轻松入门开发 ARM 程序

（为初学者写的入门手册）

万利电子有限公司

www.manley.com.cn

目　　录

前　言 ……2

第一章　EWARM 集成开发环境及 J-Link 仿真器……..…3

第二章　在 EWARM 中生成一个新项目..……..11

第三章　编译和连接应用程序..……...……15

第四章　用 C-SPY 调试应用程序..……..21

第五章　如何编写 ST ARM 应用程序..…….....29

前 言

很多朋友都想自己动手开发 ARM 应用程序，却不知从哪儿下手。我们编写本手册的目的就是想帮助这些初学者轻松的入门。

应该说，入门学会开发 ST ARM 应用程序并不难，但要全面掌握和融会贯通则需要付出许多努力。因为本手册的目的只是帮助新手入门，所以我们在编写时大胆地砍去许多旁支侧节，避免过多介绍细节，紧紧地抓住如何使用 ARM 开发工具和使用芯片商提供的软件库为主线，单刀直入，使读者无须阅读许多资料，短时间就能掌握开发 ARM 应用程序的基本技巧。

本手册以 IAR 公司的 EWARM 开发环境和 ST 公司的 STR710-EVAL 评估板为基础，介绍如何编写 STR710（ARM7 内核）的 C 语言应用程序。手册的前半部分介绍 IAR Systems 公司的 ARM 开发环境 EWARM 的使用方法，后半部分介绍如何利用 ST 公司提供的 STR710 软件库和例子编写自己的应用程序。用户只需按照本手册介绍的步骤，一步一步去做，相信只需几天的时间就能初步掌握使用 IAR EWARM 开发环境和编写 ST ARM 的应用程序的基本技巧。

相对 51 单片机，ARM 的结构和应用确实要复杂的多。要真正熟练使用 ARM 处理器需要学习许多相关知识。希望学习 ARM 的朋友在初步入门后，继续学习本手册中提到的相关资料，通过不断的学习和实践扩充知识和提高自己的水平。

实践 ARM 开发需要准备以下软硬件：

1. IAR 的 JTAG 仿真器 J-Link （从万利公司购买）
2. IAR Embedded Workbench for ARM version 4.31 （从 IAR 网站下载）
3. ST 公司的 ATR710-EVAL 评估板 （从万利公司购买）
4. ST 公司的 ATR710-EVAL 评估板源程序包 01STR71x （从万利网站下载）

需要获得更多知识请参阅以下资料：

1. *ARM Embeded Workbench User Guide* （在 EWARM 主菜单的 Help 下）
2. *ARM C/C++ Compiler Reference Guide* （在 EWARM 主菜单的 Help 下）
3. *ARM_Assembler Reference Guide* （在 EWARM 主菜单的 Help 下）
4. *Linker and Library Tools Reference Guide* （在 EWARM 主菜单的 Help 下）
5. *STR71xF 用户手册* （从万利网站下载）
6. *STR71xF 教程（PPT）* （从万利网站下载）

第一章 EWARM 集成开发环境及 J-Link 仿真器

1．IAR EWARM 简介

IAR Embedded Workbench for ARM（下面简称 IAR EWARM）是一个针对 ARM 处理器的集成开发环境，它包含项目管理器、编辑器、C/C++编译器和 ARM 汇编器、连接器 XLINK 和支持 RTOS 的调试工具 C-SPY。在 EWARM 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。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。有关 EWARM 详细特点介绍请参考万利公司编写的《IAR EWARM 快速用户指南》。

IAR EWARM 的最新版本是 4.31。为了方便用户学习评估，IAR 提供一个限制 32k 代码的免费版本。用户可以到 IAR 公司的网站 www.iar.com/ewarm 或万利电子的网站 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）。使用它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。有关 EWARM 模拟程序的用法请参考万利公司编写的《IAR EWARM 快速用户指南》。但本手册着重介绍针对 ST 公司的 STR710 评估板真实硬件的开发流程。

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。结合 IAR EWARM 集成开发环境可支持所有 ARM7/ARM9 内核芯片的仿真。

J-Link ARM 主要特点如下：

- 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
- 自动监测所有 JTAG 信号和目标板电压
- J-Link 本身使用 PC 机给 USB 端口的电源

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。连接时应首先将 J-LINK 与 PC 连接,再将 J-LINK 与目标系统连接，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）。

注意：

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，再将连接目标板的 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。

图 1-1. 带 5V 适配器的 J-Link

3．安装 EWARM

用户可以从万利电子有限公司的网站主页 www.manley.com.cn 下载 EWARM 4.31A(32K)版本的软件(包含 J-LINK 驱动程序)及许可证。

图1-2．万利公司主页

安装EWARM 4.30A软件方法如下：

① 将安装程序包解压

② 选择并运行文件 autorun.exe

③ 点击 Install the IAR Embedded Workbench，开始安装。见图 1-3。

图 1-3. 安装 EWARM

④ 输入许可证号（License）和密钥（License key）

用户从下载的软件包中的文本文件中提取许可证号（License）和密钥（License key），分别输入下面两个窗口（见图 1-4 和图 1-5）。许可接受后建议按默认设置安装。

