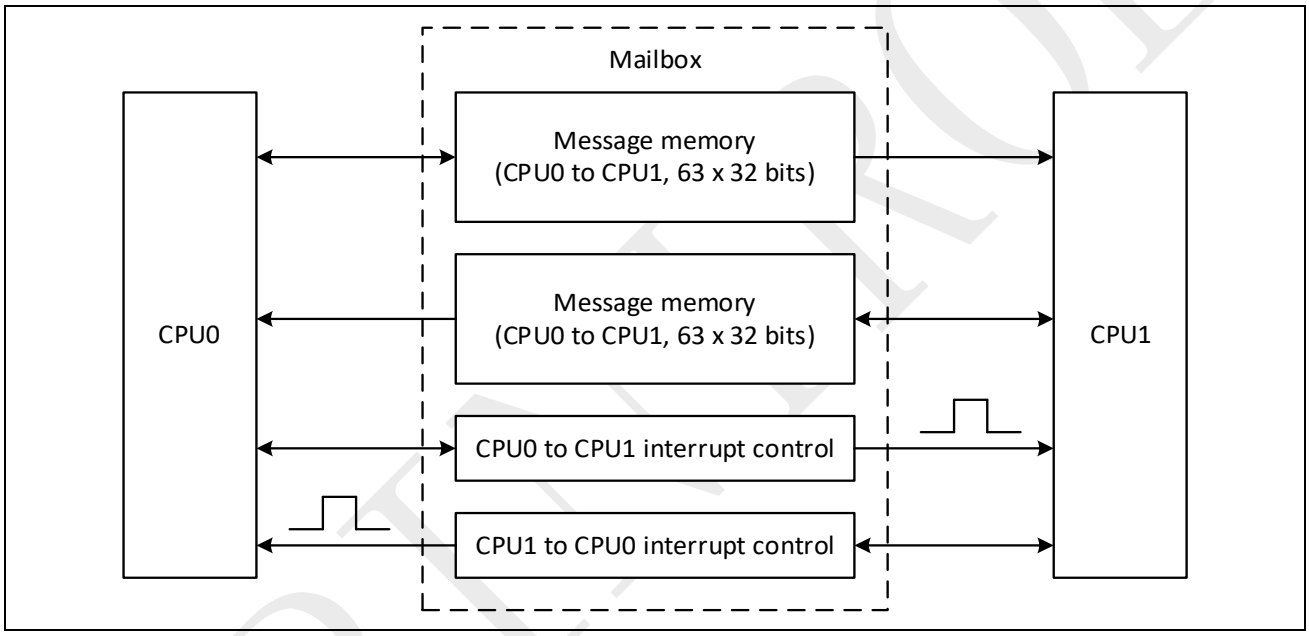
SPC2188 双核调试指南

概述

SPC2188 是双核 CPU，两颗 CPU 都是 ARM CM4 的核。CPU 之间通过 Mailbox 进行握手和共享数据（CPU0 和 CPU1 通过中断来相互请求，请求的消息内容通过 Message RAM 传递）。



目录

图片列表

表格列表

版本历史

版本	日期	作者	变更
A/0	2023-08-31	X.He	首次发布。
A/1	2023-09-10	X.He	1. 添加章节 4JTAG 菊花链调试；
A/2	2023-11-20	X.He	1. 添加章节 4.3Vscode 工程菊花链调试；

术语或缩写

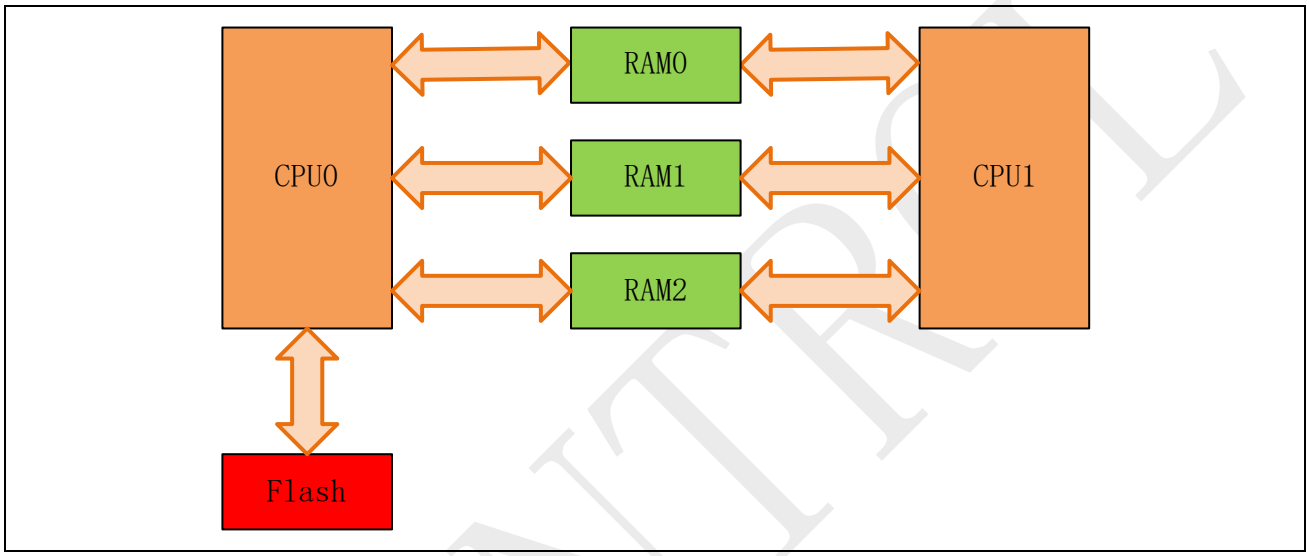
术语或缩写	描述

1 CPU1 代码执行过程

如图 1-1 所示，对于存储区域而言，CPU1 只能访问 RAM 空间，并没有访问 Flash 的能力，所以 CPU1 的代码在存储到 Flash 之后，只能依靠 CPU0 将其代码搬运到 CPU1 可以访问的 RAM 区域，然后有 CPU0 配置 CPU1 使能位，让 CPU1 开始执行其代码。

本手册的后续章节，将对如何烧录 CPU1 程序以及调试方法作详细说明。

图 1-1：CPU0 和 CPU1 访问存储器权限

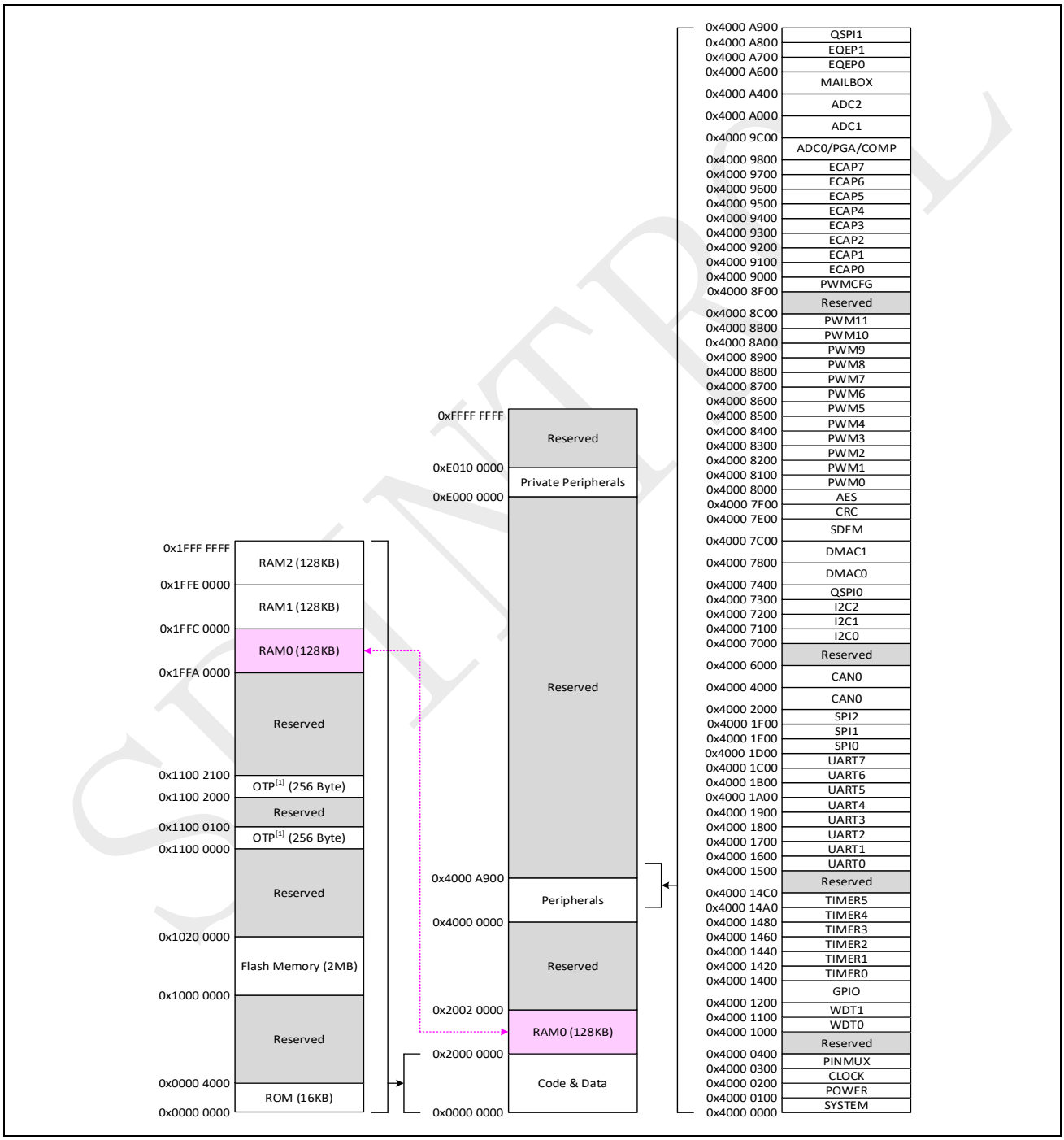


2 双核代码烧录

2.1 采用 ISP Download Tool 烧录镜像

本小节采用的范例是 SDK Demo 中的 CPU0_Write_to_CPU1 以及 CPU1_Write_to_CPU0 两个示例工程，以此来说明使用 ISP Download Tool 下载双核程序时 CPU0 和 CPU1 的工程配置。

