在 SAM9261 上调试 U-boot

Team MCUZone
http://www.mcuzone.com

版本：Rev1.0

www.mcuzone.com

2007-08

更新记录

2008.08.29

文档创建。

在 SAM9261 上调试 U-boot

—基于 Virtual PC

Team MCUZone

本文叙述 U-boot 调试过程，调试软件选择了 ARM 的 AXD 与 RVDEBUG，仿真器选择的是 jlink，其他仿真器可以类推。

一， 准备工作

1. 安装调试软件

PC 上的调试软件建议安装 ADS 或者 RealView 2.2，以后者为好。

2. 安装调试器及相关软件

由于使用的 jlink，首先必须安装 jlink 的软件，可以到 segger 网站下载。安装完成后需要将 jlink 整合到 axd 及 rvdebug 中。

连接上 jlink 到 PC 后选择自动安装驱动即可。

然后连接好 jlink 到 9261 板子 JTAG 口的 20 芯排线，给目标板上电(连接 mini USB 线)。

到 jlink 软件安装目录下运行 jlink.exe:

出现：

```
Feature(s) : RDI,FlashDL,FlashBP,JFlash,GDBFull
VTarget = 3.539V
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41069265: ARM, Architecure 5TEJ
Info: CP15.0.1: 0x1D152152: ICache: 16kB (4*128*32), DCache: 16kB (4*128*32)
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x0792603F
Found ARM with core Id 0x0792603F (ARM9)
  ETM V1.3: 4 pairs addr.comp, 2 data comp, 8 MM decs, 2 counters, sequencer
J-Link>_
```

那么就说明核心已经能被正确识别并连接。

3. 完成编译 U-boot 的工作

请参照本站另一片文章《为 SAM926X 编译 U-boot》为 9261 编译好 u-boot。

二， 使用 AXD 调试

1. 编写 AXD 格式的初始化脚本

由于要将 u-boot 加载到 sdram 中运行，且 u-boot 中不会再执行 low level 的初始化，因此相关的初始化必须由脚本完成，或者由复位的时候可运行的代码完成。

