



飞凌嵌入式
FORLINX EMBEDDED



OK6410

RVDS 调试手册

Devoted to create the best embedded products

www.witech.com.cn

www.forlinx.com

www.helloarm.com

目录

第一章	OK6410 简介.....	6
第二章	安装 RVDS2.2 开发环境.....	9
2-1	安装 RVDS2.2 到 Windows XP 中.....	9
第三章	JLINK 驱动的安装.....	16
第四章	JLINK 的连接方法.....	21
第五章	RVDS2.2 介绍.....	22
5-1	RVDS 开发工具介绍.....	22
5-2	集成开发环境（IDE）.....	22
第六章	RVDS2.2 的使用方法以及调试方法、调试的原理.....	23
6-1	CodeWarrior for RVDS.....	23
6-1-1	打开 CodeWarrior.....	23
6-1-2	新建 OK6410 裸机工程的方法.....	24
6-1-3	为工程添加源码文件.....	29
6-1-4	为工程进行必要的设置.....	31
6-1-5	工程编译方法.....	35
6-2	设置 AXD1.3.1.....	37
6-2-1	打开 AXD1.3.1.....	37
6-2-2	针对 JLINK 设置 AXD.....	37
6-2-3	更新 Jlink 固件.....	39
6-2-4	检测目标 CPU.....	40
6-3	仿真一下，小试牛刀.....	42
第七章	Jlink.....	45
7-1	全速运行.....	45
7-2	断点调试.....	45
7-3	单步调试.....	46
第八章	S3C6410 GPIO.....	47
第九章	利用 GPIO 控制 OK6410 的 LED.....	48
9-1	实验目的.....	48
9-2	实验设备.....	48
9-3	实验内容.....	48
9-4	实验原理.....	48
9-5	实验电路.....	49
9-6	实验程序.....	49
9-7	实验步骤.....	50
9-8	实验结果.....	51
第十章	GPIO 控制 OK6410 的蜂鸣器.....	52
10-1	实验目的.....	52
10-2	实验设备.....	52
10-3	实验内容.....	52
10-4	实验原理.....	52

10-5	实验电路.....	53
10-6	实验程序.....	53
10-7	实验步骤.....	54
10-8	实验结果.....	55
第十一章	GPIO 检测 OK6410 的按键操作.....	56
11-1	实验目的.....	56
11-2	实验设备.....	56
11-3	实验内容.....	56
11-4	实验原理.....	56
11-5	实验电路.....	57
11-6	实验程序.....	58
11-7	实验步骤.....	60
11-8	实验结果.....	61
第十二章	利用定时器制作精确延时来控制 OK6410 的 LED.....	62
12-1	实验目的.....	62
12-2	实验设备.....	62
12-3	实验内容.....	62
12-4	实验原理.....	62
12-5	实验电路.....	63
12-6	实验程序.....	63
12-7	实验步骤.....	66
12-8	实验结果.....	66
第十三章	利用定时器精确控制 OK6410 的蜂鸣器频率.....	67
13-1	实验目的.....	67
13-2	实验设备.....	67
13-3	实验内容.....	67
13-4	实验原理.....	67
13-5	实验电路.....	68
13-6	实验程序.....	68
13-7	实验步骤.....	70
13-8	实验结果.....	70
第十四章	总结.....	71

1、资料唯一更新方式：www.forlinx.com 下载专区。

在 www.forlinx.com 注册新用户后方可下载。

在注册时必须提供开发板序列号（序列号在开发板背面的白色纸条上）。正确注册后，24 小时内飞凌会为您开通下载权限，注册后请耐心等待。如果提供的信息错误，将不能通过验证。

2、在您的使用过程中如果遇到相关技术问题，欢迎访问飞凌官方论坛寻求答案，或者发帖求援。论坛地址：
<http://bbs.witech.com.cn/>

3、飞凌技术服务热线：0312-3119192

4、本手册版权归属飞凌嵌入式有限公司所有，并保留一切权利。任何单位及个人不得擅自摘录本手册部分或全部内容。

5、本手册重点讲述 RVDS 调试 OK6410。如果您对开发板的硬件不熟悉，或者对开发板接口不熟悉，强烈建议把飞凌 6410 硬件手册通读一遍。如果您对 S3C6410 本身不熟悉，将强烈建议结合 S3C6410 Datasheet，边学边读。

第一章 OK6410 简介

OK6410 开发板立足客户全面、稳定的应用需求，是专为企业级客户设计的高效、安全型产品开发平台。实用性强是它的最大优势，OK6410 可以对核心板、底板、内存、FLASH、扩展卡等部件进行灵活特配。凭借高品质、低价格、易扩展三大顶尖特性，将科技转化为企业竞争力，助力客户实现可持续发展。

飞凌长达 10 年关注商用客户需求的丰富经验，覆盖嵌入式各个领域的研发体系，值得客户信赖。

OK6410 历经多道质量管控环节层层把关及环境、电磁、安规等实验室严格测试，确保产品在客户应用中的高可靠表现。

严格的部件优选制度，使整板具有稳定的品质保障。开发板通过不同温（湿）度、不同电压、多任务满负荷长时间运行测试，平均无故障运行时间（MTBF）达到 10000 小时，代表了世界一流的稳定性和兼容性。

我们的工程师一如既往地追求至高标准，目标是让您一眼就能看出这款开发板完全满足嵌入式学习开发的所有期望：具有出色的稳定性、精确的操控性和便捷的扩展性。这是如何实现的呢？是通过我们对飞凌基本原则的一贯坚持，即：丰富的功能设计与实用性相结合。还因为我们始终坚持最初的开发板设计理念：取消多余的部分，保留最重要的元素。

OK6410 以其明确的功能设计、详实的技术文档以及专业的售后服务体现了这种理念。其中最突出的例子是，核心板+底板的结构，绝对让人印象深刻。那么，这一切都只是为了造型吗？恰恰相反，增加这一设计纯粹是为了提高性能，这种改进的结果是增强了开发板的灵活性、操控性和扩展性能。当您独立开发时，这样的设计能够完全满足各种扩展的需要。

PCB 布线间距越大，抗干扰能力越强。对于我们的工程师而言，这种概念显得过于简单。在拥有 10 多年工业级板卡设计经验的飞凌，重要的是性能尺寸比：减小核心板尺寸可以提高灵活性，提高抗干扰能力则意味着出色的稳定性。因此，飞凌在增强 CPU 主频的同时，将核心板大小控制在 5cm*6cm。使得开发板在 667MHz 主频上仍可稳定工作。

CPU，内存和 FLASH，这些都是程序运行的主要组成。从这个角度来看，小容量的 NORFLASH 似乎有些多余。因此，工程师毅然取消了 NORFLASH，这降低了核心板价格，在后续开发中，这将为您节省大量的成本。

OK 系列标配 100M 网口 (DM9000AE)、具有带连接和传输指示灯。这种力求简洁稳定的风格，明确体现了对经典设计的传承。

在数据传输的实用性和快捷性方面，我们只认同与实际使用密切相关的部件，例如 USB。飞凌 OK6410 具有 HOST 和 Slave 两种接口模式。1 个 USB HOST 插口，支持 USB1.1 协议，可插鼠标、U 盘等；1 个 USB Slave 接口，支持 USB2.0 协议，使用 mini-USB 插座，

可与 PC 连接。这不仅显著提高了传输能力，而且 CPU 在总线的数据处理上也不会受到丝毫影响。

对于开发板上的串口而言，探索新的设计方向同时也意味着回顾过去。OK6410 板上配备了串口扩展板，为需要多串口的客户提供了接口保证。

1 个无线网卡。WIFI 以其覆盖范围广、传输速度快、使用便捷成为备受关注的焦点。OK6410 自带 wifi 接口，为数据传输和信息交互提供了更大空间。

工程师采用了 18B20 温度采集模块和红外接收器，使得 OK6410 更加与众不同。独有的硬件接口创造了更大的使用空间。配置这两个模块后，可以让您轻松掌握总线与 CPU 之间通信、复杂信号编解码以及典型的驱动程序编写的方法。其设计恰到好处，堪称完美。尽管 OK6410 的外观简单，但其卓越性能却发挥得淋漓尽致。

此外，我们还决定采用 40pin 软排线触摸屏和 8 个底板通孔的设计，即可固定 3.5\4.3\5.6 寸不同型号液晶屏。这一基本理念可以将开发板搭配多种液晶屏使用。其中最引人注目的边缘四个通孔设计，令开发板与 5.6 寸液晶屏 1:1 完美匹配。

性能出色的 SD 卡座，增强了数据的存储量并可以实现 SD Memory 功能和 SDIO 功能。结合大容量 SD 卡及接口后置结构，为产品开发带来了更大的灵活性。

音频操作在几毫秒内就能轻松完成。3 个 3.5mm 标准立体声音频插座。其中包括 1 个音频输出插座，可与耳机连接；1 个话筒输入插座；1 个线路输入插座流畅的传输，能够让使用者享受到更快的声音录入与更佳的音响效果。

OK6410 拥有 1 路复合视频广播信号 CVBS 输出接口，可直接将开发板与电视相连，播放图片、声音、视频等多种媒体文件。

1 个 CMOS 摄像头接口，支持 ITU-RBT601/656 8 位模式，使用 10*2 插针连接器，提供一种交互式非线性连接。可支持 130 万、300 万、500 万等多种分辨率摄像头高速采集。

8 位拨码开关，使开发板边缘更加紧凑、充实。这个美观的设计细节能够带来实际的效果：轻松设置系统启动方式，可以自由选择 NandFLASH 启动或 SD 卡启动。

提供强大的内部实时钟管理方案，并特别备有锂电池插座，完全解除断电后系统时间丢失的后顾之忧。

1 个使用 10*2 插针连接器的 JTAG 接口，可用于芯片内部测试和在线编程。

4 个 LED 和 1 个蜂鸣器可以作为开发中测试的一部分，对于初学者朋友而言也是学习的最佳起点。

OK6410 提供丰富的外围扩展接口，包括：3 个 10×2 插针扩展口（1 路 GND、1 路 DA、8 路 AD、10 路 IO、1 路 SPI）。一个用来扩展 8×8 矩阵键盘，另一个可连接 3 个 TTL 电平的串口和 6 路 IO 接口。

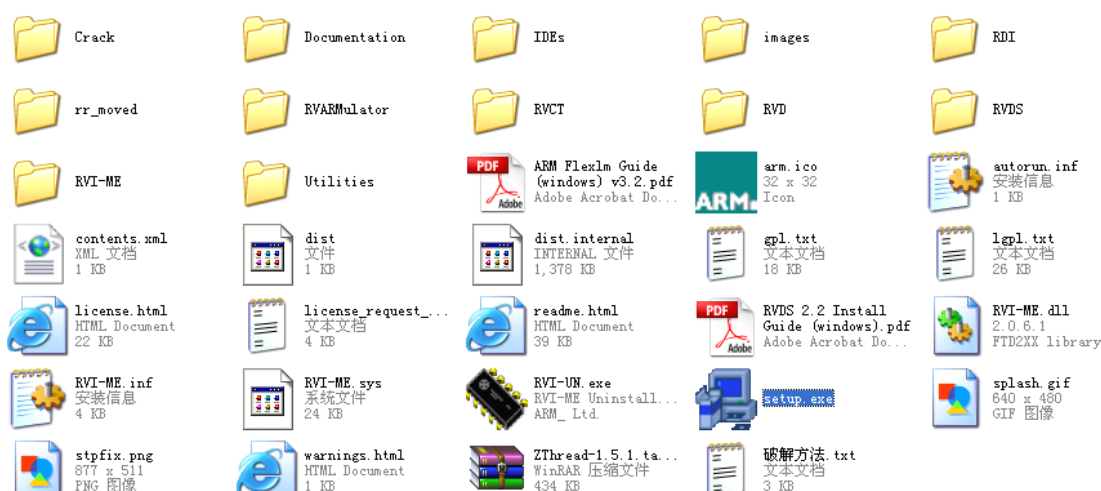
OK6410 的板色秉承飞凌长久以来的传统。海蓝色的纯粹，能够让人想起过去与飞凌并肩前行的时光。可选的翡翠色如同板身正面的飞凌标志一样，令人回想起电子行业的辉煌历史，这也是是 20 世纪 80 年代 PCB 的鲜明写照。

