个 X 显示断点的存在。

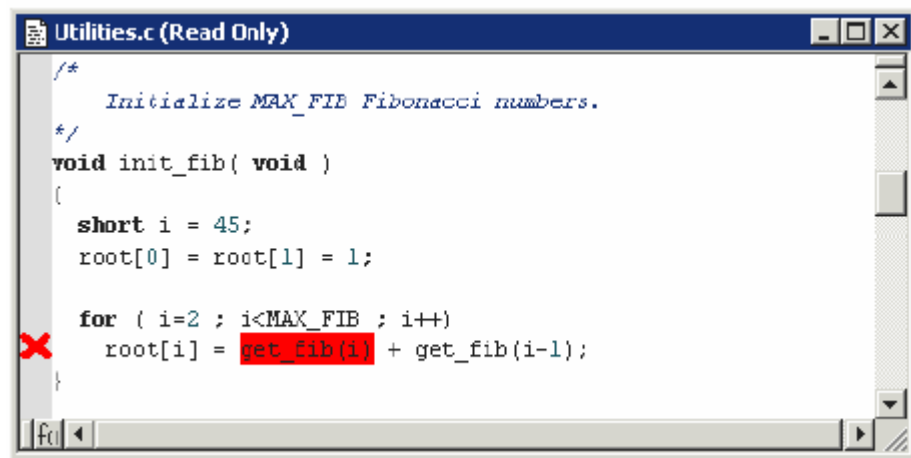


图 17 断点设置

为了查看所有断点,可以选择 **View>Breakpoints**,以打开断点窗口。在调试日志(Debug Log)窗口用户可以找到断点执行的相关信息。

运行至一个断点

2. 要使用户的应用程序运行至断点,选择 **Debug-> Go** 命令。或者点击工具栏上的 **Go** 按钮。

应用程序运行到用户设定的断点处。监控窗口会显示 **root** 表达式的值,而调试日志窗口则显示断点的相关信息。

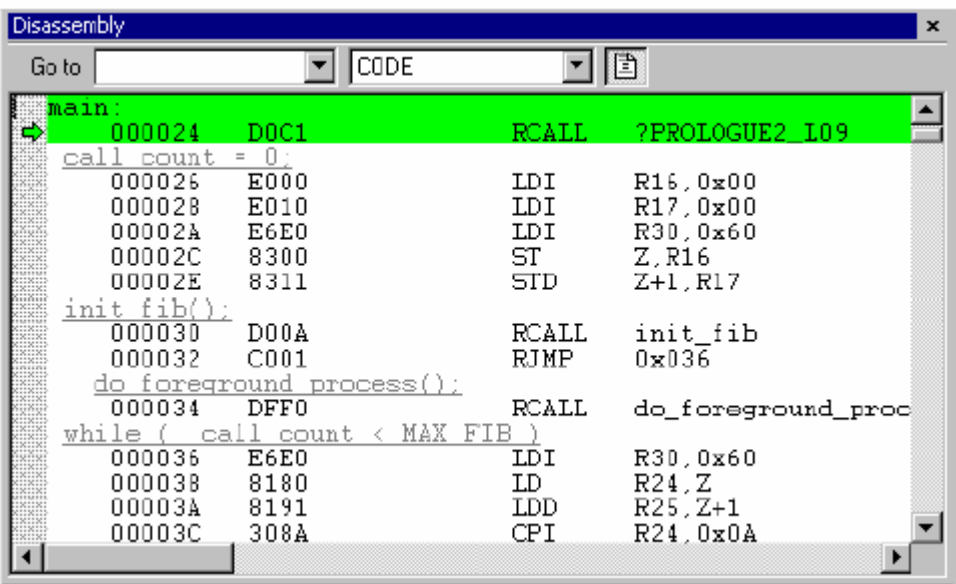
3. 选中断点,然后选择 **Edit-> Toggle Breakpoint** 命令,删除断点,使程序继续向下运行。

在反汇编模式中调试

使用 **C-SPY** 进行调试在 **C/EC++**环境中通常更快速和更简洁。然而,如果用户想对低层的进程进行完全控制,可以采用反汇编调试模式,即每一步都对应一条汇编指令。**C-SPY** 允许用户在这两种模式间自由切换。

1. 首先点击工具栏上的 **Reset** 按钮,重启动户的应用程序。

2. 如果反汇编窗口没有打开,就选择 **View-> Disassembly** 命令,打开反汇编调试窗口。用户会看到当前 **C** 语言语句的对应汇编语言编码。



```
main:
000024 D0C1 RCALL ?PROLOGUE2_I09
call count = 0;
000026 E000 LDI R16,0x00
000028 E010 LDI R17,0x00
00002A E6E0 LDI R30,0x60
00002C 8300 ST Z,R16
00002E 8311 STD Z+1,R17
init_fib();
000030 D00A RCALL init_fib
000032 C001 RJMP 0x036
do_foreground_process();
000034 DFF0 RCALL do_foreground_proc
while ( call count < MAX_FIB )
000036 E6E0 LDI R30,0x60
000038 8180 LD R24,Z
00003A 8191 LDD R25,Z+1
00003C 308A CPI R24,0x0A
```

图 18 在反汇编模式下调试
在反汇编调试窗口中,用户还可以使用其他反汇编调试命令。

监控寄存器

寄存器窗口允许用户监控并修改寄存器的内容。

1. 选择 **View->Register** 命令,打开寄存器窗口。

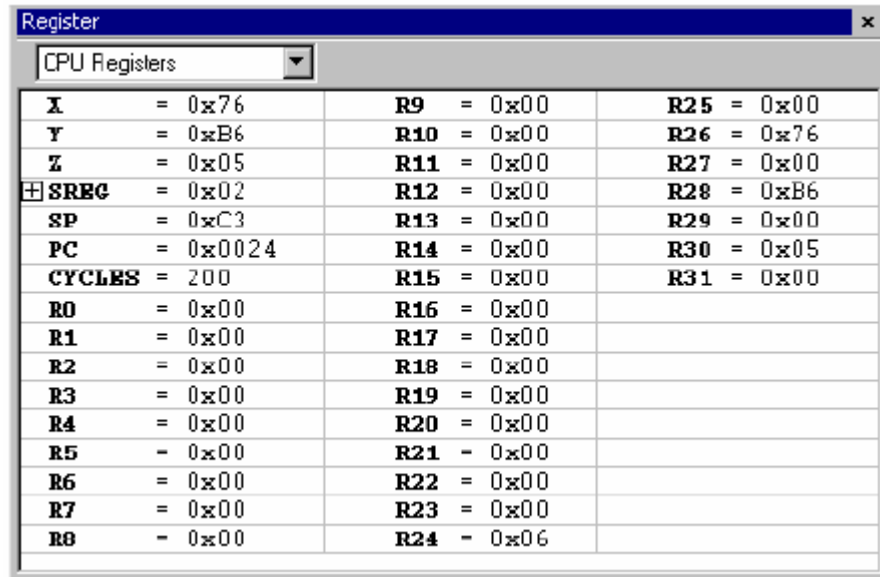


图 19 寄存器窗口

2. Step Over 命令将执行下一行指令，可以观察寄存器窗口中的值的变化。
3. 关闭寄存器窗口。

监控存储器

存储器窗口允许用户监控存储器的指定区域。在下列例子中，将监控对应 root 变量的存储器。

1. 选择 View->Memory，打开存储器窗口。
2. 激活 Utilities.c 窗口，选择 root。然后将它从 C 源代码窗口拖到存储器窗口中。同时在存储器窗口中对应 root 的值也被选中。