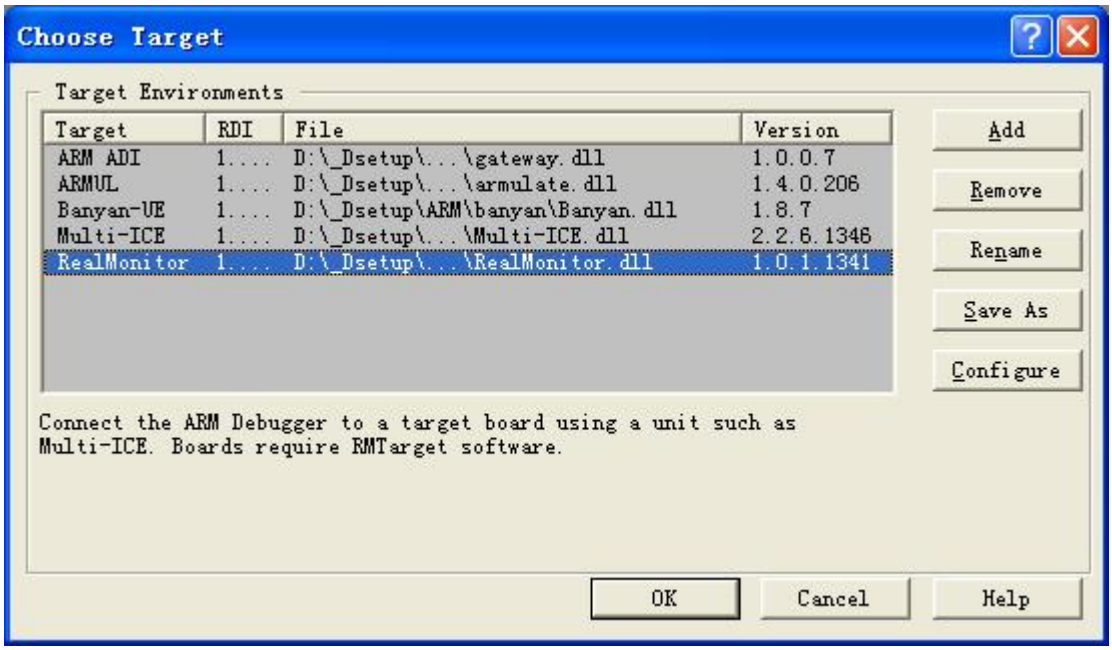
2. 加载 u-boot 的 elf 文件

连接底板的 J4 到 PC，出现一个 usb 转的串口，打开超级终端，并连接到该串

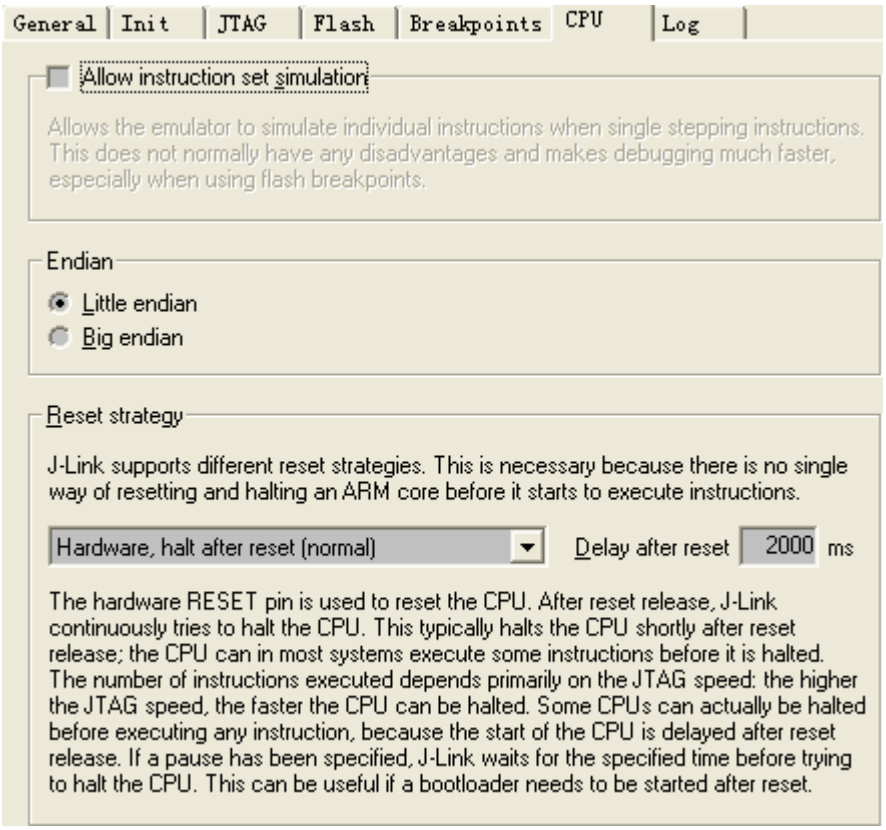
口。

连接好 miniUSB 的电源线，板上的程序会开始运行，如果是原来的 u-boot,记得要在 linux 或 win ce 起来之前将其停住。

运行 AXD，在菜单 Options->Config Target 中添加 jlink 的 dll(jlinkrdi.dll,在 jlink 软件的安装目录下)：



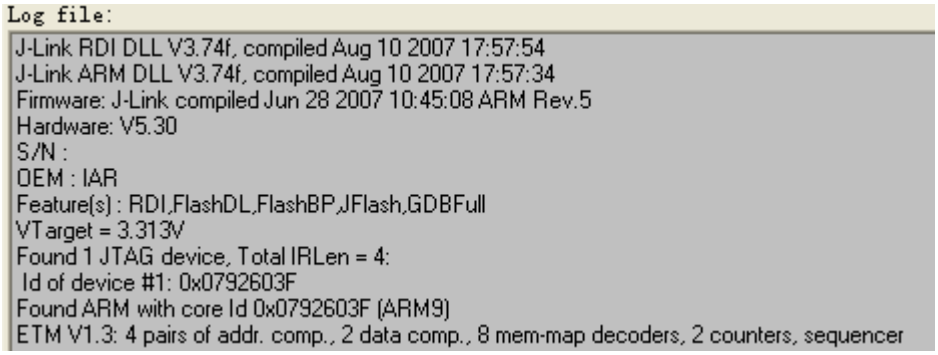
在 jlink 的设置选项卡中设置复位方式：



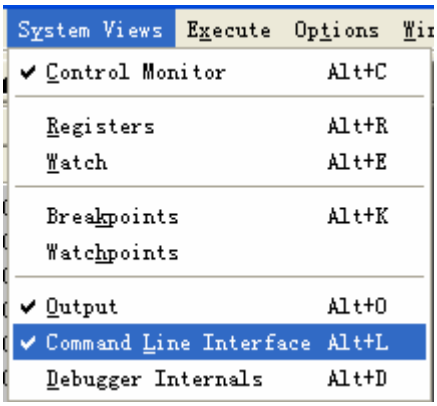
复位后的延时可以稍长。设置这个时间的目的就是让复位之后，系统中现有的代码有机会运行一段时间，这段时间正好完成系统相关的初始化,这样就可以不必使用初始