总之，OK6410 是一款能够唤起使用者的产品。它融合了历代开发板的精髓，由飞凌专为当今追求经典和超越的人们量身打造。

第二章 安装 RVDS2.2 开发环境

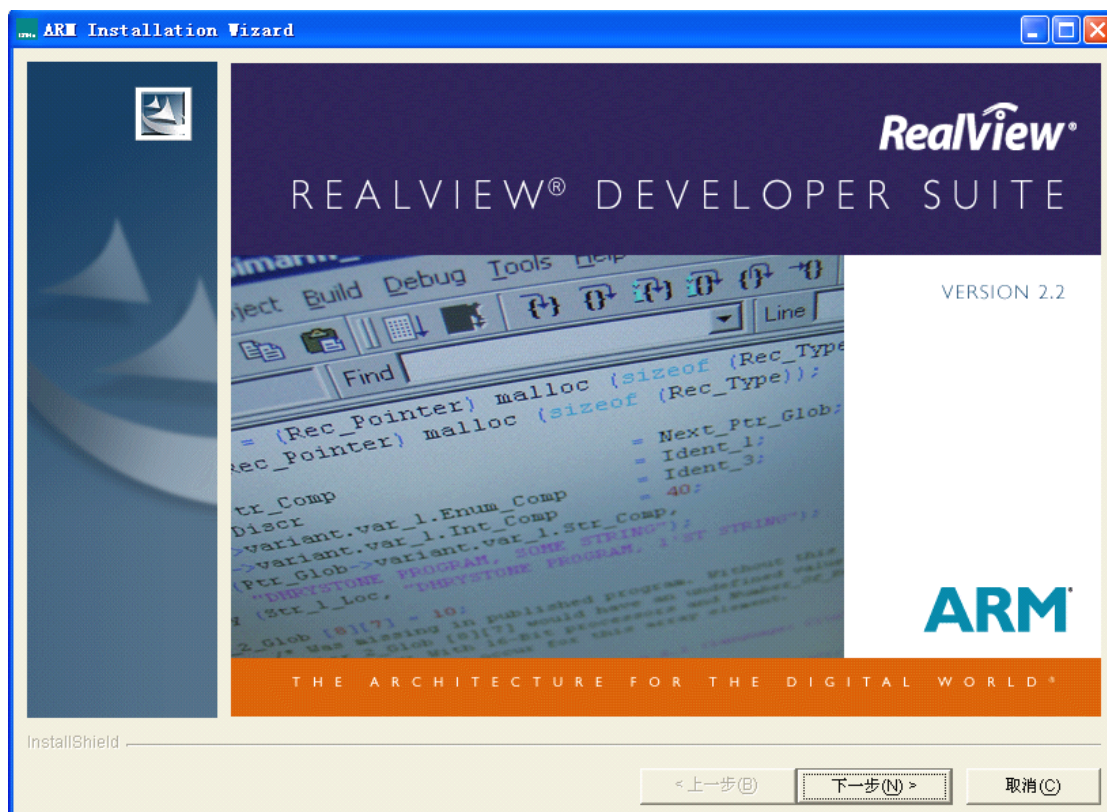
2-1 安装 RVDS2.2 到 Windows XP 中

飞凌 6410 基础光盘中提供了一个 RVDS2.2 的安装包，路径是：基础光盘\实用工具\rvds2.2 安装程序.rar。解压缩该软件包得到如下图安装文件：

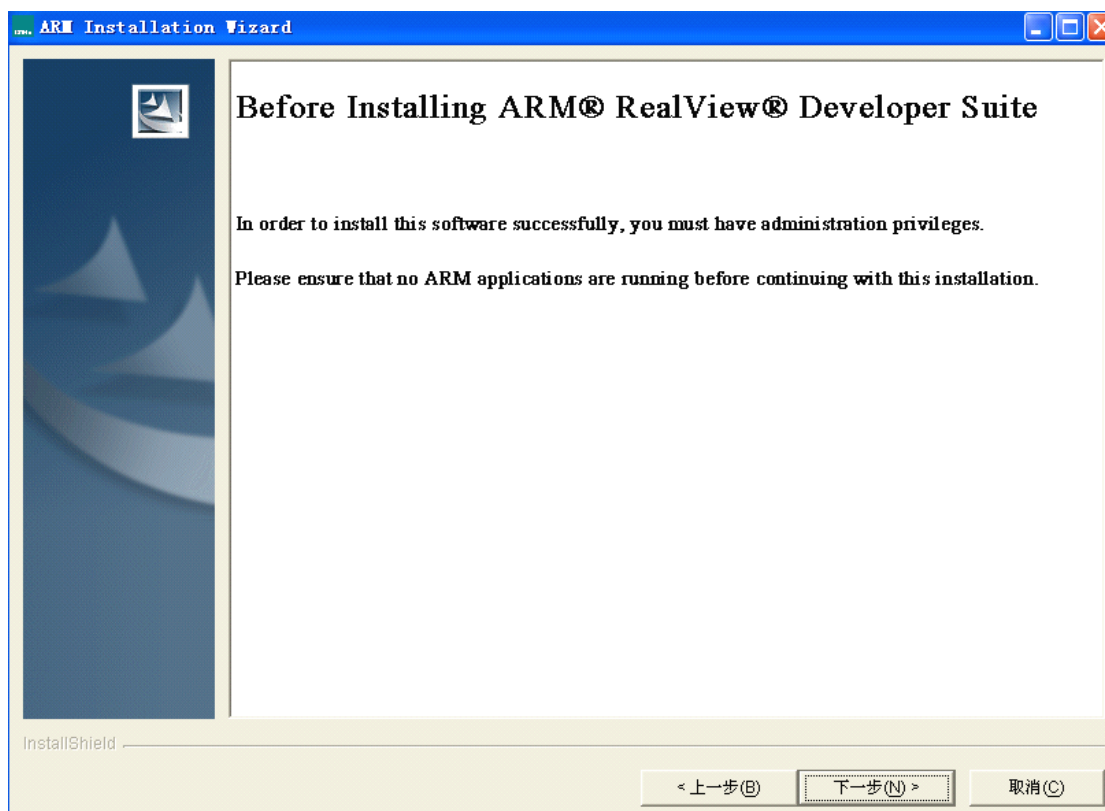


双击打开 setup.exe，开始安装 RVDS2.2。

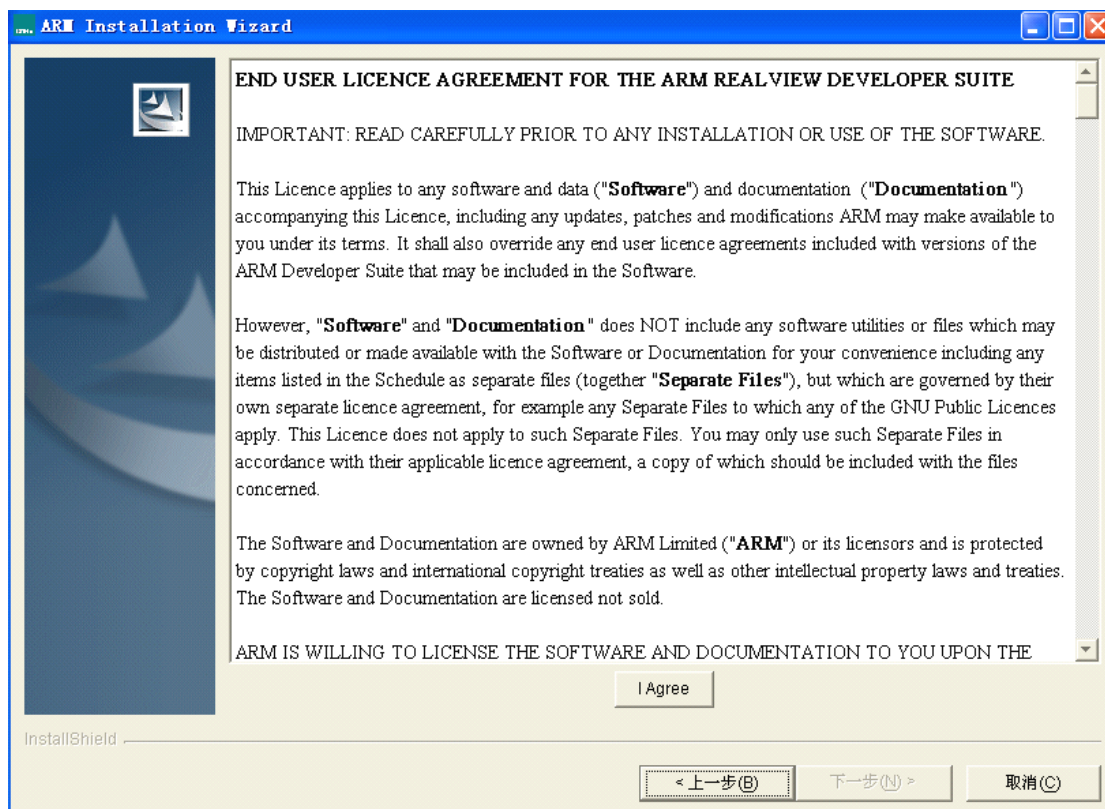
步骤 1. 点击”下一步”。



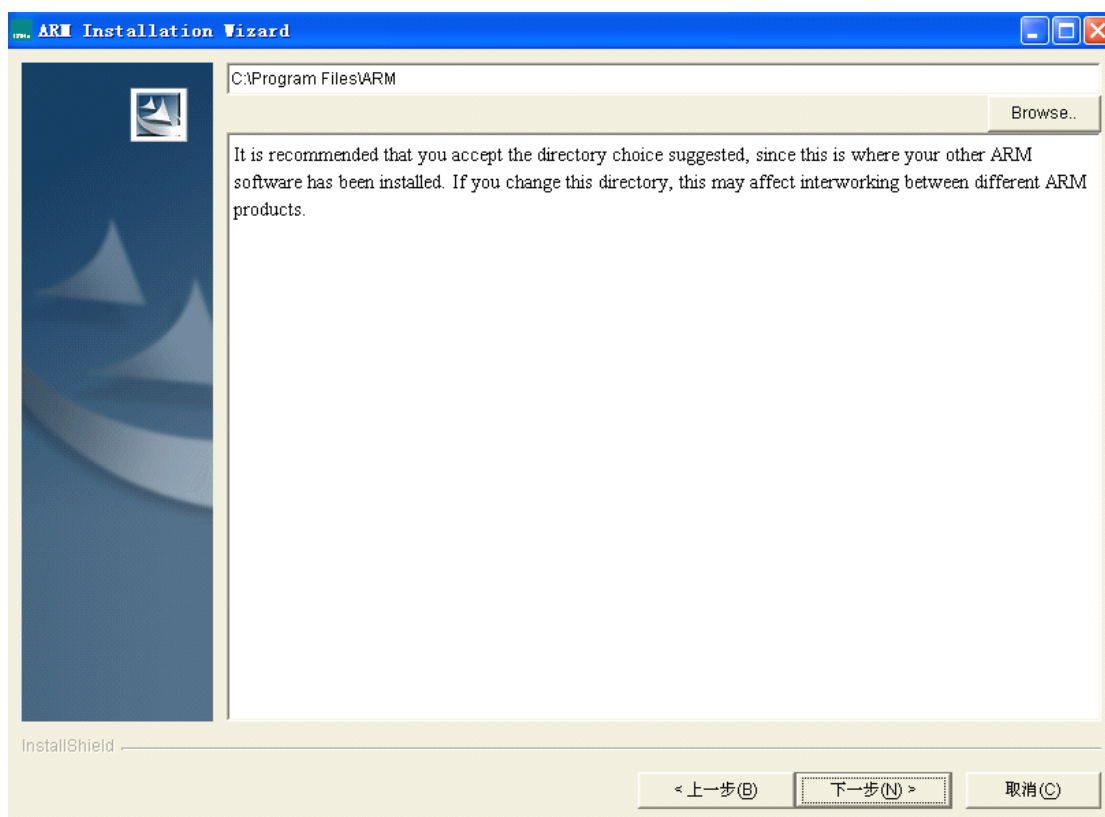
点击”下一步”。



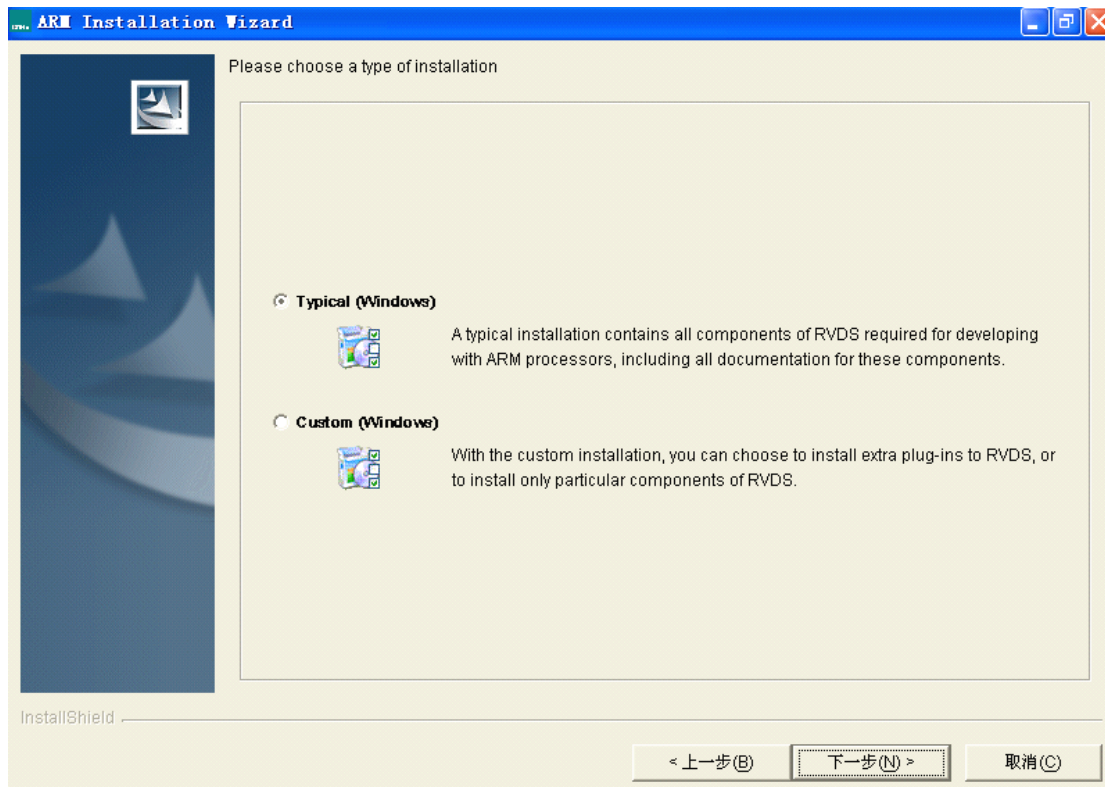
点击”agree”，然后点击下一步。



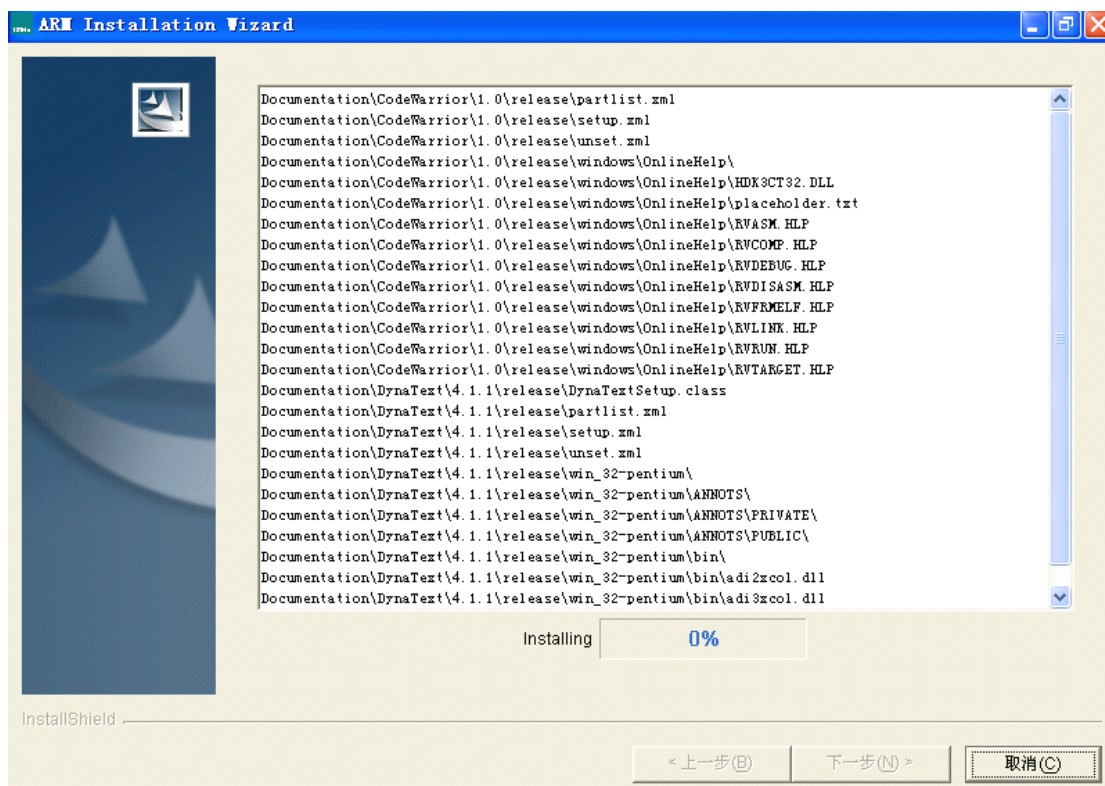
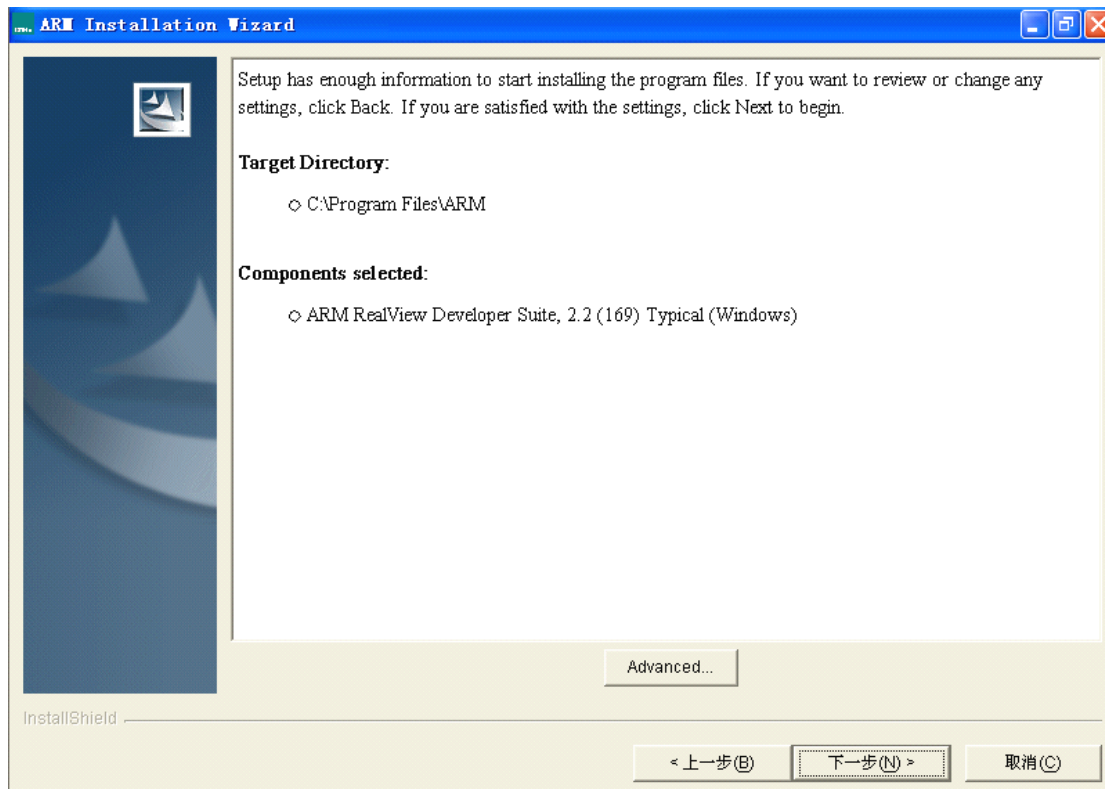
点击”Browse”选择安装路径，推荐使用默认路径。点击”下一步”。



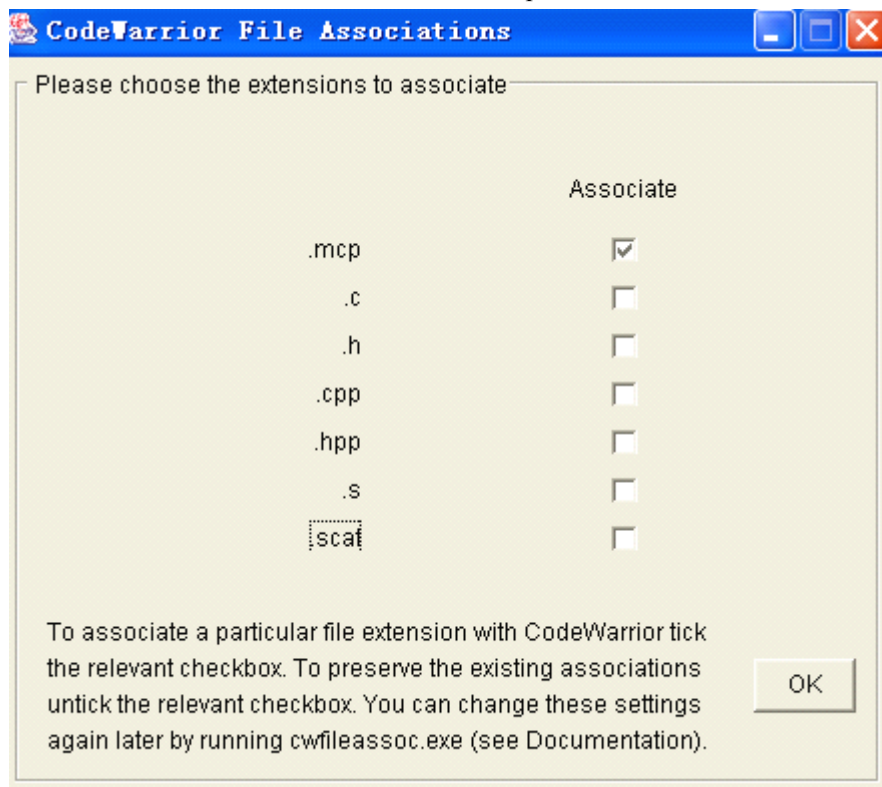
点击”下一步”，使用典型安装。



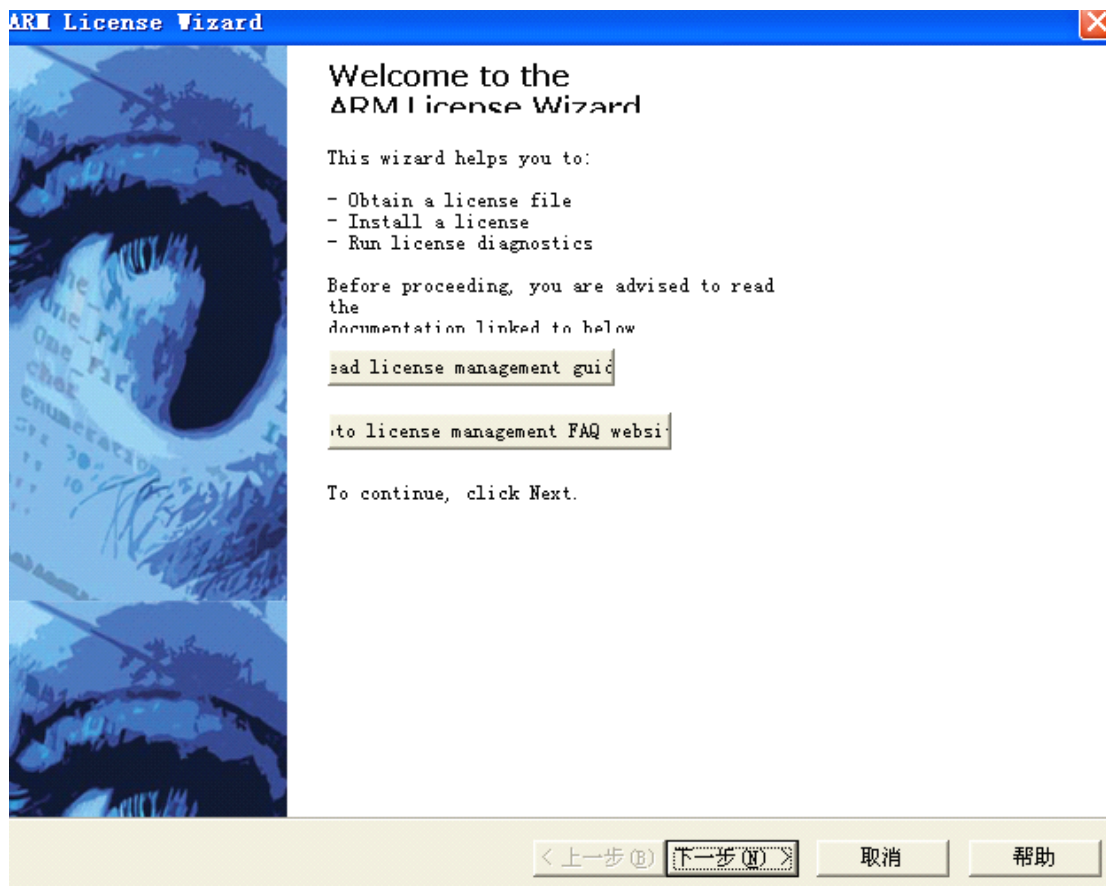
点击”下一步”



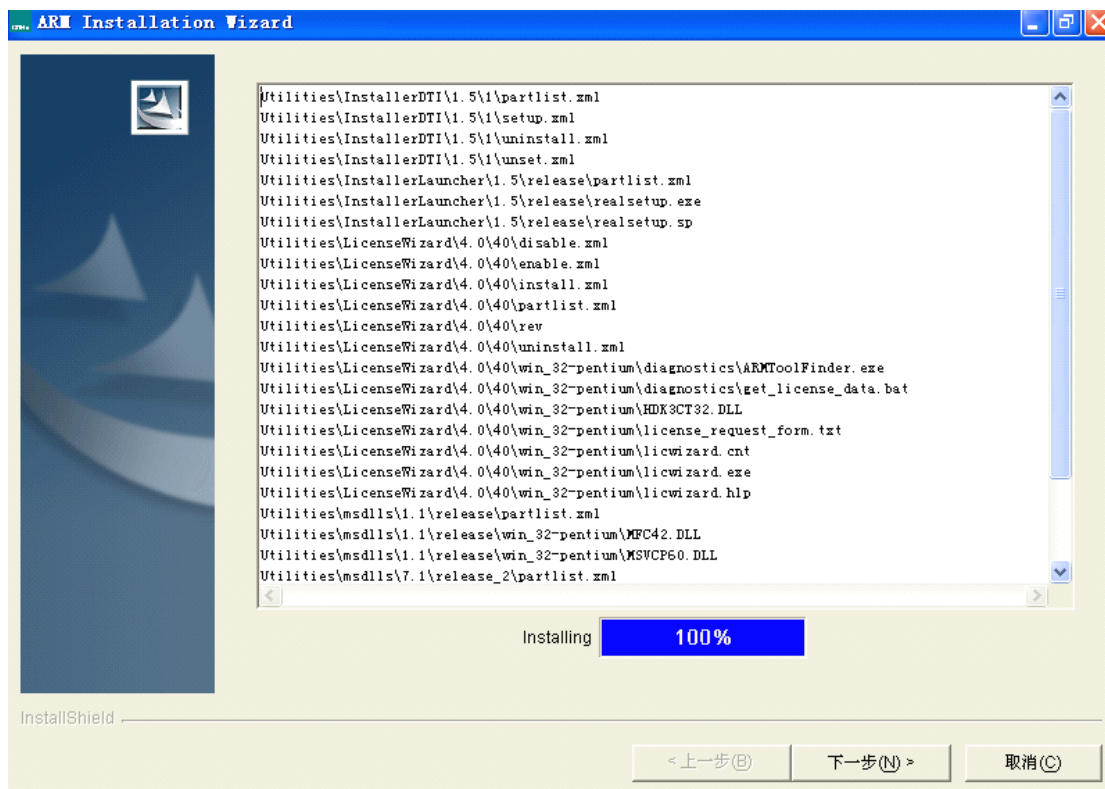
选择关联文件类型，这里只保留对 mcp (rvds 工程) 的关联。



这是注册 license 的步骤。注册步骤不再多讲，请看飞凌的关于 rvds 的教学视频。



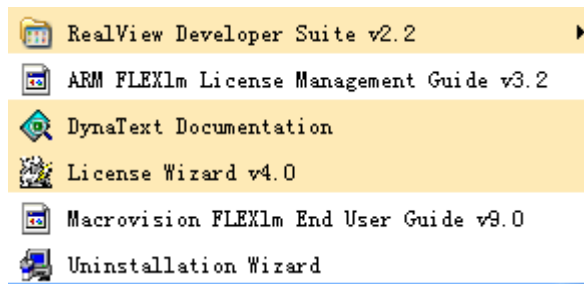
点击”下一步”



点击”完成”



在”开始->所有程序->ARM-> RealView Developer Suite v2.2->”中，可以看到安装好的rvds2.2。

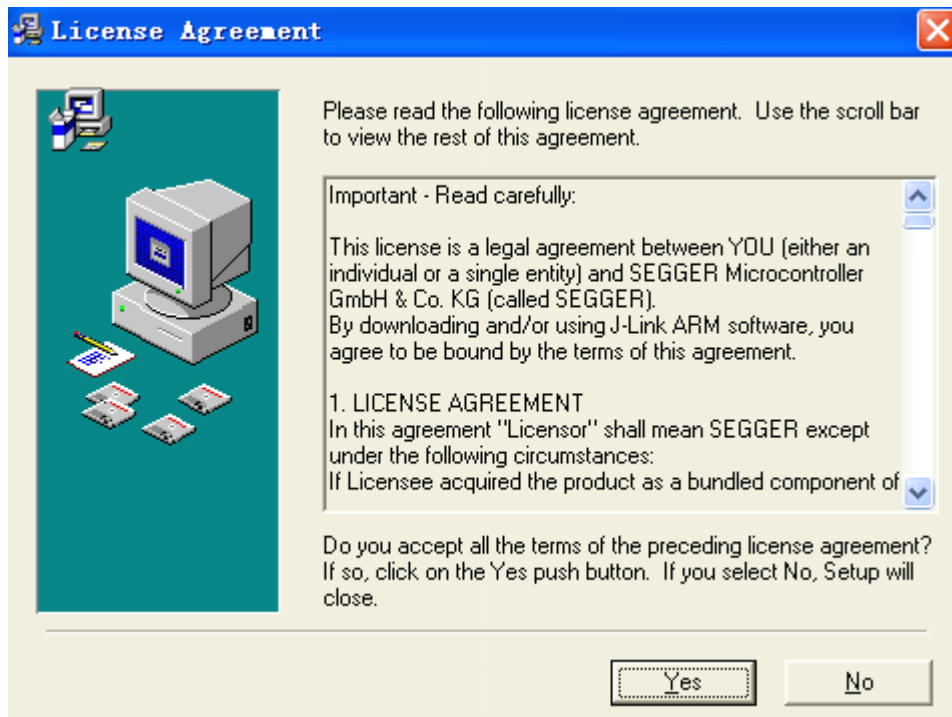


第三章 JLINK 驱动的安装

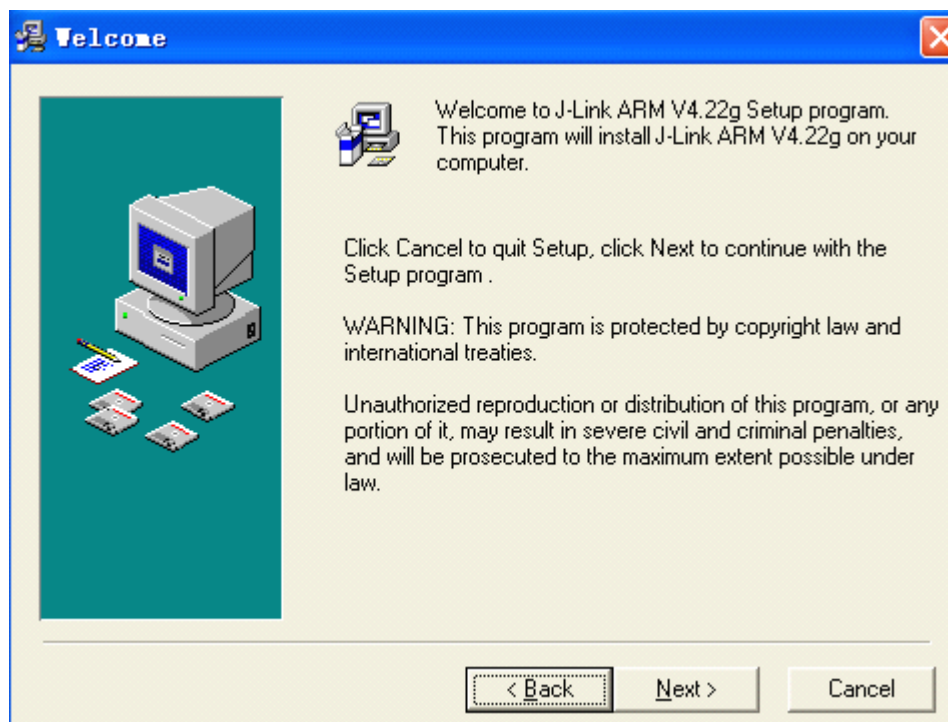
双击 setup_JLinkARM_V422g.exe 开始安装 Jlink 驱动。

