# IAR 下 9200EK 范例使用设置

——By Team Mcuzone

IAR 提供了几个 9200 的范例，是基于 9200EK 板的，9200EK 板上用的 FLASH 好像是 AT49BV1604/1614/6416/，但是 16XX 已经比较难买，而且价格很高，所以我们的 9200 核心板上采用了引脚兼容的 AT49BV163，如果不加修改，在把程序 download 到 FLASH 的时候 IAR 会提示没有找到 AT49BVXXXX 器件，这样子我们就没有办法使用这些范例了。

研究了一下，其实可以通过重新编译生成一个 d79 文件，来代替 IAR 默认使用的 d79 文件。这样子就不会出现那个错误，程序也可以正确下载。由于 AT49BV163 可以直接替换 1614，所以，可以直接使用那些范例，不过最好把一些 ID 修改一下。

已经有网友提供了两套 mac 文件和 d79 文件，大家可以直接使用，如果觉得自己可以写出更好的 mac 和 d79 也欢迎尝试，如果方便的话，可以和大家分享。

下面大致讲一下一些简要设置。

首先，打开 Project 的 Options 菜单：

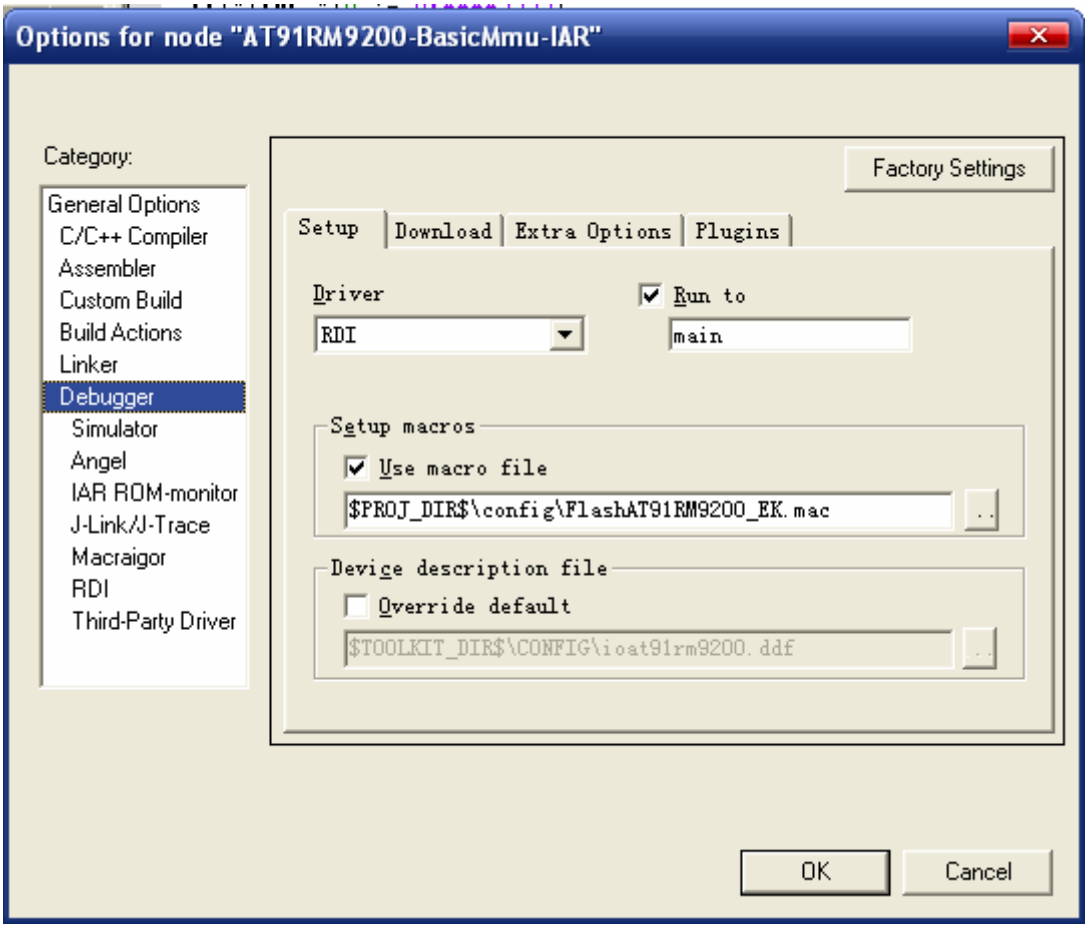


图 1，Options 菜单，Debugger 子菜单

将 macro file 通过右边的按钮指向到重新编译生成的 mac 文件，如图 2：

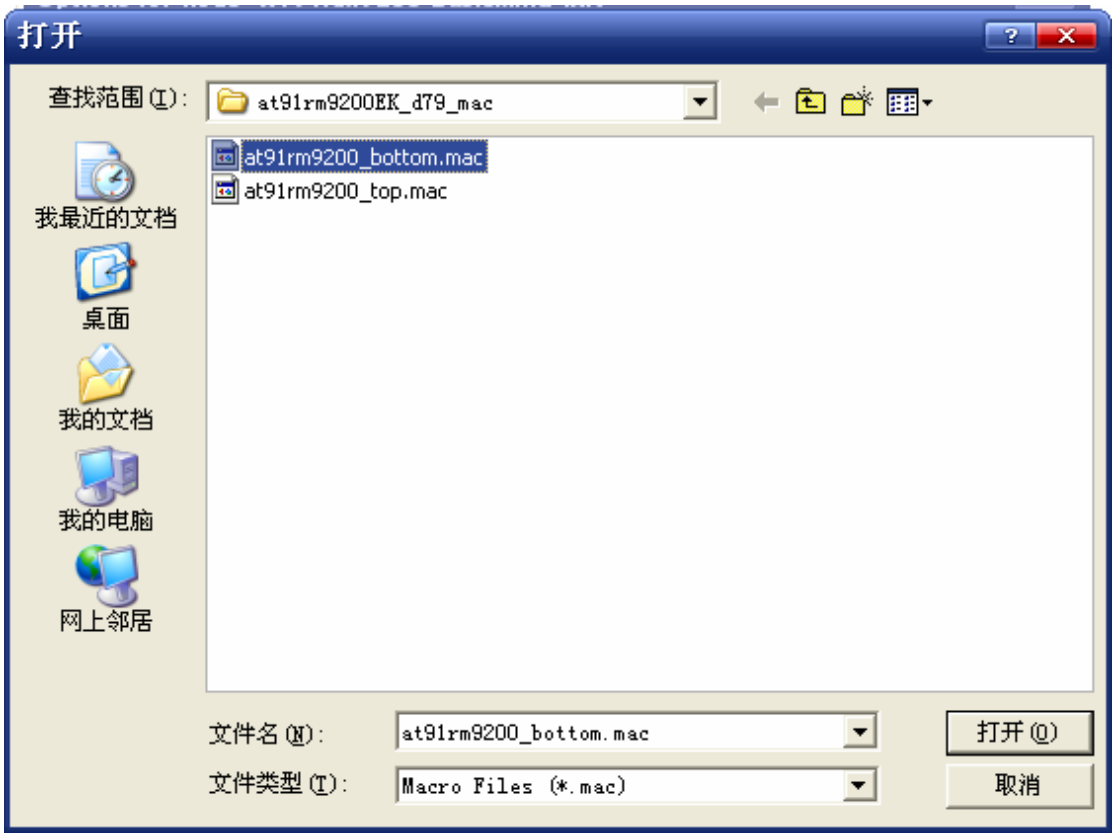


图 2，选择 mac 文件

然后进入 Debugger 下的 Download 菜单：

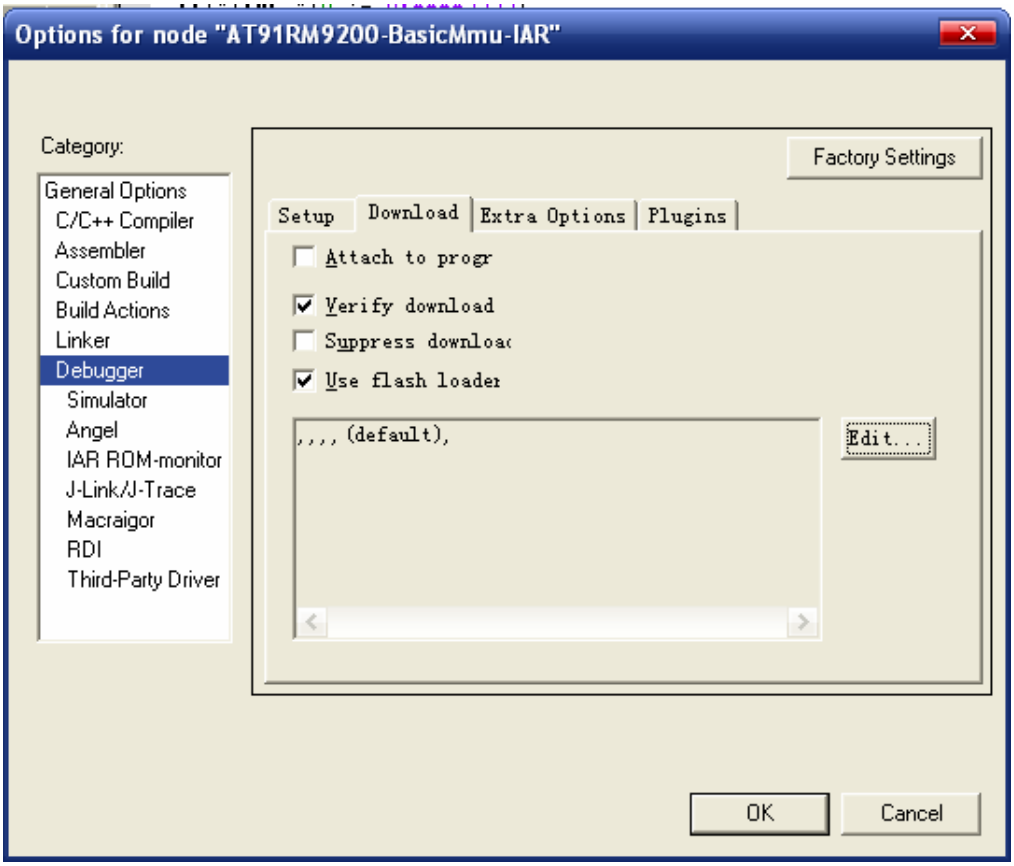


图 3，Download 菜单

点击 Edit，然后点中（default），再点击右边的 Edit 按钮

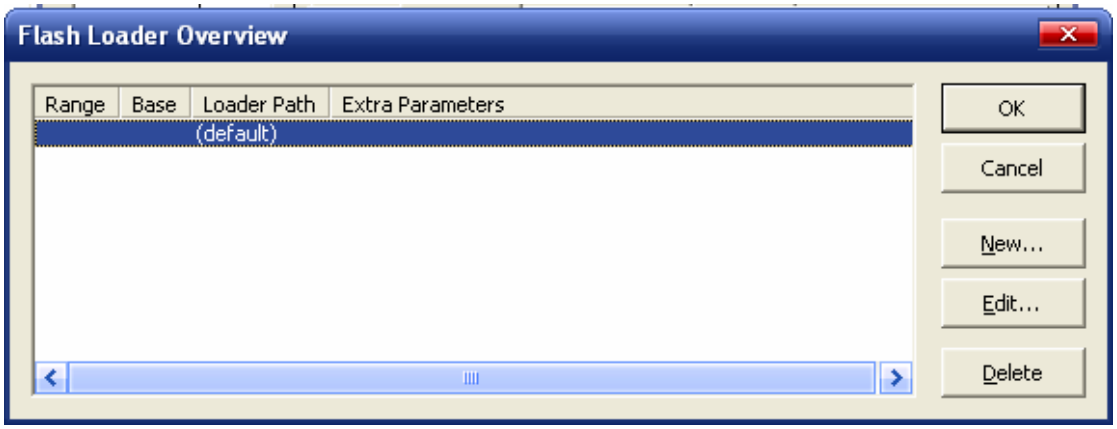


图 4，Flash loader

点 Edit 按钮后出现以下页面：

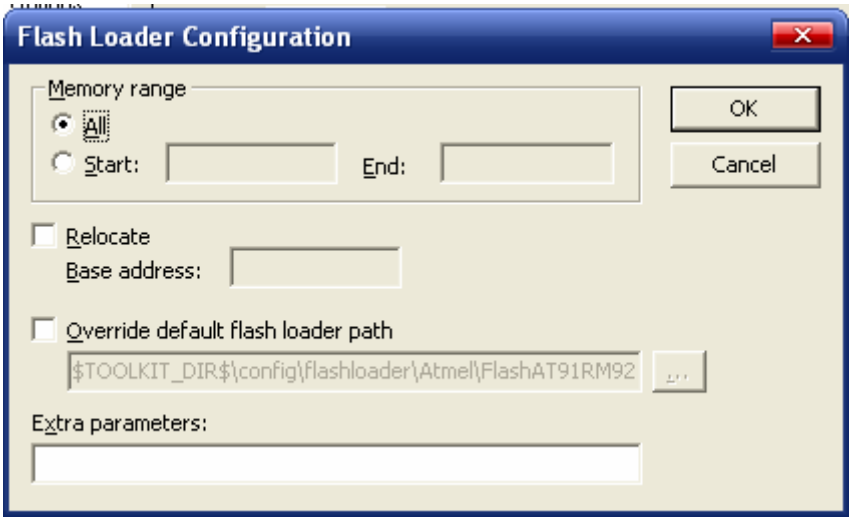