图 2-1：SPC2188 内存映射图

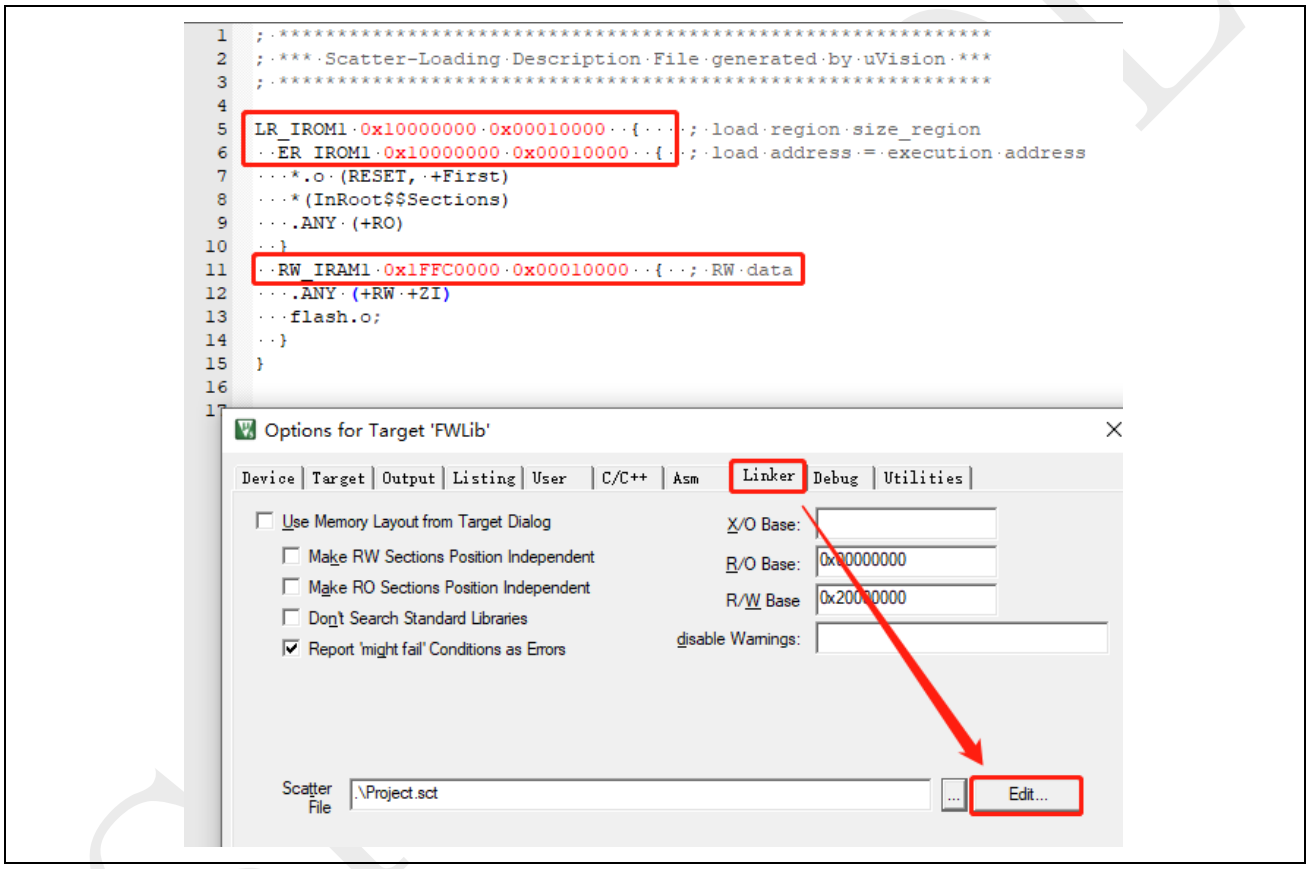


如图 2-1：所示，CPU0 和 CPU1 可以 RAM0、RAM1、RAM2，所以可访问的地址空间范围是 0x1FFFA0000~0x20020000(0x20000000~0x2002000 和 0x1FFA0000~0x1FFB0000 是使用同一块 RAM 实体)，仅 CPU0 可以访问的 Flash 地址空间范围是 0x10000000~0x10200000。

2.1.1 CPU0 的工程配置

在 SDK Demo 中的 CPU0_Write_to_CPU1 示例工程中，如图 2-2 所示，CPU0 的代码加载域和执行域是以 0x10000000 地址开始，长度为 0x10000 的 Flash 区域，数据堆栈区域则为 0x1FFC0000 地址开始，长度为 0x10000 的 RAM 区域。

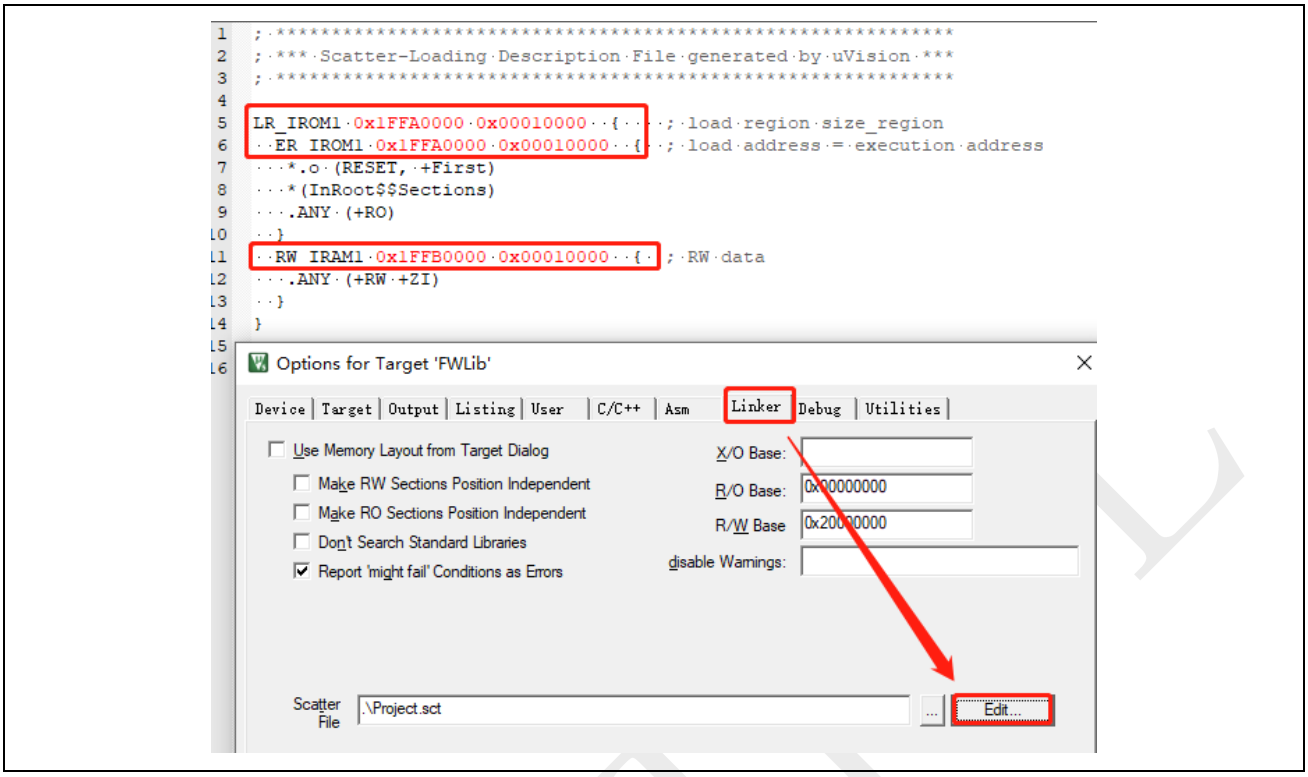
图 2-2：CPU0 工程配置



2.1.2 CPU1 工程配置

在 SDK Demo 中的 CPU1_Write_to_CPU0 示例工程中，如图 2-3 所示，CPU1 的代码加载域和执行域是以 0x1FFA0000 地址开始，长度为 0x10000 的 RAM 区域，数据堆栈区域则为 0x1FFB0000 地址开始，长度为 0x10000 的 RAM 区域。

图 2-3：设置 CPU0 程序在 RAM 中运行



2.1.3 CPU0 搬移 CPU1 镜像

如章节 1 描述，首先 CPU1 所属代码需要借由 CPU0 从 flash 搬运至 CPU1 可以访问的 RAM 地址中，CPU0 搬移动作的代码如图 2-4 所示，搬运完 CPU1 代码之后，需要使能 CPU1，此时 CPU1 开始执行程序。

图 2-4：CPU0 搬运 CPU0 程序

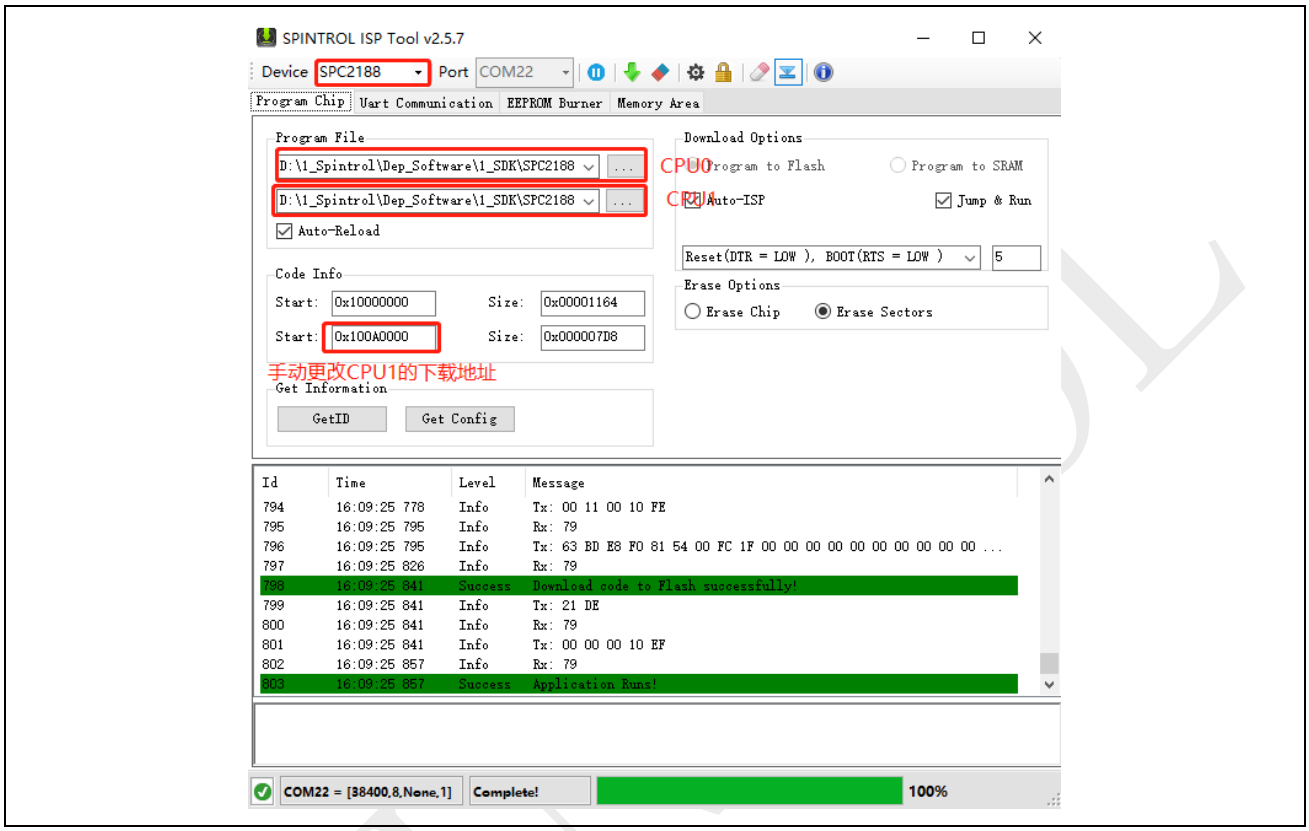
```
79  /*
80   * Init the UART
81   */
82  PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
83  PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
84  UART_Init(UART0, 38400);
85
86  Delay_Init();
87
88  u32Flag = 0;
89  u32Data = 0;
90
91  printf("Enter CPU0...\n");
92
93  u32AddrSrc = CPU1_CODE_START_ADDR;
94  u32AddrDst = CPU1_CODE_DEST_ADDR;
95  /* Code is initially in flash. Need to copy to the CPU1 code-RAM before
96  CPU1_CodeInit(u32AddrSrc, u32AddrDst, 1024);
97
98  /* Enable CPU1 */
99  SYSTEM_RunCPU1(RAM_0_BASE);
100
101 NVIC_EnableIRQ(MAILBOX_IRQn);
102
103 /* Set the message that CPU0 to CPU1 */
104 MAILBOX_CPU0Write32BitMessageToCPU1(0, 0x11223344U);
105
```