驱动的位置：飞凌基础光盘\实用工具\JLink4.22 驱动\Setup_JLinkARM_V422g.exe

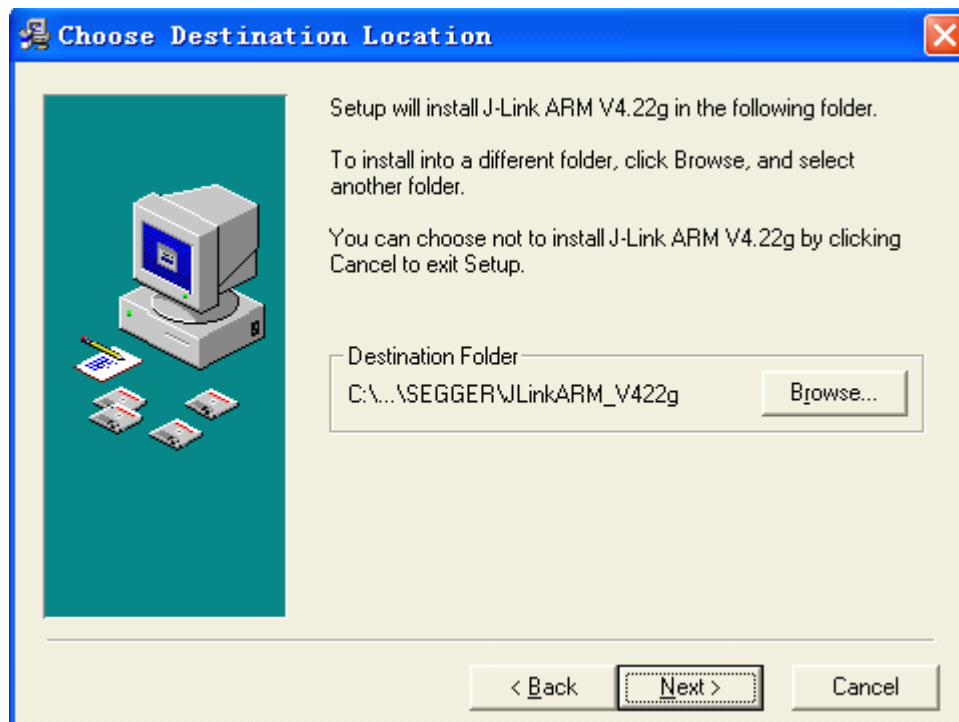
如图：



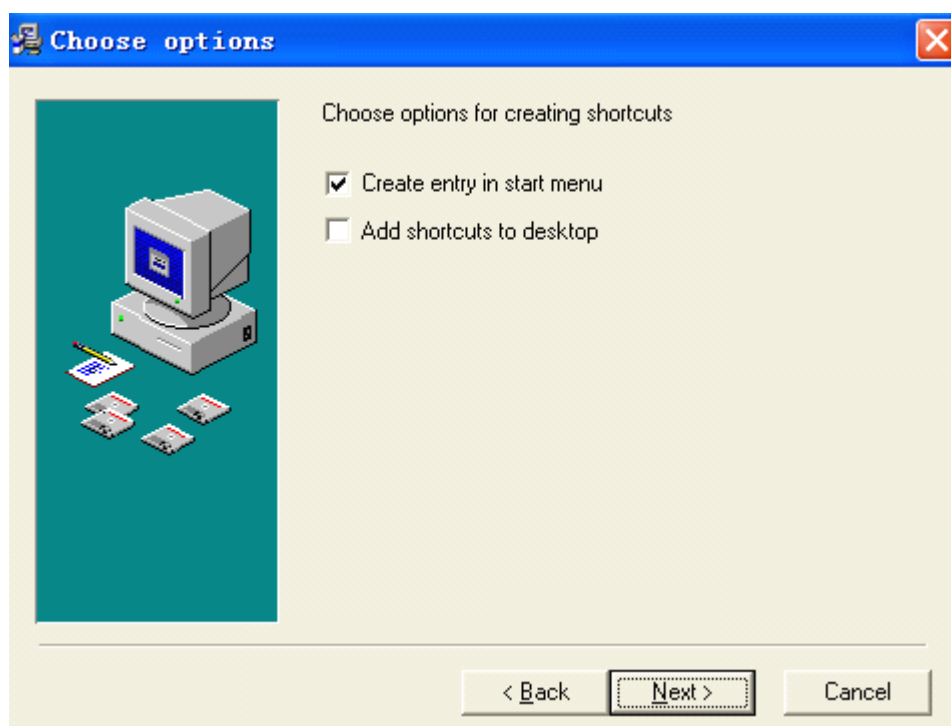
点击“Next”



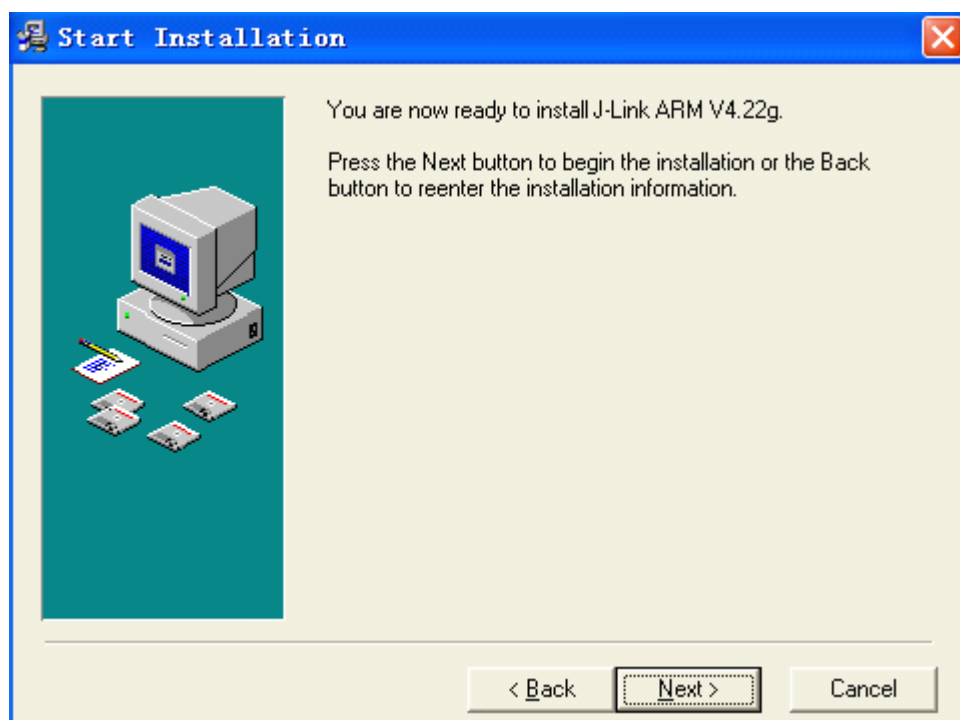
点击“Next”

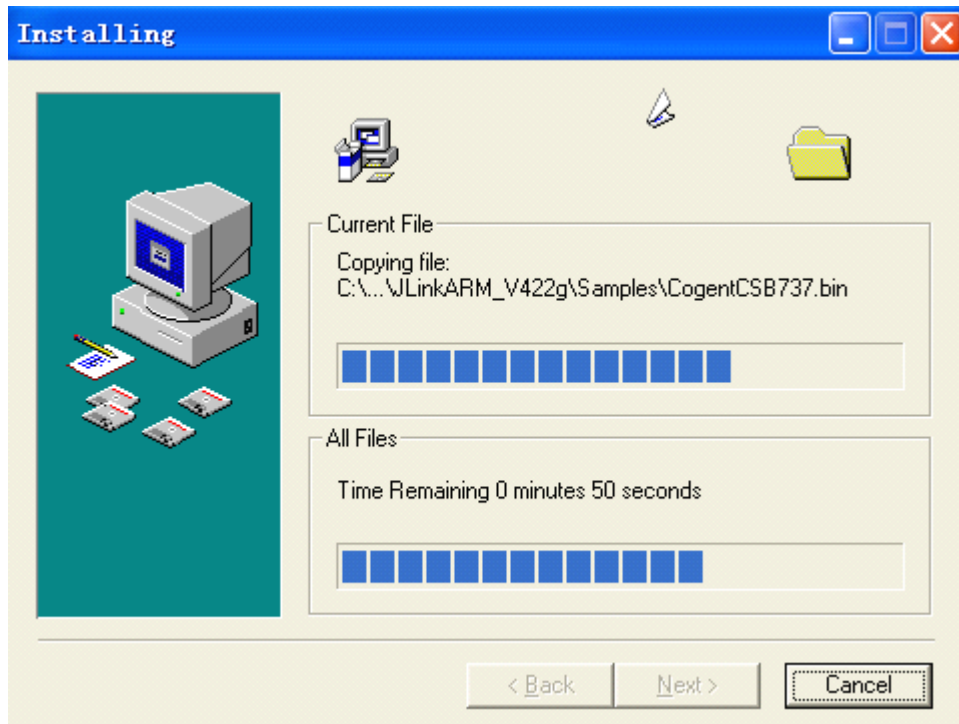


点击“Next”

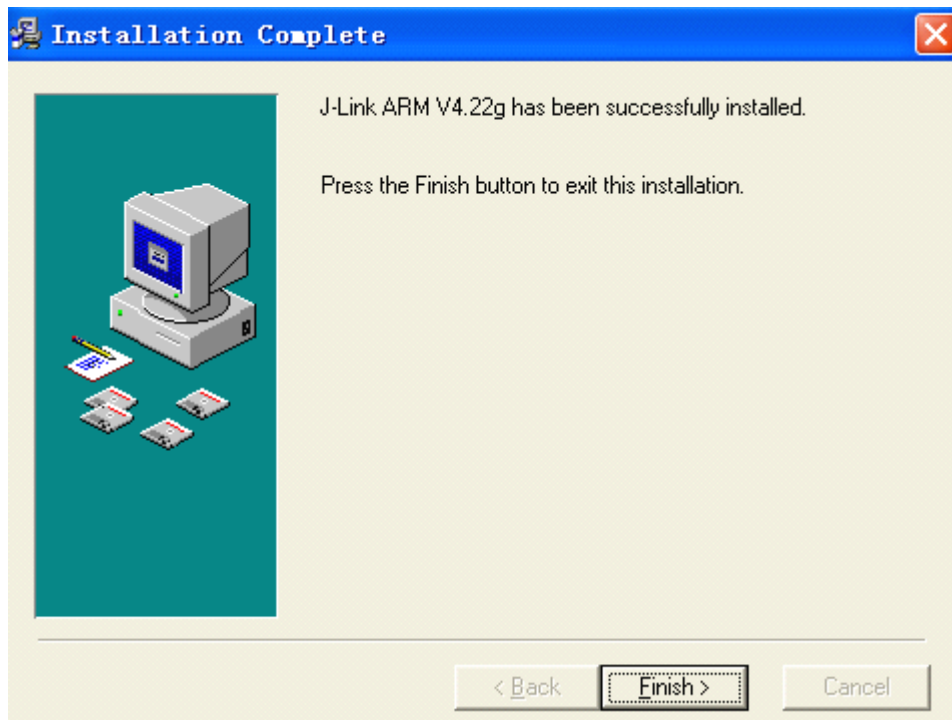


点击“Next”

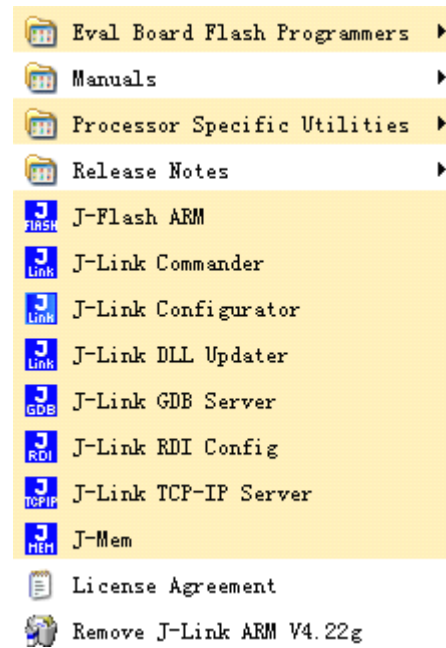




点击“Finish”



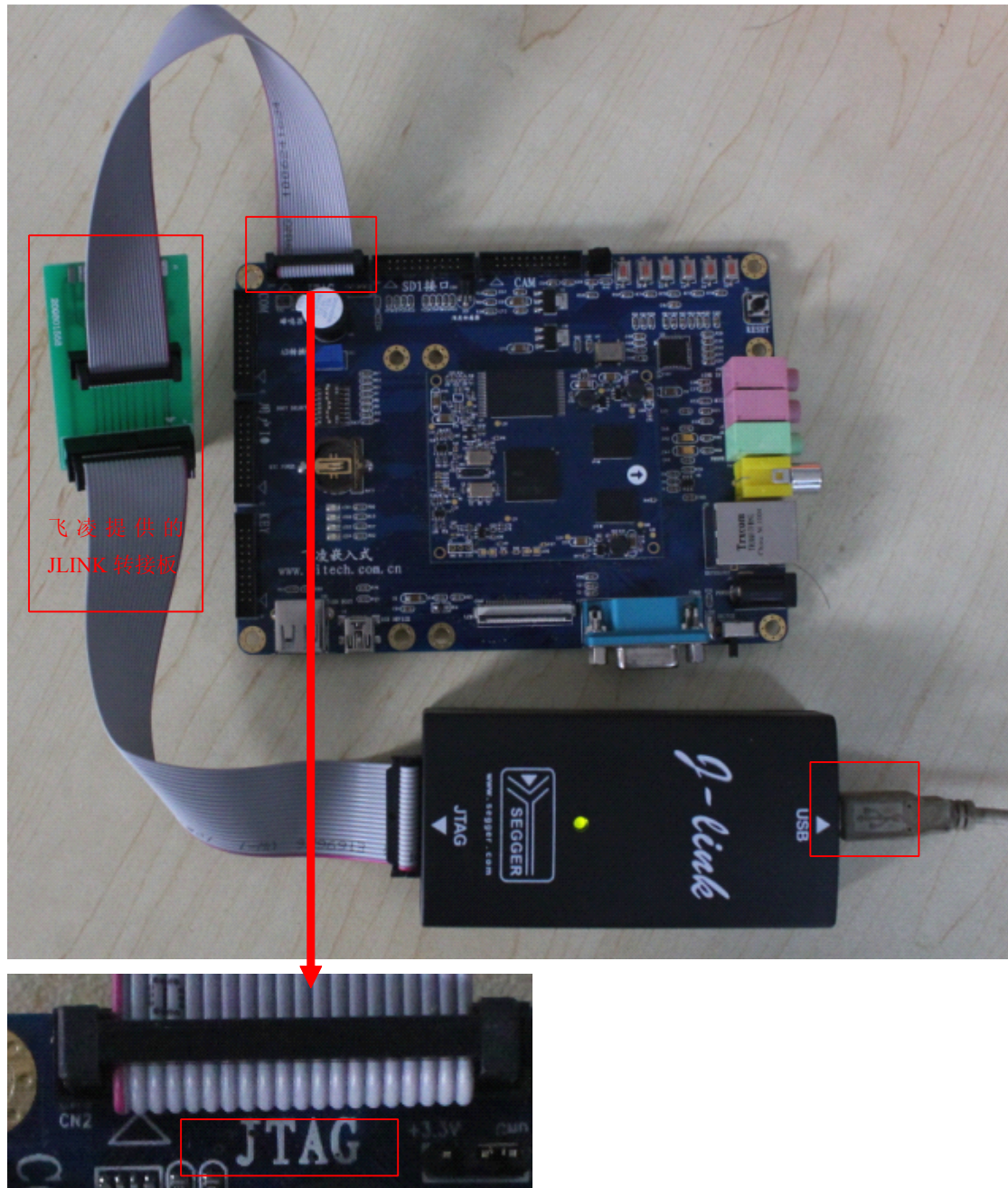
安装完毕后，可以在”开始->所有程序->SEGGER-> J-Link ARM V4.22g”中看到以下文件：



Jlink 驱动安装到此结束。

第四章 JLINK 的连接方法

PC、Jlink、OK6410 开发板连接方法如下图：



如图中，是飞凌官方出售 JLINK v8。每一块飞凌的 JLINK v8 在出厂前都会手动调试。请放心使用。

飞凌各位代理也会出售类似的 JLINK，可能与图中略有不同。如果可以正常的转接到飞凌的开发板上，都是可以正常使用的。

第五章 RVDS2.2 介绍

5-1 RVDS 开发工具介绍

RealView® Development Suite (RVDS) 是 ARM 公司继 SDT 与 ADS1.2 之后主推的新一代开发工具。RVDS 集成的 RVCT 是业内公认的能够支持所有 ARM 处理器, 并提供最好的执行性能的编译器; RVD 是 ARM 系统调试方案的核心部分, 支持含嵌入式操作系统的单核和多核处理器软件开发, 可以同时提供相关联的系统级模型 构建功能和应用级软件开发功能, 为不同用户提供最为合适的调试功效。

目前全球基于 ARM 处理器的 40 亿个产品设备中, 大部分的软件开发是基于 RealView 开发工具。安全、可靠和高性能地设计产品的最好选择就是购买 ARM RealView 开发工具。

RVDS 向下兼容以前的版本 (ADS v1.2.1、1.1、1.0.1)。

RealView Development Suite (RVDS) 是为从事 SoC、FPGA 和 ASIC 设计的工程师, 进行复杂的嵌入式应用和平台接口而设计的。RVDS 向硬件设备的设计者提供多核调试、应用与所有的 ARM 处理器的代码产生和 CORTEX CPU 的配置等功能。它提供了到达第三方元件的接口 (如 ARM ESL tools)。

5-2 集成开发环境 (IDE)

Rvds2.2 集成了 Codewarrior for rvds 开发环境。

Codewarrior for rvds 提供基于 Windows 使用的工程管理工具。它的使用使源码文件的管理和编译工程变得非常方便。但 CodeWarrior IDE 在 UNIX 下不能使用 RealView 编译工具 (RVCT)。

优化的标准 C/ C++编译器

链接器

汇编器

映像转换工具

ARM 目标文件管理

C 语言库

RogueWave C++标准模版库

RealView 编辑工具

为了给 ARM 架构提供最优异的支持, ARM 公司经过十六年的研究推出了 ARM RealView 编译工具。他们包含了能够将 C 或 C++编译成 32 位 ARM 指令集、16 位 Thumb 指令集和 Thumb-2 指令集所必需的软件部件。

RVDS 编译工具为 ARM 架构提供了最优异的支持, 它在代码的速度和大小上有了许多重要改进。

GNU Interoperability

使用 RVDS 中的编译工具能为嵌入式 Linux 和 Symbian 系统提供最优化的应用程序。RVDS 为那些一直在寻求互用 ARM 与 GNU 工具链,并且基于 ARM 架构的兼容 Application Binary Interface (ABI)的客户 提供空前灵活的支持,使开源工具与商业工具的使用贯穿整个软

件开发团队。

第六章 RVDS2.2 的使用方法以及调试方法、调试的原理

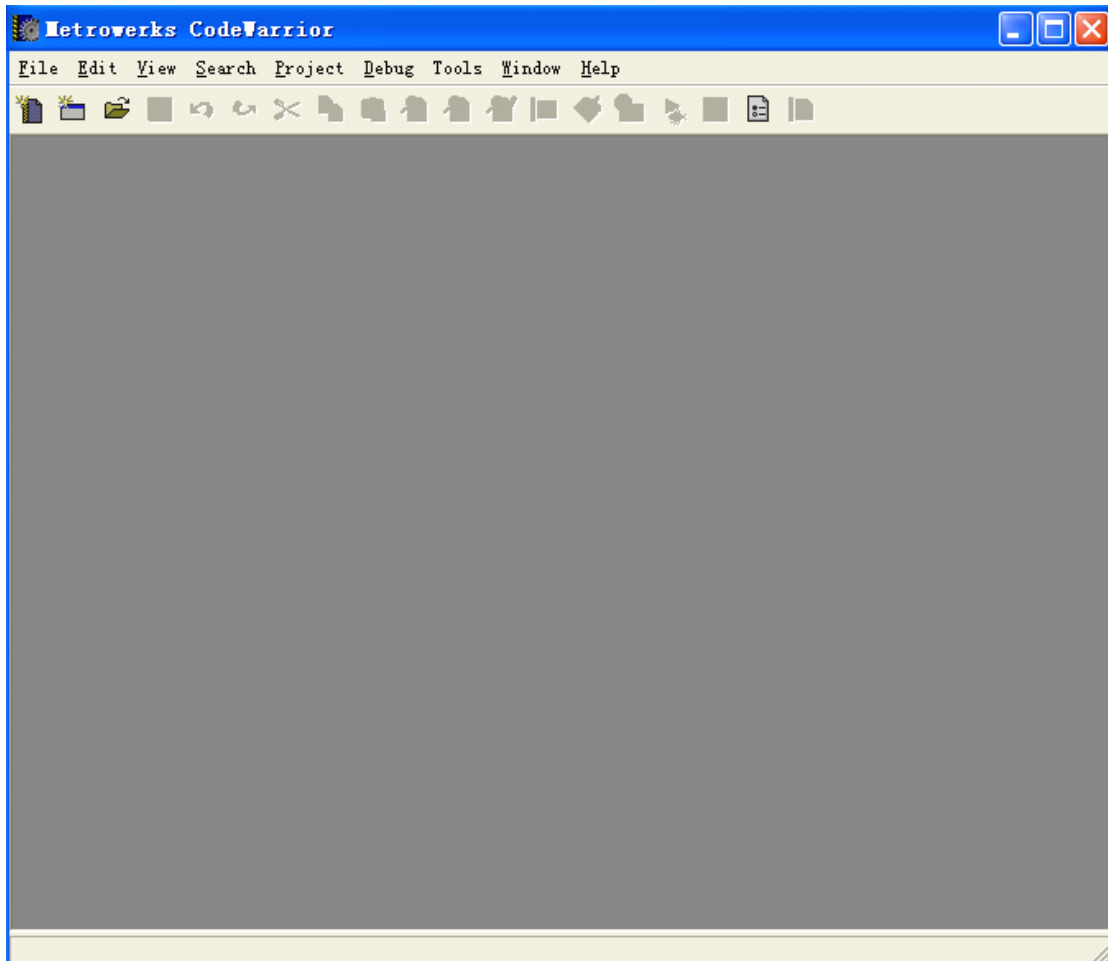
6-1 CodeWarrior for RVDS

6-1-1 打开 CodeWarrior

打开路径：

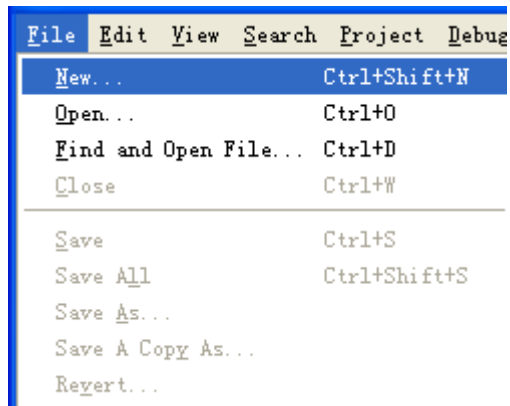
开始->所有程序->ARM-> RealView Developer Suite v2.2-> CodeWarrior for RVDS