图 1-4. License 输入窗口

图 1-5. License key 输入窗口

⑤ 安装完成后添加 J-LINK 驱动程序

退回安装主界面，点击 Install drivers 后弹出图 1-6 的目录。选择其中的 JLink 目录，将其整个复制到另一个文件夹中（用户可以新建一个文件夹）。

图 1-6. 安装 J-Link 驱动程序

用 USB 电缆将 J-LINK 仿真器与 PC 机连接，这时会弹出图 1-7 所示硬件安装向导窗口

图 1-7. 硬件安装向导窗口

选择从列表或指定位置安装，指向先前新建文件夹中的 JLink 目录，如图 1-8 所示。

图 1-8. 指向 JLink 目录

安装完成后启动 EWARM 环境，如图 1-9 所示。

将 J-Link 的 20 芯电缆与 STR710-EVAL 评估板的 JTAG 插座相连。接通评估板电源，按第二章中步骤执行后面的操作。

第二章 在 EWARM 中生成一个新项目

要为某个目标系统开发一个新应用程序，必须建立一个新项目。步骤如下：

1．建立一个项目文件目录

首先应该为新项目创建一个目录，用来存放与项目有关的各种文件。项目开发过程中生成的一系列文件，如：工作区文件，开发环境的配置，编译、连接和调试选件配置，各种列表文件和输出文件等都将被存放在这个目录下。用户也可以选择把各种原文件也放在这个目录下。在下面的例子中我们生成一个 C:\Program files\IAR System\My projects 目录。为了方便，我们把 ST 的例子软件包 01STR71x 也复制到此目录下。

2．生成新的工作区

EWARM 虽然是按项目进行管理，但是要求把所有的项目都放在工作区内（Workspace）。用户如果是第一次使用 EWARM 开发一个新项目，必须先创建一个新工作区，然后才能在工作区中创建新项目。一个工作区中允许存放一个或多个项目。如果用户过去已经建立了一个工作区并且希望把目前要建的新项目放在老工作区内，则可以直接打开老工作区并执行第三步生成新项目。

创建新工作区方法是：选择 EWARM 开发环境主菜单的 File > New > Workspace 命令。如果目前正在开发或调试另一个项目，EWARM 开发环境将提示用户保存和关闭当前活跃工作区，然后开启一个空白工作区窗口，如图 2-1 所示。

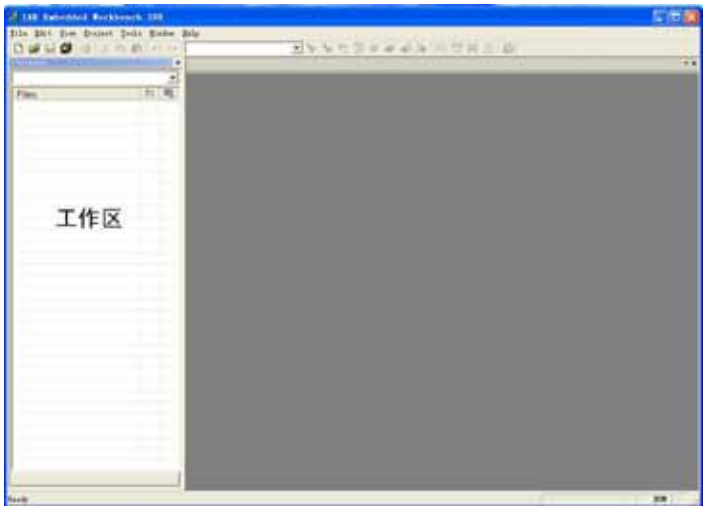

图2-1．创建新工作区

注意：第一次使用 EWARM 开发环境时也显示上述空白工作区窗口。

3．生成新项目

下一步就是在工作区中创建新项目，方法如下：

① 选择主菜单 Project > Create New Project，弹出生成新项目窗口，见图 2-2。EWARM 提供几种应用程序和库程序的项目模板。如果选择 Empty project，表示采用默认的项目选件设置。在本例中我们选择 Empty project。

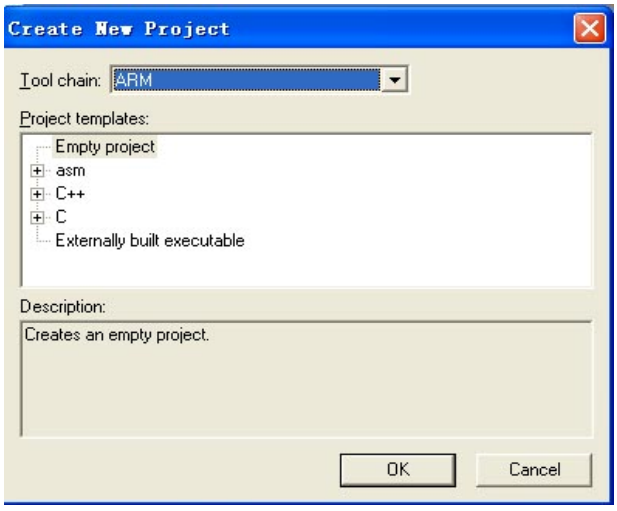

图 2-2. 生成新项目窗口

② 在 Tool chain 栏中选择 ARM，然后点击 OK 按钮，弹出 “ 另存为 ” 窗口。

③ 在 “ 另存为 ” 窗口中浏览和选择新建的 My projects 目录，输入新项目的文件名为 STR710-LED，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和输出代码模式。见图 2-3 所示：