图 5，Flash loader Configuration

重新设置.d79 文件，如图 6：

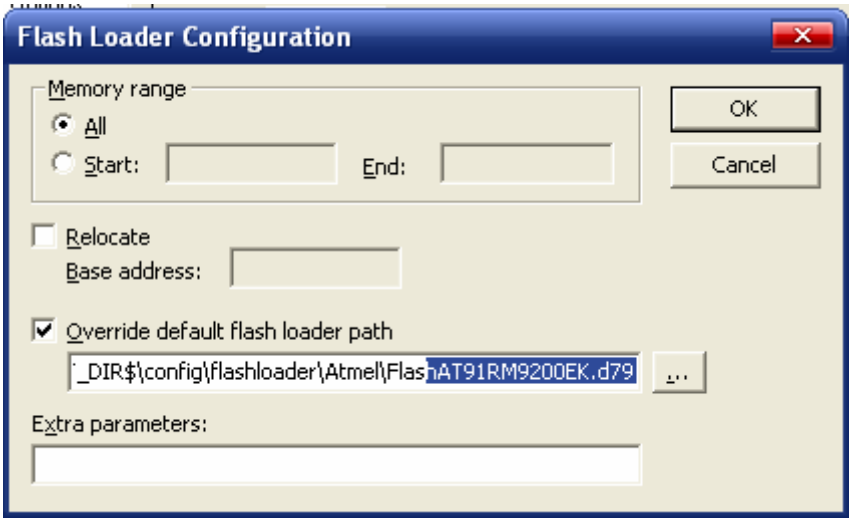


图 6，设置.d79 文件

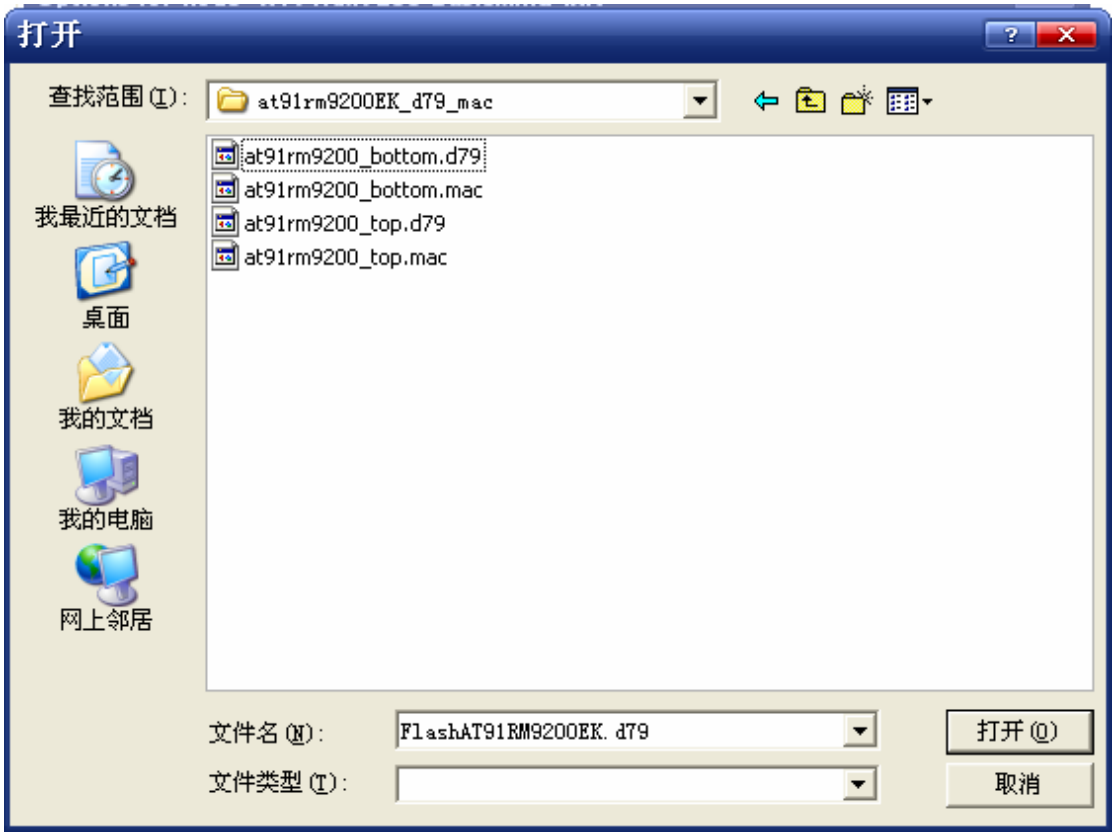


图 7，选择.d79 文件

完成 mac 文件和 d79 文件设置后回过头来设置仿真器：

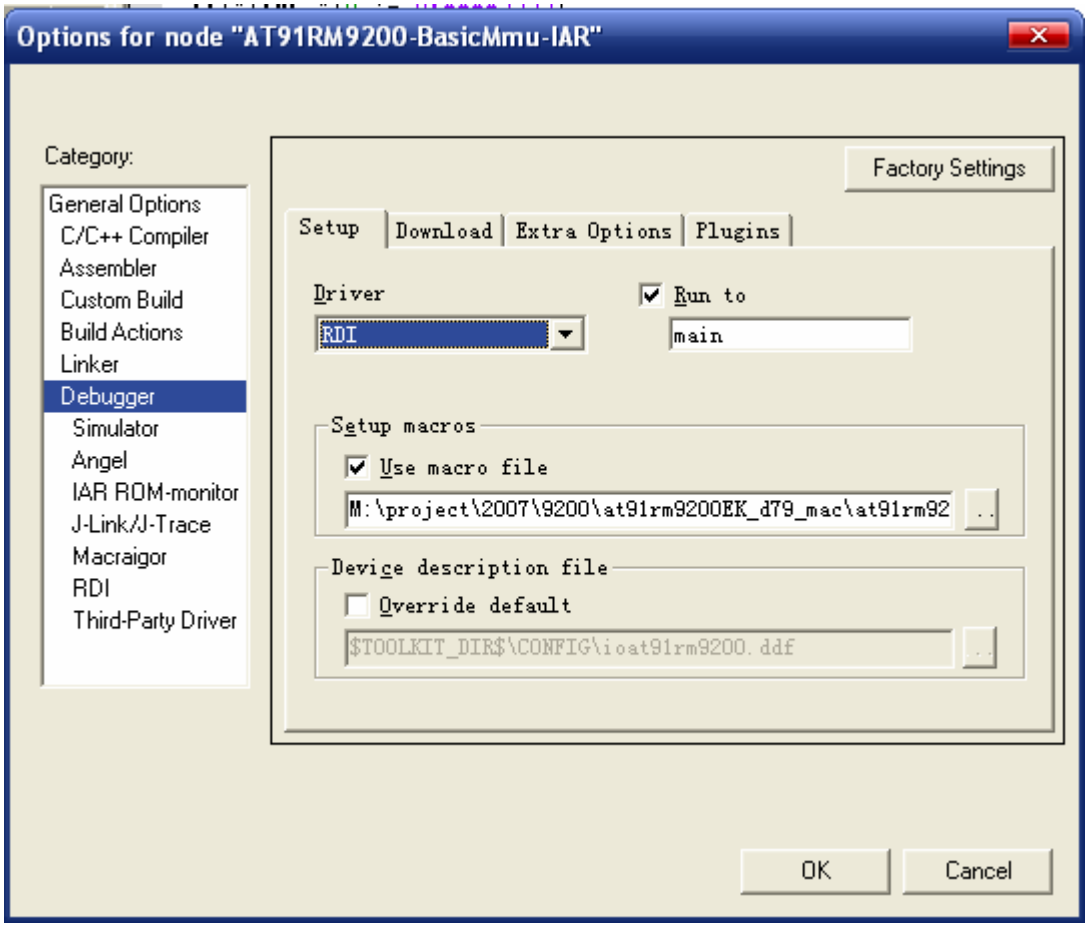


图 8，设置 debugger

如果你使用 wiggler＋H-JTAG，就选择 RDI，然后在 RDI 菜单下指向到-JTAG 的 Dll 文件；如果你使用 Multi-ICE 仿真器，也选择 RDI，然后在 RDI 菜单下指向到 Multi-ICE 安装目录下的 DLL 文件；如果你使用带 RDI 授权的 J-LINK，也选择 RDI，然后在 RDI 菜单下指向到 JLINK 提供的 DLL 文件：