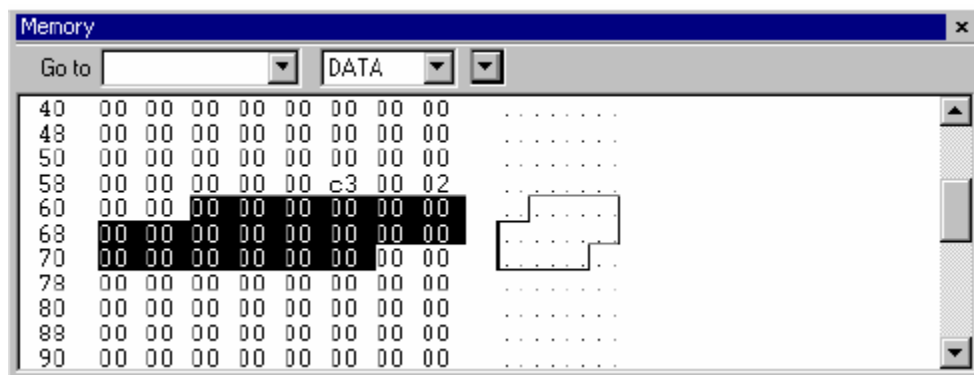


图 20 存储器监测

3. 如果要按 16 位来显示存储器中的数据值，可以点击存储器窗口工具栏的下拉菜单中的 x2 Units (2 倍单位) 命令。

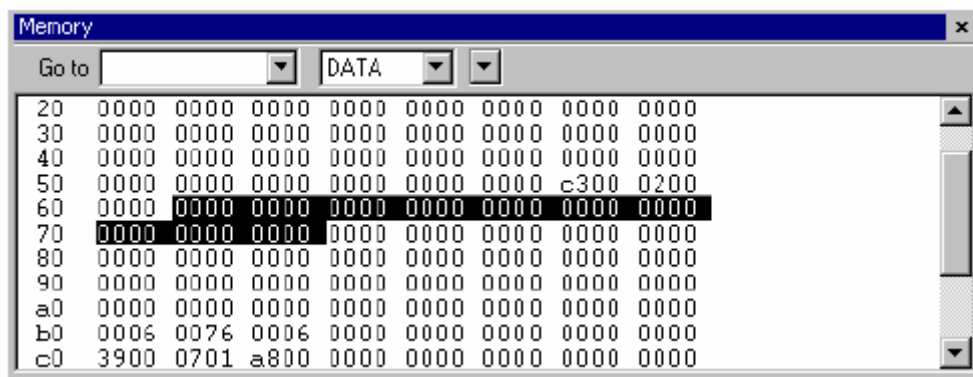


图 21 图 22 以 16 位显示内存中的数据值

如果存储器单元还没有被 `init_fib` 命令（C 语言程序）全部初始化，接着执行单步命令，用户会观察到存储器中的数值是如何更新的。

用户还可在存储器窗口中对数据值进行编辑，修改。只需在用户想进行编辑的存储器数值处放置插入点，然后键入期望值即可。

最后，关闭存储器窗口。

查看终端 I/O

有时用户也许需要对程序中的指令进行调试，以便在没有硬件支持的情况下使用 `stdin` 和 `stdout`。C-SPY 通过终端 I/O 窗口来模拟 `stdin` 和 `stdout`。

注意：终端 I/O 窗口只有在 C-SPY 使用了“With I/O emulation modules”（使用 I/O 仿真模块）输出模式对工程进行连接之后才是可用的。这意味着某些低层的程序将直接连接 `stdin` 和 `stdout` 到中断 I/O 窗口，请参见第 34 页，连接应用程序。

1. 选择 View->Terminal I/O 命令显示 I/O 操作的输出结果。

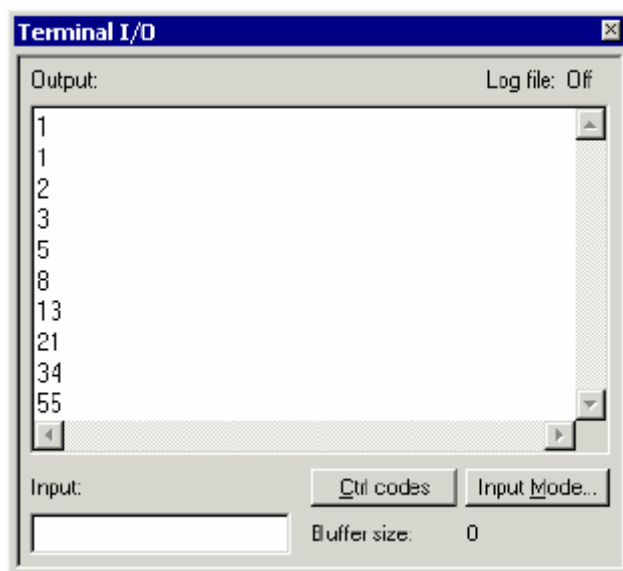


图 22 I/O 操作的输出

窗口中的内容取决于用户的应用程序运行到哪里。

程序运行完毕

1. 要完成程序运行，选择 **Debug-> Go** 命令。或者点击工具栏上的 **Go** 按钮。
如果没有新的断点，C-SPY将一直运行到程序的末尾，然后在调试日志窗口中就会显示“a program exit reached（已到程序末尾）”的信息。