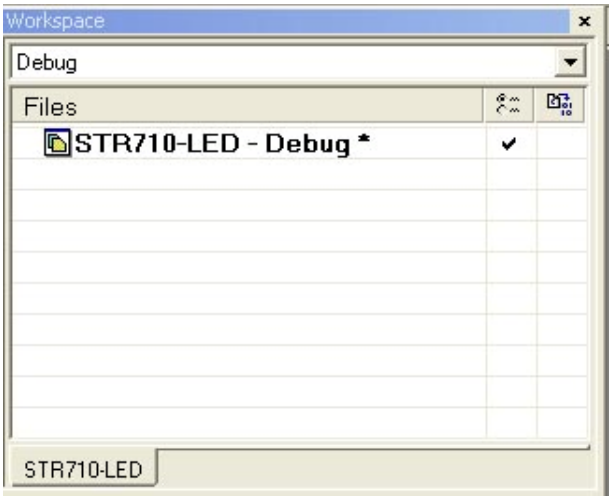

图 2-3. 生成新项目

项目名后面的 Debug 表示输出含调试信息的代码文件。EWARM 能为项目提供两种输出代码模式：Debug 和 Release。Debug 模式生成含调试信息的程序代码，用户利用它可以在 EWARM 环境下调试应用程序。而 Release 模式生成不含调试信息的发行版本的程序代码，其代码比较紧凑。用户可以从 Workspace

窗口顶部的下拉菜单中选择两种项目配置之一，本例我们选择 Debug。

现在 My projects 目录下已生成一个 STR710-LED.ewp 文件，该文件中将包含与 STR710-LED 项目设置有关的信息，如编译、连接（build）的选件等。

注意：STR710-LED-Debug 后的 * 号表示当前的工作区和项目经修改后还没有被保存。

④ 保存工作区

新生成的工作区需保存才有效，所以在添加项目后 EWARM 要求执行保存工作区操作。保存工作区选择主菜单 File > Save Workspace，浏览并选择 My projects 目录。然将工作区取名为 ST710-DEMO 输进 File name 输入框，按保存按钮退出。这时在 My projects 目录下又生成一个 ST710-DEMO.eww 文件。同时在 My projects 目录下还生成一个 settings 子目录，这个目录下存放保存窗口设置和断点设置等与当前操作有关信息的其他文件。

注意：保存操作完成后项目名后的 * 号已经消失。

4．给项目添加文件

保存工作区后的下一步就是往项目中添加文件。项目中的文件允许分组，用户可以根据项目的需要和自己的习惯来组织源文件。为举例说明，我们在下面的例子中建立一个 startup 文件组。

注意：往项目中添加文件时只需添加汇编语言和 C 语言的源程序，不需要添加头文件。但是用户必须在配置项目的编译器、连接器选件时指明包含头文件的路径和目录。关于项目配置选件的设定我们会在后面详细介绍。

新建项目 STR710-LED 是一个循环点亮 ATR710-EVAL 评估板上的 4 个 LED（LED11- LED14）的简单例子。我们用它来介绍 STR710 应用程序的开发流程。这个应用例子所需的所有源程序都在 STR710-EVAL 目录下。其中有两个 c 源程序：71x_it.c 和 main.c。另有两个 ARM 汇编语言程序：71x_init.s 和 71x_vect.s。后两个属启动程序，放在 01str71x\STR710-EVAL\startup 目录下。

添加文件的操作方法如下：

① 选择工作区窗口中的 ST710-LED–Debug 名，使其高亮。按鼠标右键在弹出菜单中选择 Add Group…或选择主菜单 Project > Add Group…执行添加文件组命令。在弹出菜单中将文件组也起名为 startup。

② 在工作区窗口中选择刚建立的文件组目录 startup，使其高亮。然后按鼠标右键在弹出菜单中选择 Add Files…或选择主菜单 Project > Add Files…执行添加文件命令。 在弹出添加文件窗口中浏览 01str71x\STR710-EVAL\startup 目录，从中选择 71x_init.s 和 71x_vect.s。这两个文件将被添加到工作区窗口的 startup 目录下。

③ 重新选择工作区窗口中的 ST710-LED – Debug 名，按鼠标右键在弹出菜单中选择 Add Files…或选择主菜单 Project > Add Files…执行添加文件命令。在弹出添加文件窗口中浏览 01str71x \STR710-EVAL 目录，从中选择 71x_it.c 和 main.c。