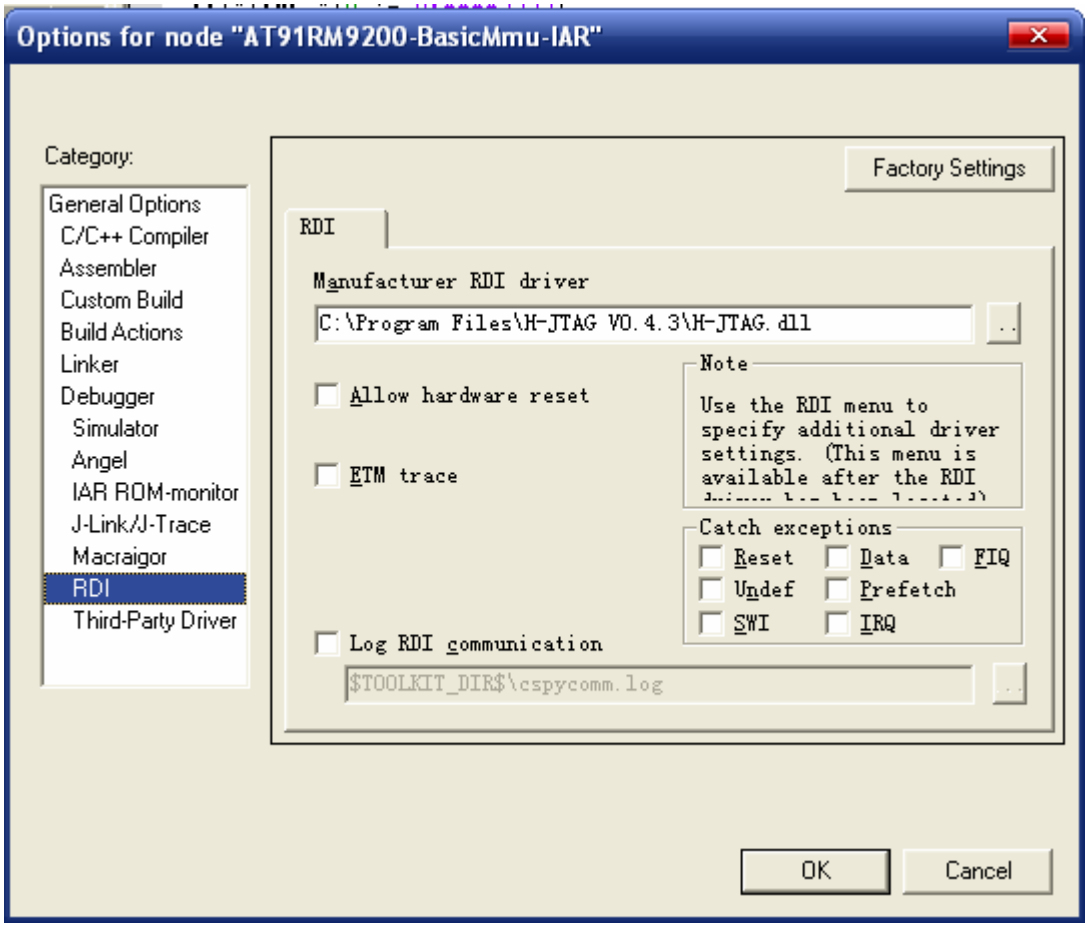


图 9，H-JTAG 设置

如果你使用的仿真器是 Multi-ICE，则指向到 Multi-ICE 的 DLL 文件：

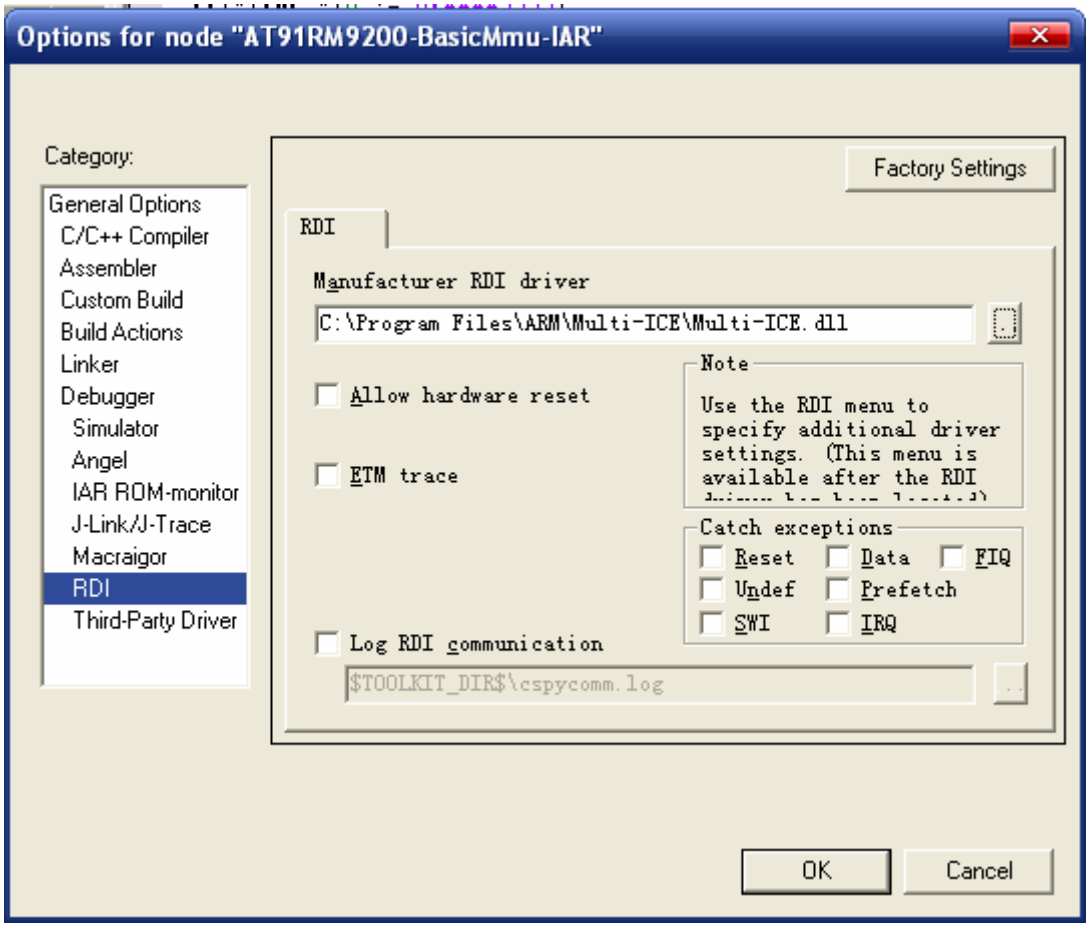


图 10，Multi-ICE 设置

当然，你也可以用 macraigor 的 wiggler 驱动：

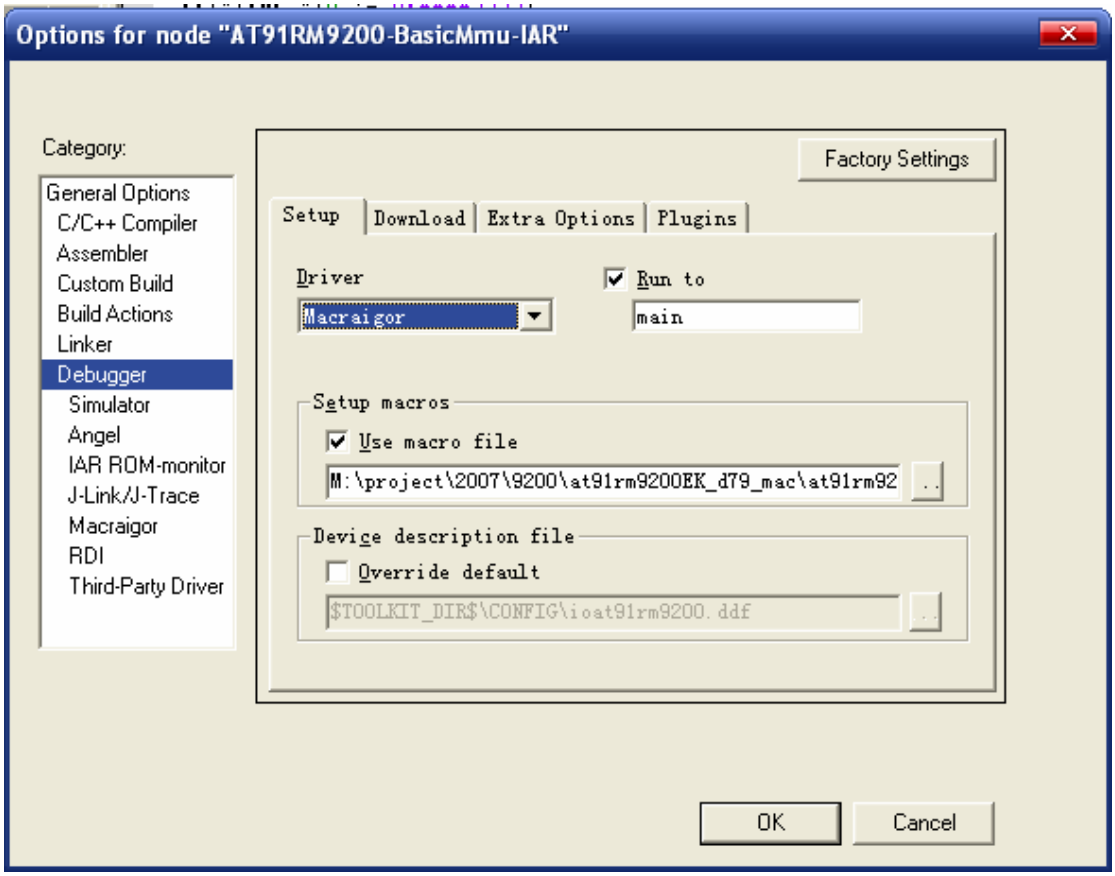


图 11，macraigor 驱动

Macraigor 提供的 wiggler 的驱动下载速度及其有限，推荐使用 H-JTAG 作为 JTAG 调试代理软件，可以有效提升下载速度。

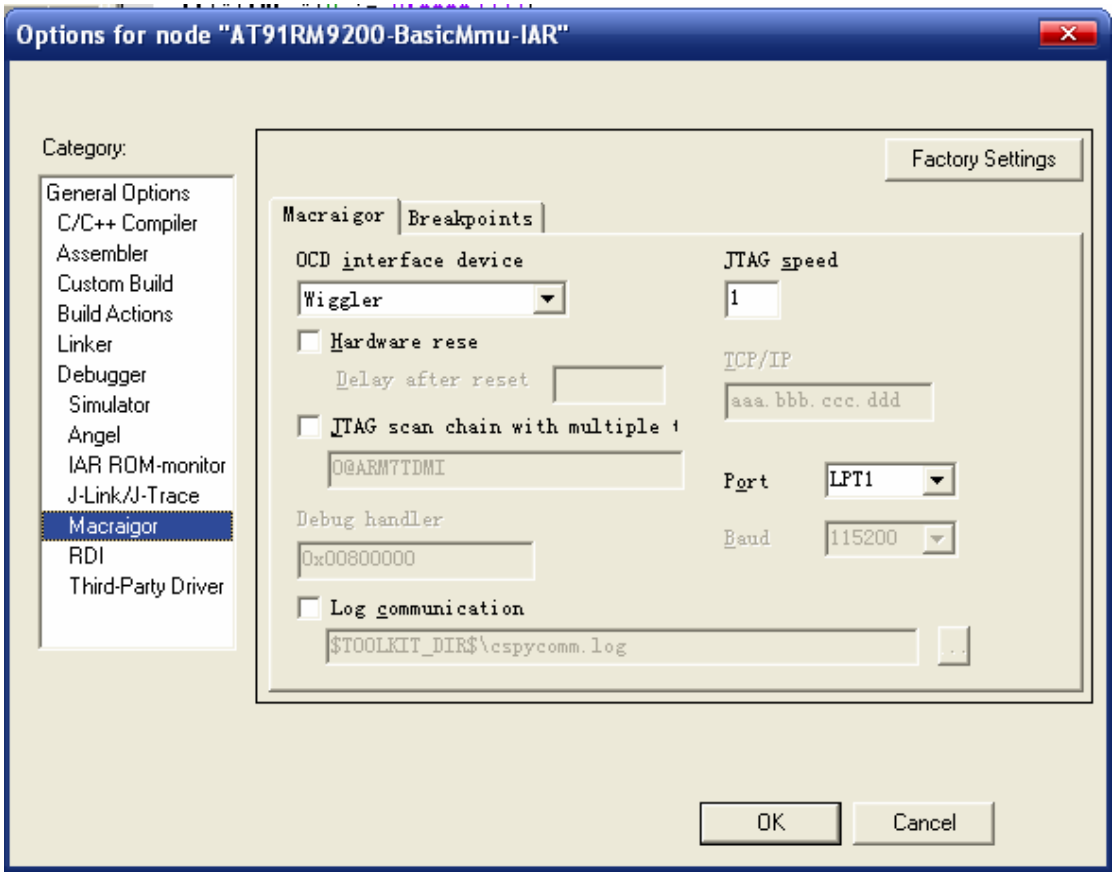


图 12，macraigor 的 wiggler

设置完成，即可选择 Project 下的 Debug 按钮，进入调试状态，

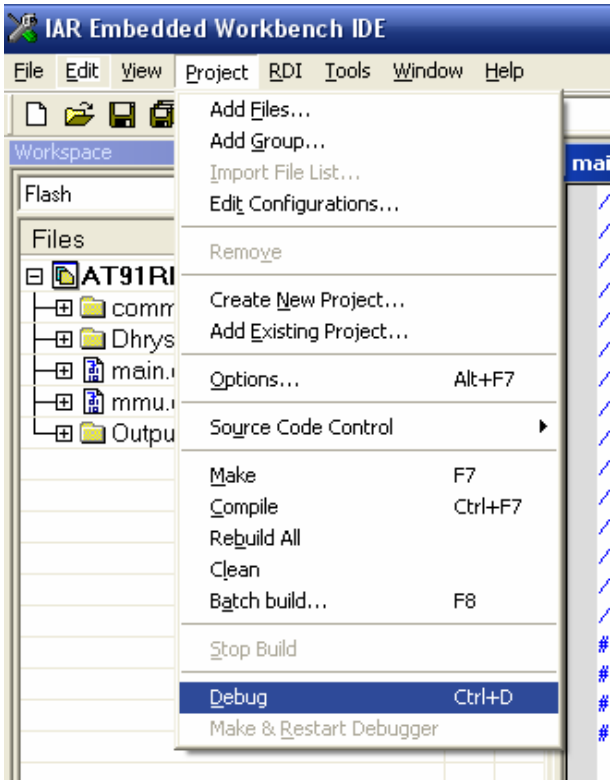


图 13，进入 Debug 状态

把串口模块连接到 9200 核心板上，观察输出信息。如果正常，按 Debug 按钮后，马上会进入编程 FLASH 的状态，大概 10 秒左右，程序下载完成，按照默认设置，程序会直接 Run to main，按下执行按钮后，Dhrystone 测试程序开始运行，并通过调试接口输入信息，如下图所示：