2.1.4 烧录

在章节 2.1.1 以及章节 2.1.2 **错误!未找到引用源。**已经介绍了 CPU0 以及 CPU1 的工程配置情况，在工程编译之后，就可以使用 ISP Download Tool 烧录镜像。

如图 2-5 所示，当选择 Decie 为 SPC2188 后，在“Program File”中分别选择 CPU0 和 CPU1 工程编译之后的 hex 文件，然后在“Code Info”中手动更改属于 CPU1 的 hex 文件的加载地址（CPU0 的加载地址无需更改）。

图 2-5：ISP Download Tool



2.2 采用 JLink 烧录镜像

在开始介绍 JLink 烧录镜像之前，建议用户首先阅读《SPC2188 JLink 使用指南》手册，了解 JLink 的基本使用方法。本手册将着重介绍使用 Jlink 方式烧录及调试 CPU1 的方法。

2.2.1 CPU0 工程配置

对于 CPU0 而言，使用 Jlink 的方式进行烧录及调试可以完全参照《SPC2188 JLink 使用指南》文档中的步骤进行操作。

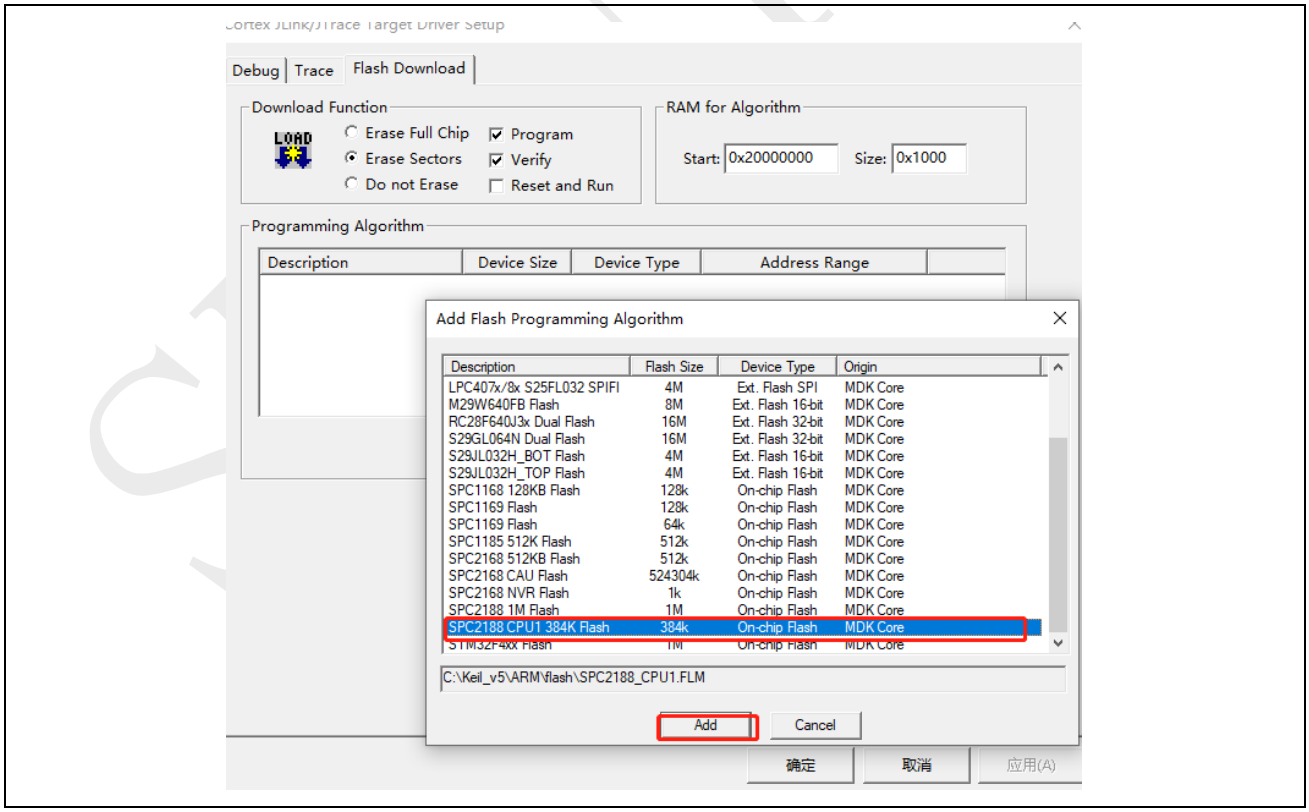
2.2.2 CPU1 工程配置

在详细介绍 CPU1 的 JLink 烧录步骤之前，首先需要把 SPC2188_CPU1.FLM 这个文件拷贝至 Keil 安装目录下的 **ARM\Flash**文件夹下，另外，SPC2188_CPU1.FLM 算法文件将会把 CPU1 的镜像烧录到 Flash 0x100A0000~0x100FFFFF 384KB 的大小的 Flash 区域内。

- **添加 Flash 算法文件**

如图 2-6 所示，选择 SPC2188_CPU1.FLM 算法文件。

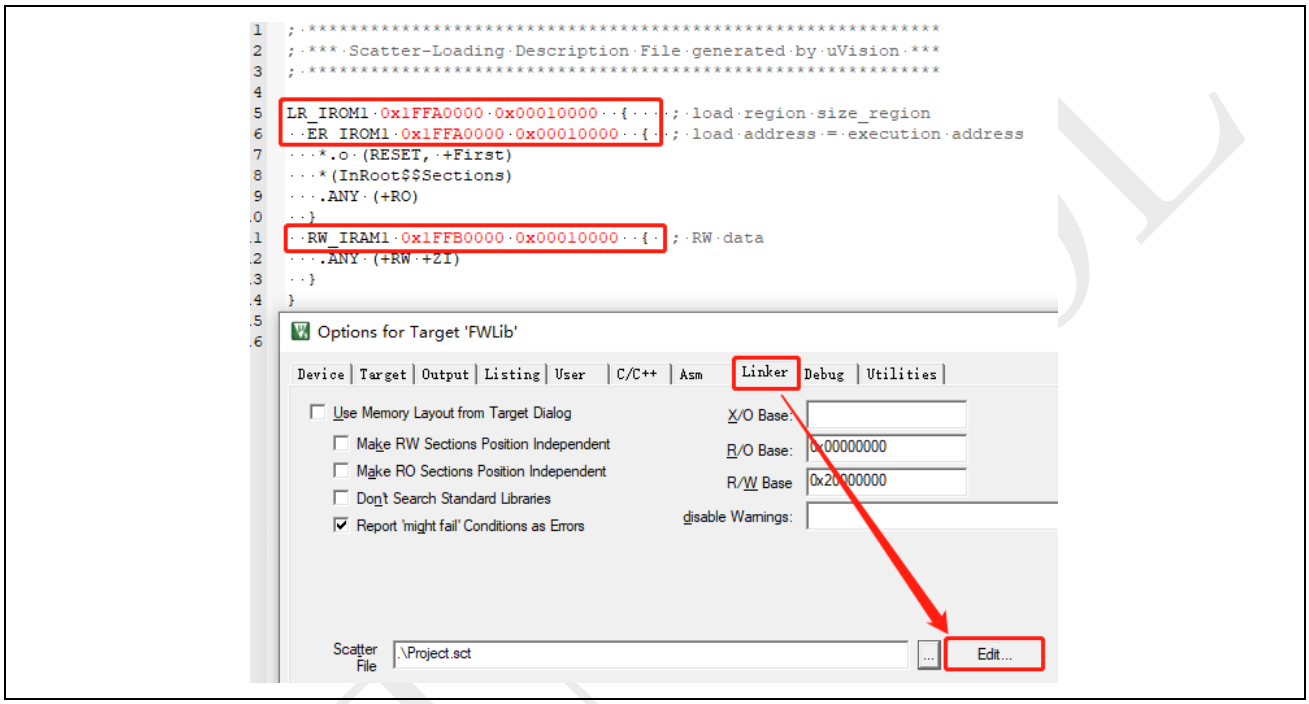
图 2-6：CPU1 工程添加 Flash 算法文件



- **配置加载地址、执行地址、数据堆栈地址**

CPU1 工程的加载地址和执行地址都必须配置在 0x1FFA0000~0x1FFFFFFF 的 RAM 地址，并且加载地址和执行地址必须 16KB 地址对齐。如图 2-7 所示，CPU1 工程的分散文件中配置其加载及运行地址均为 0x1FFA0000~0x1FFB0000，但 SPC2188_CPU1.FLM 算法文件会将加载地址映射至 0x100A0000~ 0x100FFFFF，也就是说，CPU1 工程的代码实际上存储的位置是 0x100A0000~0x100FFFFF 地址段，这一点需要特别注意。