现在的工作区窗口中的显示如图 2-4 所示。

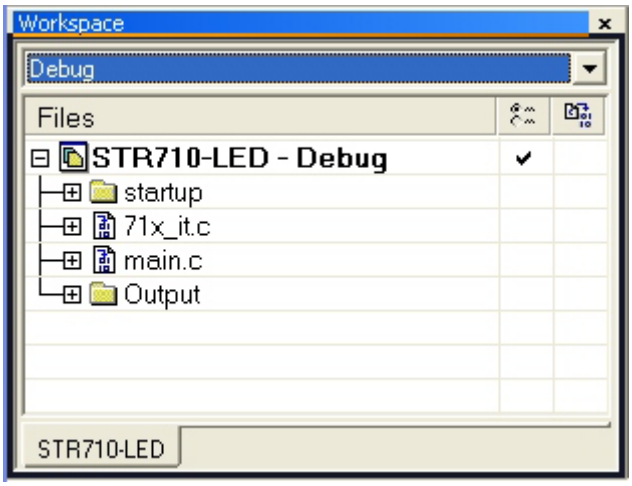

图 2-4. 添加文件后的工作区窗口

5．设置项目选件

生成新项目和添加文件后的下一步是为项目设置选件。设置项目选件是非常重要的一步，如果设置不当，编译、连接就会出错，就无法生成正确的代码文件。大家记得，在创建新项目时我们选择了 Empty project 模板，表示采用默认的项目选件设置。但是这些默认的设置还要根据具体项目的需要进行修改。

IAR EWARM 提供的项目选件内容繁多，初学者可能会感到不摸头脑、无从下手。实际上最关键的选件并不多，只要把它们设置正确了，其它的采用默认设置就不会出错。下面我们把这些关键选件设置逐条介绍：

① 通用选件设置

IAR EWARM 允许为工作区中的任何一级目录和文件单独设置选件，但是用户必须首先为整个项目设置通用的选件 General Option。

设置方法是：选中工作区中的项目名 STR710-LED – Debug，按鼠标右键在弹出菜单中选择 Options…或选择主菜单 Project > Options…。在弹出的 Options 窗口左边的目录（Category）中选择第一项 General Options。然后分别在：

- Target 页面中，Processor Variant 框中选择 Device。并点击右边的器件选择按钮选择芯片型号 ST STR710。同时 Processor mode 选择 ARM，Endian mode 选择 Little，Stack align 选择 4 byte。如图 2-5 所示；

  注意：Empty project 模板默认的处理器模式是 Thumb，即生成 16 位指令代码。但是在 STR710-LED 项目中，软件要求生成 32 指令代码（CODE32），所以应改选 ARM。
- Output 页面中，Output file 框下选择 Executable；
- Library Configuration 页面中，Library 条目下选择 Normal。
- Library Option 页面中采用默认选择

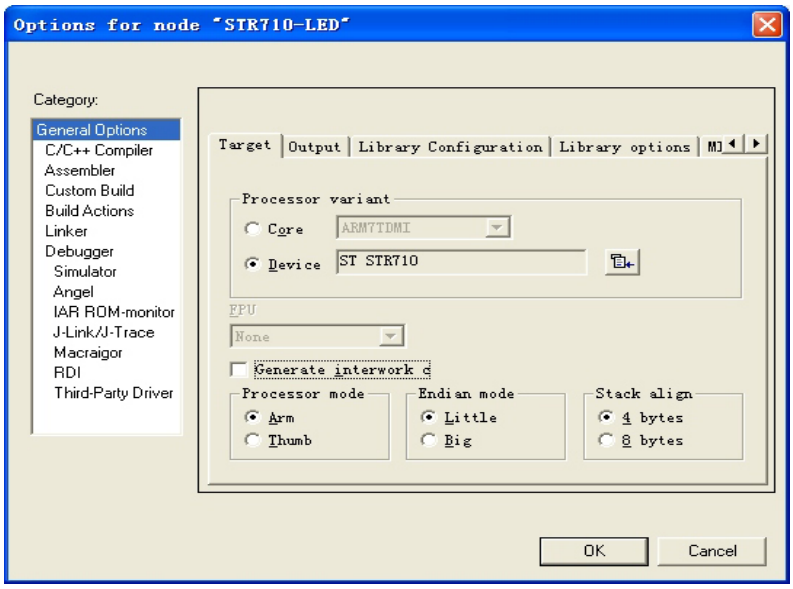

图 2-5. General Option 选件设置

② C/C++编译器选件设置

在 Options 窗口的目录 Category 中选择第二项 C/C++ Compiler。C/C++编译器的选件设置如下：

- Language 页面中，分别在三个选择框中选择 C，Allow IAR extensions，unsigned；
- Optimization 页面中，根据项目需要选择按 size 优化还是按 speed 优化，再选择优化等级。本例选择 Size，Low；
- Output 页面中，选择 Generate debug information；
- List 页面中，选择 Output list file，并选择 Assemble mnemonics 和 Diagnostics；
- Preprocessor 页面中，列有标准的 include 文件的目录。如果用户的 include 文件不在标准目录下时，必须在 Additional include directories 输入包含该项目 include 文件的目录。一个目录用一行描述，有多个目录时允许用多行。在本例中，我们选择 01str71x\STR710-EVAL\include，见图 2-6。