化文件。

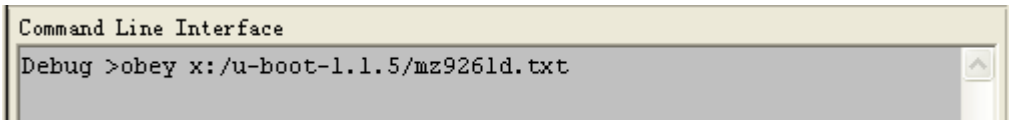
添加了 dll 后就可以进行联接，成功后如图：



选择菜单

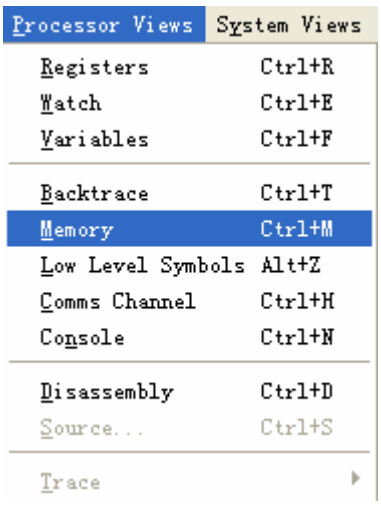


打开命令行窗口，然后在此窗口中运行 9261 的初始化脚本：

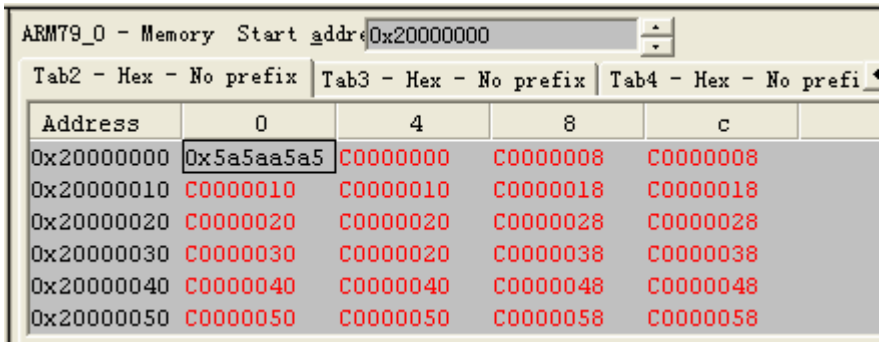


如果板子上有可以运行的代码，比如已经烧写了代码到 dataflash 中，那么根据前面的 jlink 设置，该代码会运行并初始化好环境，就不需要再运行初始化脚本。

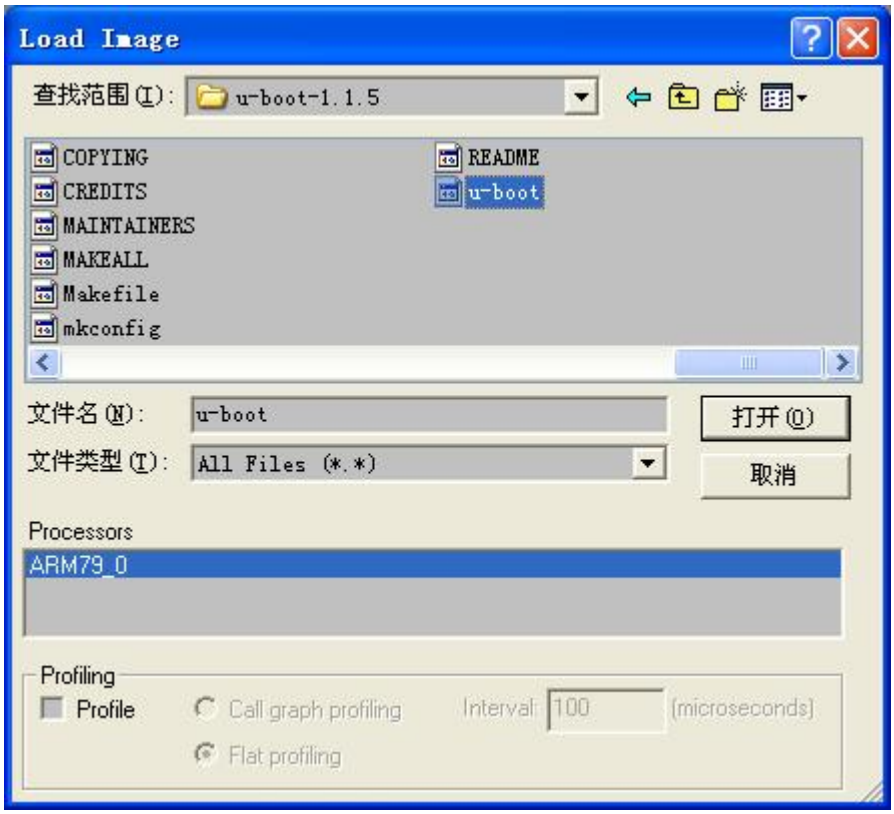
为了检验初始化结果，可以打开内存窗口:



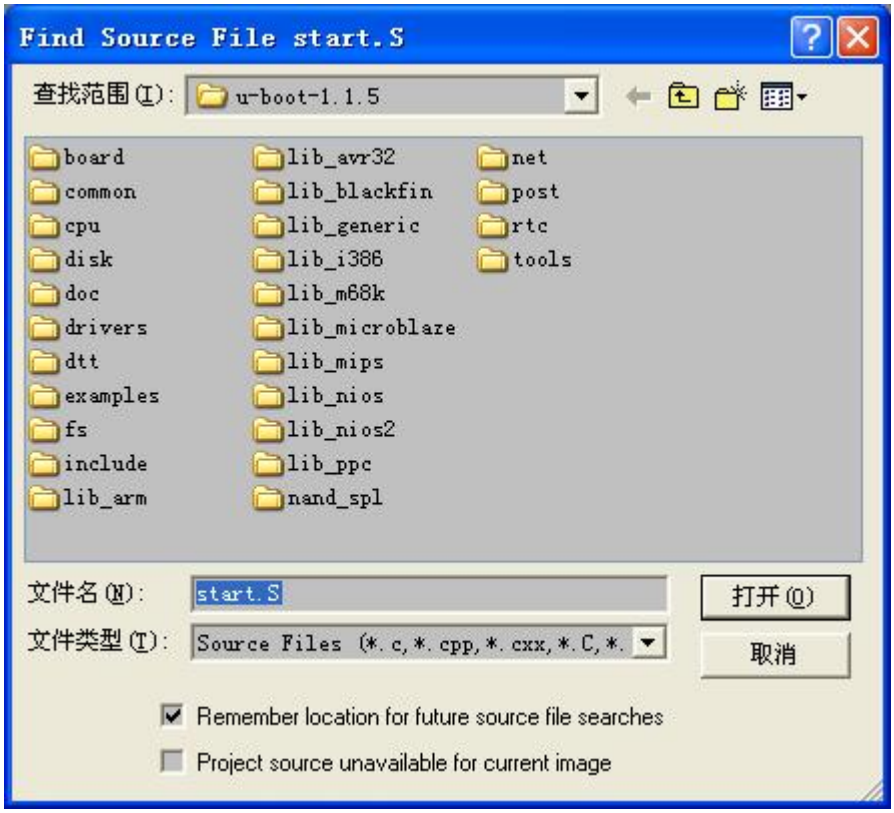
查看 SDRAM，向某个地址写入一个值，看看是否正确初始化：



选择菜单 File-> Load Image…,加载编译生成的 u-boot 文件:

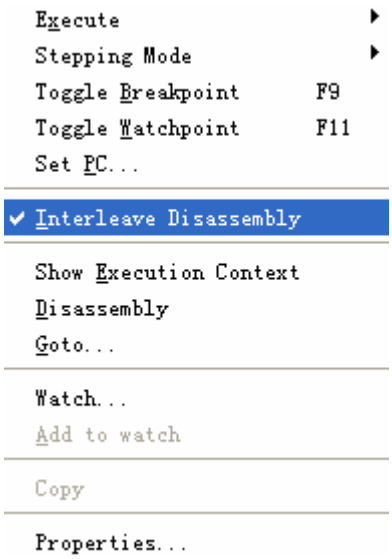


加载完成后 AXD 会试图显示第一个源代码文件，如果找不到就会出现：



手动指定到./cpu/arm926ejs/start.S 即可。以后也可能出现这个情况，也可以手动指定。

在代码窗口的右键菜单中选择反汇编模式：



检查代码是否正确加载：

```
52          .globl _start
53          _start:
54              b   reset
23f00000 [0xea000012]   b       (pc)+0x50 ; #0x23f00050
55              ldr pc, _undefined_instruction
23f00004 [0xe59ff014]   ldr     pc,0x23f00020 ; = #0x23f000a0
56              ldr pc, _software_interrupt
23f00008 [0xe59ff014]   ldr     pc,0x23f00024 ; = #0x23f00100
57              ldr pc, _prefetch_abort
23f0000c [0xe59ff014]   ldr     pc,0x23f00028 ; = #0x23f00160
58              ldr pc, _data_abort
23f00010 [0xe59ff014]   ldr     pc,0x23f0002c ; = #0x23f001c0
59              ldr pc, _not_used
23f00014 [0xe59ff014]   ldr     pc,0x23f00030 ; = #0x23f00220
60              ldr pc, _irq
23f00018 [0xe59ff014]   ldr     pc,0x23f00034 ; = #0x23f00280
61              ldr pc, _fiq
23f0001c [0xe59ff014]   ldr     pc,0x23f00038 ; = #0x23f002e0
```