打开后如图：

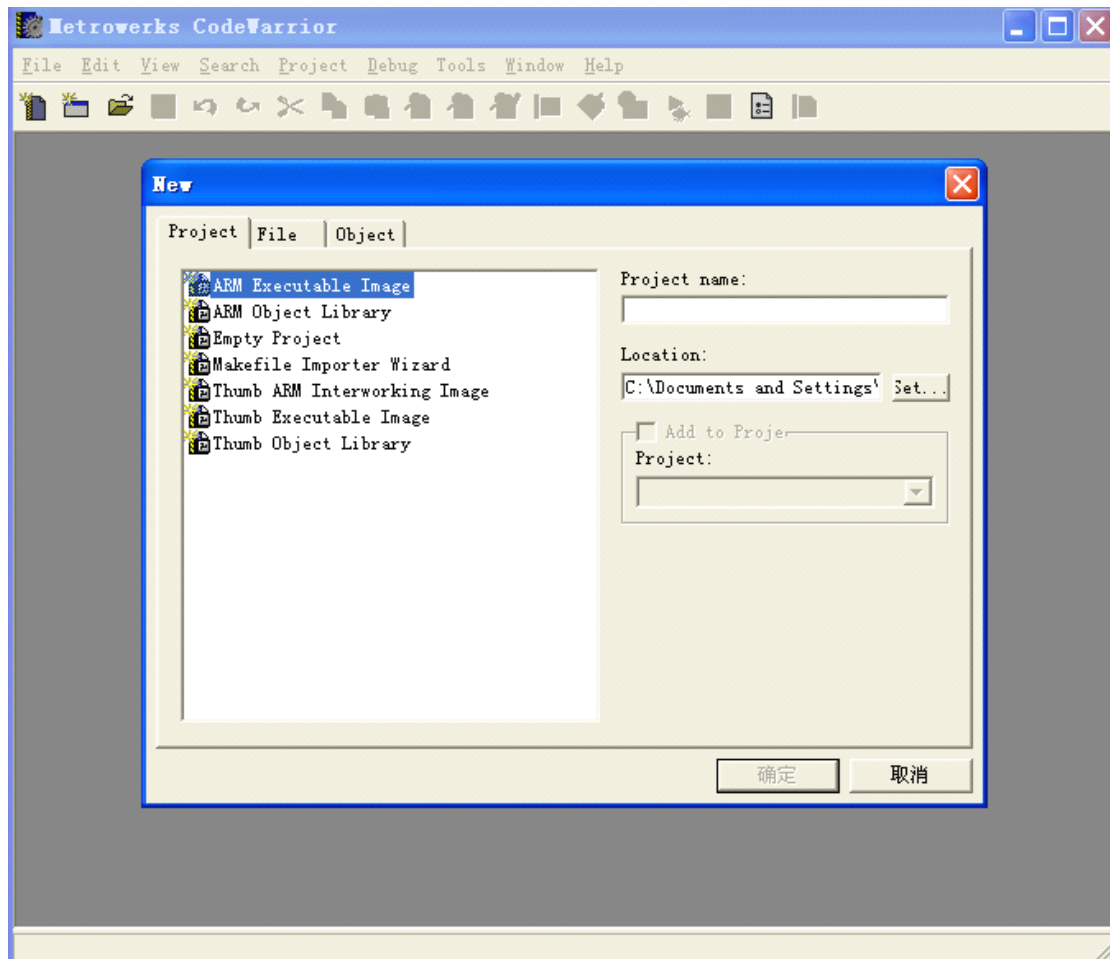


6-1-2 新建 OK6410 裸机工程的方法

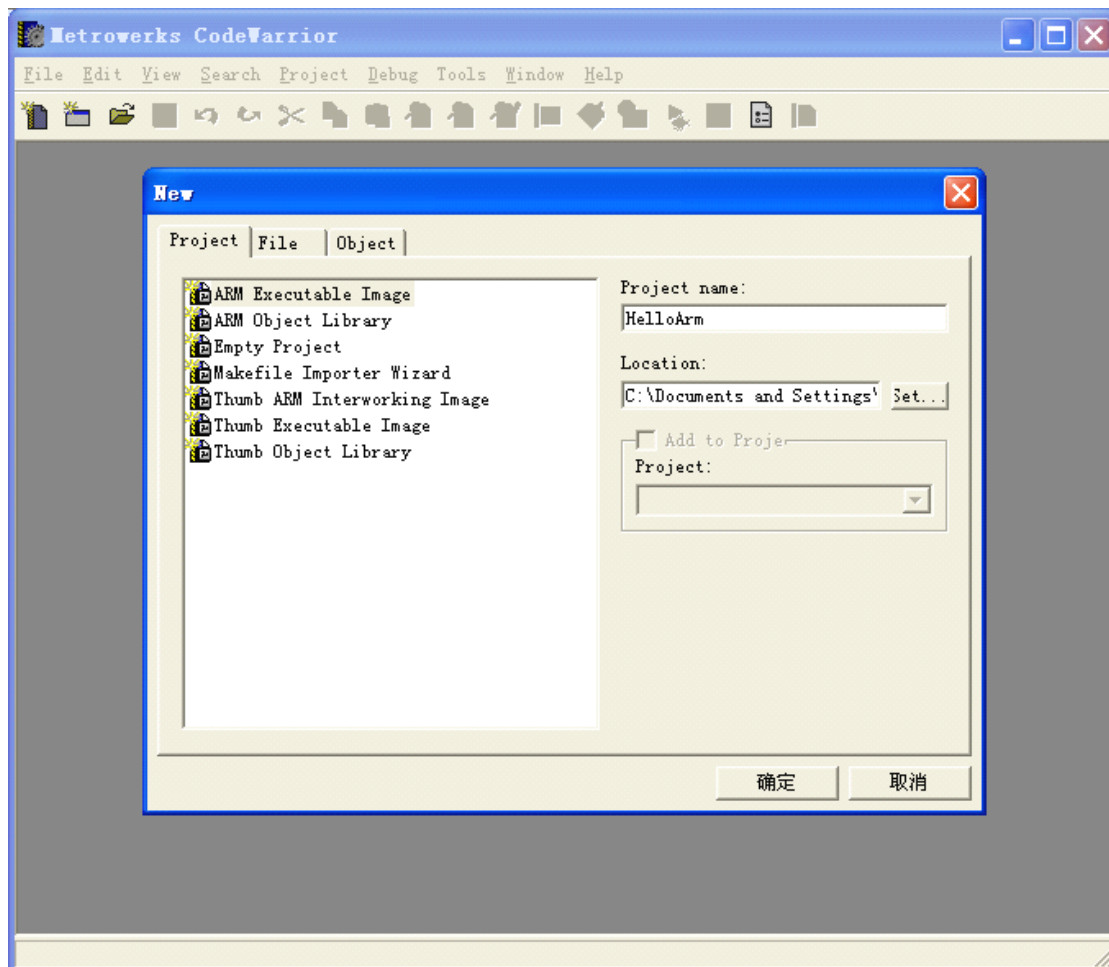
点击打开 File->New



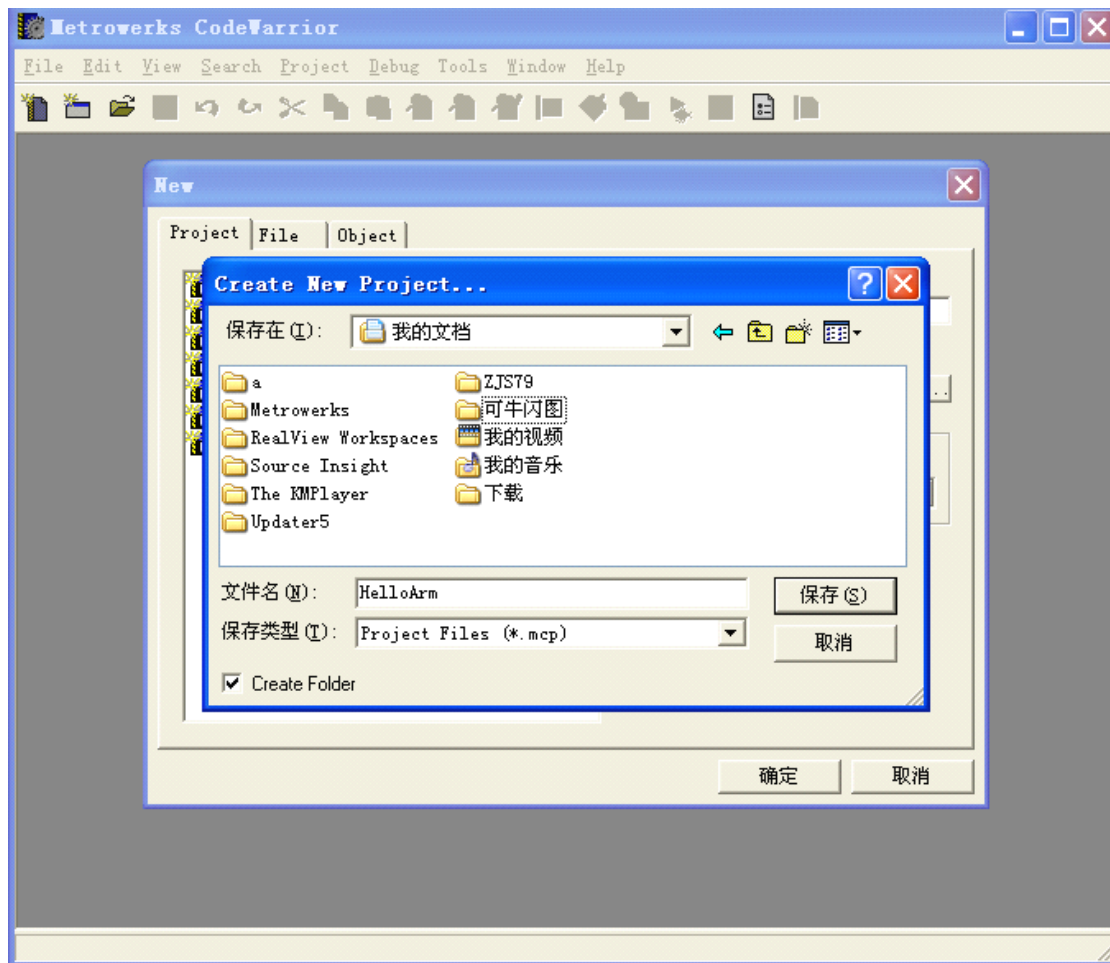
弹出新的新建对话框，如图：



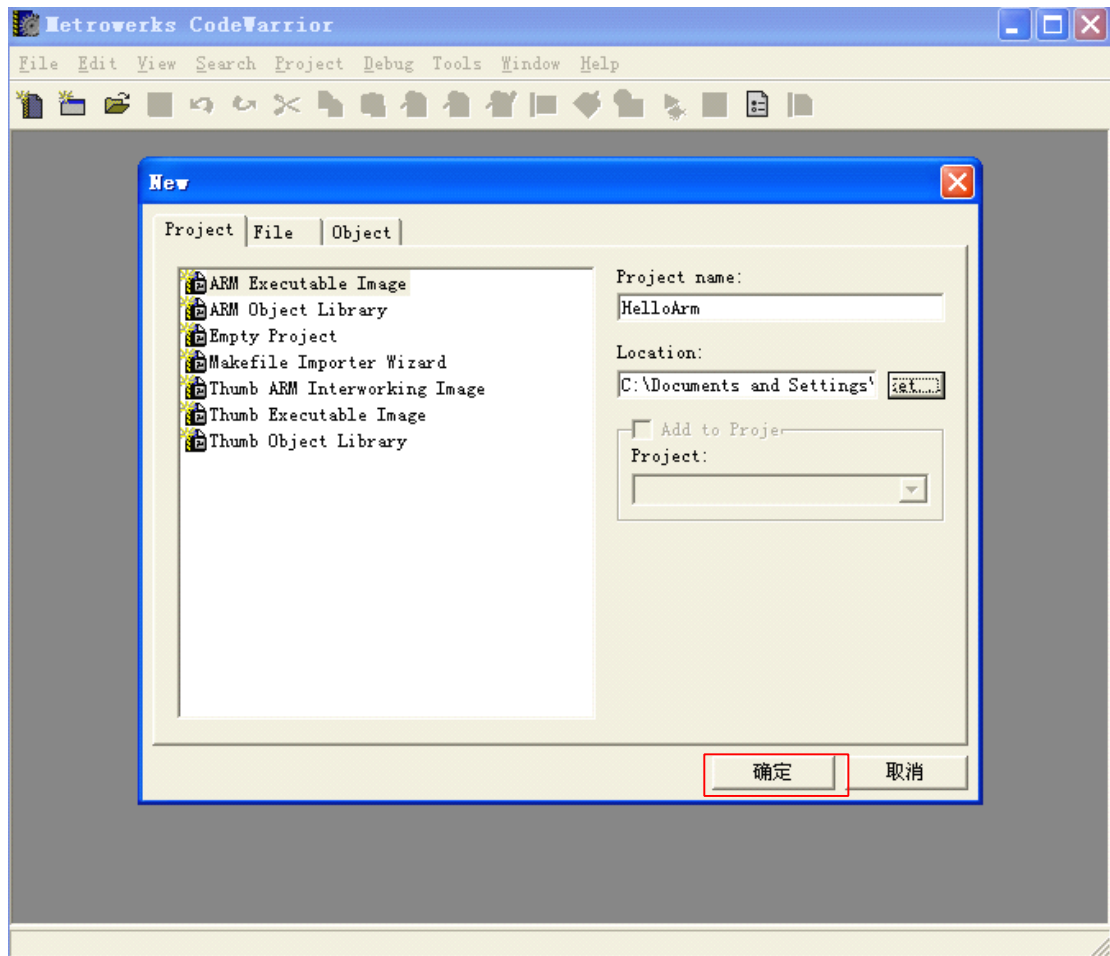
在 Project 选项签中，选择 ARM Executable Image。然后在右侧”Project name”中填写一个工程名。以 HelloArm 作为工程名，点击”Set...”，选择工程存放的目录。



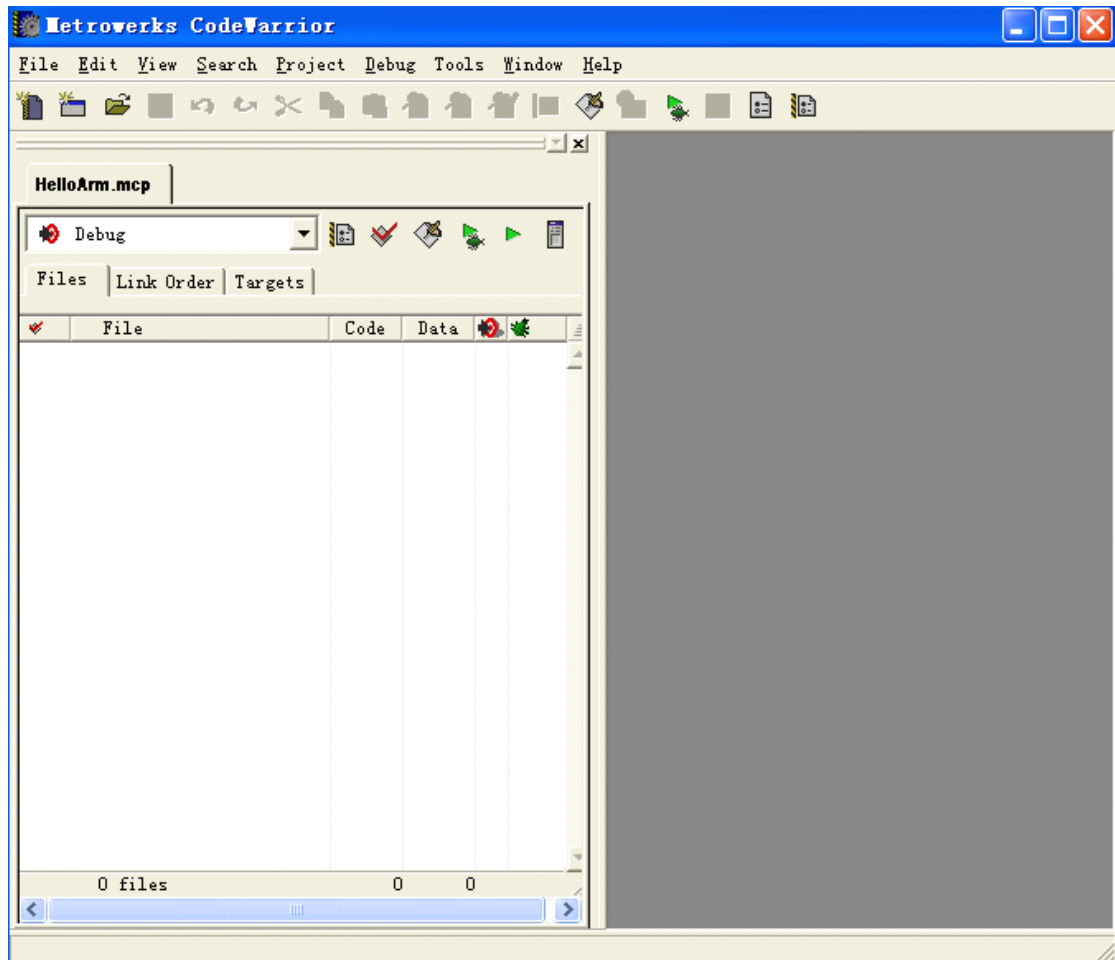
选择工程保存的目录:



填写工程名和选择路径后，点击确定，如图：



这样，就完成了新建工程的工作：

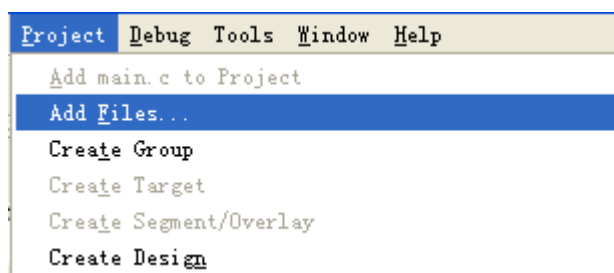


6-1-3 为工程添加源码文件

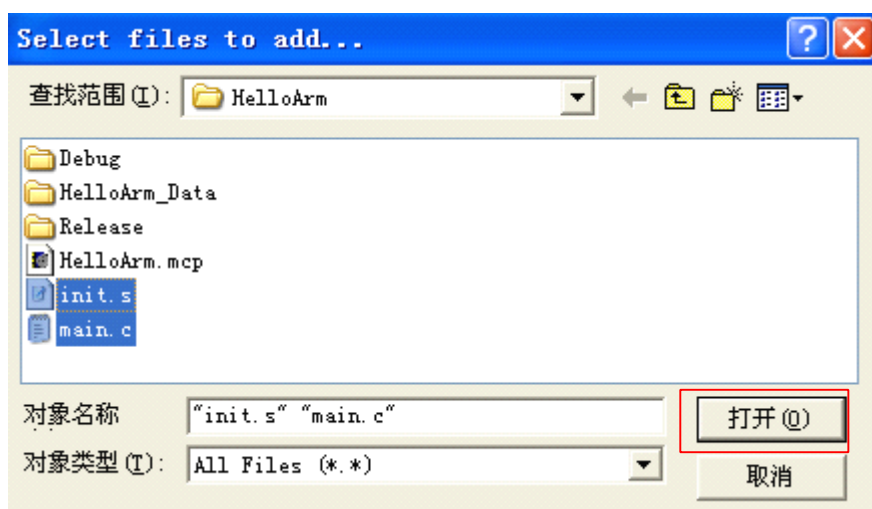
作为例子，将飞凌 led 实验中的 main.c 和 init.s 文件拷贝到工程目录下。



在 CodeWarrior for RVDS 中，点击 Project->Add Files

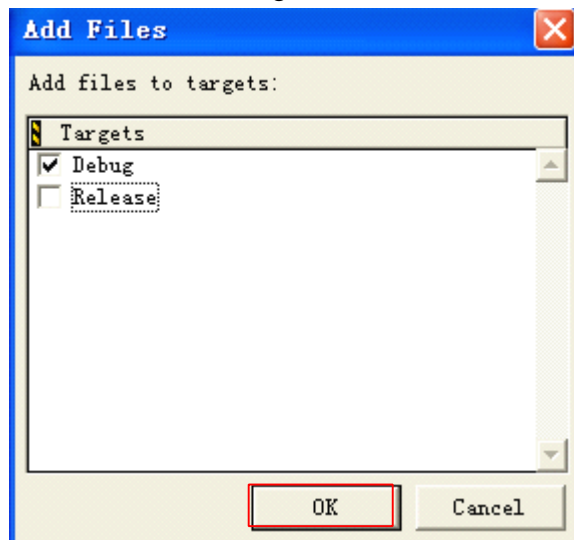


选择 init.s 和 main.c 文件，然后点击”打开”

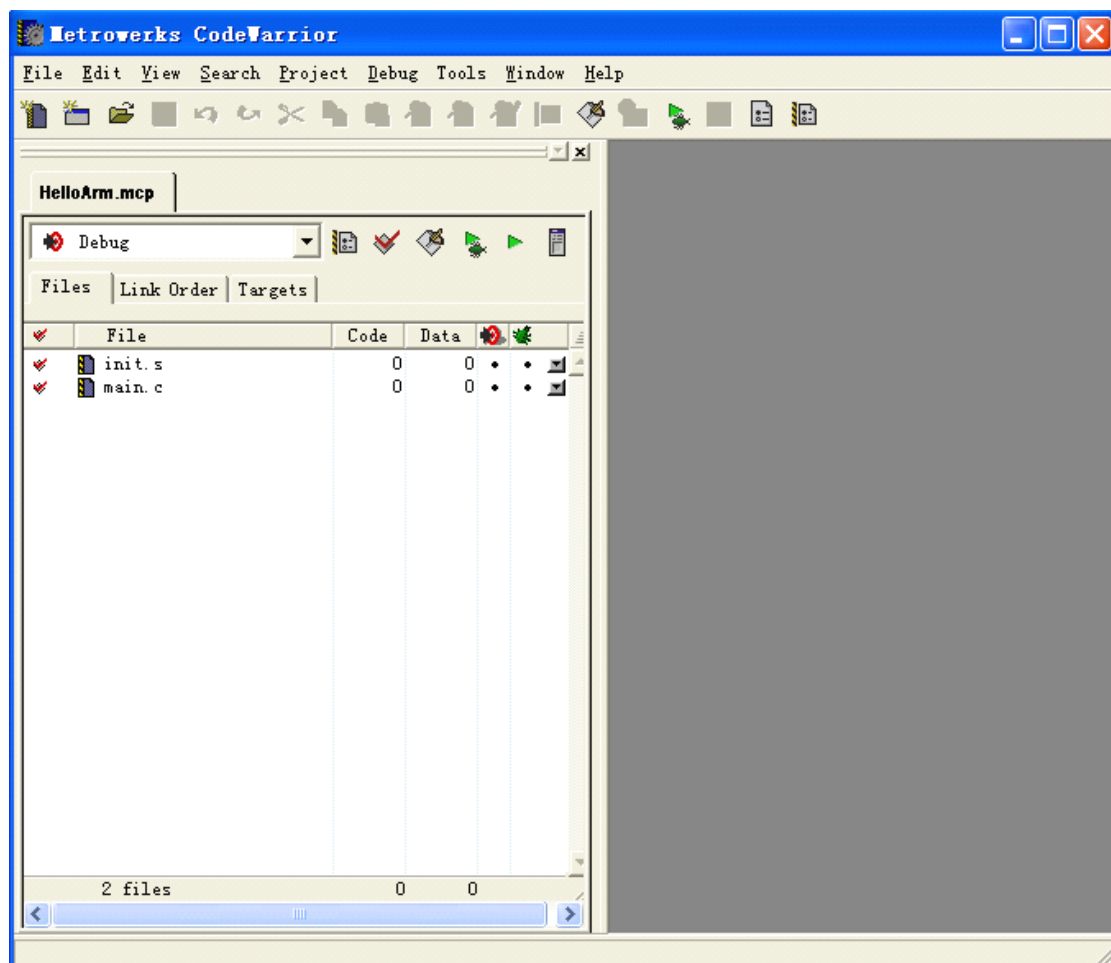


选择源码文件后，自动弹出这个对话框。这个对话框是要我们为新添加文件选择一个目标属性。在试验中，通常选择 Debug，作为调试。所以，在这里也只保留 Debug。

为目标选择 Debug 属性后，点击”OK”