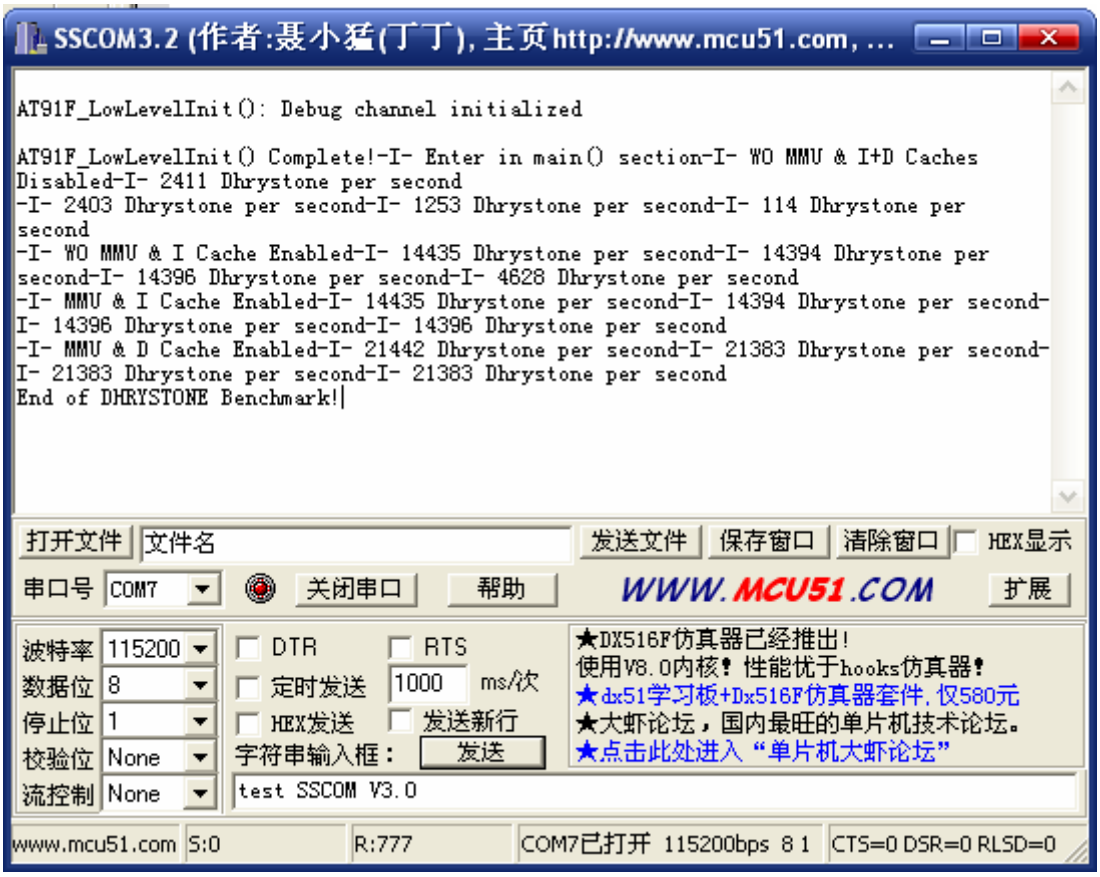


图 14，运行信息

从目前的测试情况来看，如果退出调试环境，想再次进入的话，就会提示 CFI Query Error：


图 15，CFI Query Error

如果使用网友提供的那两个.d79 文件，该错误提示出现较为频繁，而如果使用本站提供的 d79 文件，相对而言出错的概率要小很多。网友提供的 d79 文件对 FLASH 编程算法有所改动，而本站提供的 d79 文件使用的是 AT49BV1614 的驱动，只是把 ID 检查部分注释掉了。这样 IAR 就将 9200 核心板上的 AT49BV163D

当成 AT49BV1614 进行编程，而实际上 AT49BV163 即可以直接替换 AT49BV1614。

当出现 CFI Query Error!的时候只能通过 H-Flasher，或者 J-FLASH 软件先把 AT49BV163 擦除，才能再次进入调试状态。0.4.4 版本的 H-JTAG 附带的 H-Flasher 开始支持 AT49BV 系列 NOR FLASH，感谢 H-JTAG 的作者，使得我们可以以很低的门槛进入 ARM9 的学习阶段。