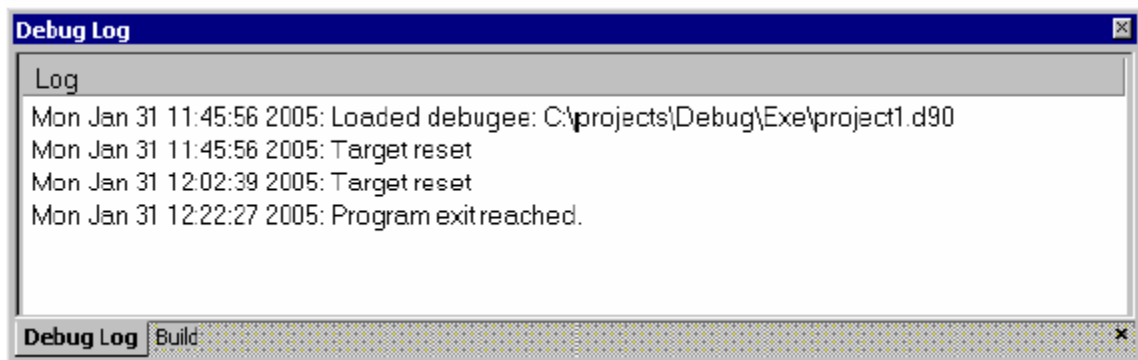


图 23 在 C-SPY 中到达程序末尾

程序中所有的输出结果都显示在终端 I/O 窗口中了。

如果用户想再次运行当前程序的话，选择 **Debug-> Reset** 命令，或者点击工具栏上的 **Reset** 按钮。

2. 要退出 C-SPY，请选择 **Debug>Stop Debugging**。也可点击工具栏上的 **Stop Debugging** 按钮，然后就看到嵌入式 **Workbench** 的工作区了。

C-SPY 还提供很多其他的调试工具。其中一些在下面的章节中有介绍，例如宏和中断仿真等。

如想了解更多关于 C-SPY 的用法，请参看第 4 部分，“调试”。

想了解 C-SPY 特性的参考信息，请参看第 7 部分，“参考信息”。

2.3 C 与汇编混合模式

在某些工程中，有时需要以汇编语言来编写某段的源代码。本章首先介绍了编译器在检验调用规则时发挥的作用，其中用户需要熟悉从 C/C++代码中调用汇编代码的过程，反之亦然。

此外，本章还教用户如何轻松在 C 源代码中加入汇编代码，对含有嵌入式 C++代码的工程，这个过程也是适用的。

此时，假设用户已经熟悉了前面章节中介绍的 IAR Embedded Workbench 的基本信息了。

2.3.1 检查调用规则

当编写被一个 C 语言程序调用的汇编程序时，就必须熟悉编译器使用的调用规则。以 C 语言编写主要代码，然后使用编译器生成相应的汇编文件，用户可以研究这一过程和相应的汇编生成文件，就会对调用规则有详细的认识。

在以下例子中，将通过编译器，从 **Utilities.c** 文件生成汇编文件。

1. 在 **tutorials** 工作区中创建一个新工程，并命名为 **project2**。
2. 在该工程中，添加 **Tutor.c** 和 **Utilities.c** 文件。

要查看工作区内的全部工程，点击工作区窗口底部的 Overview 标签。如果只查看新创建的工程，点击 project2 标签。而现在，project2 应该位于当前可视窗口中。

- 3. 要设置选项，请选择 Project>Options，然后选择 General Options 选项。
在 Target 页，从 Processor configuration 下拉菜单中选择--cpu=m128，然后从 Memory model 下拉菜单中选择 Small。
- 4. 要在工作区窗口中设定文件级节点选项，选中文件 Utilities.c
选择 Project->Options 命令。用户会发现只有 C/C++ Compiler(C/C++ 编译器) 和 Custom Build （自定义编译）列表是可选的。
- 5. 在 C/C++ Compiler（C/C++ 编译器）列表中，选择 Override inherited settings （覆盖继承设置），并确认以下设置：

| Page | Option |
|---------------|---|
| Optimizations | Size: None (Best debug support) |
| List | Output assembler file Include source Include compiler runtime information (deselected). |

表 5 工程 2 的编译器选项

注意： 在这个例子中，编译代码以显示局部和全局变量访问时应该使用低级优化模式。如果使用了高级优化模式，指向局部变量的信息将被删除。实际的函数声明没有随优化等级而改变。

- 6. 点击 OK，返回到工作区窗口中。
- 7. 对 Utilities.c 进行编译。用户可以在 projects\debug\list 子目录下找到 Utilities.s90 这个输出文件。
- 8. 要检查调用规则，并观察 C 或 C++代码在汇编语言中如何表示，请打开 Utilities.s90 文件。
用户可以研究参数在哪里，怎样被传递，当完成一个函数调用后如何回到程序体中，以及如何接收返回值。用户还可以研究一个汇编级的子函数必须占用哪些寄存器。
要为用户的应用函数获取正确的接口，用户必须为每一个必需的函数编写主干代码。
关于更为详细的编译器中的调用规则，请参见 AVR® IAR C/C++ 编译器参考手册。

2.3.2 在工程中添加一个汇编模块

这个教程演示了如何轻松地创建一个带有汇编模块和 C 模块的工程。用户还可以对它进行编译，并查看汇编输出的列表文件。

创建工程

- 1. 通过删除文件 Utilities.c 并添加文件 Utilities.s90，修改工程 2。
注意：要在 Add files（添加文件）对话框中查看汇编文件，选择 Project->Add Files 以及在 Files of type（文件类型）下拉菜单中选择 Assembler Files（汇编文件）。
- 2. 在工作区窗口中选择工程级节点，选择 Project->Options。在 General Options, C/C++ Compiler 和 Linker 表中都使用默认设置。选中 Assembler 列表，点击 List 标签，然后选择 Generate list file（生成列表文件）选项。