图 2-7：CPU1 工程配置



当以上步骤都操作完毕，就可以使用 SWD 或者 JTAG 烧录 CPU1 镜像。

注意： 烧录 CPU1 的镜像需要使用是 CPU0 的 SWD 或者 JTAG 接口，因为只有 CPU0 具有访问 Flash 的权限。

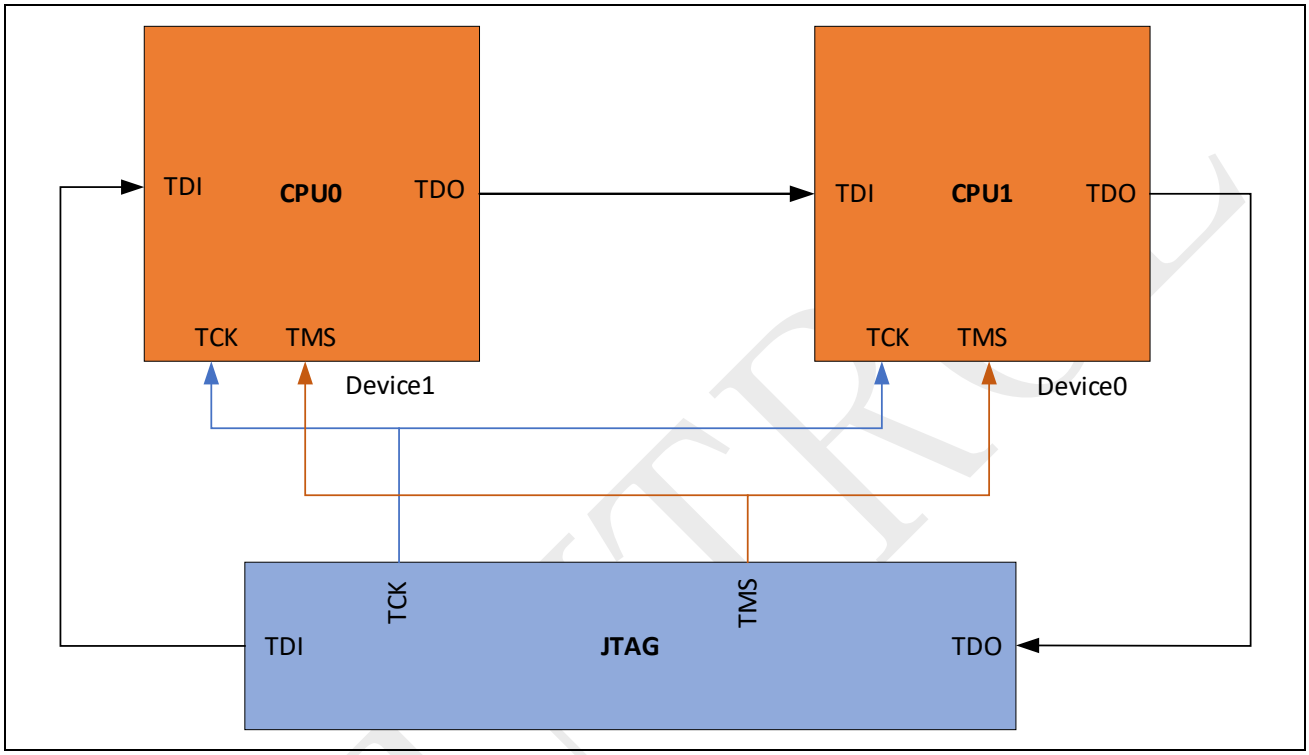
3 CPU1 程序仿真

CPU1 的程序最终是由 CPU0 把 CPU1 的程序搬运到对应的 RAM 地址，CPU1 所属的全部程序（包括代码及数据）都在 RAM 地址，所以对 CPU1 程序的仿真需要使用 RAM 调试的办法，可以参考《SPC2188 RAM 程序调试使用指南》手册。

4 JTAG 菊花链调试

SPC2188 具有双核，也支持 JTAG 菊花链，菊花链的硬件连接如图 4-1 所示中，所有设备共用 TCK 和 TMS，JTAG 连接器的 TDI 从 CPU0 的 TDI 输入，然后 CPU0 的 TDO 连接到 CPU1 的 TDI，然后 CPU1 的 TDO 连接到 JTAG 连接器的 TDO。

图 4-1：菊花链硬件连接



使能菊花链调试之前需要使能一下两个配置：

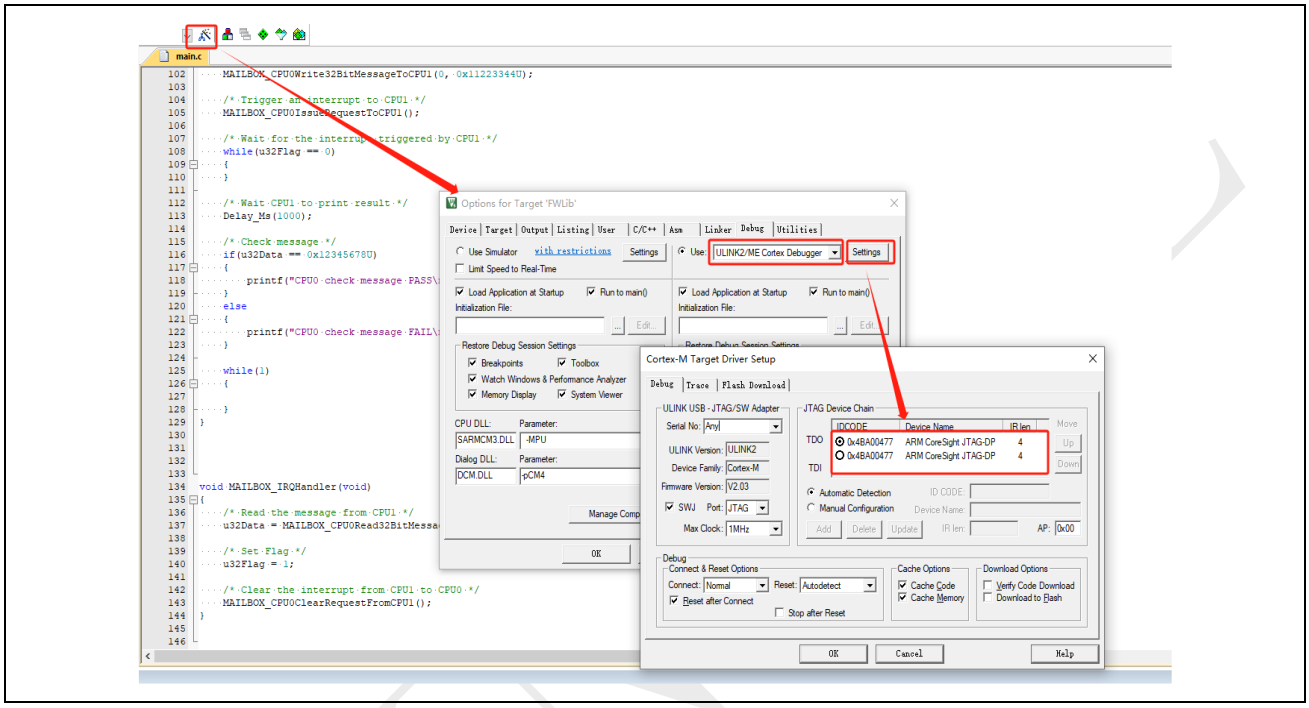
- 使能 JTAG 菊花链接口
- CPU0 和 CPU1 都必须使能

对 DBGIFCTL 寄存器的 MODE 位段配置 JTAG_CHAIN 模式可以使能 JTAG 菊花链接口，CPU0 默认是使能的，而使能 CPU1 通过配置 CPU1CTL 寄存器的使能位段进行配置，DBGIFCTL 寄存器的数据复位不丢失数据，只有断电才会丢失，CPU1CTL 寄存器复位数据会丢失，**因此在调试过程中不能进行复位外设操作（因为复位外设操作会导致 CPU1CTL 寄存器的数据丢失），仅可以复位 Core**。

4.1 KEIL 工程菊花链调试（使用 ULink）

BOOT 引脚拉高，TRSR 拉高，芯片进入调试模式，然后根据图 4-2 步骤，可以查看到 KEIL 识别到两个设备节点，如果 Keil 没有识别到节点，需要检查调试工具和芯片是否正常连接、JTAG 菊花链的调试接口是否使能、CPU1 是否使能。

图 4-2：查询设备节点

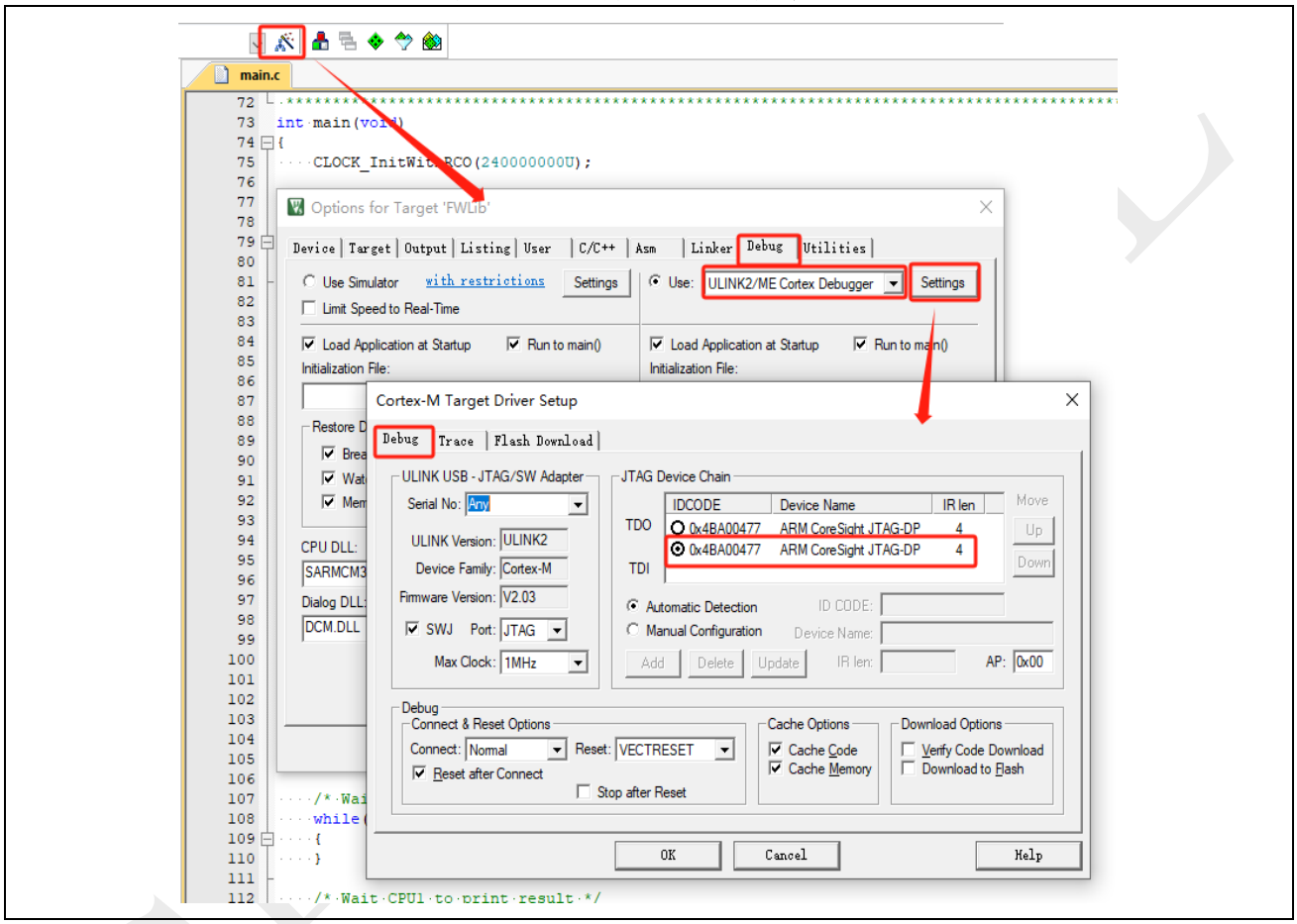


4.1.2 CPU0 工程配置

- **选择 CPU0 的设备节点**

SPC2188 双核菊花链连接如图 4-1 所示，CPU0 核靠近调试器 TDI 引脚，所以在 KEIL 中选择 CPU0 核时应选择靠近 TDI 的设备节点，如图 4-3 所示。