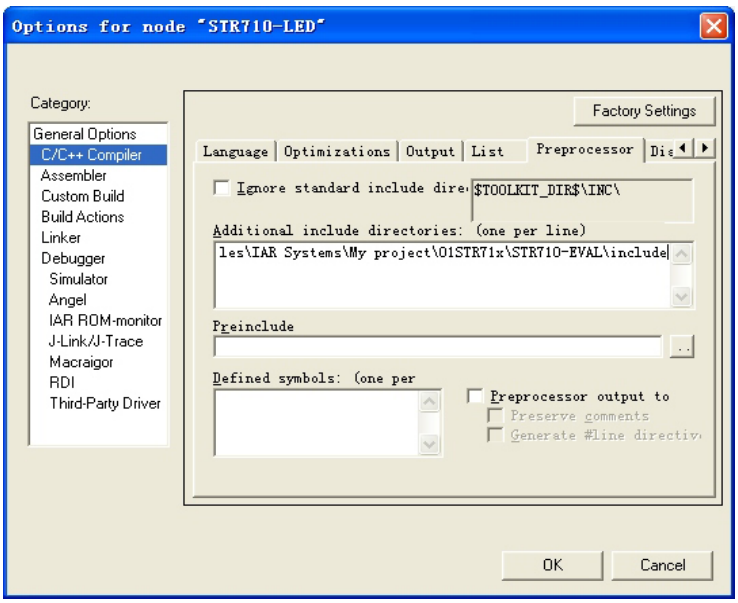

图 2-6. C/C++ 选件窗口

③ Assembler 选件设置

在 Options 窗口的目录 Category 中选择第三项 Assembler。汇编器的选件设置如下：

- Language 页面中，选择 User symbols are sensitive，并选择合适的宏汇编括号（Macro quote character）。本例选用<>；
- Output 页面中，选择 Generate debug information；
- List 页面中，选择 Output list file，并根据需要选择各子项；
- Preprocessor 页面中，列有标准的 include 文件的目录。如果用户的 include 文件不在标准目录下时，必须在 Additional include directories 输入包含该项目 include 文件的目录。一个目录用一行描述，有多个目录时允许用多行。
- Diagnostic 页面中，选择 Enable 和 All warning。

④ Linker 选件设置

在 Options 窗口的目录 Category 中选择第六项 Linker。连接器的选件中最重要的设置是以下几个：

- Output 页面，选择合适的输出格式十分重要。如果需要将输出文件送给一个调试器进行调试，则要求输出格式带有调试信息。本例选择 Debug information for C-SPY。以及此选项下的选件 With runtime control modules，With I/O emulation modules 和 Allow C-SPY-specific extra output file。见图 2-7。

  注意：I/O emulation modules 指示连接器将 stdin 和 stdout 指向 C-SPY 的 I/O 窗口的低级例程。在没有真实硬件采用模拟仿真时应选择此项选件。
- Extra Output 页面选择 Generate extra output file。输出格式选择 simple-code 和 None；

- Diagnostics 页面选择 Generate errors；
- List 页面，选择 Generate linker listing 和 Segment map（见图 2-8），允许生成存储器分配 MAP 文件。