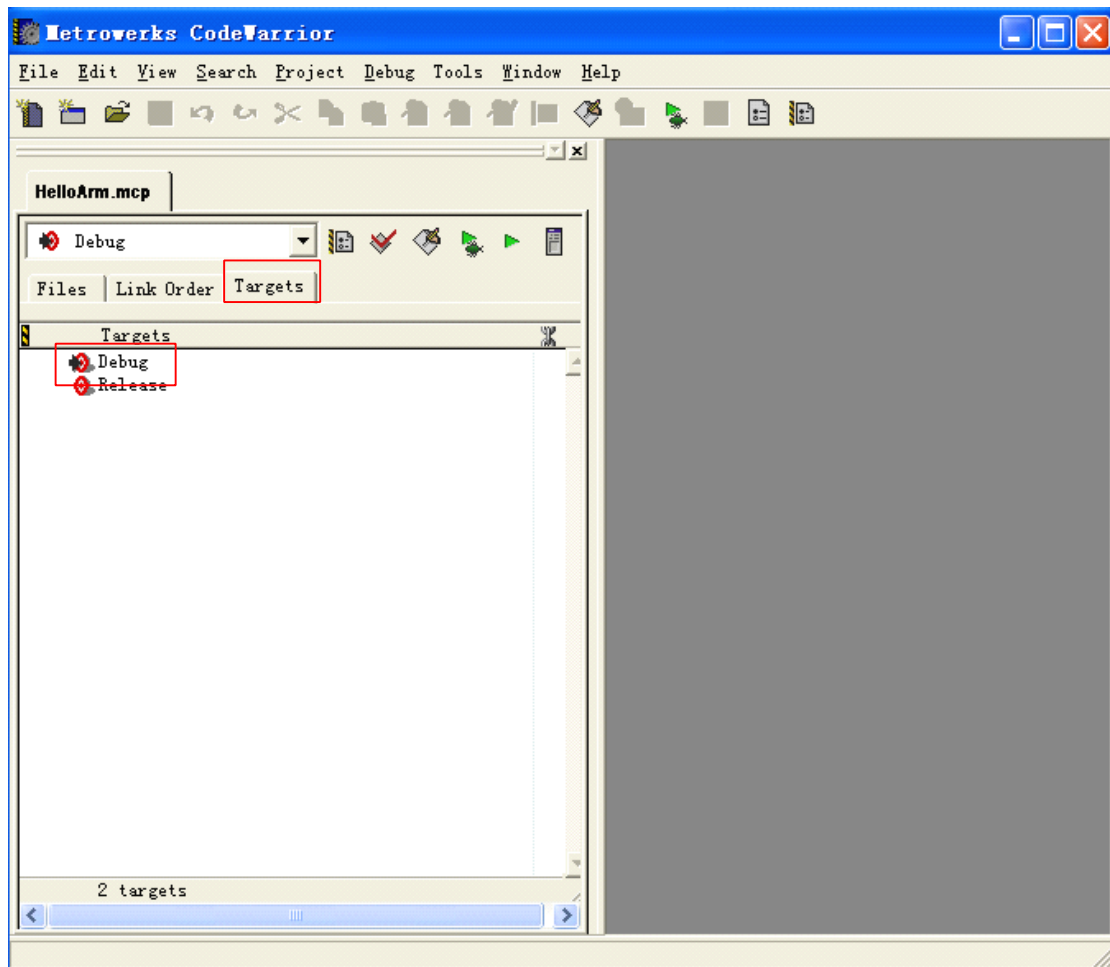


添加完成后，就可以看到 main.c 和 init.s 包含在工程中了。

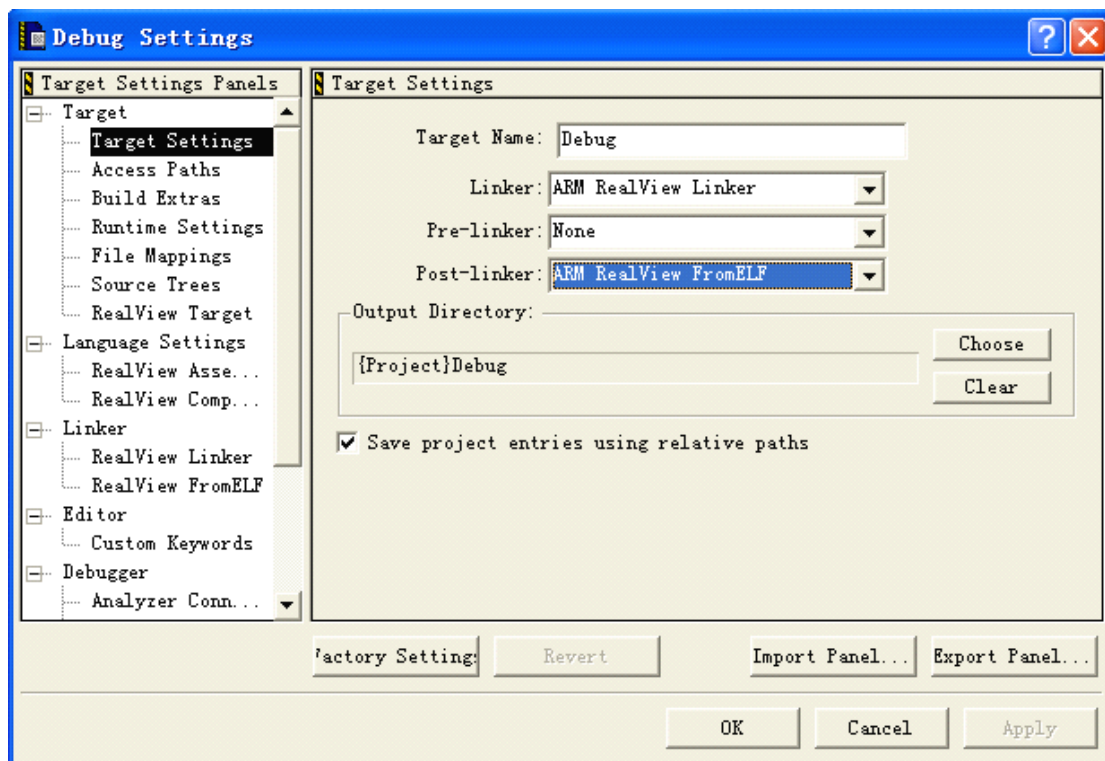


6-1-4 为工程进行必要的设置

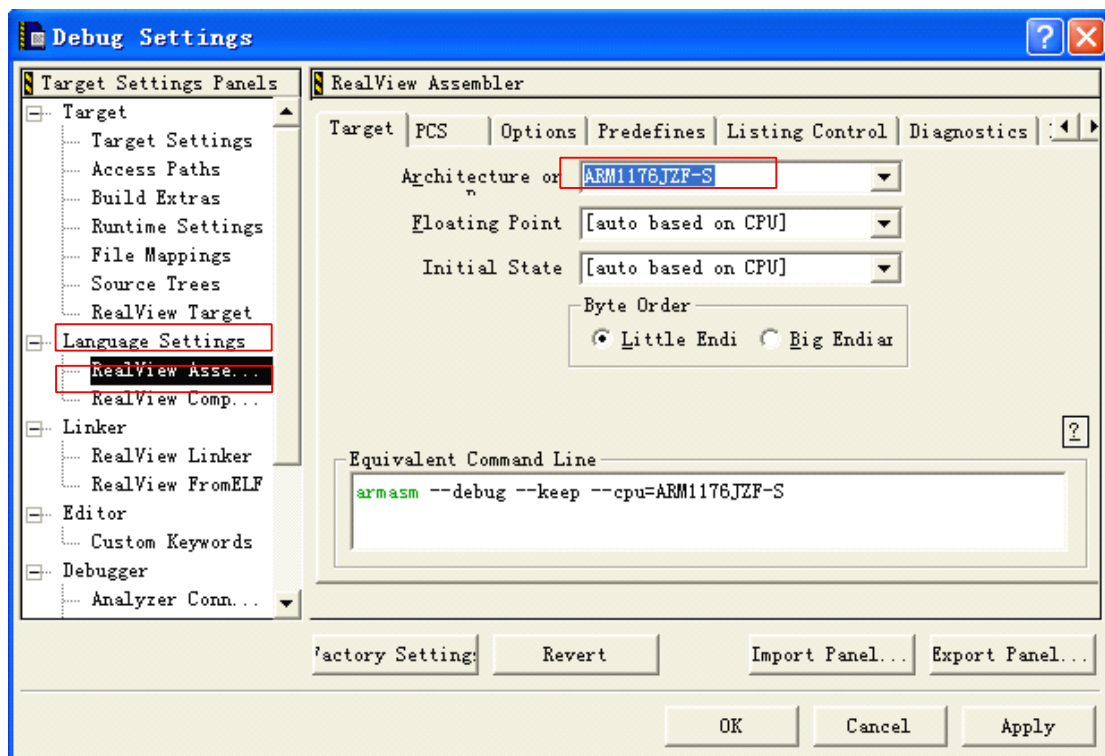
点开工程的”Targets”选项签，鼠标左键双击”Debug”。



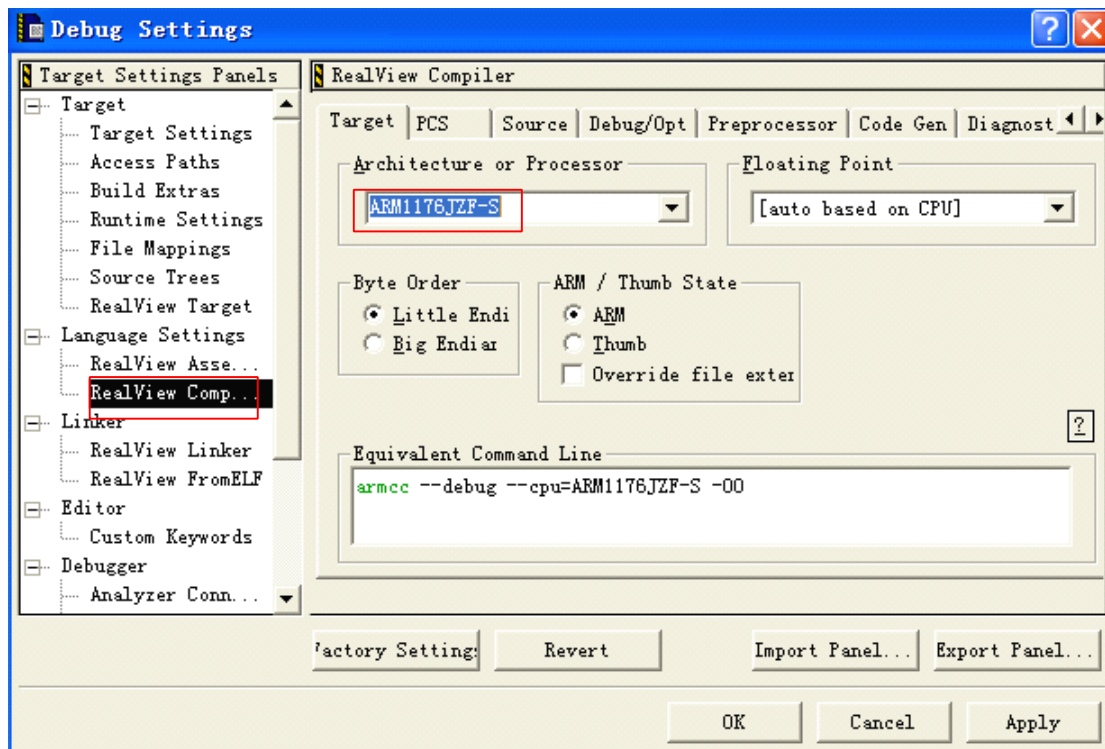
打开设置后如图：



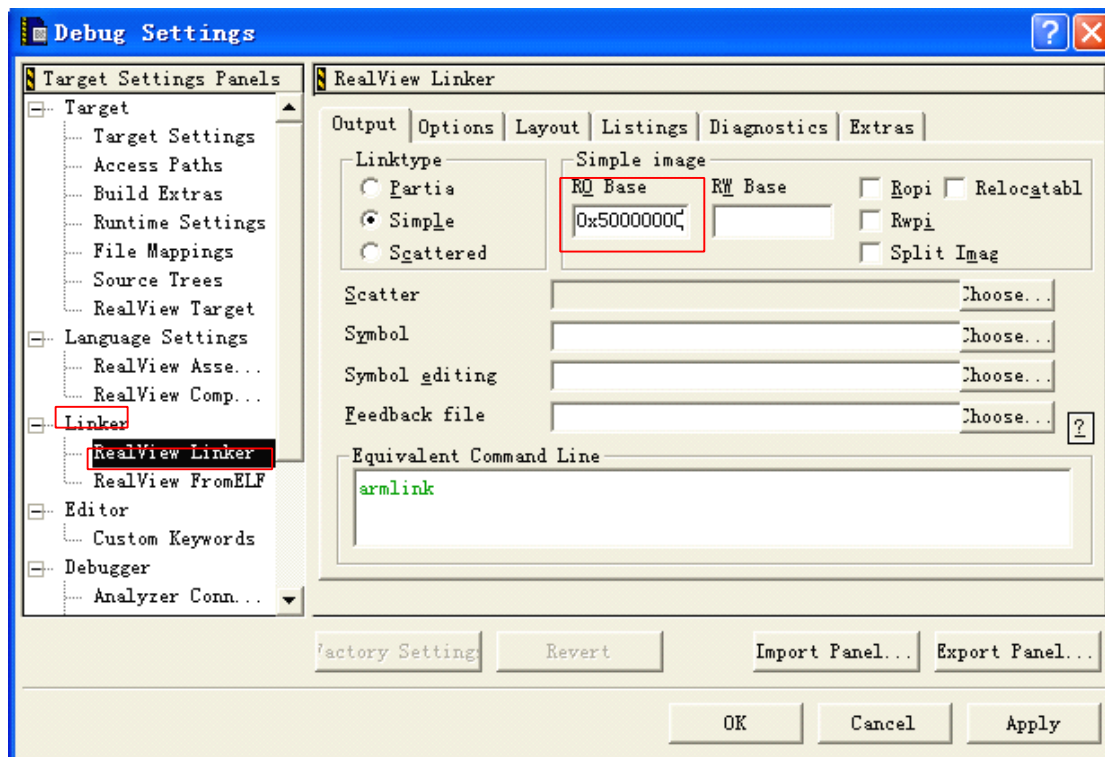
打开”Language Settings”。打开 RealView Assembly”。将右侧” Architecture or Processor”的下拉菜单改为”ARM1176JZF-S”。



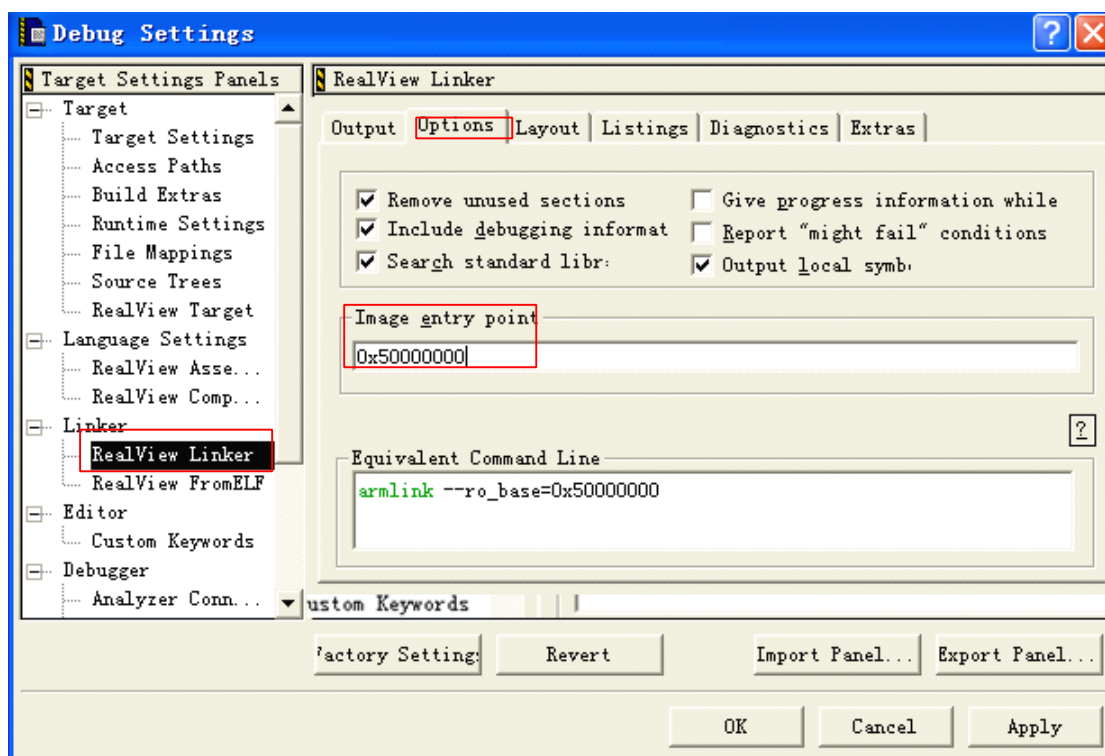
然后点击"RealView Compile"，将"Architecture or Processor"的下拉菜单改为"ARM1176JZF-S"。



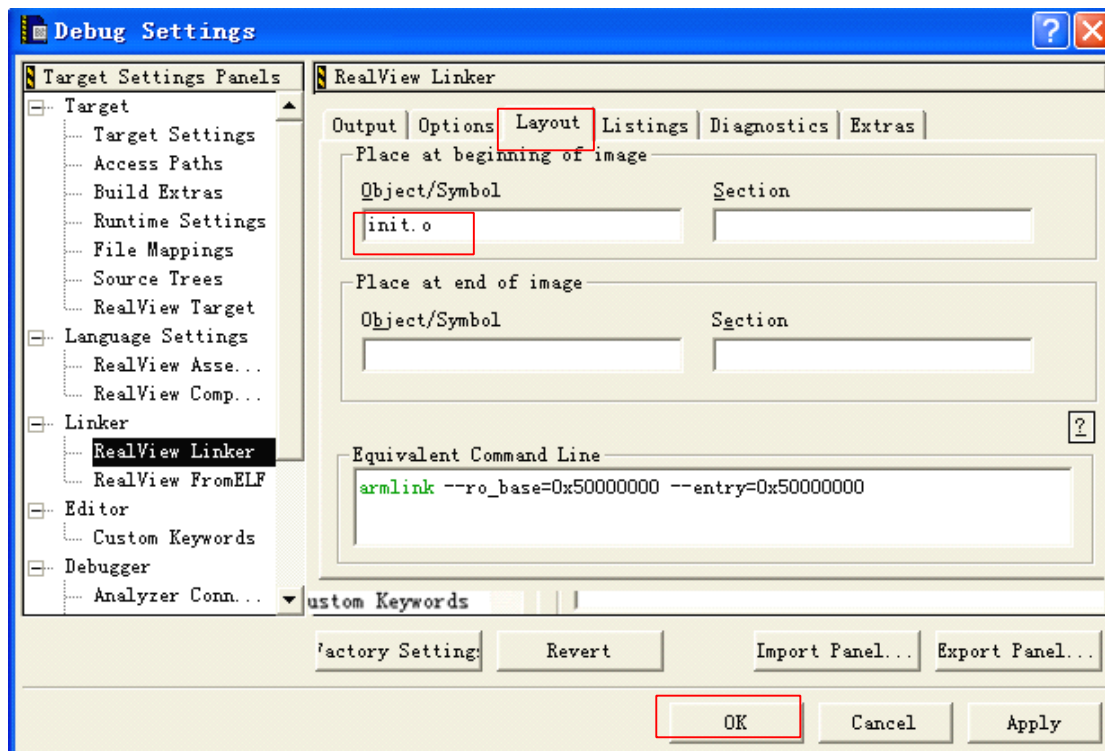
点击"Linker"，点击"RealView Linker"，RO Base 下面的文本框中填写"0x50000000"。



点击“Opints”选项签，将“Image entry point”中填写 0x50000000

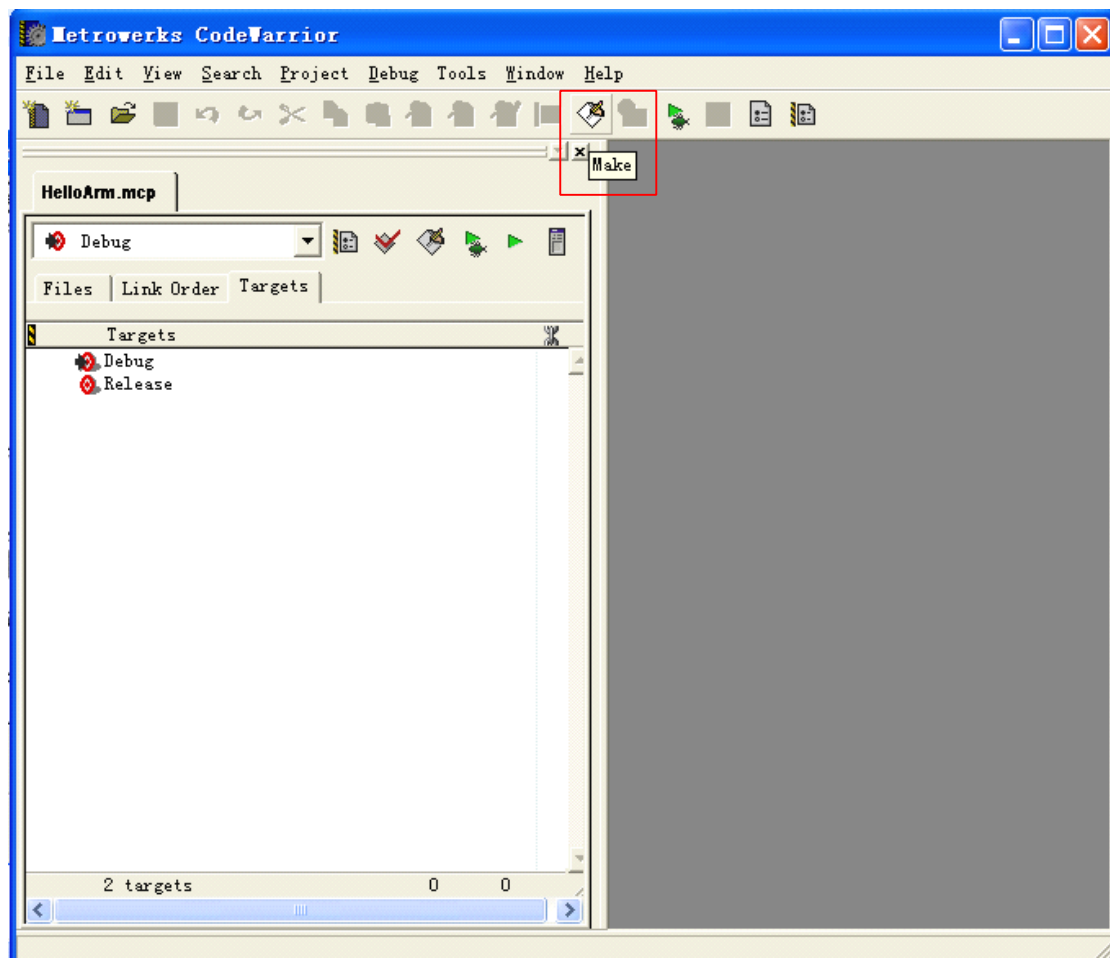


点击“Layout”选项签，将“Object/Symbol”中填写 init.o。设置完成后点击“OK”