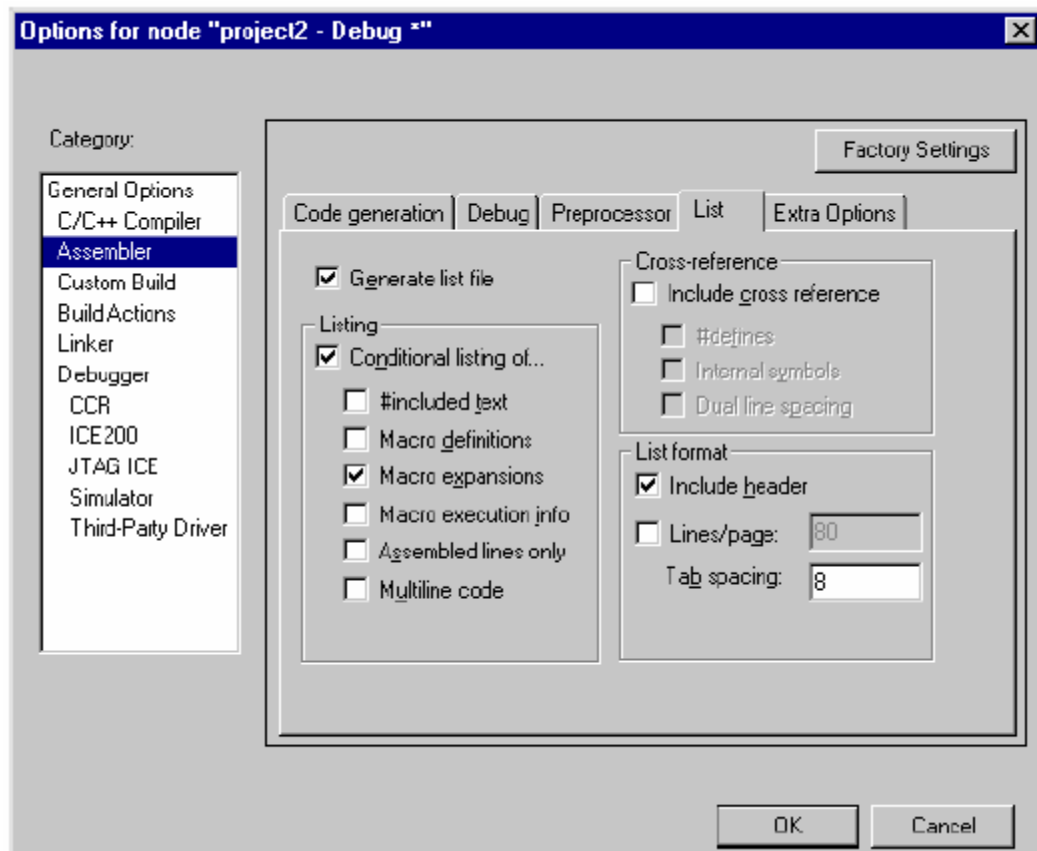


图 24 创建列表文件时的汇编设置

点击 OK。

3. 在工作区窗口中选择 Utilities.s90 文件，然后点击 Project->Compile 命令，开始汇编。假设其源文件已经编译成功，并生成了可连接的目标文件 Write.r90。

查看汇编列表

4. 在工作区窗口中的 Output 文件夹中双击 Utilities.lst 文件，展开列表。

文件末尾包含一段生成的错误和警告信息，以及校验信息（CRC）。

想了解列表文件格式的更多信息，请参看 AVR @IAR Assembler Reference Guide（汇编器参考手册）。

5. 选择 Project>Rebuild All，以重新创建工程 2。

6. 启动 C-SPY，运行 project2.d90 应用程序，并且像在以前的教程中一样，观察其行为。

7. 完成后退出调试器。

2.4 使用 C++

在本章中，C++ 用来创建一个 C++ 类。这个类则用来创建两个独立的对象，并创建应用程序及相应调试。

我们还演示了如何设置一个条件断点。

此时，假设用户已经熟悉了前面章节中介绍的 IAR Embedded Workbench 的基本信息了。

注意，是否支持 C++取决于用户所安装的 IAR 产品的版本。本教程假定支持 C++。

创建一个 C++应用程序

本教程将演示如何使用 AVR IAR Embedded Workbench 的 C++特性。文中包含两个文件：

- Fibonacci.cpp 创建一个 fibonacci 类，用来提取一系列 Fibonacci 数；
- CPPtutor.cpp 在 fibonacci 类中创建两个对象，fib1 和 fib2，并使用 fibonacci 类来提取两列 Fibonacci 数。

为验证这两个对象是相互独立的，我们以不同的速度提取这些数字。从 fib1 中提取数字采用循环中每步一取，而从 fib2 中提取数字则采用每秒一取的方式。

Fib1 目标文件使用默认定义创建，而 fib2 则以整数自变量来定义。

编译并连接 C++程序

- 1 在 tutorials 工作区中，创建一个新工程 project3。
2. 在 project3 中添加 Fibonacci.cpp 和 CPPtutor.cpp 文件。
3. 选择 Project->Options，确定下列选项被选中：

| Category | Page | Option/Setting |
|-----------------|-----------------------|--|
| General Options | Target | Processor configuration: --cpu=m128 |
| | | Memory model: Small |
| | Library Configuration | Library: Normal DLIB (C/EC++ library) |
| C/C++ Compiler | Language | Embedded C++ |
| | Code | Place string literals and constants in initialized RAM |

表 6 嵌入式 C++ 教程的工程选项

设置 C++编程语言的开关选项，只需要选择 Normal DLIB (C/EC++ library) 和 Embedded C++选项即可。

- 4 选择 Project->Make 选项，进行编译并连接用户的应用程序。
或者点击工具栏上的 Make 按钮。Make 命令对已修改过的文件进行编译与连接。
5. 选择 Project->Debug，启动 IAR C-SPY 调试器。

设定一个断点并执行到它

1. 打开 CPPtutor.cpp 窗口。
2. 要观察目标文件如何创建，在 C++目标文件 fib1 中的 fibonacci fib1 行上设置一个断点。
fibonacci fib1;