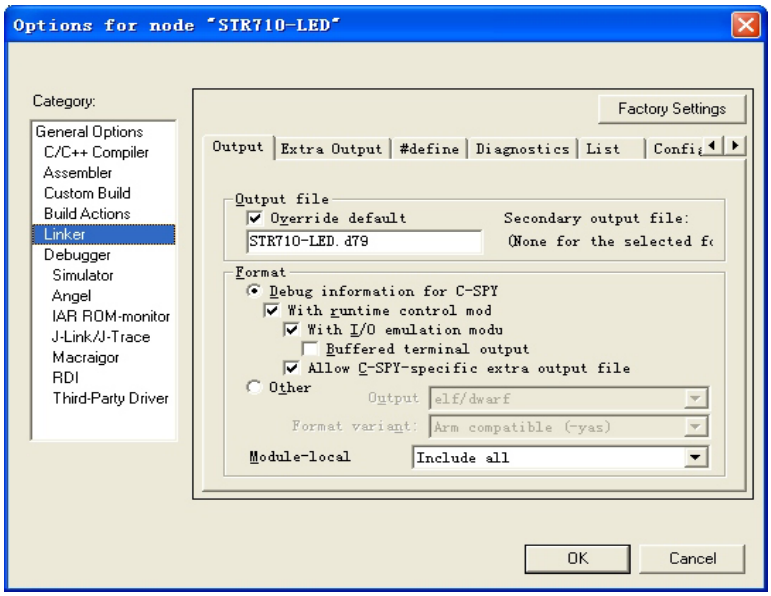

图 2-7. 连接器 Output 选件设置

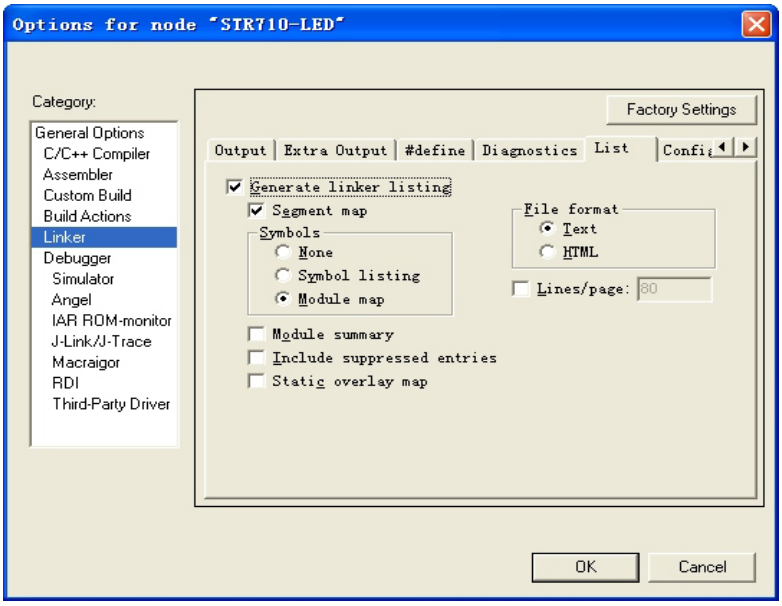

图 2-8. 连接器 List 选件设置

- Config 页面，主要是定义连接器命令文件（Linker Command File）。这是连接器选件中最重要同时也是最复杂的设置。连接器命令文件中包含连接器的各项命令行参数，主要用于控制程序各种代码段和数据段在存储器中如何分布。用户一定要吃透和掌握如何生成正确的连接器命令文件。为了帮助初学者理解，我们增加了下面一段介绍：

附注：段的存储器分配方法

用户在开发以 ARM7 或 ARM9 为内核的嵌入式应用系统时会采用不同半导体厂家的产品，每种芯片内部 SRAM 和 FLASH 的大小和地址分布都不同，另外用户目标系统配置的外部存储器也不同，用户应用软件要求的存储器分配也不相同。以上所有的不同最后落实到在运行时不同的代码段和数据段的存储器地址分配方案。而这种**运行时存储器分配**必需在连接器命令文件中说明，并由连接器 IAR XLINK 生成。

经 XLINK 连接生成的代码文件下载到目标板时的地址，由 FlashLoader 执行，后面将介绍。

IAR EWAR 提供默认的连接器命令文件，它在 IAR EWAR 安装目录的 ARM\config 目录下，名字叫 lnkarm.xcl。但是默认的连接器命令文件 lnkarm.xcl 不能完全适用特定的目标系统，必须加以修改。下面介绍几个连接器命令选件的功能和用法：

命令选件	功 能	举 例	
-D	说明存储器块的起始地址和结束地址	-DROMSTART=0x40000000 -DROMEND=0x4003FFFF	说明 STR710 内部 Flash 的地址分布
		-DROMSTART=0x2000000 -DROMEND=0x2000FFFF	说明 STR710 内部 RAM 的地址分布
	说明段（堆栈）的空间大小	-D_CSTACK_SIZE=100	CSTACK 是在启动程序 71x_init.s 中定义的段，此命令给它分配 100 字节
-Z	按段出现的顺序进行段定位	-Z(CODE)ICODE=0x40000000-0x4003FFFF -Z(CODE)CODE_ID= 0x40000000-0x4003FFFF	把代码段 ICODE 和 CODE_ID 顺序放置在 STR710 内部 Flash 地址 0x4000000 - 0x4003FFFF 内。
	按段出现的顺序进行段定位，同时分配空间	-Z(DATA)CSTACK+2000=20000000-2000FFFF -Z(DATA)IRQ_STACK+100=20000000-2000FFFF	分配堆栈 CSTACK 2000 个字节，分配)IRQ_STACK 100 个字节，顺序放置在 STR710 内部 RAM 地址 0x20000000 - 0x2000FFFF 内。
-P	把离散的存储器空间分配给段	-P(DATA)MYDATA=10000-101FFF，110000-111FFF	用-P 命令后，就不能用汇编命令 SFB 和 SFE 获得该段的起始地址和结束地址。
-carm	采用的 CPU	-carm	

表 2-1. 主要的连接命令选件

用户可以使用上述命令选件进行段的存储器地址分配。

在连接器命令文件中需要对两种段进行说明。一是 IAR C/C++编译器定义的段名，二是启动程序（汇编语言程序）中定义的堆栈。下面分别叙述：

ⅰ）IAR C/C++编译器定义了许多不同用途的段名，其种类繁多，简单介绍请见表 2-2。用户应该查阅《EWARM C Compiler.pdf》中的 Segment reference 章节详细了解。在默认的 lnkarm.xcl 文件中，几乎对所有 C/C++编译器定义的段都已经有默认的说明，用户不必去修改。但是需要在默认的 lnkarm.xcl 文件的头部（ -carm 命令选件的前面）增加以下语句：

```
-DROMSTART=0x????????
-DROMEND=0x????????
-DRAMSTART=0x????????
-DRAMEND=0x????????
-DVECSTART=ROMSTART
```