图 4-3：选择 CPU0 设备节点



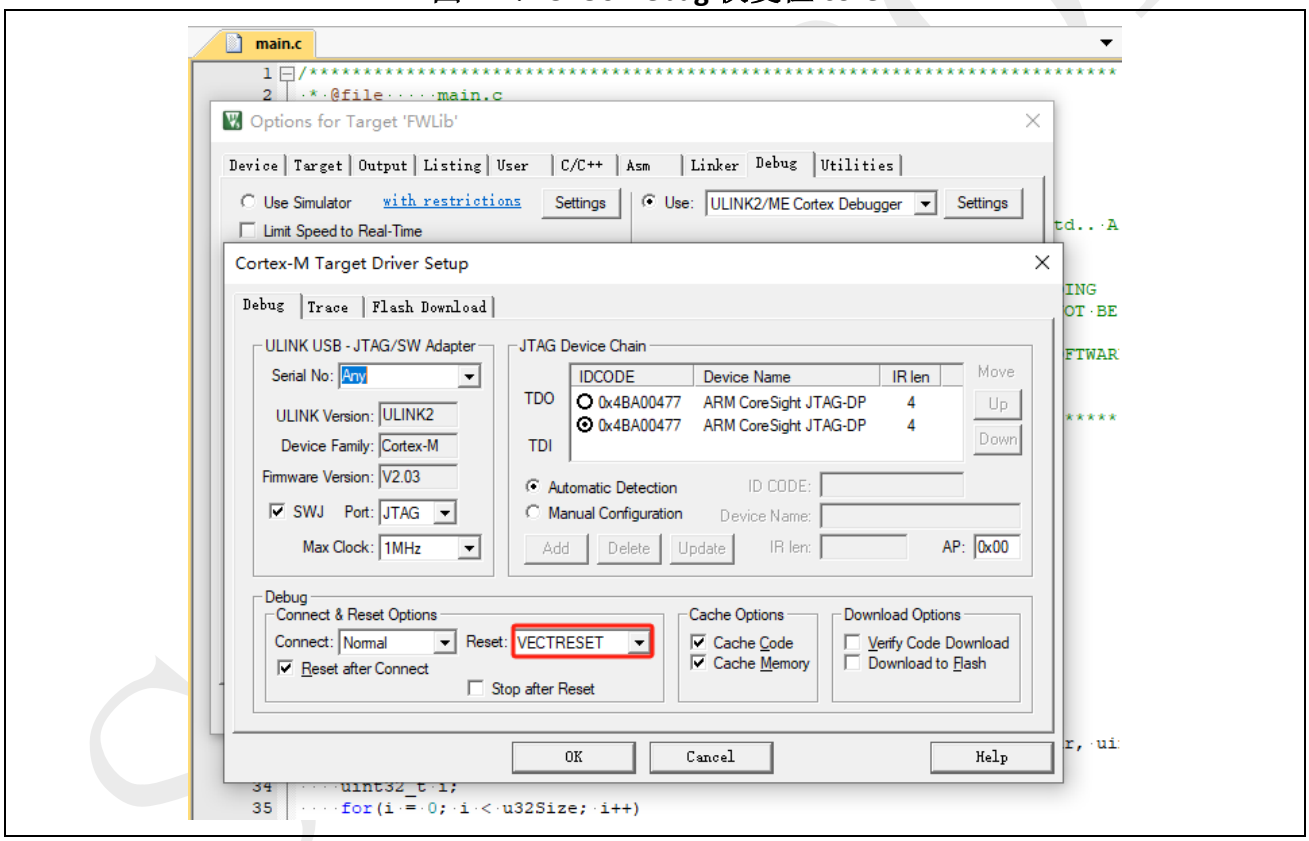
- **选择 Debug 时，仅复位 core**

使用 Ulink 调试器进行 Debug 时，有四种复位方式：

- **Autodetect**：根据目标设备选择 HW RESET、SYSRESETREQ、VECTRESET 其中一种复位类型；
- **HW RESET**：通过置位调试器的 TREST 引脚信号来对芯片进行复位，core 和外设都会复位；
- **SYSRESETREQ**：软件进行系统复位，core 和外设都会复位；
- **VECTRESET**：软件复位，仅复位 core；

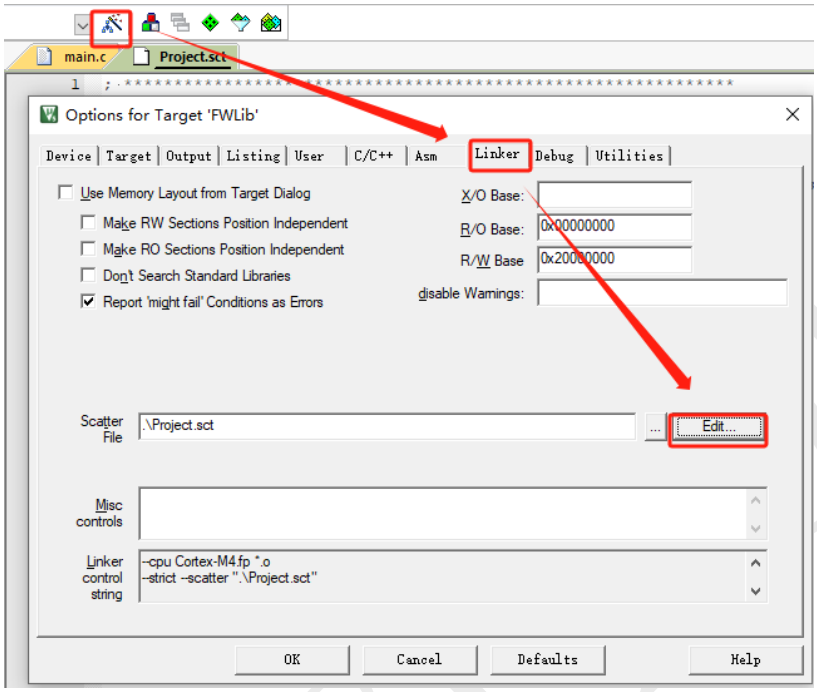
如果进行复位外设操作，会将 CPU1 的使能位进行复位，导致 CPU1 失能，菊花链断开，所以菊花链 Debug 时只能选择 VECTRESET 复位。

图 4-4：CPU0 Debug 仅复位 core



- 配置 CPU0 的 Flash 下载地址

图 4-5：配置 CPU0 Flash 地址



点击”Edit”按钮，会出现如下文件，其中运行地址和加载地址都设置为 0x10000000 地址，因此 CPU0 的 code 存放在 0x10000000 的 Flash 地址，数据地址为 0x1FFC0000 的 RAM 地址。