从反汇编结果可以看到代码被正确加载，且代码 link 在 0x23F00000 位置。然后就可以使用 AXD 提供的功能调试代码。比如单步，跳出等等。

3. 代码调试

如果不像看汇编代码，可以直接到 start.S 的 185 行打个断点：

```
185             ldr pc, _start_armboot
186
187         _start_armboot:
188             .word start_armboot
189
```

然后点击运行，即会停在该行。

按 F8 单步，选择./lib_arm/board.c，会停在 C 函数 start_armboot 处。这个函数中有相当多的初始化代码。

下面以调试 NAND 代码为例，大体讲下调试的过程。

在 nand_init(L305, board.c)处打断点并直接运行到该处：

```
302
303         #if (CONFIG_COMMANDS & CFG_CMD_NAND)
304             puts ("NAND:  ");
23f00a28 [0xe59f0114]   ldr     r0,0x23f00b44 ; = #0x23f20610
23f00a2c [0xeb003f77]   bl      puts
305             nand_init();        /* go init the NAND */
23f00a30 [0xeb001262]   bl      nand_init
306         #endif
```

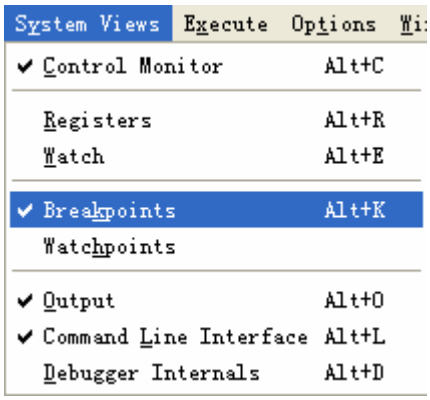
此时超级终端中能收到前面的函数执行完后所打印出的信息：

```
U-Boot 1.1.6 (Aug 26 2007 - 12:52:20)

DRAM:  64 MB
NAND:
```

单步跟进，打开./drivers/nand/nand.c。

打开断点窗口：



在窗口右键 add：

添加一个断点，地址为 0x23f1a0f4，其实就是 board_nand_init 函数。地址信息来源于 map 文件：

```
.text           0x23f19fd8      0x198 board/atmel/at91sam9261ek/libat91sam
                0x23f19fd8                at91sam9261ek_nand_init
                0x23f1a0f4                board_nand_init
.text           0x23f1a170       0xcc cpu/arm926ejs/libarm926ejs.a(cpu.o)
```

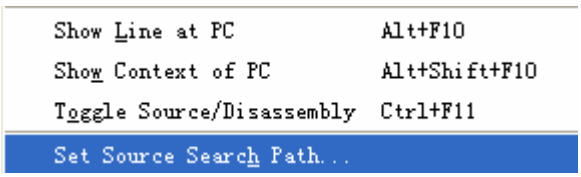
点运行后会执行到文件./board/atmel/at91sam9261ek/nand.c。

```
103     void board_nand_init(struct nand_chip *nand)
104     {
105         /* Init due to switch 8/16 bits mode */
106         if (nand->write_byte)
107             nand->write_byte = NULL;
108         if (nand->read_byte)
109             nand->read_byte = NULL;
110         if (nand->write_buf)
111             nand->write_buf = NULL;
112         if (nand->read_buf)
113             nand->read_buf = NULL;
114         if (nand->verify_buf)
115             nand->verify_buf = NULL;
```