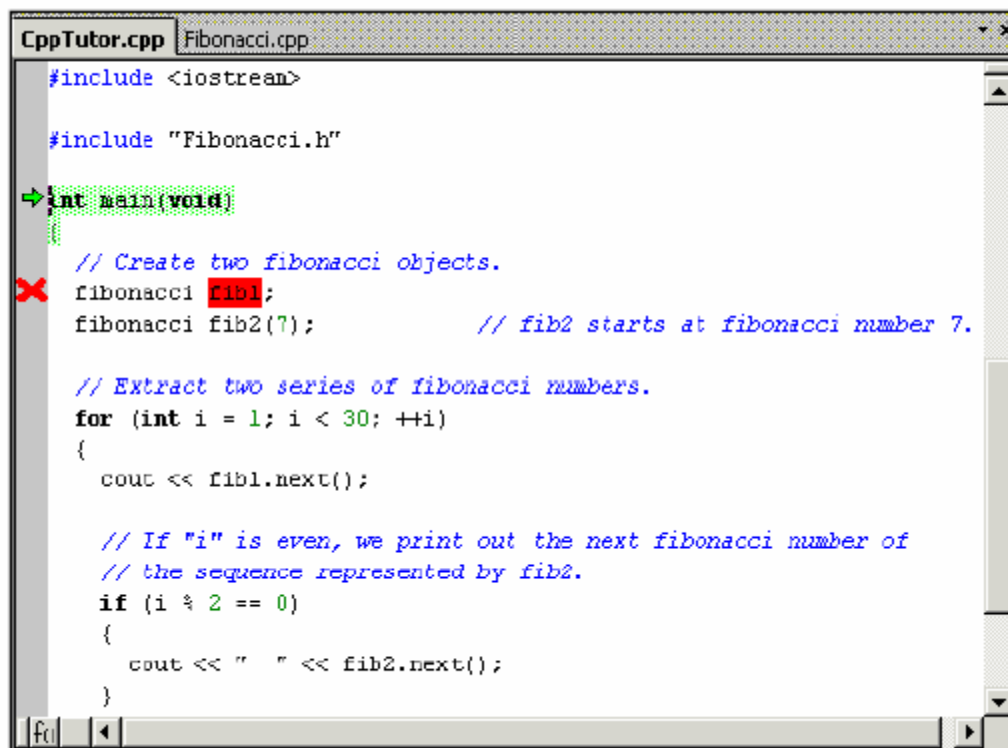


图 25 在 CppTutor.cpp 中设置断点

3. 选择 Debug ->Go,或者点击工具栏上的 Go 按钮。 光标将置于断点处。
4. 要进入构造函数, 选择 Debug->Step Into 或者点击工具栏上的 Step Into 按钮。然后再点击 Step Out。
5. Step Over 三次, Step Into 一次直到运行到 Fibonacci.cpp 程序的 next 函数。
6. 在编辑窗口的左下角点击 Go to function 按钮, 然后双击 nth 函数名, 跳转至该函数处。在 value = nth(n-1) + nth(n-2)行上的调用 nth(n-1) 函数处设置一个断点。
7. 对函数调用进行逆向追踪以及对每个函数调用的参数值进行检验都是很有趣的。对断点增设条件限制, 中断不会被触发除非条件值为真, 并且用户可以在调用堆栈 (Call Stack) 窗口中查看到每一个函数调用。

选择 View>Breakpoints, 以打开断点窗口。在断点窗口中选择断点, 右击上下文菜单, 然后选择 Edit, 以打开 Edit Breakpoints 对话框。设置 Skip count text box 为 4, 然后点击 Apply。关闭对话框。

查看函数调用

8. 选择 Debug ->Go, 运行程序直至断点条件得到满足。
- 9 当 C-SPY 在断点处停下的时候, 选择 View->Call Stack,打开调用堆栈窗口。

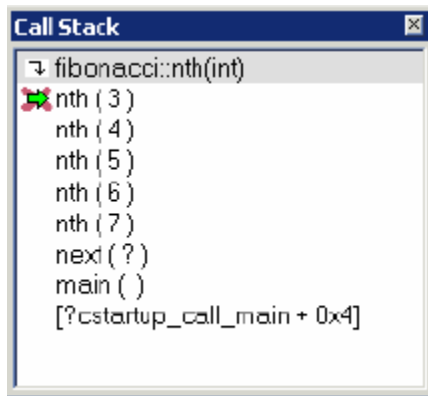


图 26 检查函数调用

现在有 5 个关于函数 `nth()` 的例子，在调用堆栈窗口中有相应显示。因为调用堆栈窗口显示的是函数变量的值，用户可以在不同的函数实例中观察到不同的值。

当用户双击这些函数实例来追踪函数调用时，也可以打开寄存器窗口来观察它是如何更新的。

打印 Fibonacci 数列

- 1 在 View（视图）菜单中打开终端 I/O 窗口。
- 2 去掉断点，执行完整程序，确认打印 Fibonacci 数列。

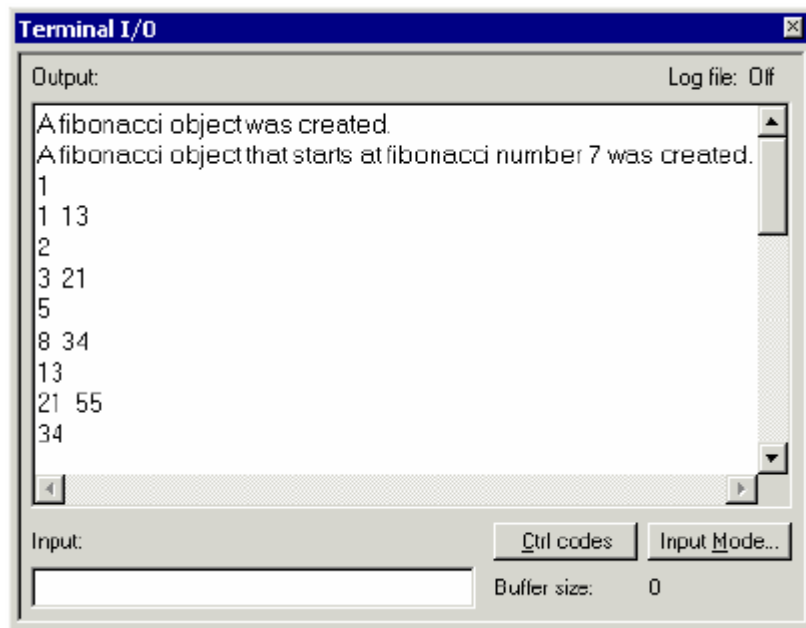


图 27 打印 Fibonacci 序列

2.5 模拟一个中断

在本章中，将介绍如何在工程中添加一个串口设备的中断处理函数。以及如何通过一个片内通信的外围设备（UART）读取 Fibonacci 数列。

文中还演示了 AVR IAR C/C++编译器的关键字和#pragma 矢量指示符是如何使用的。以及怎样通过支持中断、断点和宏来实现中断仿真。注意：这个例子并不是描述一个确切的仿真，其目的在于展示 C-SPY 宏、断点和中断系统如何有利于硬件仿真的实现。

此时，假设用户已经熟悉了前面章节中介绍的 IAR Embedded Workbench 的基本信息了。注意，只有用户使用 IAR C-SPY 软仿真器才能实现中断仿真。

2.5.1 加入一个中断句柄

这部分讲述如何轻松地编写一个中断。首先简单介绍了工程中所用的应用程序，然后讲解了如何建立一个工程。

应用程序——简介

中断句柄将从串口通信口接收寄存器（USART0）UDR0 读出数值。然后，打印其值。主程序允许中断并在等待中断过程中开始打印（.）。

编写一个中断处理函数

以下的程序行是本文档使用的中断处理函数定义（完整的源代码在 avr/tutor 子目录下的 Interrupt.c 文件中找到）。

```
// define the interrupt handler
#pragma vector=USART0_RXC_vect
__interrupt void irqHandler( void )
```

#pragma 矢量指示用于详细说明中断向量地址—本例的 USART0 中断向量收到中断—关键字 __interrupt 用于引导编译器调用中断服务程序。
要了解关于本文中使用的扩展关键字以及编译指示符的详细信息，请参看AVR® IAR C/C++ 编译器参考手册。