下面讲一下 0.4.4 版本的 H-JTAG 和 H-Flasher 的设置：

第一步，打开 H-JTAG，找到目标器件

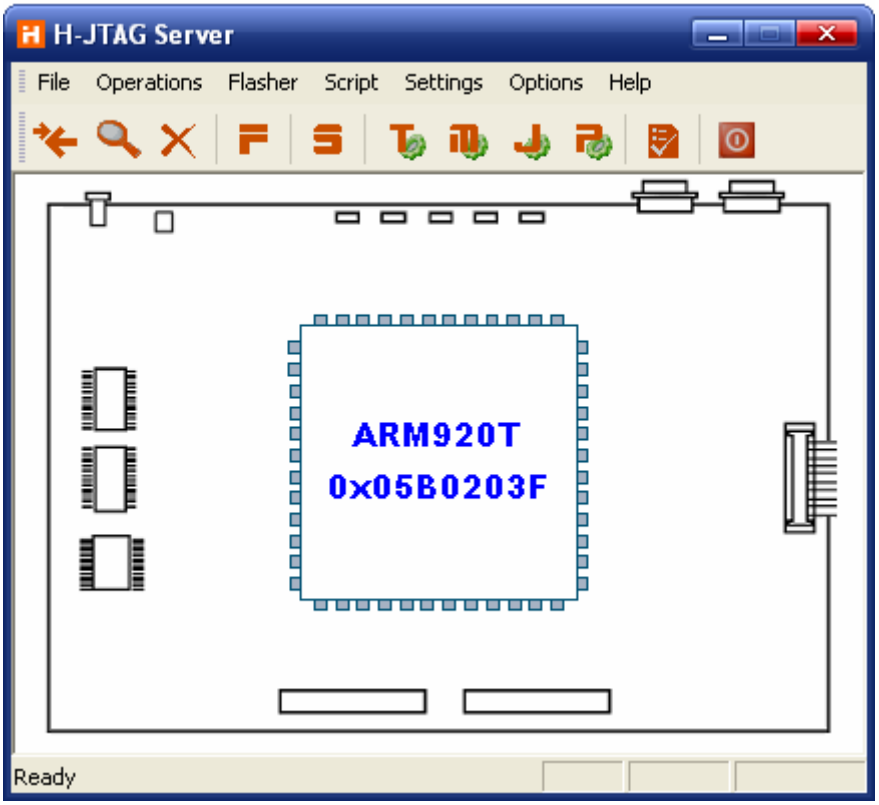


图 16，找到目标芯片内核

第二步，打开 H-Flasher

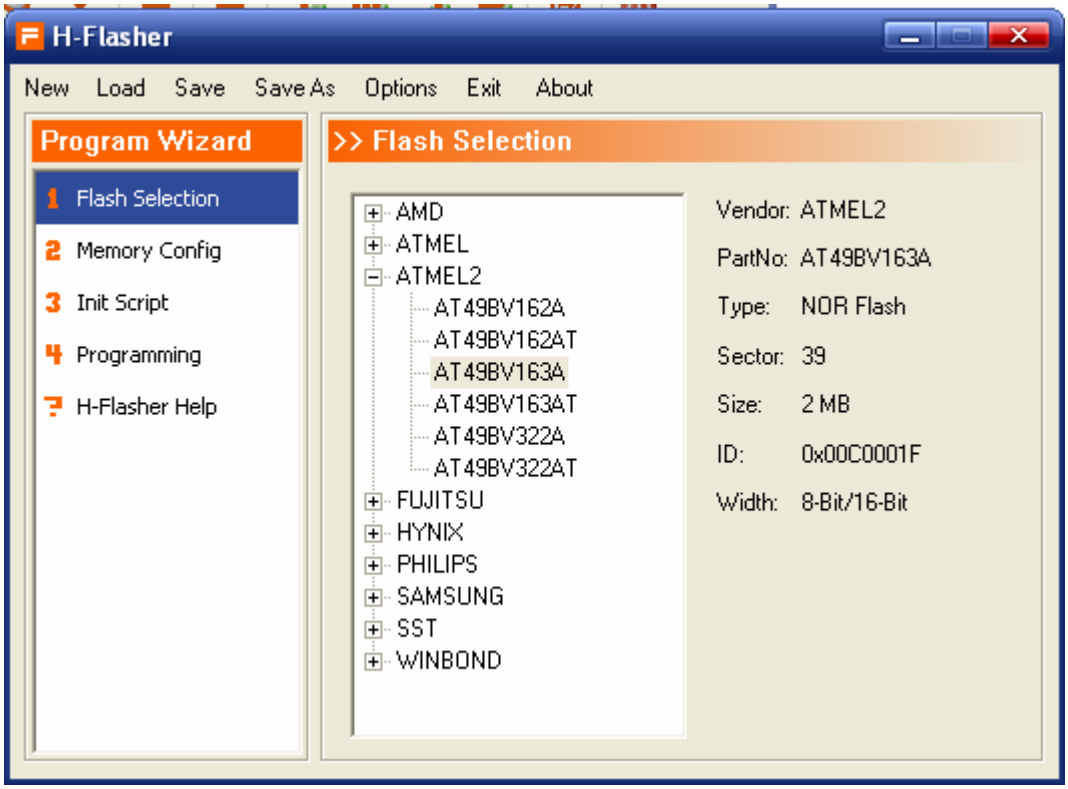


图 17，H-Flasher

下面我们只要按照 H-Flasher 左边的 Program Wizard 一步一步进行设置即可。首先是选择 FLASH，这里我们选择 AT49BV163A。

如果觉得 AT49BV163A 和核心板上使用的 AT49BV163D 后缀不符，可以自行加入一个文件，方法是到 H-JTAG 的安装目录下找到 AT49BV163A 的文件，然后把该文件里面的 ID 改动一下，然后另存为 AT49BV163D，当重新打开 H-Flasher（一定要在右下方任务栏上退出 H-JTAG 和 H-Flasher 后再重新打开）后就会发现多出了一个 AT49BV163D 的型号。


图 18，添加一个 AT49BV163D 文件

然后设置存储器信息，请按照实际情况进行设置，对于我们的 9200 核心板，我们使用的是 16 位宽的 FLASH，FLASH 的起始地址是 0x1000 0000，RAM 的起始地址是 0x2000 0000。如下图进行设置：

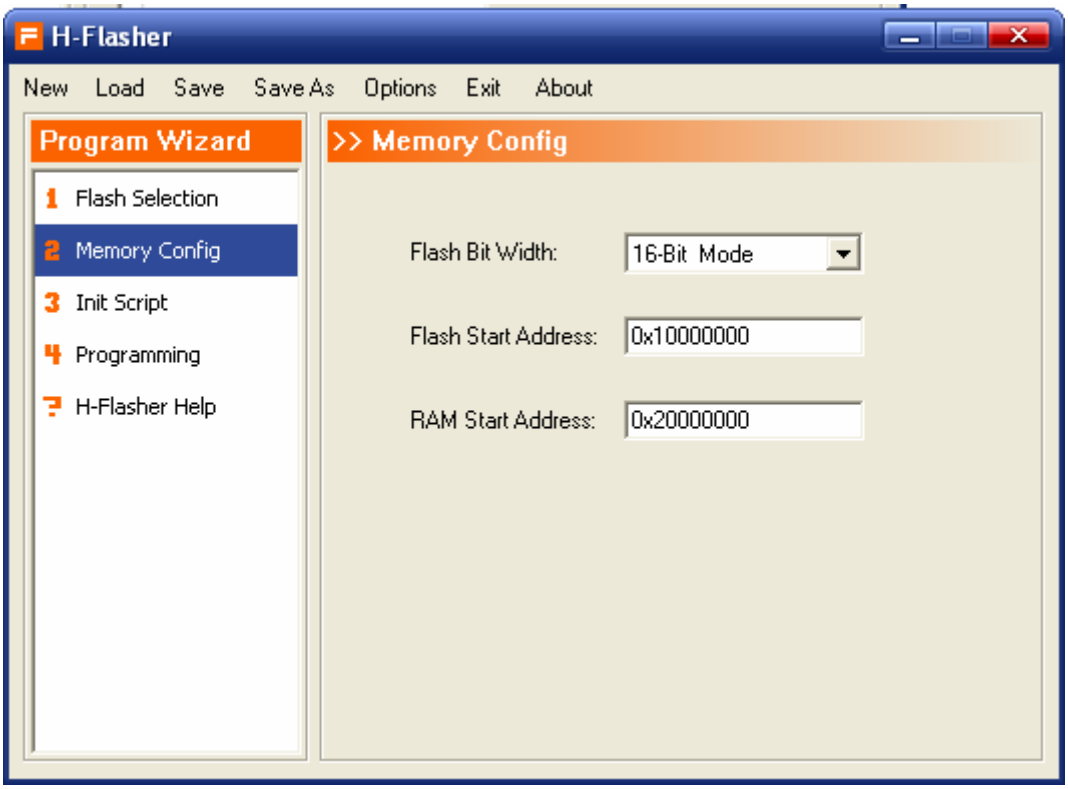


图 19，Memory Config

Init Script 可以直接跳过。

Program wizard 的第四步是 Programming，在这一步，里面有不少的操作。

首先，我们需要进行 Check，只有在找到 9200，并且找到 FLASH 的前提下，我们才能对 FLASH 进行读，擦除，写等操作。在进行 check 前，请确认 H-JTAG 已经启动，并已经检测到内核，如图 16。然后我们点击 check 按钮，如果一切正常，H-Flasher 会找到 FLASH，并显示 FLAHS 名称和 ID，不然将会提示错误。