以上语句是定义片内 ROM 和 RAM 的地址范围，数值???????用 16 进制表示。XLINK 对所有段的地址分配都被限定在前 4 句定义的范围内。第 5 句指定 INTVEC 段的起始地址在 ROM 起始地址，INTVEC 用于保存复位和异常向量。用户应注意 INTVEC 段必须放在 ROM（当指定程序在 ROM 中运行时）或 RAM（当指定程序在 RAM 中运行时）的起始地址 0-3F 处。

因为在 STR710-LED 项目中， INTVEC 至少需要 428 个字节，低于此数值连接处理将出错。所以将 lnkarm.xcl 文件中原来的-Z(CODE)INTVEC=0-3F 修改成：

```
-Z(CODE)INTVEC=VECSTART:+0x428
```

ⅱ）启动程序（71x_init.s）中定义的所有堆栈和堆（Heap）必须在连接器命令文件中加以说明，缺一不可，否则就会报错。根据 STR710-LED 项目启动程序选择以下命令：

```
-Z(DATA)CSTACK+2000=RAMSTART-RAMEND
-Z(DATA)IRQ_STACK+100=RAMSTART-RAMEND
-Z(DATA)FIQ_STACK+100=RAMSTART-RAMEND
-Z(DATA)SVC_STACK+100=RAMSTART-RAMEND
-Z(DATA)ABT_STACK+100=RAMSTART-RAMEND
-Z(DATA)UND_STACK+100=RAMSTART-RAMEND
-Z(DATA)HEAP+8000=RAMSTART-RAMEND
```

以上命令为各堆栈和堆分配存储空间，同时把它们顺序地放置在片内 RAM 空间内。用户可以根据应用的需要为堆栈分配空间。经过上述修改后，保存文件，并在 Linker 选件 config 页面中指定新文件的路径。此连接命令文件就可以使用了。修改后的 xcl 文件指定程序在片内 ROM 中运行。

如果要求连接后的程序在 RAM 中运行，只需修改前面 4 条命令就可以了。

```
-DRAMSTART=0x20000000
-DRAMEND=0x2000FFFF
-DROMSTART=RAMSTART
-DROMEND=RAMEND
```

上面的命令先规定片内 RAM 地址范围。然后将片内 ROM 地址范围指向与 RAM 相同的空间。编译连接后的文件可以通过 FlashLoader 直接下载到片内 RAM，并将 STR710 的 BOOT0 和 BOOT1 脚设置成从 RAM 引导，Reset 后就可以在 RAM 运行了。

下面是两个完整的连接命令文件例子。请注意其中-Z 命令的用法的差异，但连接处理结果是一样的。

连接命令文件例 1（在 ROM 中运行）：

```
// Code memory in flash
-DROMSTART=0x40000000                          定义片内 ROM 起始地址
-DROMEND=0x4003FFFF                            定义片内 ROM 结束地址
-DVECSTART=ROMSTART                            定义 INTVEC 段起始地址
// Data memory
-DRAMSTART=0x20000000                          定义片内 RAM 起始地址
-DRAMEND=0x2000FFFF                            定义片内 RAM 结束地址
-carm
-Z(CODE)INTVEC=VECSTART:+0x428                 把 INTVEC 段放在 ROM 起始地址，保留 428 字节
-Z(CODE)ICODE,DIFUNCT=ROMSTART-ROMEND          把段 ICODE 和 DIFUNCT 顺序放在片内 ROM 空间
-Z(CODE)SWITAB=ROMSTART-ROMEND                 把段 SWITAB 顺序放在片内 ROM 空间
-Z(CODE)CODE=ROMSTART-ROMEND                   把段 CODE 顺序放在片内 ROM 空间
-Z(CONST)CODE_ID=ROMSTART-ROMEND               把段 CODE_ID 顺序放在片内 ROM 空间
-Z(CONST)INITTAB,DATA_ID,DATA_C=ROMSTART-ROMEND
                                               把段 INITTAB，DATA_ID 和 DATA_C 放在片内 ROM 空间
-Z(CONST)CHECKSUM=ROMSTART-ROMEND              把段 CHECKSUM 顺序放在片内 ROM 空间
-Z(DATA)DATA_I,DATA_Z,DATA_N=RAMSTART-RAMEND
                                               把段 DATA_I，DATA_Z 和 DATA_N 顺序放在片内 RAM 空间
-Z(DATA)CODE_I=RAMSTART-RAMEND                 把段 CODE_I 顺序放在片内 RAM 空间
-QCODE_I=CODE_ID                               把 CODE_ID 中的内容拷贝到 CODE_I
-D_CSTACK_SIZE=1000                            给堆栈 CSTACK 分配 1000 字节
-D_SVC_STACK_SIZE=100                          给堆栈 SVC_STACK 分配 100 字节
-D_FIQ_STACK_SIZE=100                          给堆栈 FIQ_STACK 分配 100 字节
-D_ABT_STACK_SIZE=100                          给堆栈 ABT_STACK 分配 100 字节
-D_UND_STACK_SIZE=100                          给堆栈 UND_STACK 分配 100 字节
-D_IRQ_STACK_SIZE=400                          给堆栈 IRQ_STACK 分配 400 字节
-D_HEAP_SIZE=400                               给堆 HEAP 分配 400 字节
-Z(DATA)CSTACK+_CSTACK_SIZE=RAMSTART-RAMEND    把上述堆栈顺序放在片内 RAM 中
-Z(DATA)SVC_STACK+_SVC_STACK_SIZE=RAMSTART-RAMEND
-Z(DATA)ABT_STACK+_ABT_STACK_SIZE=RAMSTART-RAMEND
-Z(DATA)UND_STACK+_UND_STACK_SIZE=RAMSTART-RAMEND
-Z(DATA)FIQ_STACK+_FIQ_STACK_SIZE=RAMSTART-RAMEND
-Z(DATA)IRQ_STACK+_IRQ_STACK_SIZE=RAMSTART-RAMEND
-Z(DATA)HEAP+_HEAP_SIZE=RAMSTART-RAMEND
```