创建工程

1. 在 tutorials 工作区中添加一个新工程——project4。
2. 导入文件 Utilities.c 和 Interrupt.c。
3. 在工作区窗口中,选择工程级节点,然后选择 Project->Options。然后选中 General Options（基本选项）表，点击 Target（目标）标签。在 Processor configuration 的下拉菜单中，选择 --cpu=m128，然后选择 Small 存储器模式。
此外，确认 C/C++ Compiler（编译器）和 Linker（连接器）列表中使用出厂设置。
接下来，用户需要创建仿真环境。

2.5.2 创建仿真环境

C-SPY 中断系统基于一个周期计数器。在 C-SPY 生成一个中断前，用户可以指定该周期数。

要仿真 UART0 的输入，将从 InputData.txt 文件中读取数据，其中包括 Fibonacci 数列。用户可以在 UART0 接收寄存器 UDR0 上设置一个“直接读取断点”，然后连接一个用户定义的宏函数（文中用的是 Access() 宏）。这个宏函数从文本文件中读取 Fibonacci 数列。

不管何时产生中断，中断程序始终都要读 UDR0，继而触发断点，执行 Access() 宏并将 Fibonacci 数导入 UART 接收寄存器中。

直接读取断点将在处理器读取 UDR0 寄存器前触发中断，从而运行宏把通过指令直接读入的数据存入寄存器中。

这一章讲解了为实现串口中断仿真而建立的软仿真器所需的步骤。具体如下：

- 定义一个 C-SPY 安装文件，可以打开 InputData.txt，并定义 Access() 宏函数；
- 设定 C-SPY 选项；
- 创建工程；
- 启动软仿真器；
- 设定中断要求；
- 设置断点并与 Access() 宏函数关联。

注意：系统计时器中断仿真的简单示例，请参见 p168 页，简单中断仿真。

定义一个 C-SPY 的宏安装文件

在 C-SPY 中，用户可以定义宏安装文件，这些宏在 C-SPY 启动工程中将被注册。在本章中，用户将使用位于 avr\tutor 目录下的 C-SPY 宏文件 SetupSimple.mac。其创建过程如下：

首先定义宏安装函数 execUserSetup()，它将在 C-SPY 安装过程中自动运行。因此，它可以用来自动创建仿真环境。在日志窗口的信息可以证实这个宏命令确实运行过：

```
execUserSetup()  
{  
    __message "execUserSetup() called\n";
```

接着是 InputData.txt 文件，它包含了导入到 UART 的 Fibonacci 数列，打开该文件：

```
    _fileHandle = __openFile(  
        "$TOOLKIT_DIR$\\tutor\\InputData.txt", "r" );
```

然后定义 Access() 宏函数。它将从 InputData.txt 文件中读取 Fibonacci 数列，然后按接收寄存器地址分配这些数值：

```
Access()
{
    __message "Access() called\n";
    __var _fibValue;
    if( 0 == __readFile( _fileHandle, &_fibValue ) )
    {
        UDR0 = _fibValue;
    }
}
```