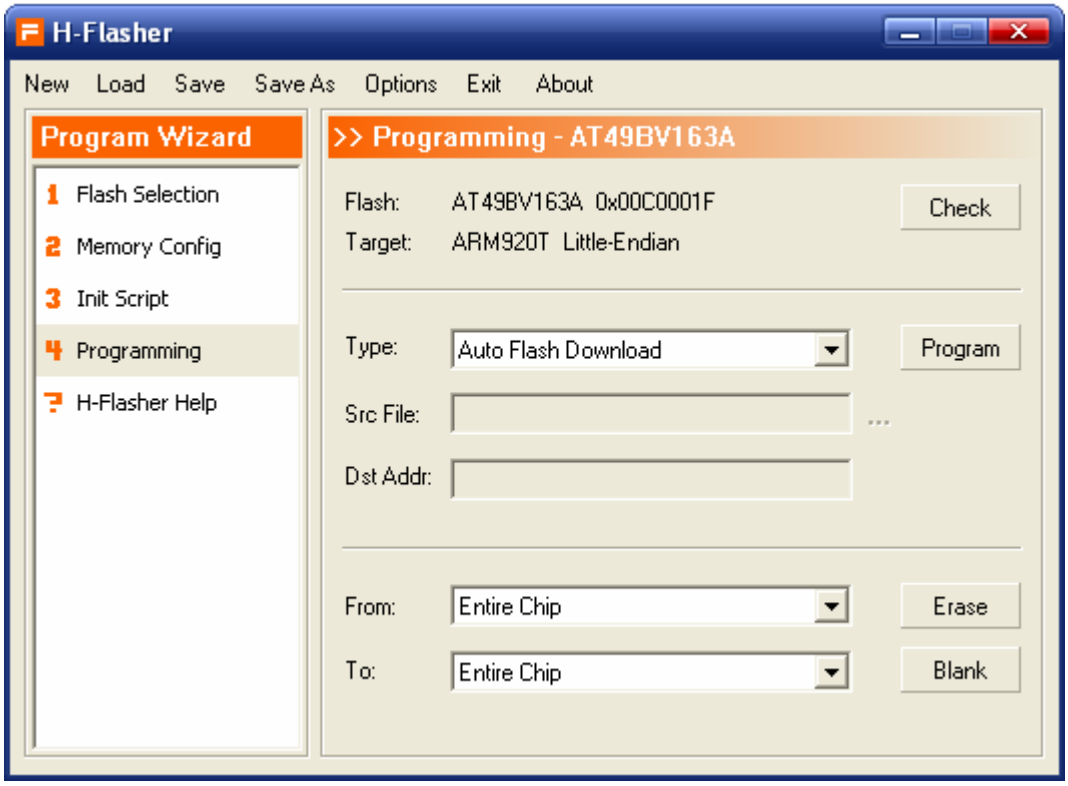


图 20，Programming，Check

如果找不到 FLASH，将会出现以下提示：


图 21，Error

这个时候，请插拔一下电源，然后重新点 check，一般情况下就会查找到 FLASH 了。如果还不行，请运行一下 IAR 的 debug 试试看。按照现有测试规律，如果 IAR 不能进入 Debug 状态，那么 H-Flasher 一般就可以检测 Check 成功，并进行 Erase 和 Program 操作；如果 H-Flasher 不能成功 check，那么 IAR 一般可以

进入 Debug 状态；如果既不能 Check 到 FLASH，也不能进入 Debug 状态，那么请插拔一下电源，然后再检测 IAR 下是否可以进入 debgu 状态，或者 H-Flasher 下能否 check 到 flash。

Check 成功后，一般我们需要 Erase，下面是 erase 全片 AT49BV163 的时间：

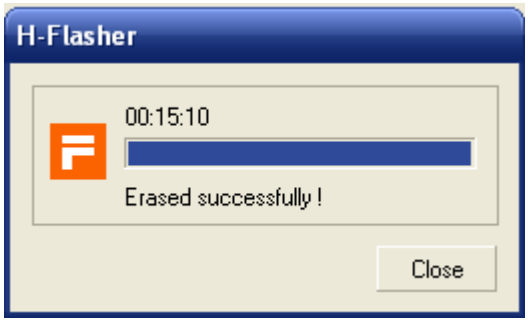


图 22，erase

2MB FLASH 整片擦除花了 15 秒左右时间。作为一个免费的软件，配合一个简易的调试器，能够达到这样的速度已经比较理想了。

再来看看用 J-FLASH 进行整片擦除的时间：

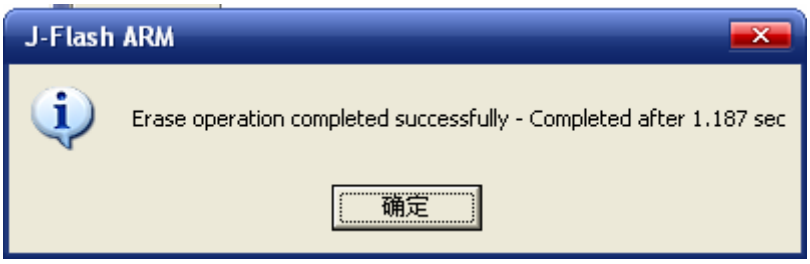


图 23，J-FLASH 擦除 AT49BV163 所需时间

可以看到，J-FLASH 的优势还是很明显的。

下面是查空的时间：

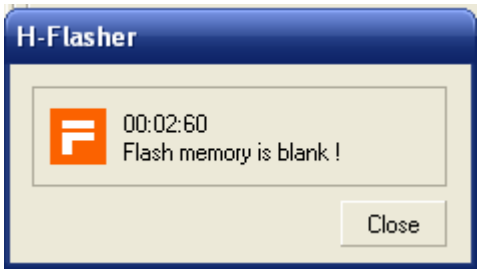


图 24，查空


图 25，查空时间

J-Flash 比 H-Flasher 的查空的速度也要快不少。

为了进一步对比 J-Flash 和 H-Flasher 的性能，我们先用 J-Flash 生成一个 1MB 的 bin 文件进行测试。下面是 J-Flash 编程 1MB 文件的过程和速度：