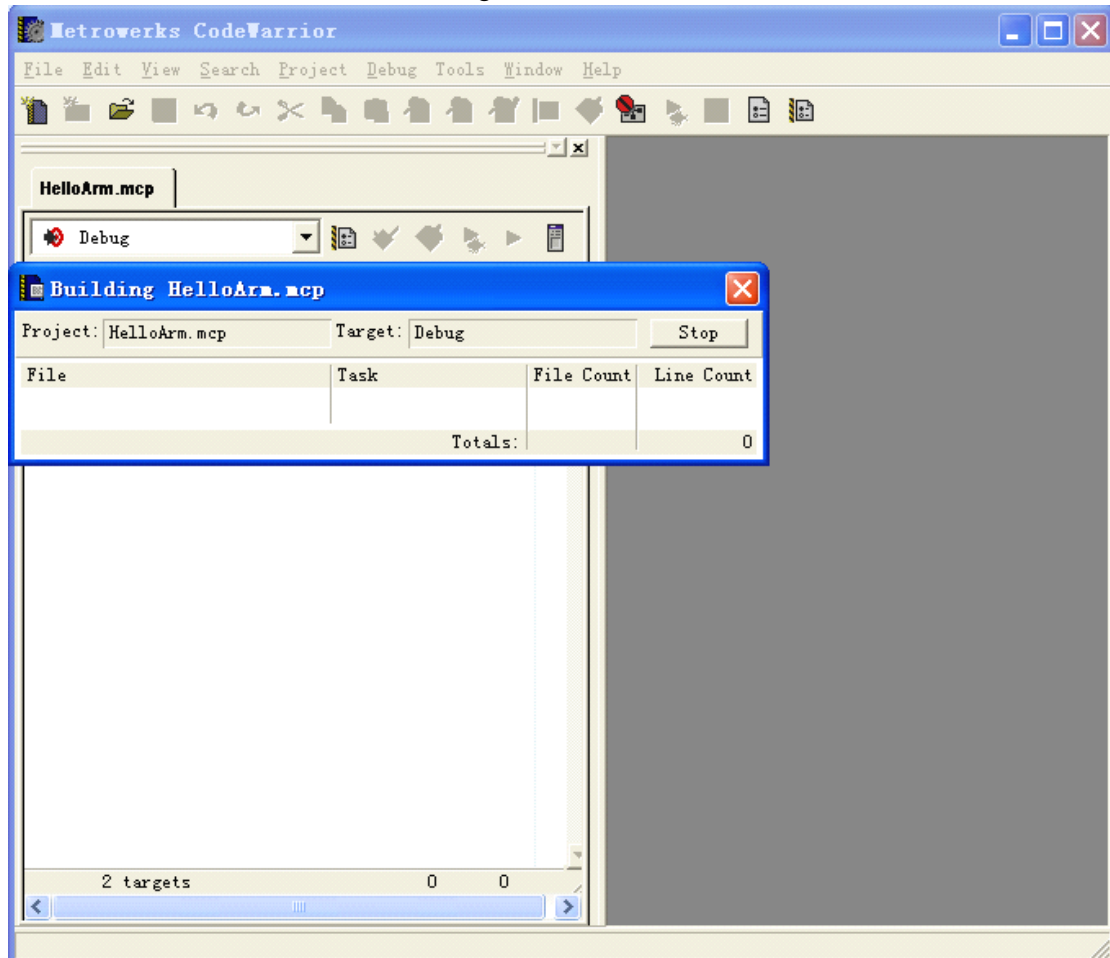


6-1-5 工程编译方法

设置完成后，点击开始编译整个工程。



编译过程当中，会弹出一个 Building 的工程对话框。这个对话框会显示编译当前的状况。



编译完成后，在工程所在目录的 Debug 目录下，可以看到我们已经编译好的 HelloArm.axf 文件。这个文件可以用 JLINK v8 进行仿真。

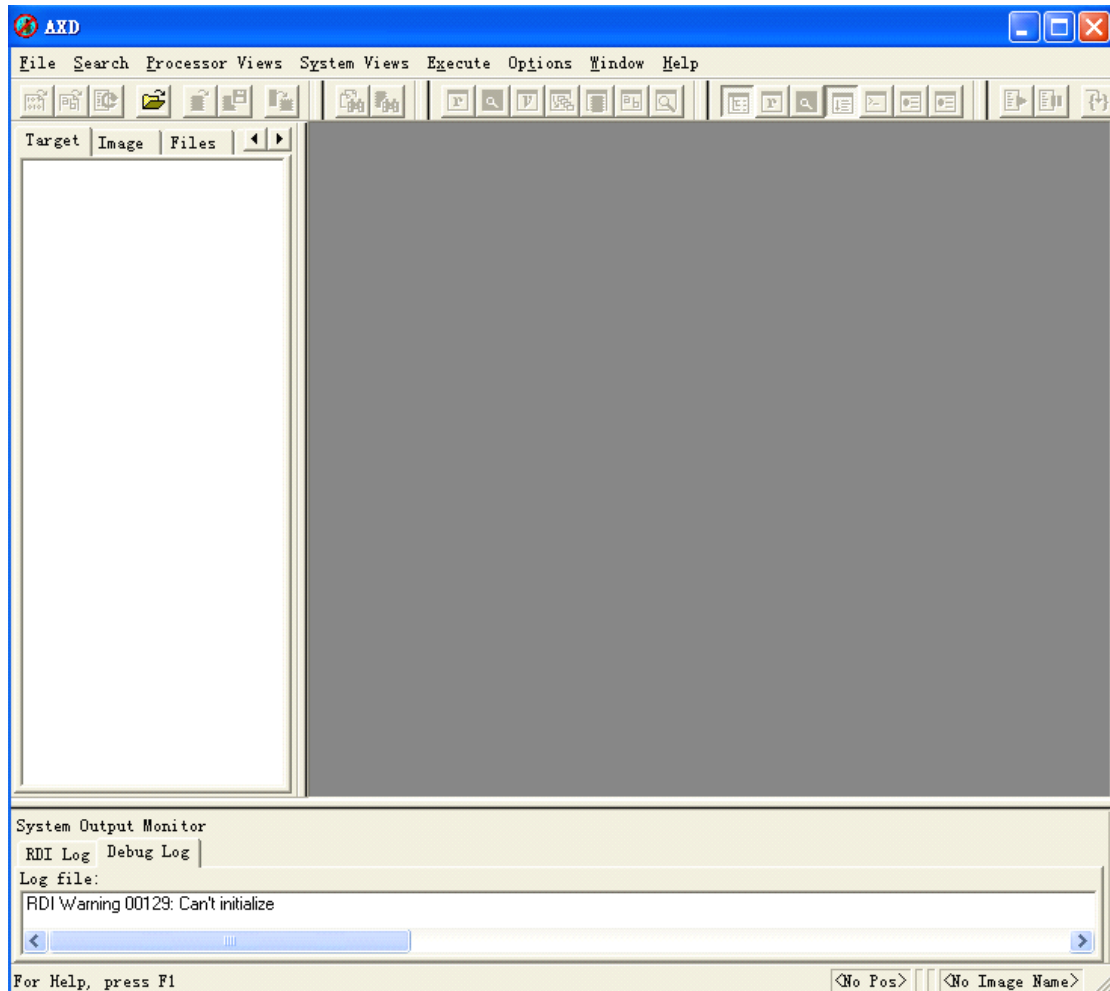
gs\fat\My Documents\HelloArm\Debug



6-2 设置 AXD1.3.1

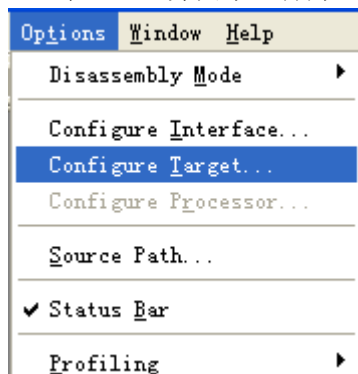
6-2-1 打开 AXD1.3.1

打开 AXD1.3.1，如图：

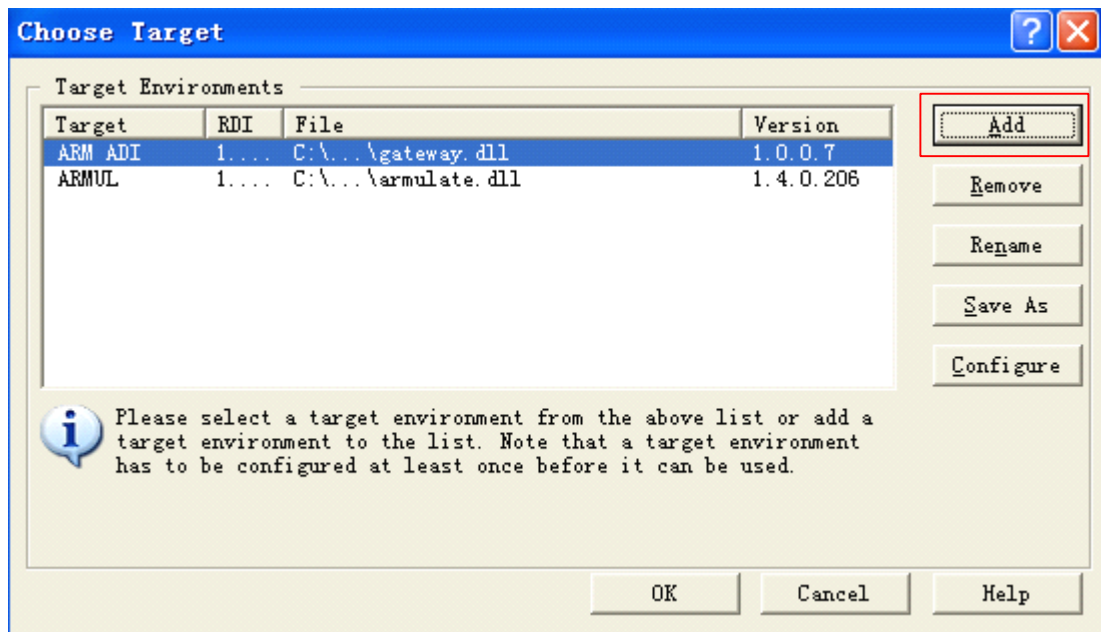


6-2-2 针对 JLINK 设置 AXD

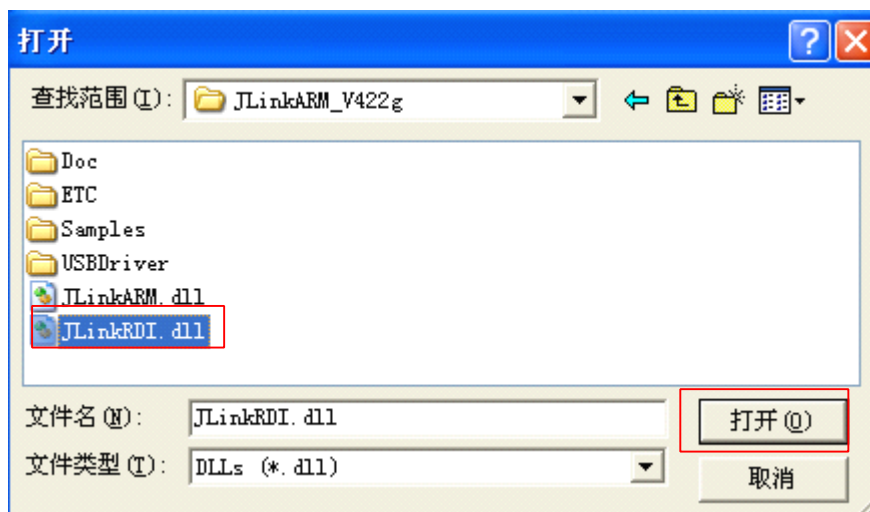
在 AXD 界面中，打开 Option->Configure Target...



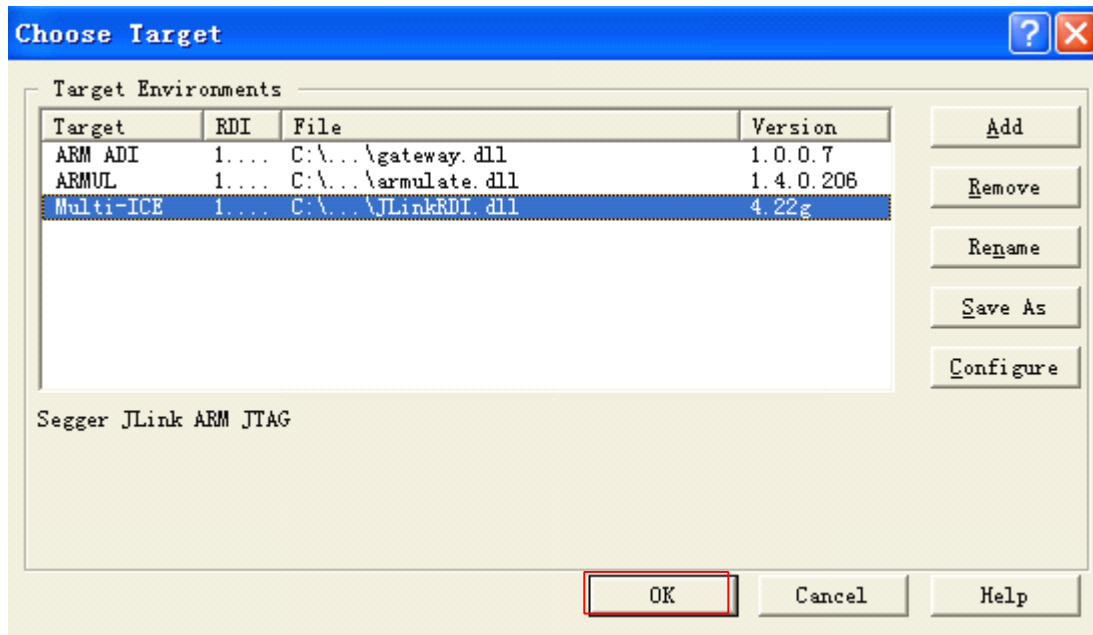
在新的对话框中，点击”Add”。



在新弹出的对话框中，找到 JlinkARM_V422g 的安装目录。选择”JLinkRDI.dll”，点击”打开”。



这是返回到选择目标的对话框。这是就能看到 JLinkRDI 出现在 AXD 的目标选项中了。点击“OK”。

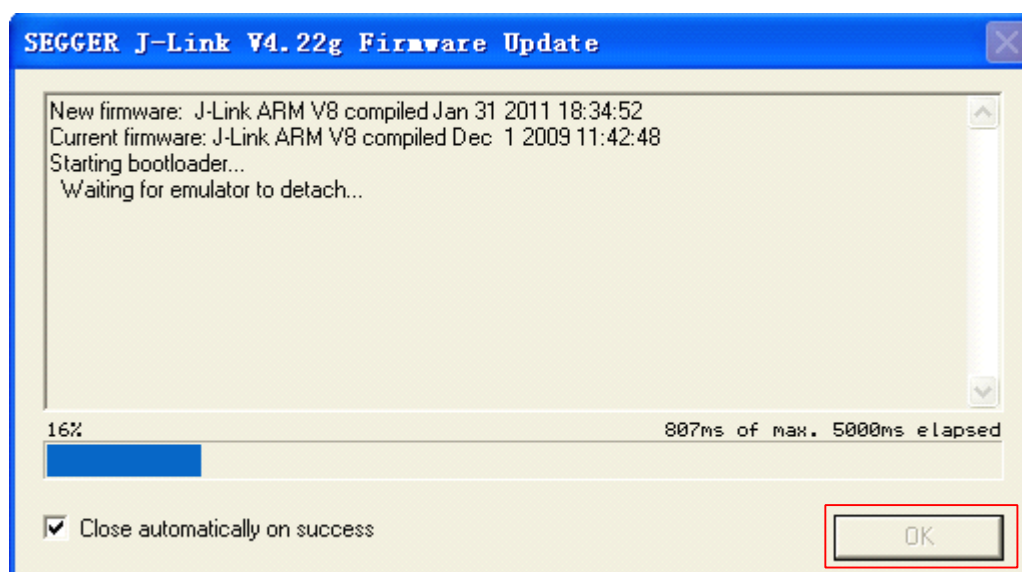


6-2-3 更新 Jlink 固件

本次提供的 JLINK 驱动是 V4.22 版本。这是一个较高的版本。V4.22 的驱动有可能需要更新 JLINK 内部的固件。如果需要更新固件，会出现下面的对话框，我们只需要点击“是”，就会自动更新固件。如果没有出现固件更新的对话框，本小节可略过直接跳到下一小节。

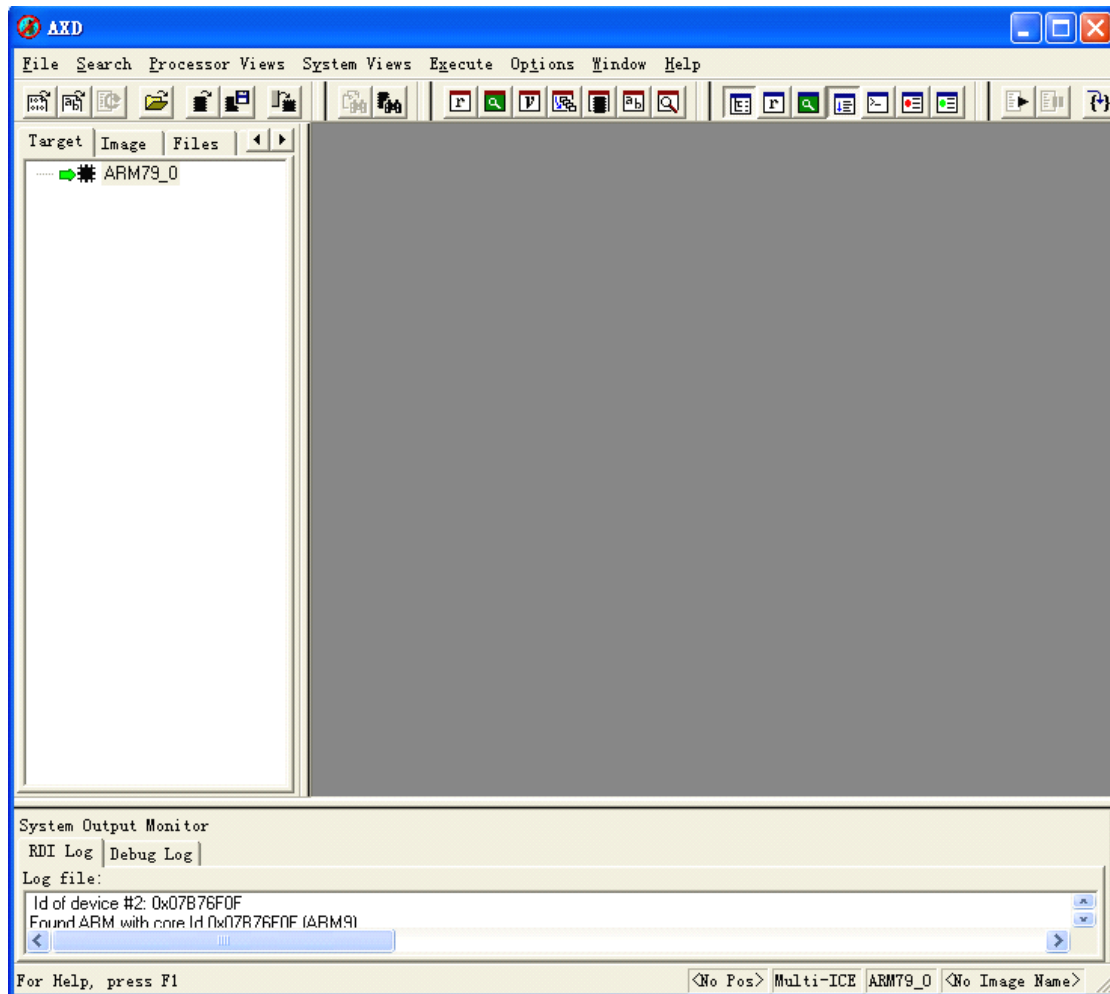


下图是升级固件的过程。升级完成后点击”OK”即可。



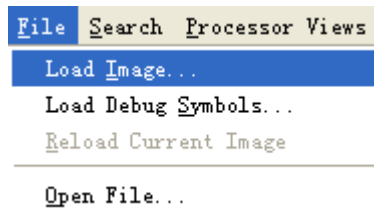
6-2-4 检测目标 CPU

每次选择 JLinkRDI.dll 目标后，都会检测到 CPU 的型号。由于 JLink 的原因，只能识别出 CPU 型号为 ARM79_0，而非我们熟知的 ARM11JZF-S。