用户需要将 Access() 宏函数连接到一个直接读取断点。但是，这一步将在稍后的步骤中完成。

最后，文件中应包括两个在重启和退出时用于文件纠错管理的宏函数。

想了解更多关于宏的信息，请参看“使用 C-SPY 宏”一章以及 C-SPY 的“宏的相关参考”。

接下来，用户需要指定宏文件，并设置 C-SPY 的其他选项。

设置 C-SPY 选项

1 选择 C-SPY 选项，点击 Project->Options。在 Debugger（调试器）列表中，点击 Setup（安装）标签。

2 使用 Use setup file（使用安装文件）浏览按钮，确认以下宏文件被使用：

SetupSimple.mac

或者，使用一个变量来确定其路径：

\$TOOLKIT_DIR\$\tutor\SetupSimple.mac

对于详细信息，请参见第 250 页，变量声明总结

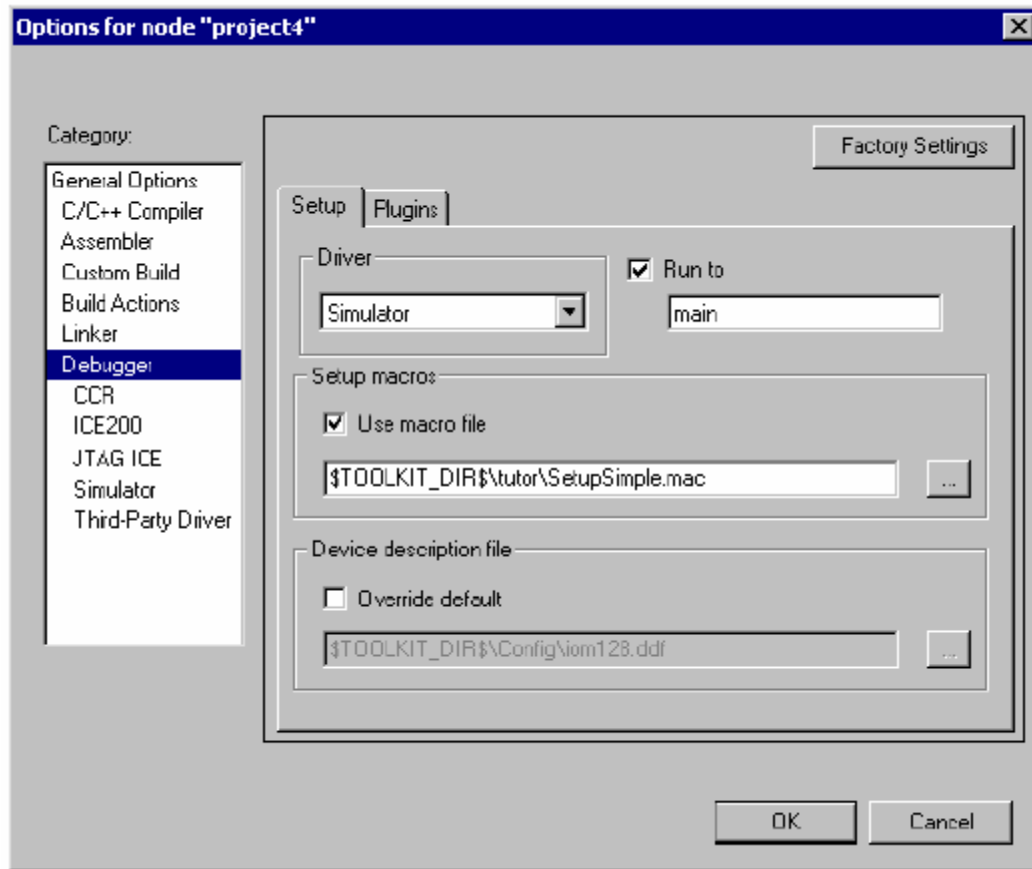


图 28 指定启动宏文件

注意： 取决于用户选择的处理器配置—iom128.ddf，合适的设备描述文件将被自动选择。该文件的信息需要用在中断系统上。

3 选择 **Run to main**（运行到主程序），然后点击 **OK**。这一步确保调试段从主程序开始。现在可以开始创建工程了。

创建工程

1 选择 **Project->Make**，编译并连接工程。

或者点击工具栏上的 **Make** 按钮。 **Make** 命令对已修改过的文件进行编译和连接。

启动软仿真器

1 启动 IAR C-SPY 调试器，运行 **project4** 工程。

出现 **Interrupt.c** 窗口（或者在其他窗口之间）。点击它，使之位于最上层（处于激活状态）。

2 从 **View** 菜单中检查 **Debug Log** 窗口。注意宏文件已经加载，并且 **execUserSetup** 函数也已被调用。

指定一个中断仿真

现在用户需要指定一个中断，使它循环 2000 次后模拟一次中断。

1 选择 Simulator->Interrupts，显示 Interrupts（中断）对话框。

2 按下表设定中断：

| Setting | Value | Description |
|-----------------|------------|--|
| Interrupt | USART0_RXC | Specifies which interrupt to use; the name is specified in the *.ddf file. |
| Activation Time | 4000 | Specifies the activation moment for the first interrupt. The interrupt is activated when the cycle counter has passed this value. |
| Repeat Interval | 2000 | Specifies the repeat interval for the interrupt, measured in clock cycles |
| Hold time | 0 | Hold time, not used here |
| Probability | 100 | Specifies probability. 100% specifies that the interrupt will occur at the given frequency. Specify a lower percentage to simulate a more random interrupt behavior. |
| Variance | 0 | Time variance, not used here. |

表 7 中断对话框

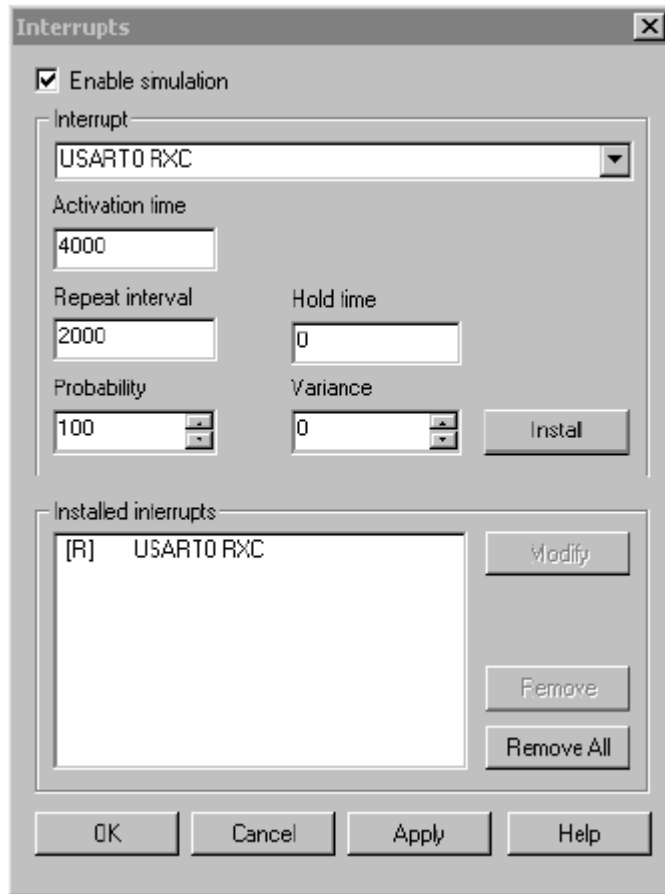


图 29 检查中断设置

在运行过程中，C-SPY 处于等待状态，直到循环计数器超过激活时间。当前运行的汇编指令执行后，C-SPY 将生成一个中断，即每循环 2000 次重复一次。

3 当用户确定设置后，点击 **Install**（安装），然后点击 **OK**，关闭 **Interrupt** 对话框。

要自动定义中断，用户可以在一个 C-SPY 安装文件中使用系统宏命令 `__orderInterrupt`。在本章结尾，我们将演示如何自动执行此过程。

设置一个立即断点

定义一个宏命令，并将它连接到一个立即断点上，由此，用户可以使这个宏命令模拟硬件设备，比如本文中提到的 I/O 接口。立即断点不会中断程序运行，只是有时用来延迟检查条件或执行相关联的宏命令。

在这个例子中，USART0 的输入是由一个设置在 UDR0 地址段上的直接读取断点动作来仿真的，并且关联了一个已定义的 `Access()` 宏命令。这个宏将完成 UART 的输入仿真。以下是其相应步骤：

- 1 选择 **View>Breakpoints** 打开中断窗口，右击以打开上下文菜单，选择 **New Breakpoint>Immediate** 以打开 **Immediate** 标签。
- 2 在断点上加入以下参数。

| Setting | Value | Description |
|-------------|------------|--|
| Break at | UDR0 | Receive buffer address. |
| Access Type | Read | The breakpoint type (Read or Write) |
| Action | Access () | The macro connected to the breakpoint. |

表 8 断点对话框

在运行过程中，当 C-SPY 在 UDR0 地址段上探测到一个读取访问时，它会暂时延缓仿真进程，然后执行 Access()宏命令。这个宏命令会从 InputData.txt 文件中读取数值，并将其写入 UDR0。C-SPY 在收到 UDR0 中返回的缓冲值后，就恢复仿真进程。

3 点击 OK，关闭 Breakpoints 对话框。

要自动执行断点配置，用户可以在一个 C-SPY 安装文件中使用系统宏命令 __setSimBreak。在本章结尾我们还会演示如何使用这个宏命令来实现自动完成。

2.5.3 中断仿真

在这部分中，用户将运行程序，并进行串口中断仿真。

运行程序

- 1 单步调试程序，当到达 while 循环时暂停，等待输入。
- 2 在 Interrupt.c 源代码窗口，放置函数 irqHandler。
- 3 在 ++callCount; 指令行上放置插入点并选择 Edit->Toggle Breakpoint，设置一个断点，或者点击工具栏上的 Toggle Breakpoint（触发断点）。或者使用上下文菜单予以选择。
如果用户想查看断点的详细信息，选择 Edit->Breakpoints。
- 4 打开终端 I/O 窗口，选择 Debug->Go,运行程序， 或者点击工具栏上的 Go 按钮。
在中断函数段，程序应该暂停下来。
- 5 再次点击 Go，查看在终端 I/O 窗口中打印的下一个数。
因为主程序对 Fibonacci 数值统计有上限，这个教学程序很快将运行到 exit 段，并停止。
终端 I/O 窗口将显示 Fibonacci 数列。