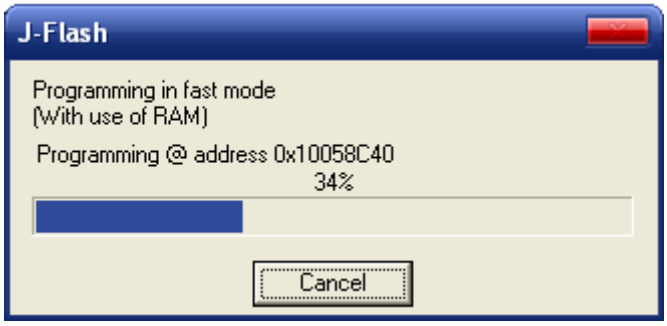


图 26，编程 1MB 文件@4M JTAG

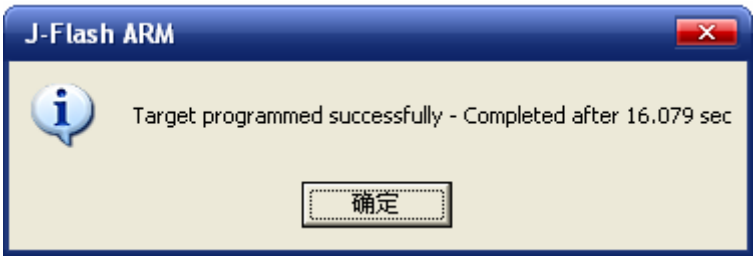


图 27，编程 1MB 文件所花时间@4M JTAG

下面我们再将 J-Flash 的 JTAG 时钟提高到 12M，相比 4M JTAG 时钟，12M 还是有不少提升。

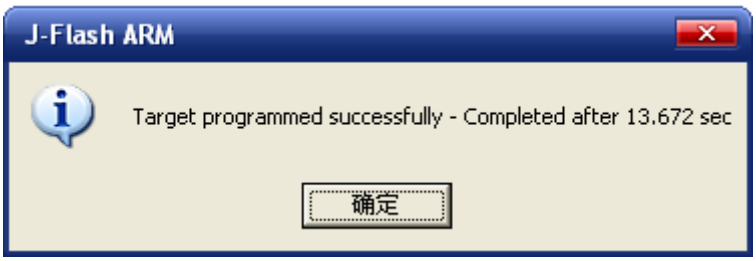


图 28，编程 1MB 文件所花时间@12M JTAG

JTAG 时钟从 4M 提高到 12M，编程 1MB 文件的时间也从 16 秒缩短到 13 秒。

下面再看一下编程加校验的速度数据：

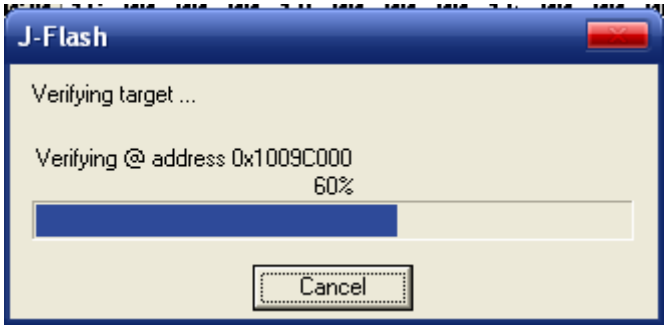


图 29，编程加校验 1MB 文件


图 30，编程加校验 1MB 文件所花时间@4M JTAG


图 31，编程加校验 1MB 文件所花时间@12M JTAG

JTAG 时钟对烧写速度还是有较大帮助，建议在条件允许的情况下尽量提高 JTAG 速度，以获取最佳性能。

下面看看 H-Flasher 的编程 1MB 文件的情况，先按照下面内容对 H-Flasher 进行设置：

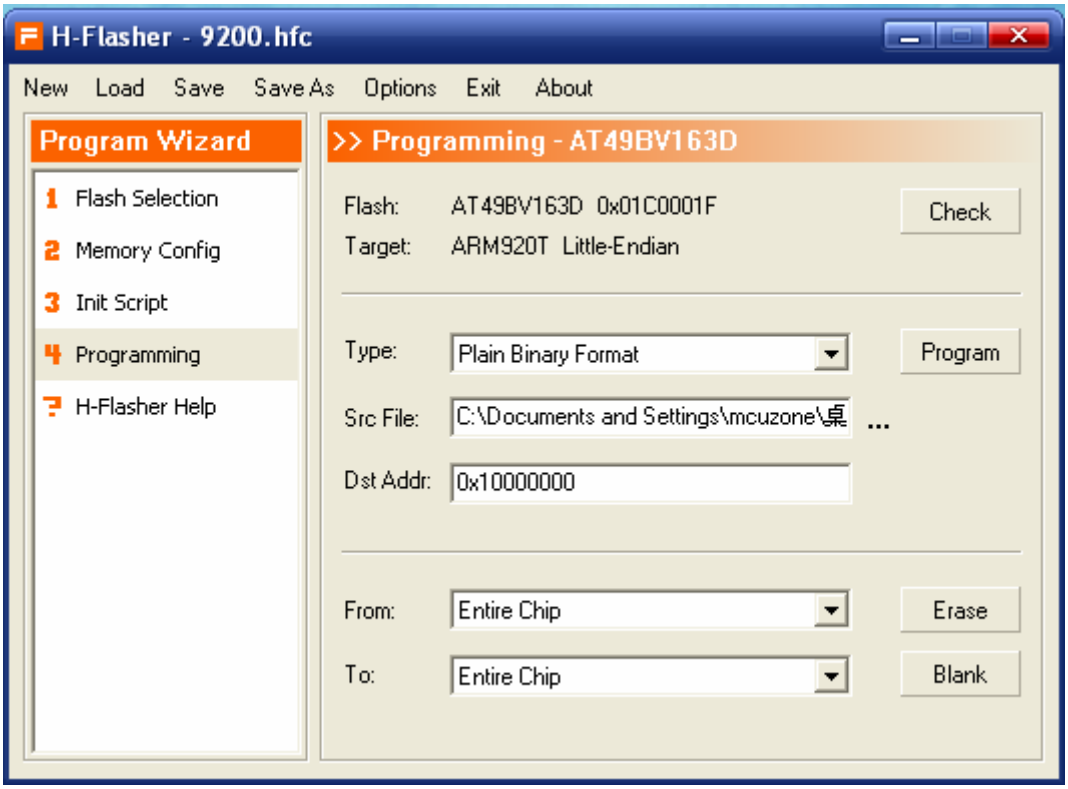


图 32，编程 FLASH 的设置

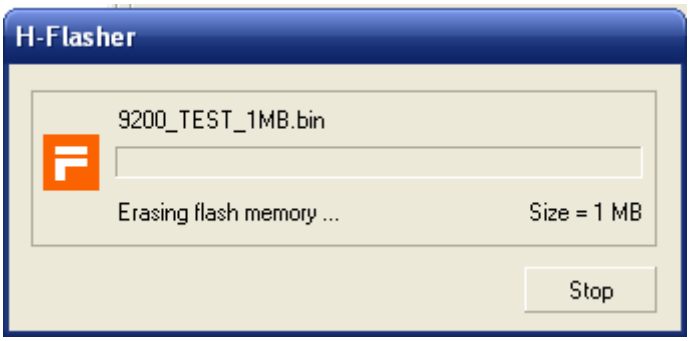


图 33，编程前自动进行擦除操作

点击“Program”后，H-Flasher 会先自动进行擦除操作，擦除后即进行 Flash 编程：

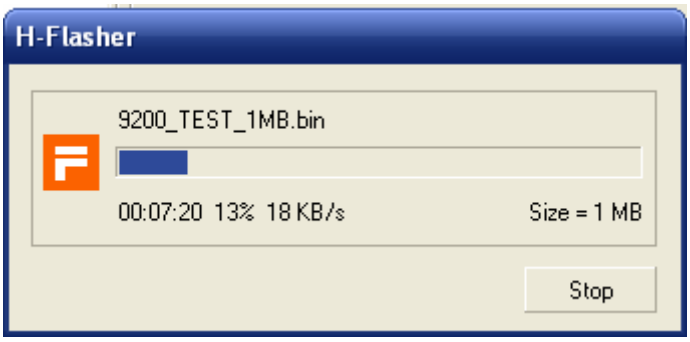


图 34，Flash 编程中

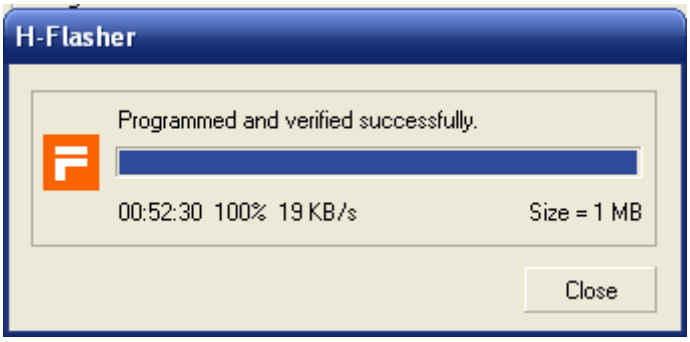


图 35，编程完成

虽然 H-Flasher 的速度和 J-FLASH 的速度有一个很大的差距，但是用来学习，基本已经够用，功能也已经比较完善，而且成本非常低廉，非常适合学习。