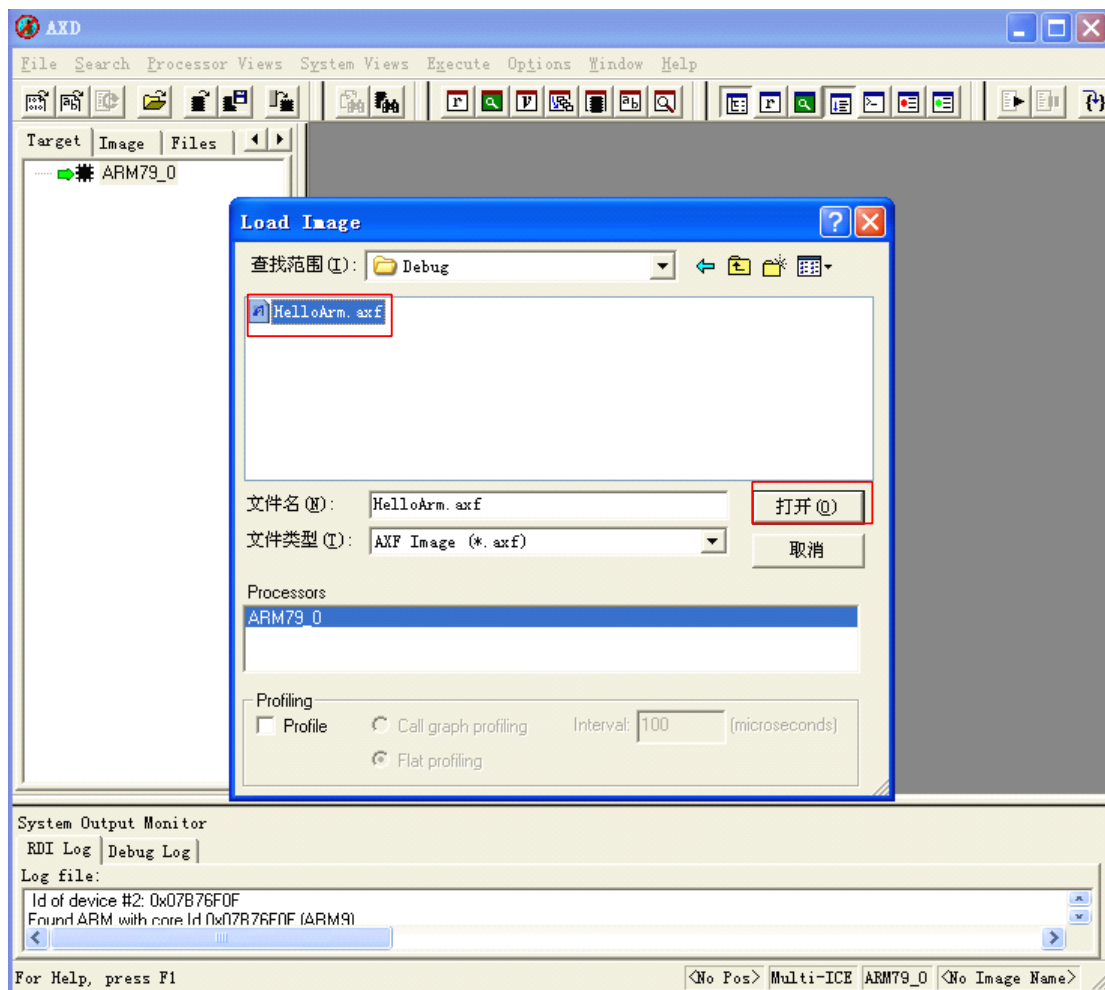




6-3 仿真一下，小试牛刀

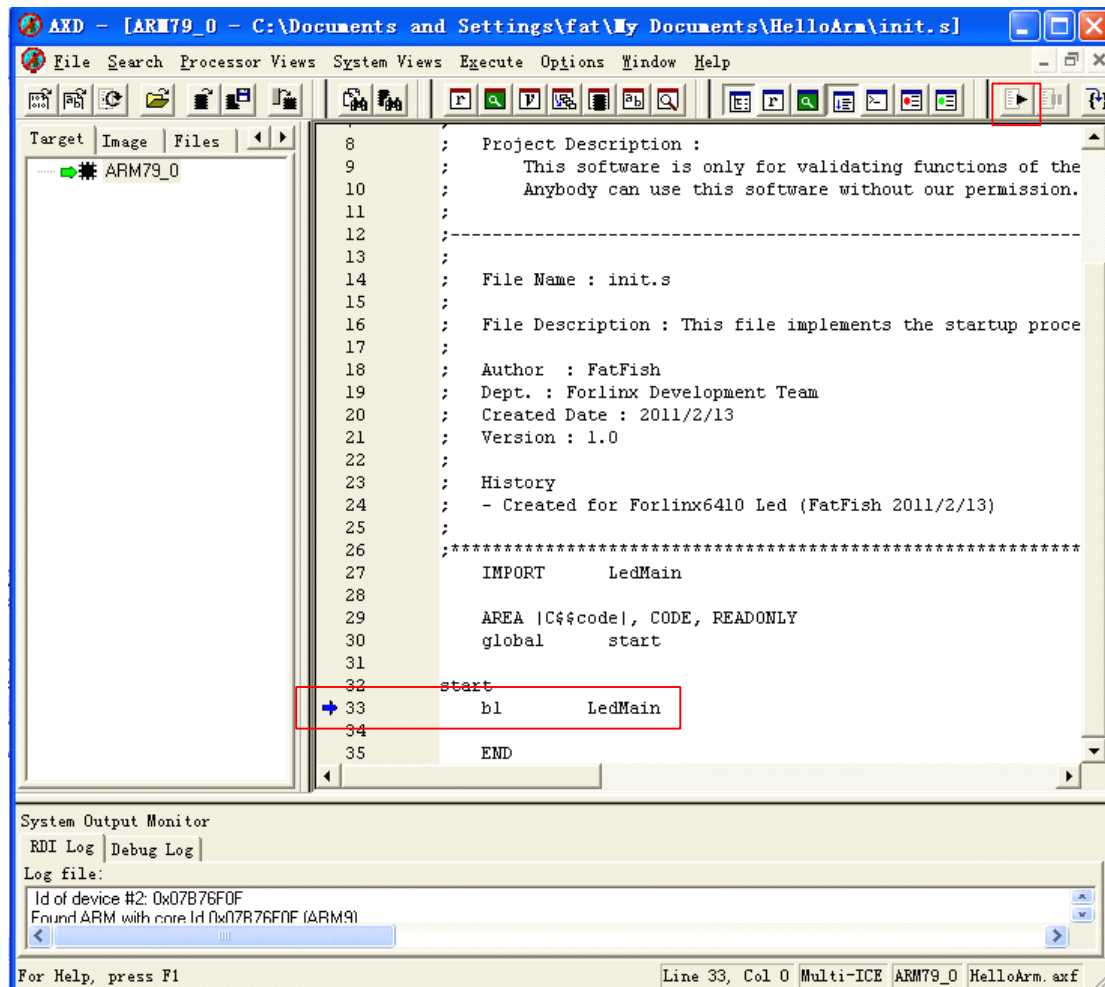
打开”File->Load Image...”。



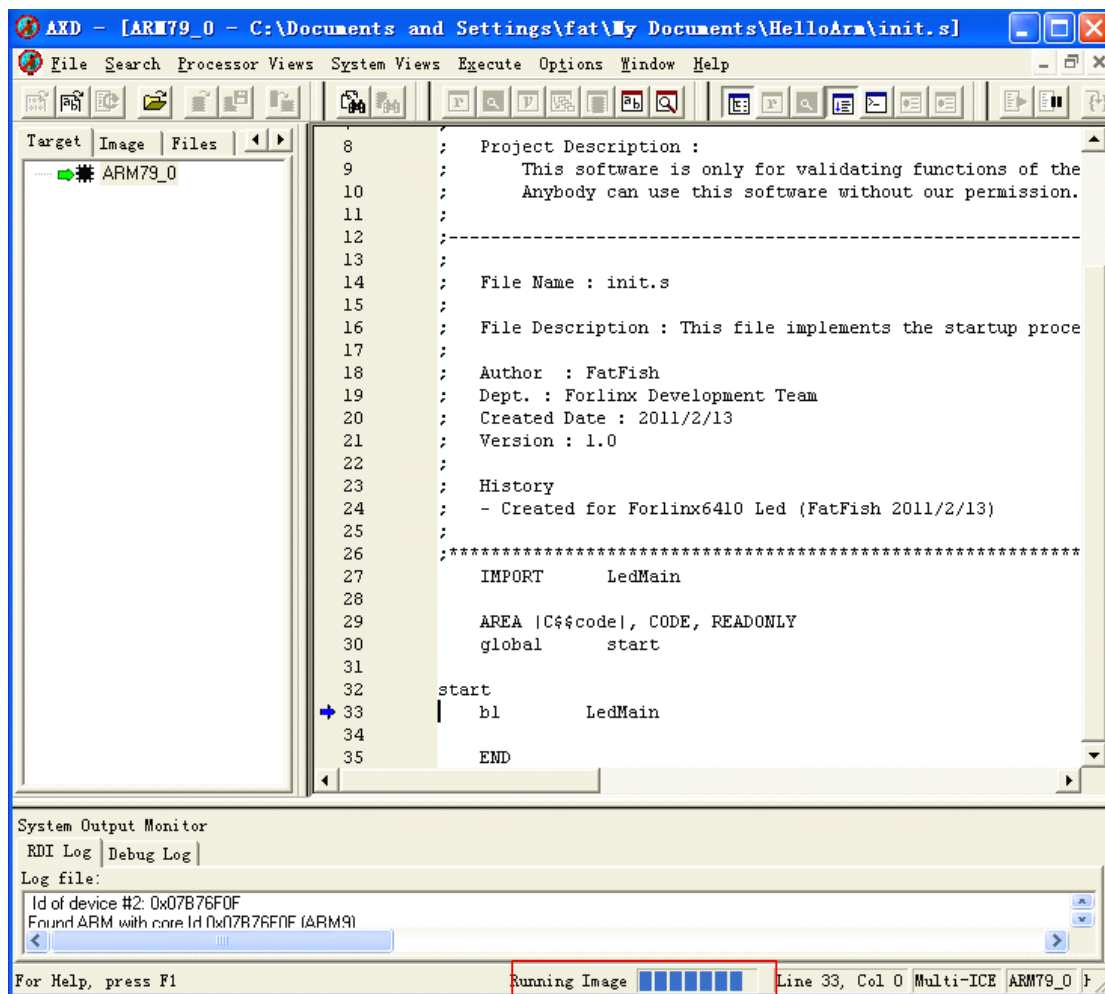
选择 HelloArm.axf，点击打开。



打开 axf 文件， 自动指向了第一条执行的语句。点击 ，开始全速仿真。



全速运行时，在 AXD 最下方会显示”Running Image”。

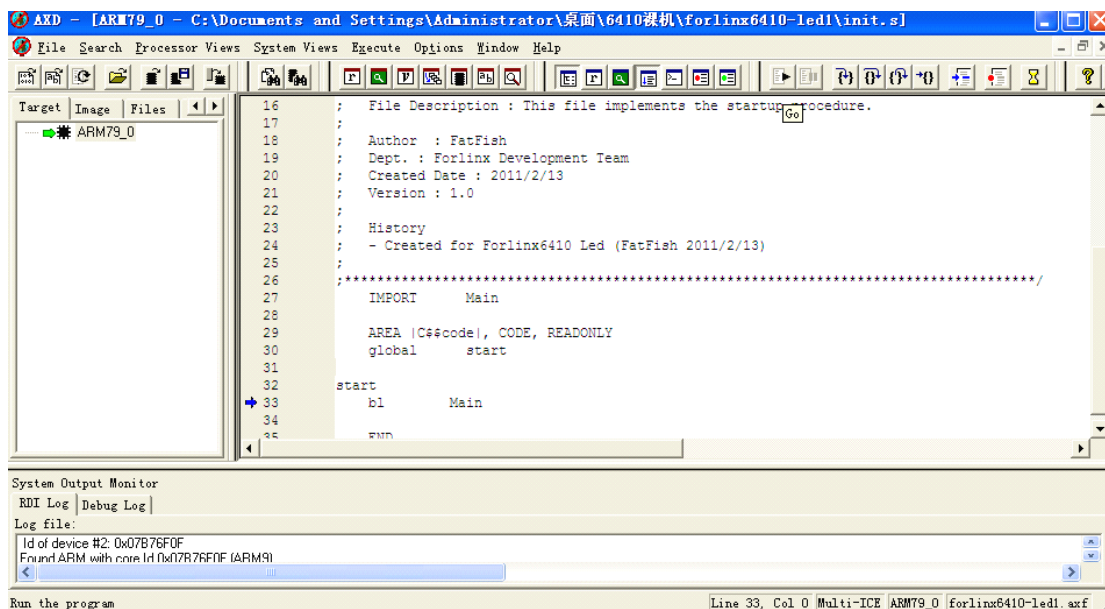


到现在，我们的 OK6410 已经很畅快的”裸奔”了。

第七章 Jlink



7-1 全速运行

点击 AXD 的“GO”，AXD 会在开发板上全速仿真程序。如图：

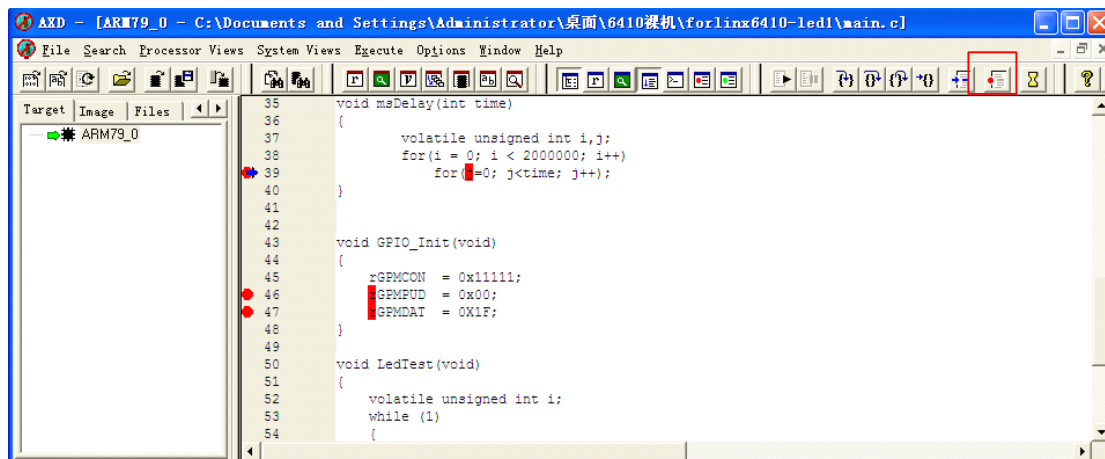


7-2 断点调试


断点调试，是一个非常重要的调试手段，可以让开发板直接运行程序到指定的断点后停下来。

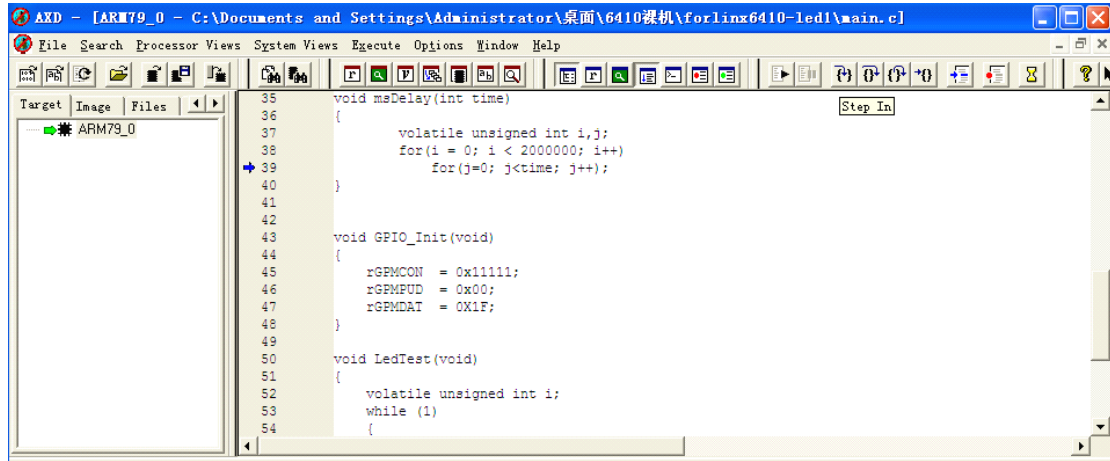
设置方法：将光标点至要设置断点的代码行，然后点击即可将该行设置为断点。设置成功后，在代码行前面会显示一个作为提示。当然，也可以同时设置一个或者多个断点，每次点击全速运行后，都会依照代码，执行到下一个断点。

图中设置了很多断点，可以参考一下：



7-3 单步调试

单步调试也是重要的一个调试手段。简单的概括一下，就是每次只会执行一行代码，就停下来了。单击”Step In”执行一次单步调试。图标为。



第八章 S3C6410 GPIO

GPIO (General Purpose I/O ports) 即通用输入/输出口。我们在实际应用中, 不管是接 LCD、接键盘, 控制流水灯等等应用, 都离不开对 I/O 的操作。可以说, GPIO 的操作是所有硬件操作的基础。

S3C6410 包含了 187 个多功能 输入/输出端口管脚, 共分十七组, 分别为: GPA、GPB、...、GPJ。我们可以通过寄存器来操作这些 IO。

- (1) 可以控制 127 个外部中断。
- (2) 有 187 个多功能输入/输出端口。
- (3) 控制管脚的睡眠模式状态, 除了 GPK、GPL、GPM、和 GPN 管脚以外。

先说说 GPIO 作为普通输入输出引脚时有几种状态:

- (1) 输出高电平。也就是引脚作为输出电压脚, 输出高电平。何为 S3C6410 的高电平? 简单的说就是 3.3V 电压。
- (2) 输出低电平。也就是引脚作为输出电压脚, 输出低电平。何为 S3C6410 的低电平? 简单的说就是 0V 电压。
- (3) 输入状态。这时, 引脚高低电平完全外界对引脚的输入电压决定。
- (4) 高阻态。引脚什么都不接, 或者说是悬空。

那么这么多状态, 和我们的学习以及开发有什么关系呢? 或者说, 在什么情况下, 我们可以使用这些功能?

简单的归纳一下, 我们不难发现, GPIO 作为普通的输入输出引脚使用时, 主要使用输入、输出的功能。高阻态呢? 不连接设备的 GPIO 就没法操作设备了。所以在开始的学习中, 主要考虑 GPIO 的输入和输出功能。

接下来, 再看几个关于 GPIO 的寄存器。

- 1. GPxCON 控制改组引脚功能的寄存器, 这组寄存器是可读取数值、也可写入数值的寄存器。一般一个 io 口会有两个或两个以上的功能, 如果 CPU 如何知道 IO 口执行什么样的功能, 就靠这组寄存器来设置。
- 2. GPxDAT 如果作为普通输入输出引脚, 这组寄存器是高低电平的状态寄存器。这组寄存器是可读取数值、也可写入数值的寄存器。某一位数值为”1”时, 这一位对应的 IO 口的电平为 3.3V (高电平); 某一位数值为”0”时, 这一位对应的 IO 口的电平为 0V (低电平)。
- 3. GPxPUD 上拉电阻寄存器。这组寄存器是可读取数值、也可写入数值的寄存器。这组寄存器控制 IO 内部上拉电阻的。由于 IO 口有时需要内部上拉来维持电压, 所以提供了上拉电阻和上拉电阻寄存器。

第九章 利用 GPIO 控制 OK6410 的 LED

9-1 实验目的

熟悉RVDS2.2开发环境。
掌握S3C6410内部相关寄存器的操作方法，最终实现对外部设备的控制。
熟悉在ARM 裸机环境下的C语言编程。

9-2 实验设备

OK6410开发板、PC 机、JLINK调试器。

9-3 实验内容

建立RVDS开发环境。
编程实现对开发板上发光二极管LED 的跑马灯控制。

9-4 实验原理

从电路图上我们可以看到，发光二极管LED 的一端连接到了ARM 的GPIO，另一端经过一个限流电阻接电源VCC3。当GPIO 口为低电平时，LED 两端产生电压降，这时LED 有电流通过并发光。反之当GPIO 为高电平时，LED 将熄灭。注意亮灭之间要有一定的延时，以便人眼能够区分出来。

实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPMCON	0x7F008820	R/W	Port M Configuration Register	0x00222222
GPMDAT	0x7F008824	R/W	Port M Data Register	Undefined
GPMPUD	0x7F008828	R/W	Port M Pull-up/down Register	0x000002AA

端口配置寄存器

GPMCON	Bit	Description		Initial State
GPM0	[3:0]	0000 = Input 0010 = Host I/F CSn 0100 = Reserved 0110 = CE_CF[0]	0001 = Output 0011 = Ext. Interrupt[23] 0101 = Reserved 0111 = Reserved	0010
GPM1	[7:4]	0000 = Input 0010 = Host I/F CSn_main 0100 = Reserved 0110 = CE_CF[1]	0001 = Output 0011 = Ext. Interrupt [24] 0101 = Reserved 0111 = Reserved	0010
GPM2	[11:8]	0000 = Input 0010 = Host I/F CSn_sub 0100 = Host I/F MDP_VSYNC 0110 = IORD_CF	0001 = Output 0011 = Ext. Interrupt [25] 0101 = Reserved 0111 = Reserved	0010
GPM3	[15:12]	0000 = Input 0010 = Host I/F WEn 0100 = Reserved 0110 = IOWR_CF	0001 = Output 0011 = Ext. Interrupt [26] 0101 = Reserved 0111 = Reserved	0010

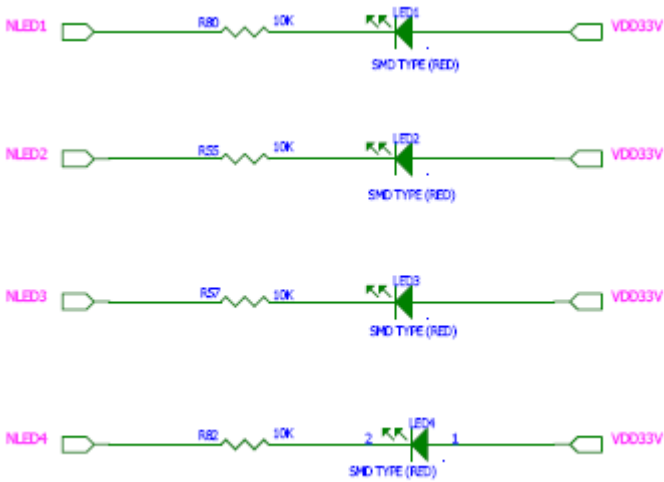
端口数据寄存器

GPMDAT	Bit	Description
GPM[5:0]	[5:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

端口上拉电路使能寄存器

GPLPUD	Bit	Description
GPM[n]	[2n+1:2n] n = 0~5	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

9-5 实验电路



9-6 实验程序

main.c

```
#define rGPMCON      (*(volatile unsigned *) (0x7F008820))
#define rGPMDAT      (*(volatile unsigned *) (0x7F008824))
#define rGPMPUD      (*(volatile unsigned *) (0x7F008828))
```

```
void msDelay(int time)
{
    volatile unsigned int i,j;
    for(i = 0; i < 2000000; i++)
        for(j=0; j<time; j++);
}
```

```

void GPIO_Init(void)
{
    rGPMCON  = 0x11111;
    rGPMPUD  = 0x00;
    rGPMDAT  = 0X1F;
}

void LedTest(void)
{
    volatile unsigned int i;
    while (1)
    {
        for(i=0; i<4; i++)
        {
            rGPMDAT  = ~(1<<i);
            msDelay(10);
        }
    }
}

void LedMain(void)
{
    GPIO_Init();
    LedTest();
}

```

init.s

```

IMPORT      Main

AREA |C$$code|, CODE, READONLY
global      start

start
    bl      Main

END

```

9-7 实验步骤

1. 准备好实验环境，将JLINK连接好。
2. 将串口线的一端插在PC 的串口上，另一端插在开发板的‘COM0’上。打开DNW.EXE 软件，给开发板上电，使 Bootloader（eboot 和 uboot 均可） 停在菜单处（在系统引导时按

空格键)。

3. 打开软件 ‘Code Warrior for RVDS’, 新建工程 ‘forlinux6410-key1.mcp’, 并添加两个程序文件 ‘main.c’ 和 ‘init.s’ (汇编文件)。
4. 对工程文件进行相应设置。
5. 编译该工程, 成功后将生成映像文件 ‘forlinux6410-key1.axf’。打开AXD, 装载映像文件 ‘forlinux6410-key1.axf’。
6. 运行程序, 观察结果。

9-8 实验结果

四个发光二极管 LED 将轮流闪烁, 最终实现流水灯效果。

第十章 GPIO 控制 OK6410 的蜂鸣器

10-1实验目的

熟悉RVDS2.2开发环境。
掌握S3C6410内部相关寄存器的操作方法，最终实现对外部设备的控制。
熟悉在ARM 裸机环境下的C语言编程。

10-2实验设备

OK6410开发板、PC 机、JLINK调试器。

10-3实验内容

建立RVDS开发环境。
编程实现对开发板上有源蜂鸣器控制，使蜂鸣器鸣叫。

10-4实验原理

蜂鸣器是通过I/O 口GPF15来间接控制的，为了增加驱动能力，增加了三级管驱动电路。当三极管的基极（B）为高电平即GPF15 为高电平时，蜂鸣器会鸣叫，反之则不响。通过设置两者之间的时间（即改变频率）可以使蜂鸣器发出不同的声音，甚至播放乐曲。

实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPFCON	0x7F0080A0	R/W	Port F Configuration Register	0x00
GPFDAT	0x7F0080A4	R/W	Port F Data Register	Undefined
GPFPUD	0x7F0080A8	R/W	Port F Pull-up/down Register	0x55555555
GPFCONSLP	0x7F0080AC	R/W	Port F Sleep mode Configuration Register	0x0
GPFPUDSLP	0x7F0080B0	R/W	Port F Sleep mode Pull-up/down Register	0x0

端口配置寄存器

GPF15	[31:30]	00 = Input 10 = PWM TOUT[1]	01 = Output 11 = Reserved	00
-------	---------	--------------------------------	------------------------------	----

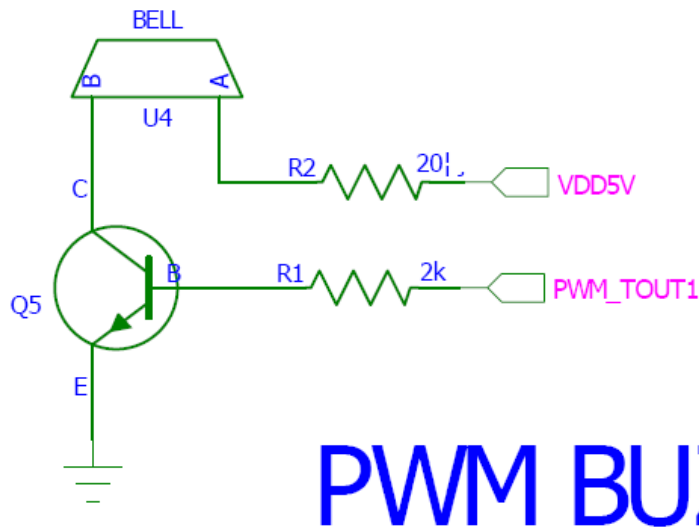
端口数据寄存器

GPFDAT	Bit	Description
GPF[15:0]	[15:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

端口上拉电路使能寄存器

GPFPUD	Bit	Description
GPF[n]	[2n+1:2n] n = 0~15	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

10-5实验电路



10-6实验程序

main.c

```
#define rGPFCON      (*(volatile unsigned*)(0x7F0080A0))
#define rGPFDAT      (*(volatile unsigned*)(0x7F0080A4))
#define rGPFPUDD      (*(volatile unsigned*)(0x7F0080A8))
```

```
void msDelay(int time)
{
    volatile unsigned int i,j;
    for(i = 0; i < 200000; i++)
        for(j=0; j<time; j++);
}
```

```
void GPIO_Init(void)
{
    rGPFCON  = rGPFCON & ~(0x1 << (31));
    rGPFCON  = rGPFCON | (0x1 << (2*15));
    rGPFPUDD = rGPFPUDD & ~(0x3 << (2*15));
    rGPFDAT  = rGPFDAT | 0x8000;
}
```

```
void BeepOff(void)
{
    rGPFDAT = rGPFDAT | 0x8000;
}
```

```
void BeepOn(void)
{
    rGPFDAT = rGPFDAT & 0x7fff;
}
```

```
void BeepTest(void)
{
    while (1)
    {
        BeepOn();
        msDelay(1);

        BeepOff();
        msDelay(1);
    }
}
```

```
void Main(void)
{
    GPIO_Init();
    BeepTest();
}
```

init.s

```
IMPORT      Main

AREA |C$$code|, CODE, READONLY
global      start

start
    bl      Main

END
```