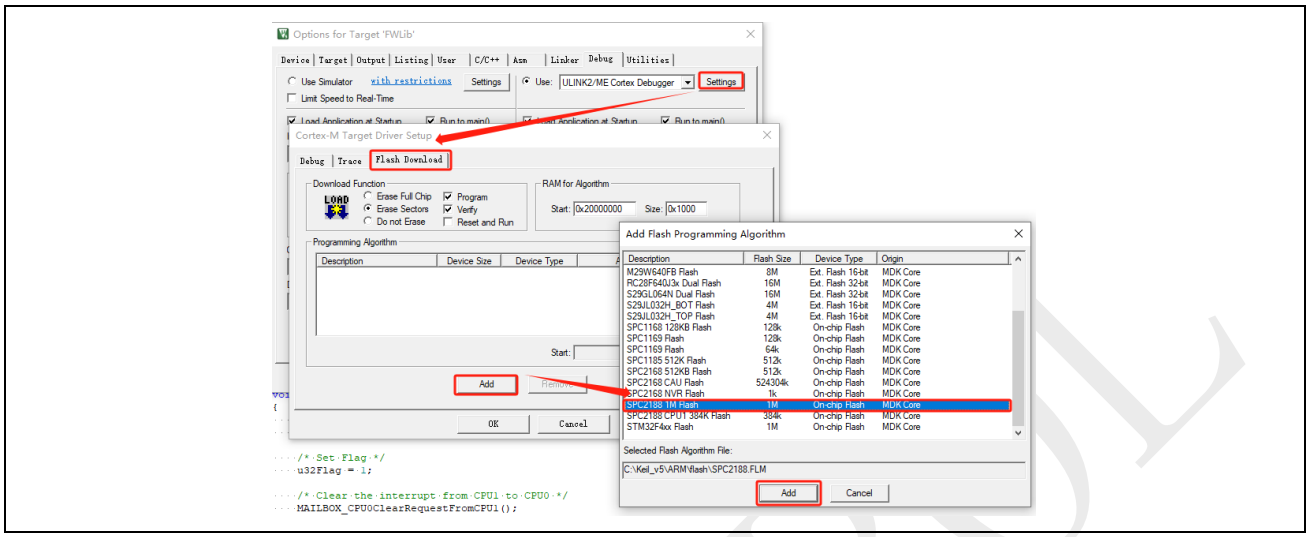
Project.sct 文件

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x10000000 0x00010000  {    ; load region size_region
  ER_IROM1 0x10000000 0x00010000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x1FFC0000 0x00010000  {  ; RW data
   .ANY (+RW +ZI)
   flash.o;
  }
}
```

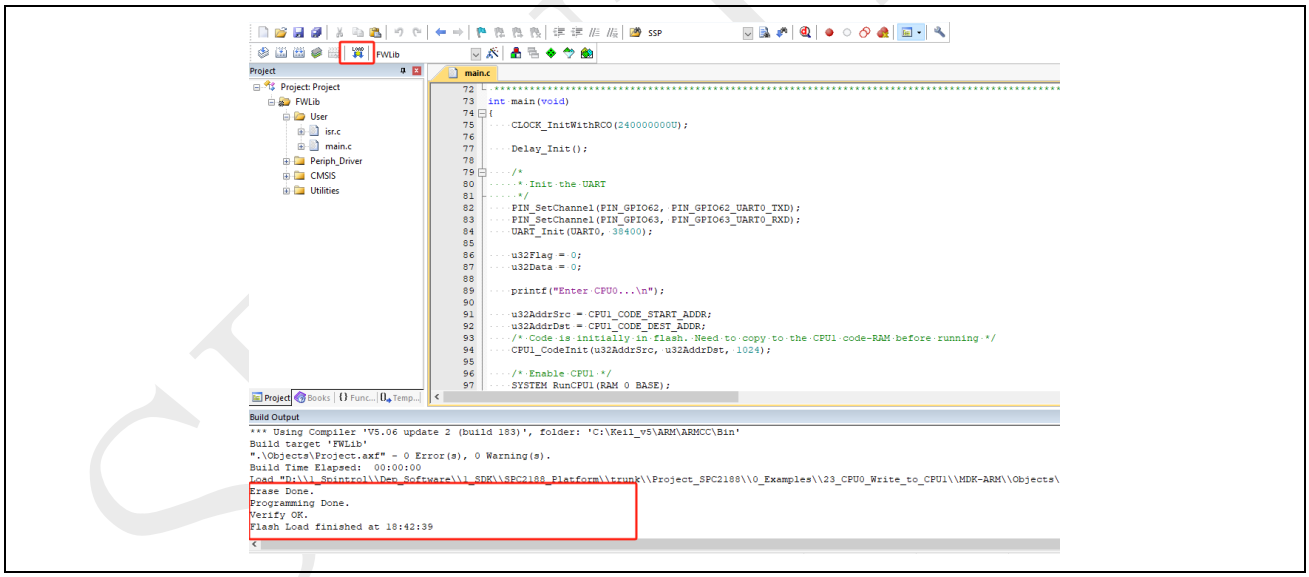
- 选择 CPU0 的算法文件

图 4-6：选择 CPU0 的 Flash 算法文件



- 烧录 CPU0 的程序到 Flash

图 4-7：烧录 CPU0 的程序到 Flash



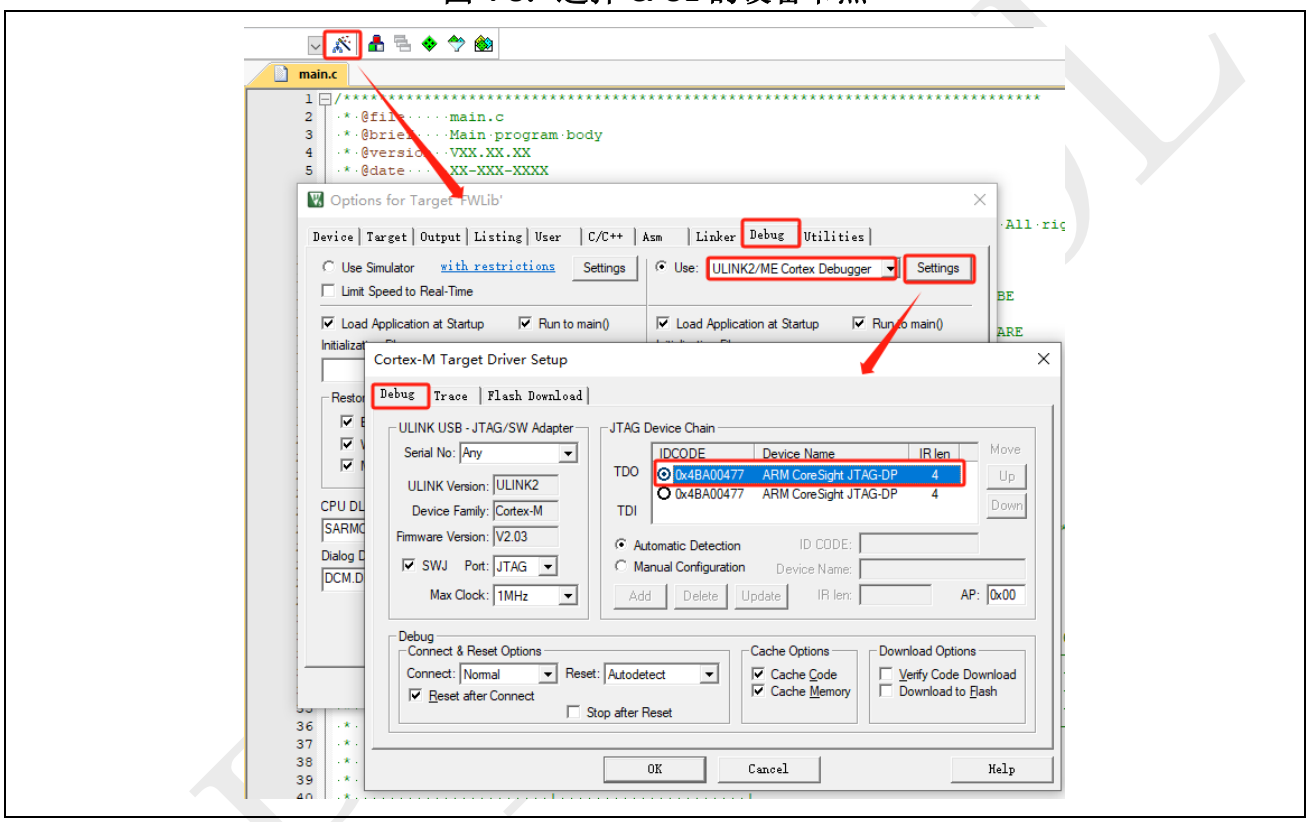
4.1.2 CPU1 工程配置

由于 CPU1 是没有访问 Flash 的权限，所以 CPU1 的程序只能在 RAM 中运行，也可以选择 CPU0 的节点先将 CPU1 的代码下载到 Flash，然后由 CPU0 将程序搬运到 RAM 中供 CPU1 执行。

- **选择 CPU1 的设备节点**

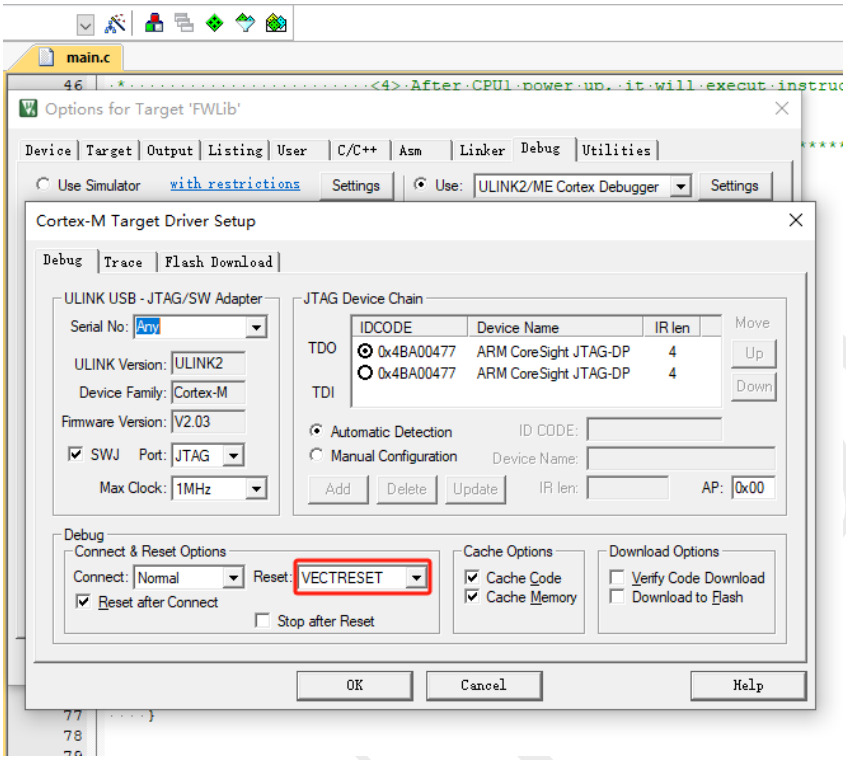
SPC2188 双核菊花链连接如图 4-1 所示，CPU1 核靠近调试器 TDO 引脚，所以在 KEIL 中选择 CPU1 核时应选择靠近 TDO 的设备节点，如图 4-8 所示。

图 4-8：选择 CPU1 的设备节点



- 选择 Debug 时，仅复位 core

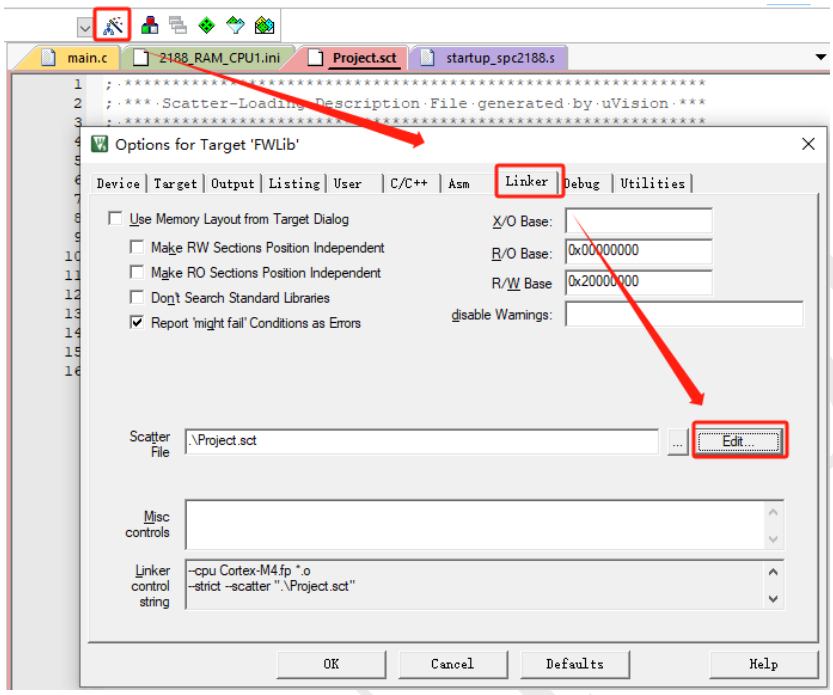
图 4-9：CPU1 Debug 仅复位 core



- 配置 CPU1 的 RAM 的下载地址

由于 CPU1 没有访问 Flash 资源的权限，需要将 CPU1 的代码放在 RAM 中进行调试。

图 4-10：配置 CPU1 的 RAM 下载地址



点击”Edit”按钮，会出现如下文件，其中运行地址和加载地址都设置为 0x1FFA0000 地址，因此 CPU1 的 code 存放在 0x1FFA0000 的 RAM 地址，数据地址为 0x1FFB0000 的 RAM 地址。

注意：　CPU0 使能 CPU1 的时候需要配置 CPU1 的起始地址为 CPU1 的代码段的起始地址。

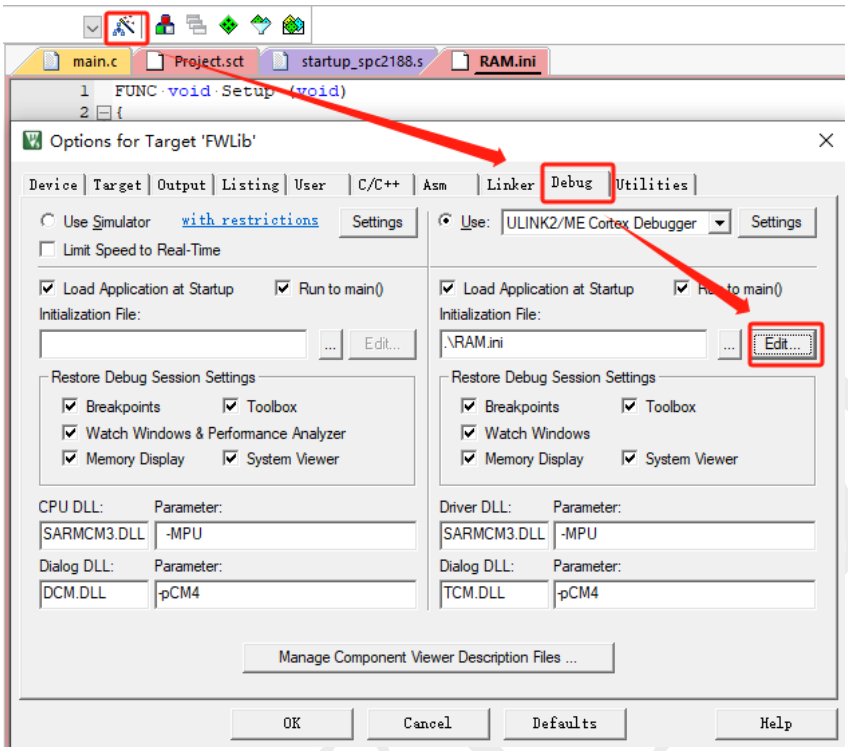
Project.sct 文件

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x1FFA0000 0x00010000  {    ; load region size_region
  ER_IROM1 0x1FFA0000 0x00010000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x1FFB0000 0x00010000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

- 配置 CPU1 的 RAM.ini 文件

图 4-11：配置 CPU1 的 RAM.ini 文件



点击”Edit”按钮，在打开的文件中添加所示内容：

RAM.ini 文件