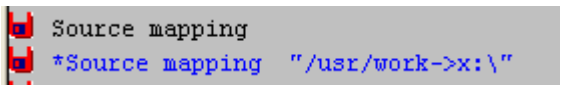
三， 使用 RVDEBUG 调试

1. 配置参数

打开 RVDEBUG，连接到 jlink。在 debug 菜单下选择配置文件搜索路径：

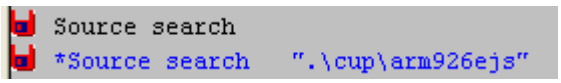


根据 u-boot 代码所在设置好 source 的 mapping，



其中/usr/work 是在 linux 上编译时 u-boot 源代码文件夹，x:为那个文件映射到本地网络驱动器的盘符。

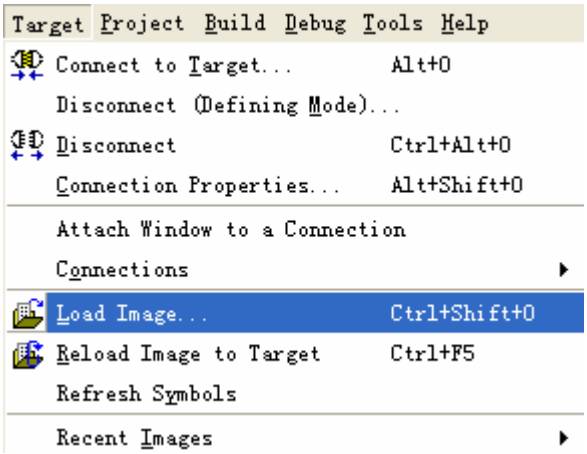
必要的时候也可以设置具体的搜索路径：



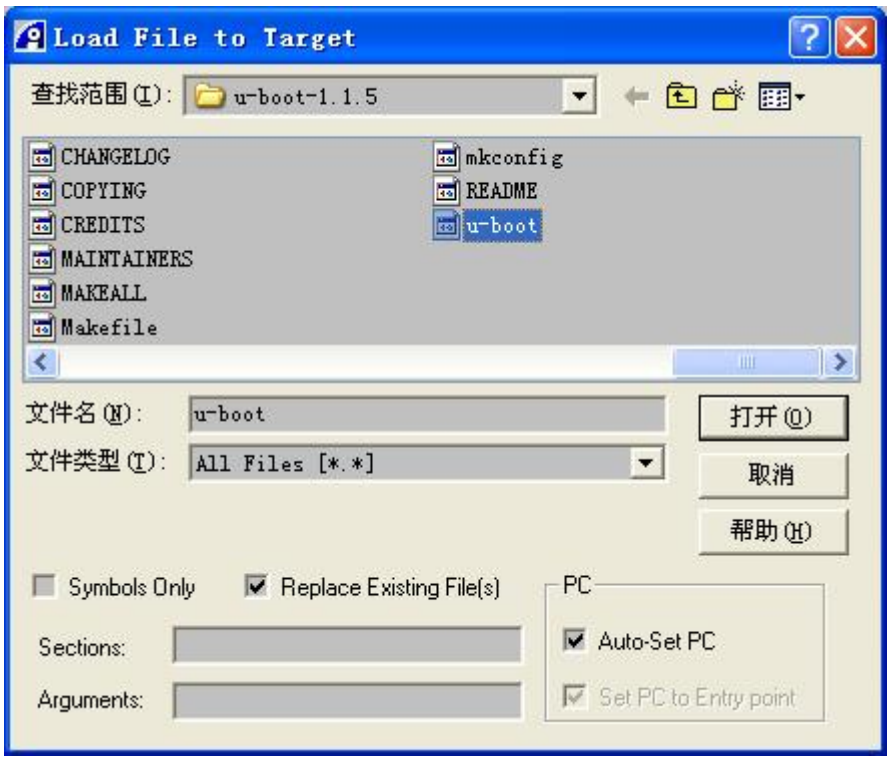
这个路径时基于 u-boot 的 elf 文件所在路径的一个相对路径。

2．加载文件

选择 target 下的菜单:

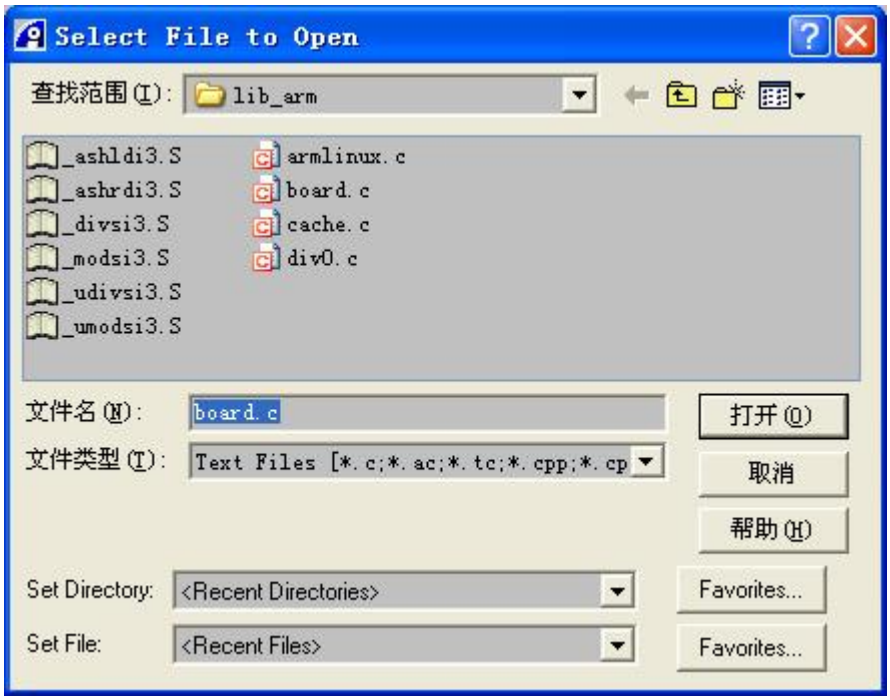


加载 elf 文件:

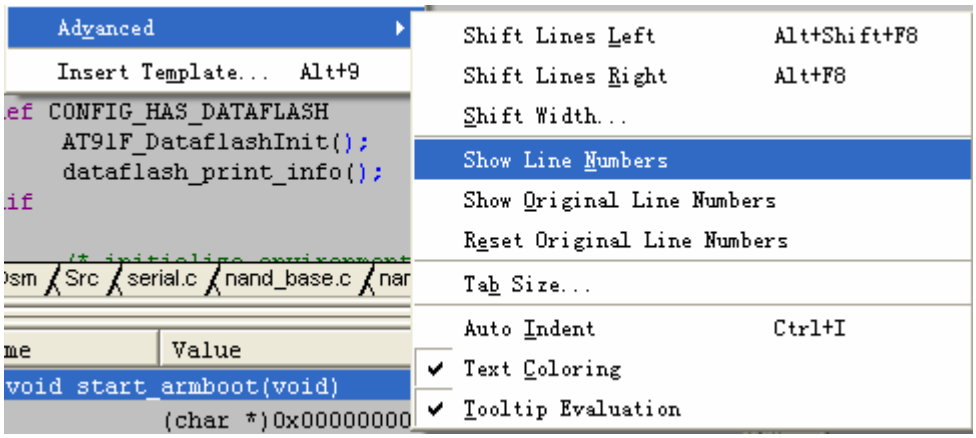


完成后即可在调试窗口看到代码，进入调试状态。

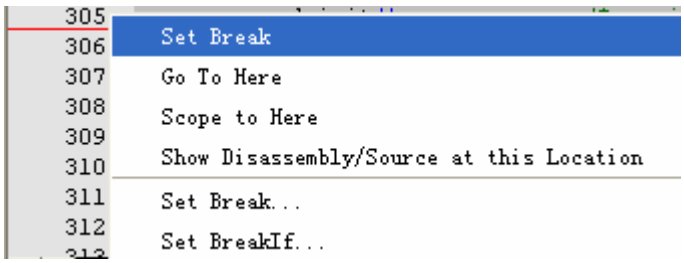
选择 File 下的 Open 打开 board.c：



打开文件后选择 Edit 下功能，显示行数：



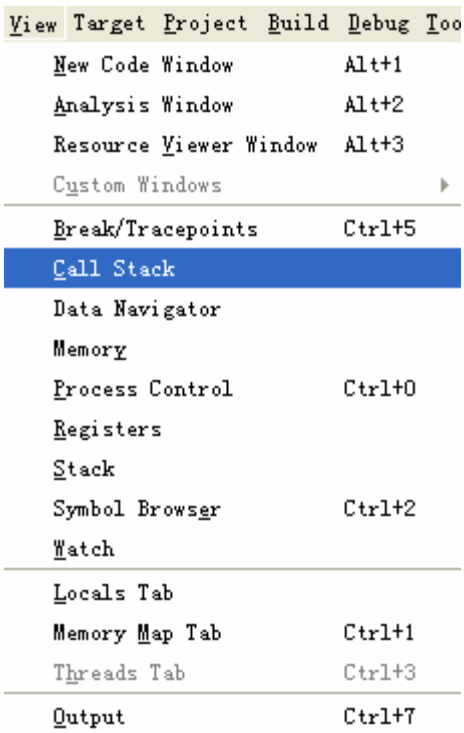
在到 board.c 的 305 行，在左侧函数位置单击鼠标左键：



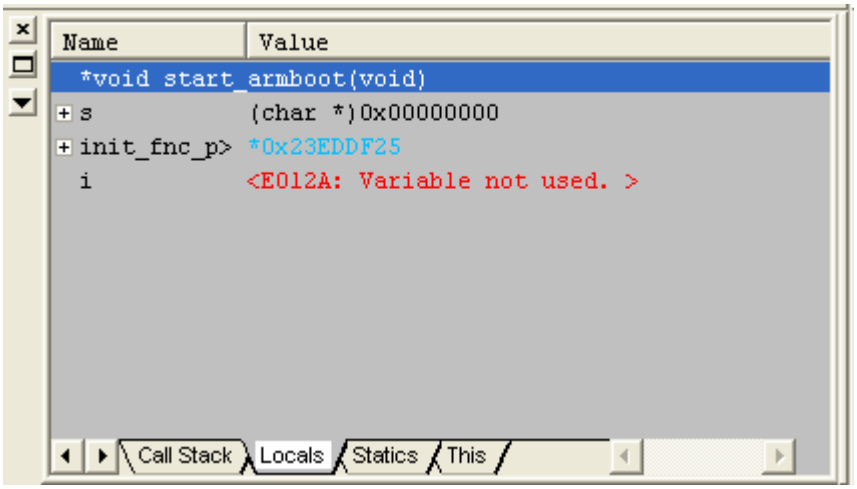
选择 Set Break 即可打断点。

在代码中可以使用单步等常规调试功能。**注意**：RVDEBUG 的调试快捷键与 AXD 不同。

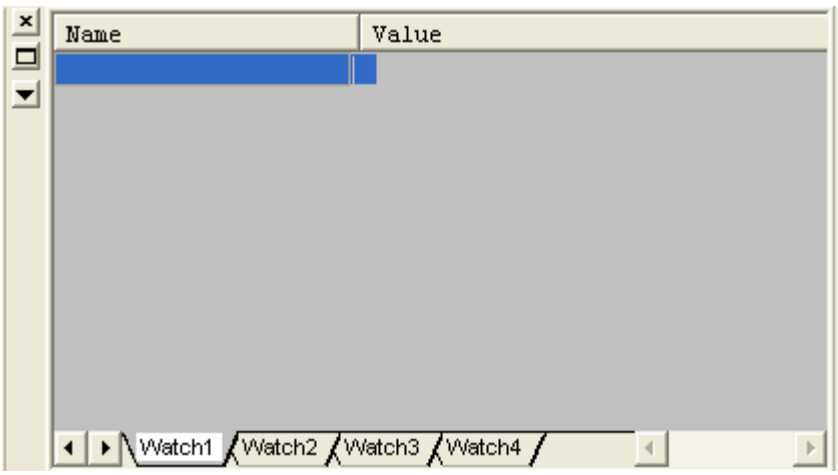
在 view 菜单下可以打开相应的资源查看窗口：



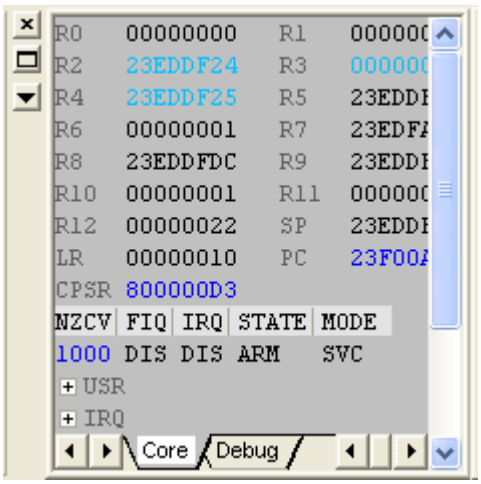
比如 call stack 窗口：



watch 窗口：



Register 窗口：



四， 一些事项

U-boot 在默认的情况下，编译的时候会进行优化，这会对调试造成影响，有的时候源码调试会比较困难。为了调试方便，可以关闭优化功能生成代码，专门用于调试，调试完成后再修改成优化方式。由于关闭优化会使得代码体积变大，因此可能需要关注

一下可能产生的问题。

关闭优化可以通过修改 U-boot 根目录下的 config.mk 来实现：

```
128 RANLIB  = $(CROSS_COMPILE)RANLIB
129
130 ARFLAGS = crv
131 RELFLAGS= $(PLATFORM_RELFLAGS)
132 DBGFLAGS= -g # -DDEBUG
133 OPTFLAGS= -Os #-fomit-frame-pointer
134 ifndef LDSCRIPT
135 #LDSCRIPT := $(TOPDIR)/board/$(BOARDDIR)/u-boot.lds.debug
```

将 133 行的 Os 改成 O0，先 make clean，然后重新编译，即可用新的 U-boot 调试。

五， 后记

以上的叙述中并没有补充 U-boot 及 AXD，RVDEBUG 的相关知识，有问题的时候可以看 U-boot 及 ARM 官网或者 google。

如果有任何建议和问题，请到 MCUZone 论坛发帖，谢谢！

www.mcuzone.com
提供：
全系列 ARM 开发工具:仿真器，开发软件
ARM7,ARM9 开发学习板
AVR 开发工具
MSP430 开发工具
ATMEL 工业级 ARM 芯片