10-7 实验步骤

1. 准备好实验环境，将JLINK连接好。
2. 将串口线的一端插在PC 的串口上，另一端插在开发板的‘COM0’上。打开DNW.EXE 软件，给开发板上电，使 Bootloader（eboot 和 uboot 均可） 停在菜单处（在系统引导时按

空格键)。

3. 打开软件 ‘Code Warrior for RVDS’, 新建工程 ‘forlinux6410-beep1.mcp’, 并添加两个程序文件 ‘main.c’ 和 ‘init.s’ (汇编文件)。
4. 对工程文件进行相应设置。
5. 编译该工程, 成功后将生成映像文件 ‘forlinux6410-beep1.axf’。打开AXD, 装载映像文件 ‘forlinux6410-beep1.axf’。
6. 运行程序, 观察结果。

10-8实验结果

蜂鸣器在固定频率下鸣叫。

第十一章 GPIO 检测 OK6410 的按键操作

11-1实验目的

熟悉RVDS2.2开发环境。
掌握S3C6410内部相关寄存器的操作方法，最终实现对外部设备的控制。
熟悉在ARM 裸机环境下的C语言编程。

11-2实验设备

OK6410开发板、PC 机、JLINK调试器。

11-3实验内容

建立RVDS开发环境。
编程实现对开发板上有源蜂鸣器控制，使蜂鸣器鸣叫。

11-4实验原理

蜂鸣器是通过I/O 口GPF15来间接控制的，为了增加驱动能力，增加了三级管驱动电路。当三极管的基极（B）为高电平即GPF15 为高电平时，蜂鸣器会鸣叫，反之则不响。通过设置两者之间的时间（即改变频率）可以使蜂鸣器发出不同的声音，甚至播放乐曲。

蜂鸣器相关实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPFCON	0x7F0080A0	R/W	Port F Configuration Register	0x00
GPFDAT	0x7F0080A4	R/W	Port F Data Register	Undefined
GPFPUD	0x7F0080A8	R/W	Port F Pull-up/down Register	0x55555555
GPFCONSLP	0x7F0080AC	R/W	Port F Sleep mode Configuration Register	0x0
GPFPUDSLP	0x7F0080B0	R/W	Port F Sleep mode Pull-up/down Register	0x0

蜂鸣器相关端口配置寄存器

GPF15	[31:30]	00 = Input 10 = PWM TOUT[1]	01 = Output 11 = Reserved	00
-------	---------	--------------------------------	------------------------------	----

蜂鸣器相关端口数据寄存器

GPFDAT	Bit	Description
GPF[15:0]	[15:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

蜂鸣器相关端口上拉电路使能寄存器

GPFPUD	Bit	Description
GPF[n]	[2n+1:2n] n = 0~15	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

按键相关实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPNCON	0x7F008830	R/W	Port N Configuration Register	0x00
GPNDAT	0x7F008834	R/W	Port N Data Register	Undefined
GPNPUD	0x7F008838	R/W	Port N Pull-up/down Register	0x55555555

按键相关端口配置寄存器

GPNCON	Bit	Description		Initial State
GPN0	[1:0]	00 = Input 10 = Ext. Interrupt[0]	01 = Output 11 = Key pad ROW[0]	00
GPN1	[3:2]	00 = Input 10 = Ext. Interrupt[1]	01 = Output 11 = Key pad ROW[1]	00
GPN2	[5:4]	00 = Input 10 = Ext. Interrupt[2]	01 = Output 11 = Key pad ROW[2]	00
GPN3	[7:6]	00 = Input 10 = Ext. Interrupt[3]	01 = Output 11 = Key pad ROW[3]	00
GPN4	[9:8]	00 = Input 10 = Ext. Interrupt[4]	01 = Output 11 = Key pad ROW[4]	00
GPN5	[11:10]	00 = Input 10 = Ext. Interrupt[5]	01 = Output 11 = Key pad ROW[5]	00

按键相关端口数据寄存器

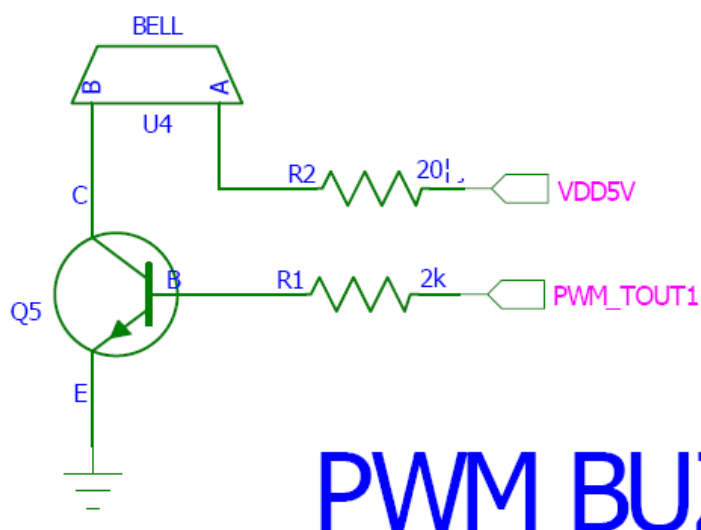
GPNDAT	Bit	Description
GPN[15:0]	[15:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

按键相关端口上拉电路使能寄存器

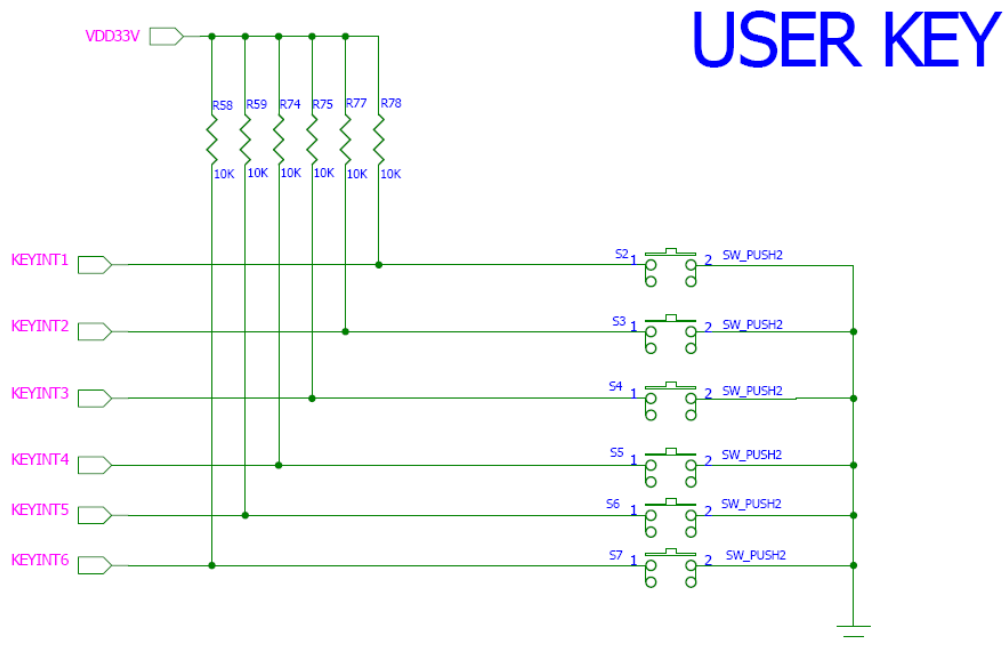
GPNPUD	Bit	Description
GPN[n]	[2n+1:2n] n = 0~15	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

11-5实验电路

蜂鸣器相关电路



按键相关电路



11-6实验程序

main.c

```
#define rGPFCON      (*(volatile unsigned *) (0x7F0080A0))
#define rGPFDAT      (*(volatile unsigned *) (0x7F0080A4))
#define rGPFPUDD      (*(volatile unsigned *) (0x7F0080A8))
```

```
#define rGPNCON      (*(volatile unsigned *)(0x7F008830))
#define rGPNDAT      (*(volatile unsigned *)(0x7F008834))
#define rGPNPUD      (*(volatile unsigned *)(0x7F008838))
```

```
void msDelay(int time)
{
    volatile unsigned int i,j;
    for(i = 0; i < 200000; i++)
        for(j=0; j<time; j++);
}
```

```
void GPIO_Init(void)
{
    /*-----init Beep-----*/
    rGPFCON  = rGPFCON & ~(0x1 << (31));
    rGPFCON  = rGPFCON | (0x1 << (2*15));
    rGPFPUUD = rGPFPUUD & ~(0x3 << (2*15));
    rGPFDAT  = rGPFDAT | 0x8000;

    /*-----init Kye-----*/
    rGPNCON = rGPNCON & (0xffffffff << (2*6));
}
```

```
void BeepOff(void)
{
    rGPFDAT  = rGPFDAT | 0x8000;
}
```

```
void BeepOn(void)
{
    rGPFDAT  = rGPFDAT & 0x7fff;
}
```

```
unsigned int KeyScan(void)
{
    if ((rGPNDAT & 0x3f) != 0x3f)
        return 1;
    else
        return 0;
}
```

```

void KeyTest(void)
{
    unsigned int KeyStatus;
    while (1)
    {
        KeyStatus = KeyScan();
        if (KeyStatus == 1)
        {
            BeepOn();
            msDelay(10);
            BeepOff();
        }
        else
            BeepOff();
    }
}

```

```

void Main(void)
{
    GPIO_Init();
    BeepOff();
    KeyTest();
}

```

init.s

```

        IMPORT      Main
        AREA |C$$code|, CODE, READONLY
        global      start

start
        bl          Main
        END

```

11-7 实验步骤

1. 准备好实验环境，将JLINK连接好。
2. 将串口线的一端插在PC 的串口上，另一端插在开发板的‘COM0’上。打开DNW.EXE 软件，给开发板上电，使 Bootloader（eboot 和 uboot 均可） 停在菜单处（在系统引导时按空格键）。
3. 打开软件‘Code Warrior for RVDS’，新建工程‘forlinux6410-key1.mcp’，并添加两

个程序文件 ‘main.c’ 和 ‘init.s’（汇编文件）。

4. 对工程文件进行相应设置。

5. 编译该工程，成功后将生成映像文件 ‘forlinux6410-key1.axf’。打开AXD，装载映像文件 ‘forlinux6410-key1.axf’。

6. 运行程序，观察结果。

11-8实验结果

蜂鸣器在固定频率下鸣叫。

第十二章 利用定时器制作精确延时来控制 OK6410 的 LED

12-1实验目的

- 熟悉RVDS2.2开发环境。
- 掌握S3C6410内部相关寄存器的操作方法，最终实现对外部设备的控制。
- 熟悉在ARM 裸机环境下的C语言编程。
- 熟悉ARMv6的VIC控制以及S3C6410的timer控制器。

12-2实验设备

OK6410开发板、PC 机、JLINK调试器。

12-3实验内容

- 建立RVDS开发环境。
- 利用定时器实现对开发板上发光二极管LED 的跑马灯控制。

12-4实验原理

从电路图上我们可以看到，发光二极管LED 的一端连接到了ARM 的GPIO，另一端经过一个限流电阻接电源VCC3。当GPIO 口为低电平时，LED 两端产生电压降，这时LED 有电流通过并发光。反之当GPIO 为高电平时，LED 将熄灭。注意亮灭之间要有一定的延时，以便人眼能够区分出来。

实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPMCON	0x7F008820	R/W	Port M Configuration Register	0x00222222
GPMDAT	0x7F008824	R/W	Port M Data Register	Undefined
GPMPUD	0x7F008828	R/W	Port M Pull-up/down Register	0x000002AA

端口配置寄存器

GPMCON	Bit	Description		Initial State
GPM0	[3:0]	0000 = Input 0010 = Host I/F CSn 0100 = Reserved 0110 = CE_CF[0]	0001 = Output 0011 = Ext. Interrupt[23] 0101 = Reserved 0111 = Reserved	0010
GPM1	[7:4]	0000 = Input 0010 = Host I/F CSn_main 0100 = Reserved 0110 = CE_CF[1]	0001 = Output 0011 = Ext. Interrupt [24] 0101 = Reserved 0111 = Reserved	0010
GPM2	[11:8]	0000 = Input 0010 = Host I/F CSn_sub 0100 = Host I/F MDP_VSYNC 0110 = IORD_CF	0001 = Output 0011 = Ext. Interrupt [25] 0101 = Reserved 0111 = Reserved	0010
GPM3	[15:12]	0000 = Input 0010 = Host I/F WEn 0100 = Reserved 0110 = IOWR_CF	0001 = Output 0011 = Ext. Interrupt [26] 0101 = Reserved 0111 = Reserved	0010

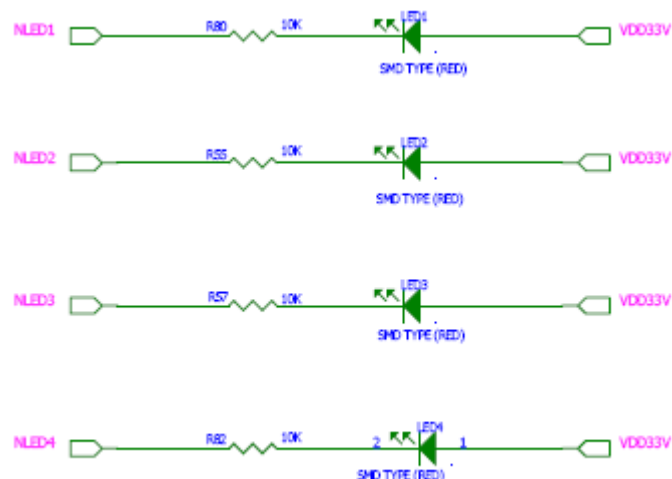
端口数据寄存器

GPMDAT	Bit	Description
GPM[5:0]	[5:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

端口上拉电路使能寄存器

GPLPUD	Bit	Description
GPM[n]	[2n+1:2n] n = 0~5	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

12-5 实验电路



12-6 实验程序

main.c

```
#define rGPMCON (*(volatile unsigned*)(0x7F008820))
```

```
#define rGPMDAT      (*(volatile unsigned *)(0x7F008824))
#define rGMPUD       (*(volatile unsigned *)(0x7F008828))
```

```
#define PCLK   66000000   //for S3C6410 66MHZ
#define HCLK 133000000   //for S3C6410 133MHZ
```

```
#define rTCFG0      (*(volatile unsigned *)(0x7F006000))
#define rTCFG1      (*(volatile unsigned *)(0x7F006004))
#define rTCON       (*(volatile unsigned *)(0x7F006008))
#define rTCNTB0     (*(volatile unsigned *)(0x7F00600C))
#define rTCMPB0     (*(volatile unsigned *)(0x7F006010))
#define rTCNT0      (*(volatile unsigned *)(0x7F006014))
#define rTCNTB1     (*(volatile unsigned *)(0x7F006018))
#define rTCMPB1     (*(volatile unsigned *)(0x7F00601c))
#define rTCNT01     (*(volatile unsigned *)(0x7F006020))
#define rTCNTB2     (*(volatile unsigned *)(0x7F006024))
#define rTCNT02     (*(volatile unsigned *)(0x7F00602c))
#define rTCNTB3     (*(volatile unsigned *)(0x7F006030))
#define rTCNT03     (*(volatile unsigned *)(0x7F006038))
#define rTCNTB4     (*(volatile unsigned *)(0x7F00603c))
#define rTCNT04     (*(volatile unsigned *)(0x7F006040))
#define rTINT_CSTAT (*(volatile unsigned *)(0x7F006044))
```

```
void uDelay(int usec)
{
    unsigned int val = (PCLK)/1000000-1;
    rTCFG0 &= ~(0xff<<8);
    rTCFG0 |= 0<<8;
    rTCFG1 &= ~(0xf<<8);
    rTCFG1 |= 0<<8;
    rTCNTB2 = val;
    rTCON &= ~(0xf<<12);
    rTCON |= 0xb<<12;
    rTCON &= ~(2<<12);
    while(usec--) {
        while(rTCNT02>=val>>1);
        while(rTCNT02<val>>1);
    };
}
```

```
void msDelay(int time)
{
    volatile unsigned int i,j;
    for(i = 0; i < 2000000; i++)
```

```
        for(j=0; j<time; j++);
    }
}
```

```
void GPIO_Init(void)
{
    rGPMCON  = 0x11111;
    rGPMPUD  = 0x00;
    rGPMDAT  = 0X1F;
}
}
```

```
void LedTest(void)
{
    volatile unsigned int i,j;
    while (1)
    {
        for(i=0; i<4; i++)
        {
            rGPMDAT  = ~(1<<i);
            for (j=0; j<1000; j++)
                uDelay(1000);
        }
    }
}

void LedMain(void)
{
    GPIO_Init();
    LedTest();
}
}
```

init.s

```
        IMPORT      Main
    .
    AREA |C$$code|, CODE, READONLY
    global          start

start
    bl      Main
    .
    END
```

12-7 实验步骤

1. 准备好实验环境，将JLINK连接好。

2. 将串口线的一端插在PC 的串口上，另一端插在开发板的‘COM0’上。打开DNW.EXE 软件，给开发板上电，使 Bootloader（eboot 和 uboot 均可） 停在菜单处（在系统引导时按空格键）。
3. 打开软件‘Code Warrior for RVDS’，新建工程‘forlinux6410-key2.mcp’，并添加两个程序文件‘main.c’和‘init.s’（汇编文件）。
4. 对工程文件进行相应设置。
5. 编译该工程，成功后将生成映像文件‘forlinux6410-key2.axf’。打开AXD，装载映像文件‘forlinux6410-key2.axf’。
6. 运行程序，观察结果。

12-8实验结果

四个发光二极管 LED 将轮流闪烁，最终实现流水灯效果。

第十三章 利用定时器精确控制 OK6410 的蜂鸣器频率

13-1实验目的

熟悉RVDS2.2开发环境。
掌握S3C6410内部相关寄存器的操作方法，最终实现对外部设备的控制。
熟悉在ARM 裸机环境下的C语言编程。

13-2实验设备

OK6410开发板、PC 机、JLINK调试器。

13-3实验内容

建立RVDS开发环境。
编程实现对开发板上有源蜂鸣器控制，使蜂鸣器鸣叫。

13-4实验原理

蜂鸣器是通过I/O 口GPF15来间接控制的，为了增加驱动能力，增加了三级管驱动电路。当三极管的基极（B）为高电平即GPF15 为高电平时，蜂鸣器会鸣叫，反之则不响。通过设置两者之间的时间（即改变频率）可以使蜂鸣器发出不同的声音，甚至播放乐曲。

实验相应寄存器

Register	Address	R/W	Description	Reset Value
GPFCON	0x7F0080A0	R/W	Port F Configuration Register	0x00
GPFDAT	0x7F0080A4	R/W	Port F Data Register	Undefined
GPFPUD	0x7F0080A8	R/W	Port F Pull-up/down Register	0x55555555
GPFCONSLP	0x7F0080AC	R/W	Port F Sleep mode Configuration Register	0x0
GPFPUDSLP	0x7F0080B0	R/W	Port F Sleep mode Pull-up/down Register	0x0

端口配置寄存器

GPF15	[31:30]	00 = Input 10 = PWM TOUT[1]	01 = Output 11 = Reserved	00
-------	---------	--------------------------------	------------------------------	----

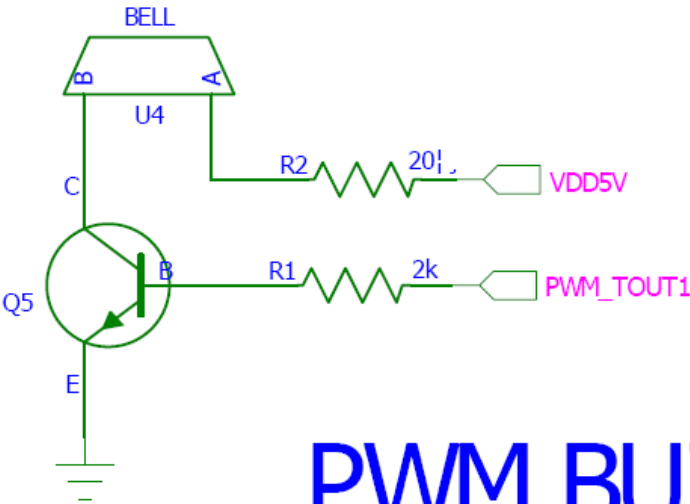
端口数据寄存器

GPFDAT	Bit	Description
GPF[15:0]	[15:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

端口上拉电路使能寄存器

GPFPUD	Bit	Description
GPF[n]	[2n+1:2n] n = 0~15	00 = pull-up/down disabled 01 = pull-down enabled 10 = pull-up enabled 11 = Reserved.

13-5实验电路



PWM BUZZER

13-6实验程序

main.c

```
#define rGPFCON      (*(volatile unsigned*)(0x7F0080A0))
#define rGPFDAT      (*(volatile unsigned*)(0x7F0080A4))
#define rGPFPUD      (*(volatile unsigned*)(0x7F0080A8))
```

```
void msDelay(int time)
{
    volatile unsigned int i,j;
    for(i = 0; i < 200000; i++)
        for(j=0; j<time; j++);
}
```

```
void GPIO_Init(void)
{
    rGPFCON = rGPFCON & ~(0x1 << (31));
    rGPFCON = rGPFCON | (0x1 << (2*15));
```



```

    rGPFPU0 = rGPFPU0 & ~(0x3 << (2*15));
    rGPFDAT = rGPFDAT | 0x8000;
}

```

```

void BeepOff(void)
{
    rGPFDAT = rGPFDAT | 0x8000;
}

```

```

void BeepOn(void)
{
    rGPFDAT = rGPFDAT & 0x7fff;
}

```

```

void BeepTest(void)
{
    while (1)
    {
        BeepOn();
        msDelay(1);

        BeepOff();
        msDelay(1);
    }
}

```

```

void Main(void)
{
    GPIO_Init();
    BeepTest();
}

```

init.s

```

    IMPORT      Main

    AREA |C$$code|, CODE, READONLY
    global      start

start
    bl      Main

    END

```

13-7实验步骤

1. 准备好实验环境，将JLINK连接好。
2. 将串口线的一端插在PC 的串口上，另一端插在开发板的‘COM0’上。打开DNW.EXE 软件，给开发板上电，使 Bootloader（eboot 和 uboot 均可） 停在菜单处（在系统引导时按空格键）。
3. 打开软件‘Code Warrior for RVDS’，新建工程‘forlinux6410-beep2.mcp’，并添加两个程序文件‘main.c’和‘init.s’（汇编文件）。
4. 对工程文件进行相应设置。
5. 编译该工程，成功后将生成映像文件‘forlinux6410-beep2.axf’。打开AXD，装载映像文件‘forlinux6410-beep2.axf’。
6. 运行程序，观察结果。

13-8实验结果

蜂鸣器在固定频率下鸣叫。

第十四章 总结

本手册目前是第一版本，着重从开发环境以及 gpio、定时器等 s3c6410 基础资源。手册中引用到了 s3c6410 用户手册、s3c6410 App Note 等官方资源。后续版本会讲解趋于复杂的 cpu 内部模块。

手册由飞凌倾心制作。在编写过程当中难免纰漏，还望各位指出错误。飞凌会尽心尽力为客户的腾飞做好每一个细节。