```
FUNC void Setup (void)
{
   SP = _RDWORD(0x1FFA0000);                  // Setup Stack Pointer
   PC = _RDWORD(0x1FFA0004);                  // Set Program Counter
   _WDWORD(0xE000ED08, 0x1FFA0000);           // Set Vector Table Offset Register
}

_WDWORD(0x40001024, 0x1ACCE551);              // Enable WDT0 operation
_WDWORD(0x40001124, 0x1ACCE551);              // Enable WDT1 operation
_WDWORD(0x40001008, 0x0);                     // Close WDT0
_WDWORD(0x40001108, 0x0);                     // Close WDT1

LOAD %L INCREMENTAL                           // Download to RAM
Setup();
//g, main
```

对文件中重要的几个设置做相关说明：

SP = _RDWORD(0x1FFA0000) ：调试开始时，将堆栈寄存器初始值设置为 0x1FFA0000。

PC = _RDWORD(0x1FFA0004)：调试开始时，将 PC 设置为 0x1FFA0004。

_WDWORD(0xE000ED08, 0x1FFA0000)：调试开始时，将中断向量表偏移寄存器（地址为 0xE000ED08）设置为 0x1FFA0000。

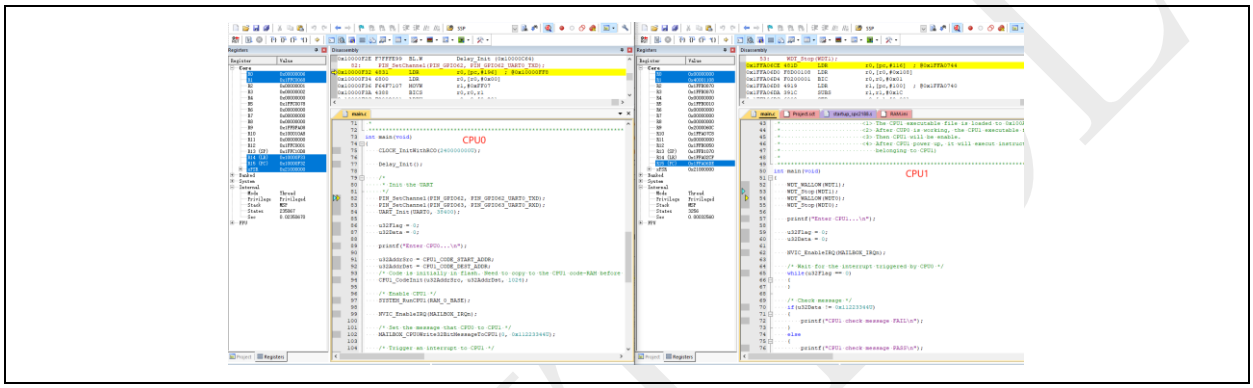
以及关闭 CPU1 的 WTD 防止在调试过程中产生复位。

4.1.1 菊花链调试

- 开始调试

根据上述方法进行完 CPU0 和 CPU1 的配置后，分别点击 CPU0 和 CPU1 工程的 Debug 按钮，就可以开始进行调试了。

图 4-12：Keil 菊花链调试