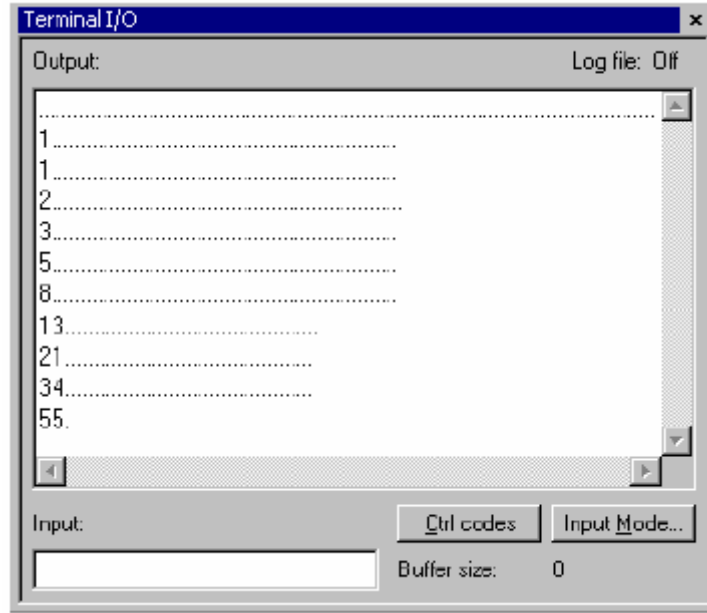


图 30 在终端 I/O 窗口中打印 Fibonacci 序列数值

2.5.4 中断和断点中宏的使用

要自动设置断点和中断定义，系统宏命令 `__setSimBreak` 和 `__orderInterrupt` 可各自相应地在宏安装命令 `execUserSetup()` 下运行。

`SetupAdvanced.mac` 文件与系统宏调用相关联，用以设置断点和确认中断：

```
SimulationSetup()
{...
    _interruptID = __orderInterrupt( "USART0 RXC", 4000,
                                     2000, 0, 0, 0, 100 );

    if( -1 == _interruptID )
    {
        __message "ERROR: failed to order interrupt";
    }

    _breakID = __setSimBreak( "UDR0", "R", "Access()" );
}
```

将前面所用的 `SetupSimple.mac` 文件替换为 `SetupAdvanced.mac`，当 C-SPY 启动时将自动设置断点，完成中断定义。因此，用户不必手动在 `Interrupts` 和 `Breakpoints` 对话框中输入数值。

注意：在用户加载 `SetupAdvanced.mac` 文件前，应该先去除以前定义的断点和中断。

2.6 使用库模块

这一章演示了如何创建库模块，以及如何将一个库与应用工程结合起来。
此时，假设用户已经熟悉了前面章节中介绍的 IAR Embedded Workbench 的基本信息了。

使用库

如果开发一个大型工程，用户很快就能积累一些有用的例行程序，并可用到应用程序中。要避免每次需要这样的程序时就汇编一次的情况，用户可以将其存为目标文件，即已汇编但还没有连接。

将这样的程序集存入一个目标文件中，即称为一个库。建议用户使用库文件来创立相关例行程序的集合，比如一个设备驱动。

我们使用 IAR XAR Library Builder 来创建库。IAR XLIB Librarian 则帮助用户操作库。它允许用户做以下操作：

- 将模块类型从 PROGRAM 改为 LIBRARY，反之亦然。
- 在一个库文件中添加或删除模块。
- 列出模块名，路径名等。

Main.s90 程序

Main.s90 程序使用一个名为 max 的函数来设置寄存器 R16 为字寄存器 R16 和 R17 的最大值。在连接的时候，EXTERN 指示符将 max 定义为外部字符。

该程序的拷贝位于 avr\tutor 目录下。

库函数

这两个库函数将构成一个独立的库。它包含由 main 调用的 max 函数，以及一个相应的 min 函数。这两个函数会对寄存器 R16 和 R17 的值进行操作，并将结果返回到 R1 中。在 R16 中返回的结果。包含这些库函数的文件名为 Maxmin.s90，在产品中有相应备份。

MODULE 指示符将这些函数定义为库模块，只有被其它模块调用时才指示 IAR XLINK Linker™ 包含这些模块。

PUBLIC 指示符将 max 和 min 符号发布给其它模块。

想了解更多关于 MODULE 和 PUBLIC 指示符的详细信息，请参看 AVR® IAR 汇编器参考手册。

创建一个新工程

- 1 在 tutorials 工作区中，创建一个新工程 project5。
- 2 在新工程中添加 Main.s90。
- 3 要设置选项，选择 Project->Options。选中 General Options 列表，并点击 Library

Configuration 图标。在 Library 的下拉菜单中选择 None，即不连接标准的 C/C++库。
其他选择列表中使用默认选项。
4 要汇编 Main.s90 文件，选择 Project->Compile。也可点击工具栏上的 Compile 按钮。

创建一个库工程

用户现在可以创建一个库工程文件了。
1 在同一个 tutorials 工作区中，添加一个新工程名为 tutor_library。
2 在这个工程中添加 Maxmin.s90 文件。
3 要设置选项，选择 Project->Options。 在 General Options（一般选项）列表中，确认以下设置：

| Page | Option |
|-----------------------|----------------------|
| Output | Output file: Library |
| Library Configuration | Library: None |

表 9 库工程的 XLINK 选项

注意：Library Builder 出现在列表中，这就意味着 IAR XAR Library Builder 已经被加入到创建工具链中。在本教程中无需对 XAR 的任何具体的选项做设定。
点击 OK。
4 选择 Project->Make。
至此，库输出文件 tutor_library.r90 已经创建成功了。

在应用工程中使用库

现在可以将带有 maxmin 程序的库添加到 project5 中了。
1 在工作区窗口中，点击 project5 标签。选择 Project->Add Files，然后添加位于 projects\Debug\Exe 目录下的 tutor_library.r90 文件。
点击 Open。
2 点击 Make，开始编译工程。
3 现在可以将一个可执行工程与一个库结合起来，所生成的应用程序可以运行。要了解如何操作库，请参看 IAR Linker 和 Library Tools 参考手册。