连接命令文件例 2： （连接结果和第一个例子一样）

```
-carm
-Z(CODE)INTVEC=40000000-40000428
-Z(CODE)ICODE,DIFUNCT=40000000-4003FFFF
-Z(CODE)SWITAB=40000000-4003FFFF
-Z(CODE)CODE=40000000-4003FFFF
-Z(CONST)CODE_ID=40000000-4003FFFF
-Z(CONST)INITTAB,DATA_ID,DATA_C=40000000-4003FFFF
-Z(CONST)CHECKSUM=40000000-4003FFFF
-Z(DATA)DATA_I,DATA_Z,DATA_N=20000000-2000FFFF
-Z(DATA)CODE_I=20000000-2000FFFF
-QCODE_I=CODE_ID
-Z(DATA)CSTACK+1000=20000000-2000FFFF
-Z(DATA)SVC_STACK+100=20000000-2000FFFF
-Z(DATA)ABT_STACK+100=20000000-2000FFFF
-Z(DATA)UND_STACK+100=20000000-2000FFFF
-Z(DATA)FIQ_STACK+100=20000000-2000FFFF
-Z(DATA)IRQ_STACK+400=20000000-2000FFFF
-Z(DATA)HEAP+400=20000000-2000FFFF
```

下表对上面出现的段名做一简单介绍：

段名	功 能 说 明	段类型	属性
INTVEC	保存 Reset 和异常向量，含跳转到 cstartup 或中断服务程序的指令。必须在 0x00-0x3F 地址范围内。	CODE	只读
ICODE	保存启动代码，位于能被 INTVEC 中的转移指令达到的前 32MB 字节内。	CODE	只读
DIFUNCT	包含指向 C++建构模块的指针。该模块在调用 main 之前由 cstartup 执行。	CODE	只读
SWITAB	保存软件中断向量表	CODE	只读
CODE	保存程序代码，可以在 ROM 中执行。	CODE	只读
CODE_I	上电后保存声明为_ramfunc 的程序代码，掉电后丢失。初始化时从 CODE_ID（ROM）拷贝到 CODE_I 中，然后在 RAM 中执行。	DATA	读/写
CODE_ID	永久性保存声明为_ramfunc 的程序代码。这些代码初始化时从 CODE_ID 拷贝到 CODE_I 中（RAM），然后在 RAM 中执行。	CONST	只读
INITTAB	保存需要在启动时初始化的段地址和段长度表格。	CONST	只读
DATA_ID	保存 DATA_I 段中变量的初值，这些值在系统初始化时由 cstartup 从 DATA_ID 拷贝到 DATA_I。	CONST	只读
DATA_C	保存常数数据，包括文字字符串	CONST	只读
CHECKSUM	保存段的校验和	CONST	只读
DATA_I	保存声明为非零初值的静态全局变量，其初值开始在 DATA_ID 中，系统初始化时由 cstartup 从 DATA_ID 拷贝到 DATA_I。	DATA	读/写
DATA_Z	保存声明初值为零或没有声明初值的静态全局变量	DATA	读/写
DATA_N	保存非易失存储器中，用_no_init 关键字声明的静态全局变量	DATA	读/写
CSTACK	C/C++内部数据堆栈（用户模式）	DATA	读/写
SVC_STACK	操作系统堆栈（管理模式，用于具有特权的操作系统任务）	DATA	读/写
ABT_STACK	数据访问中止模式堆栈（用于虚拟存储器及存储器保护）	DATA	读/写
UND_STACK	未定义指令中止模式堆栈（支持硬件协处理器软件仿真）	DATA	读/写
FIQ_STACK	快中断模式堆栈（用于高速数据传输或通道处理）	DATA	读/写
IRQ_STACK	中断模式堆栈（用于通用中断处理）	DATA	读/写
HEAP	堆（Heap）是 C\C++中为 malloc 和 free 或 new 和 delete 保留的动态分配存储空间。	DATA	读/写

表 2-2. 段名及其功能

表 2-2 中属性中的 CODE 是在 ROM 中的代码。CONST 是在 ROM 中的数据，DATA 是在 RAM 中的数据。表 2-2 中列出的所有段除了 INTVEC 必须放在运行存储器（不管是 ROM 还是 RAM）的 0-3F 地址空间和 ICODE 必须放在前 32MB 空间外，其他的段都可以放在符合其属性存储器空间的任何地址。