4.2 IAR 工程菊花链调试（使用 JLink）

本小节将以 CPU0 代码下载到 Flash、CPU1 代码下载到 RAM 为例。

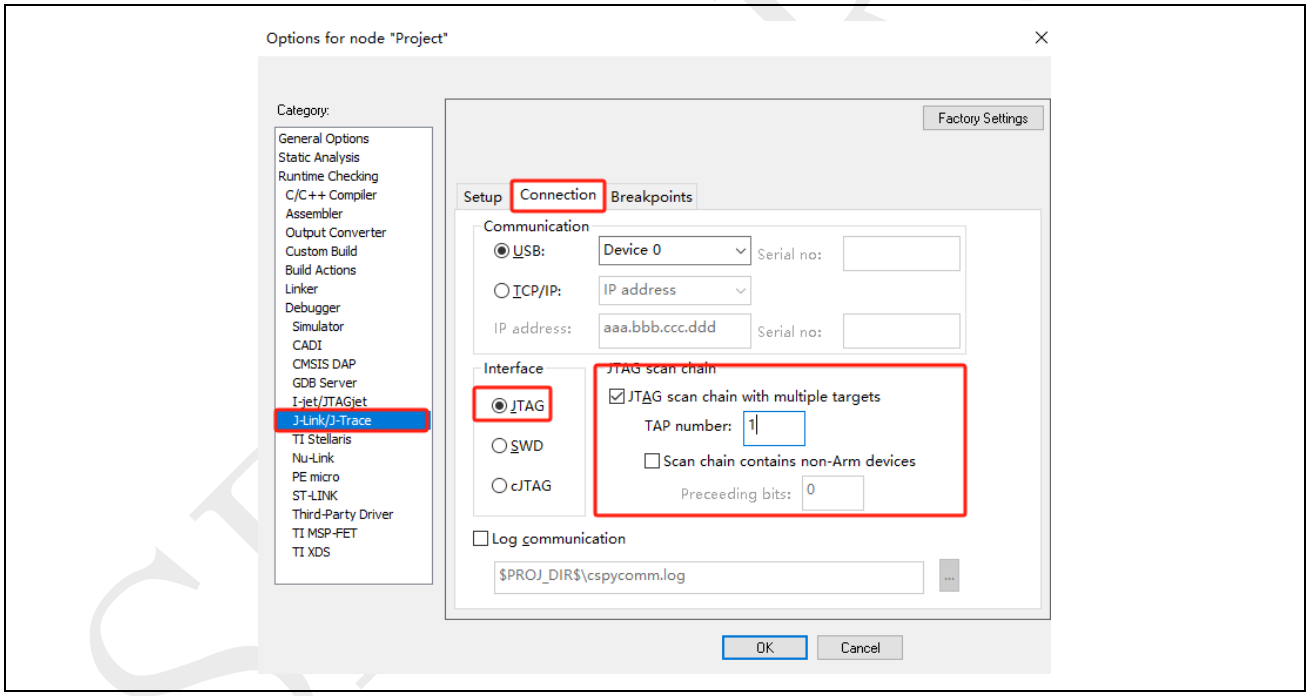
4.2.1 CPU0 工程配置配置

- **选择 CPU0 的设备节点**

在 Project -> Options -> J-Link/J-Trace -> Connection 选项卡下，选择 JTAG 接口，在 JTAG scan chain 框下选中“JTAG scan chain with multiple targets”，“TAP number”选择 1，由如图 4-1 所示。

注意： TAP 的定义为：靠近 TDO 的设备编号为 0，远离 TDO 的设备编号依次加 1，所以 CPU0 的 TAP 编号为 1。

图 4-13：选择 CPU0 设备节点

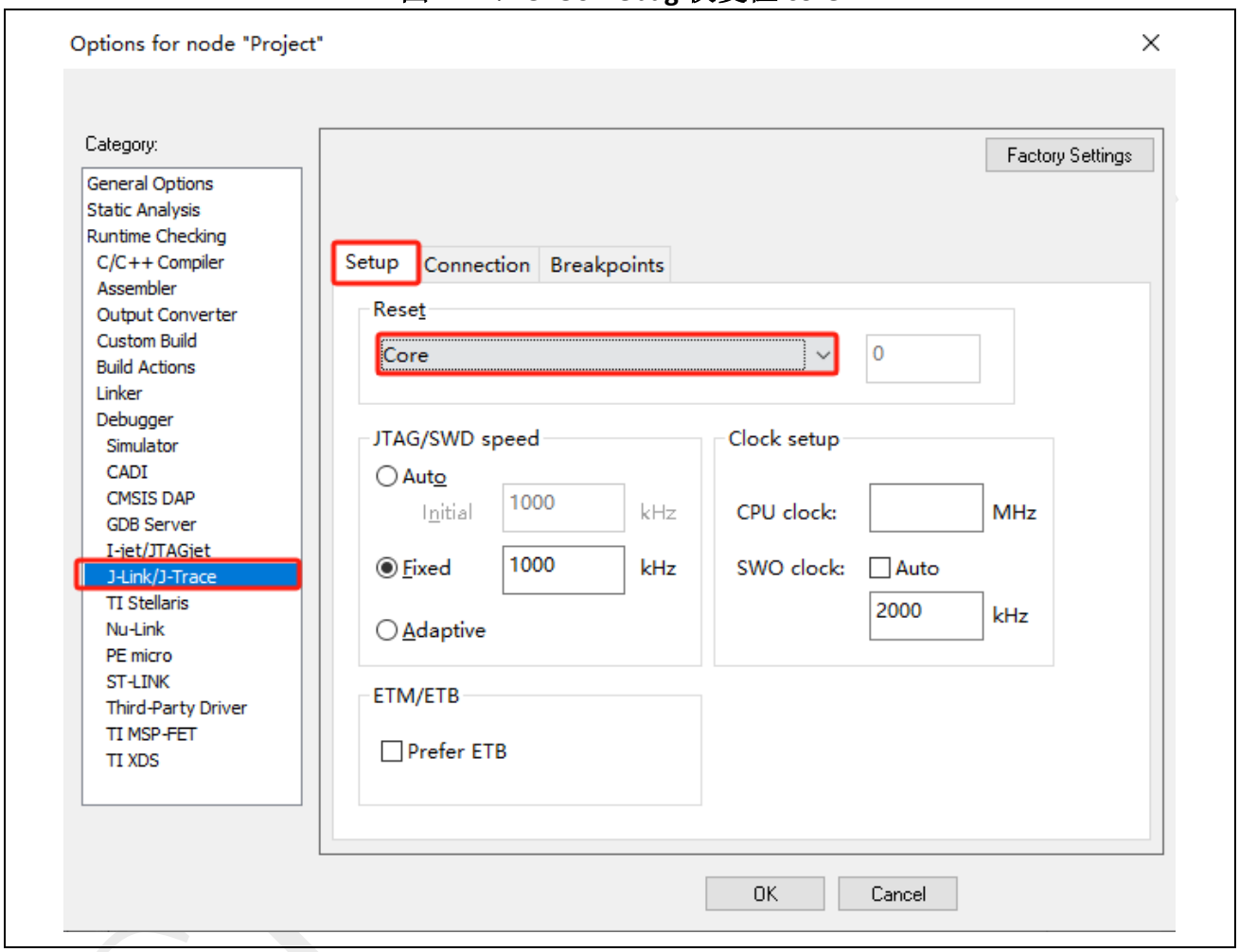


- **选择 Debug 时，仅复位 core**

在 Project -> Options -> J-Link/J-Trace -> Setup 选项卡下，选择仅复位 core。

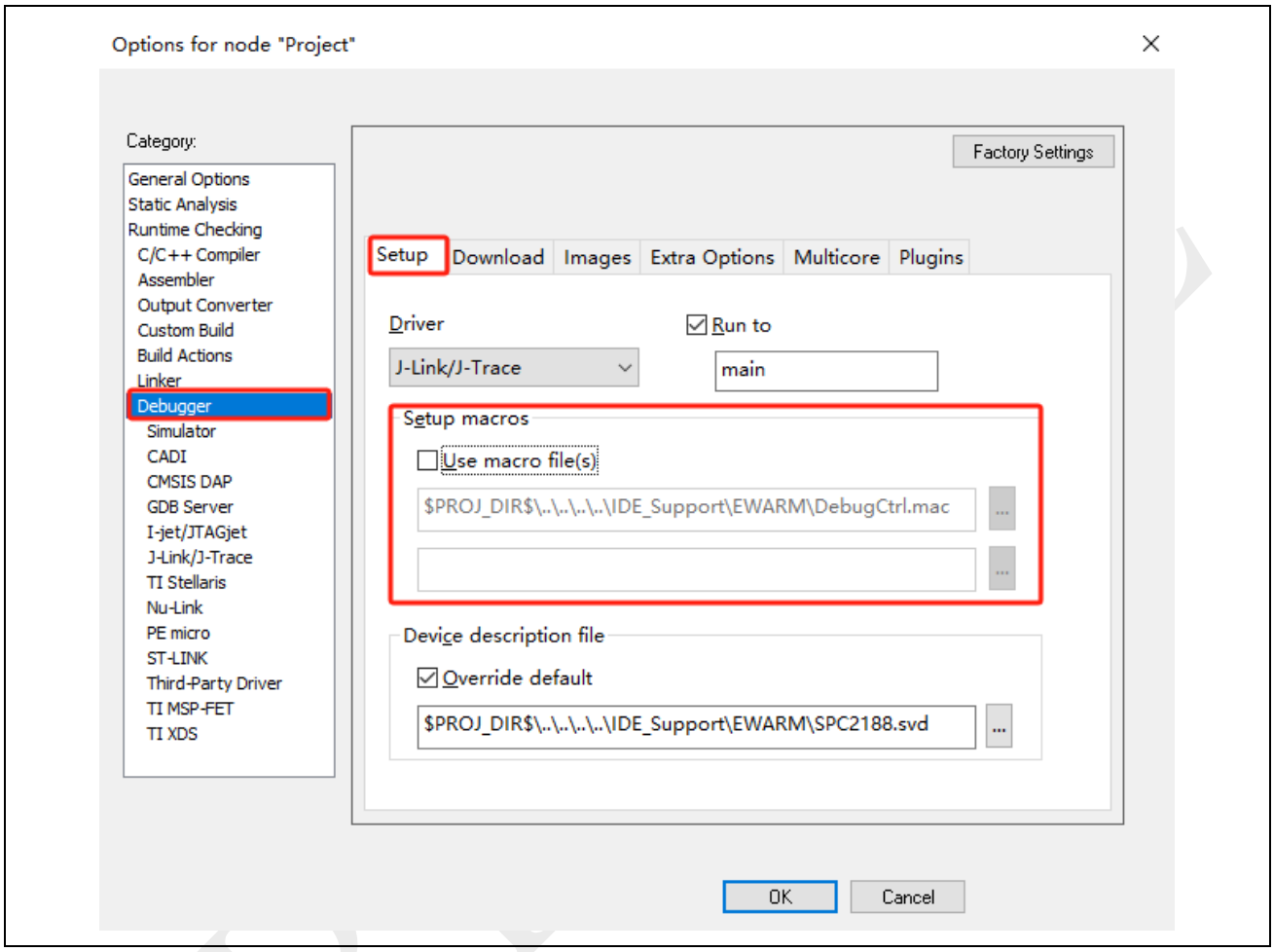
如果进行复位外设操作会将 CPU1 的使能位进行复位，导致 CPU1 失能，菊花链断开，所以菊花链 Debug 时只能选择仅 core 复位。

图 4-14：CPU0 Debug 仅复位 core



在 Project -> Options -> Debugger -> Setup 选项卡下，取消“Use macro file(s)”的选项，该文件会重新配置 Flash 的接口，因为菊花链调试只复位 Core，外设不会进行复位，所以不需要重新配置 Flash。

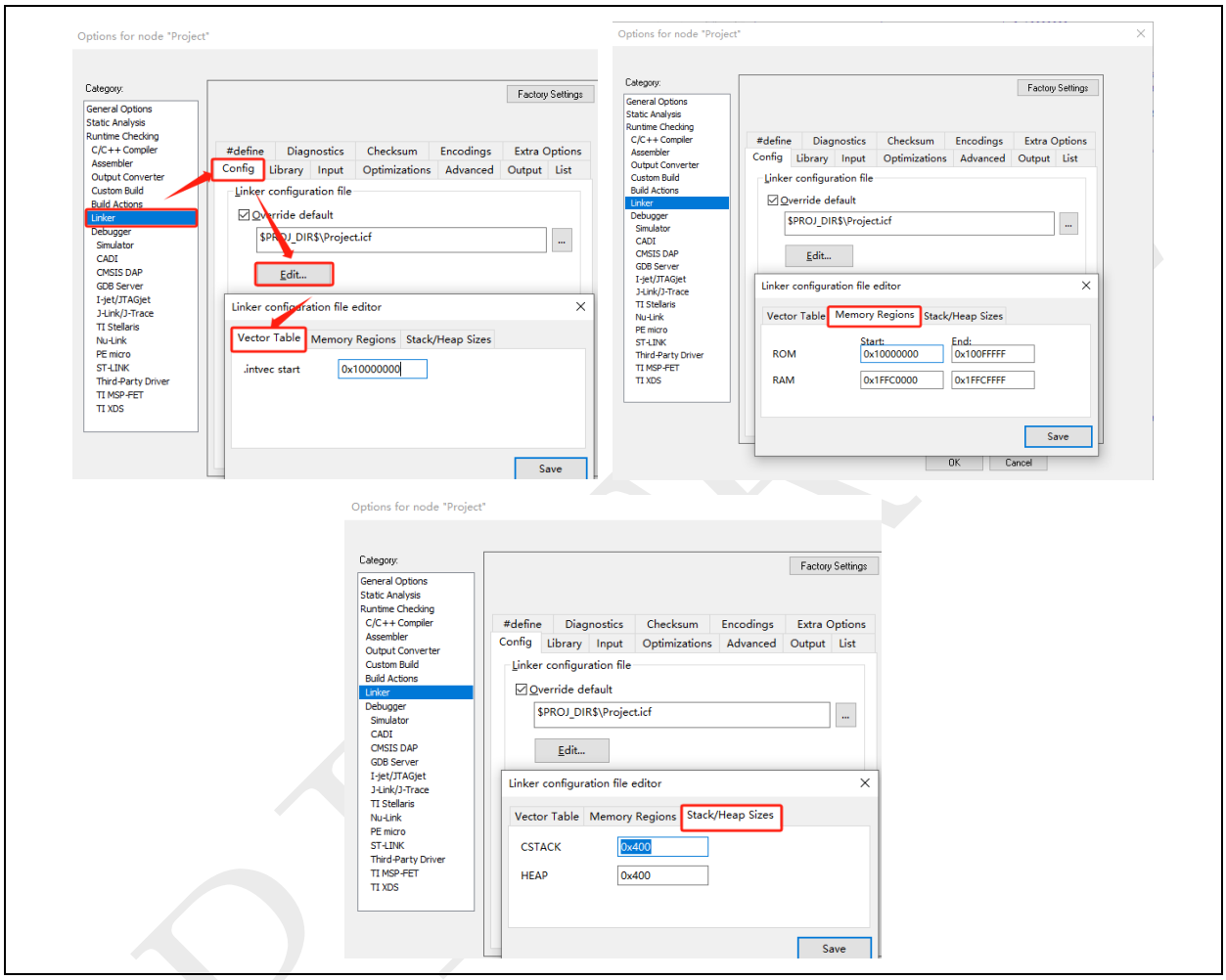
图 4-15：取消 Use macro file 选项



- **配置 CPU0 的 Flash 下载地址**

在 Project -> Options -> Linker -> Config 选项卡下，配置异常向量表地址、Flash 下载地址、RAM 的地址以及堆栈大小。

图 4-16：配置 CPU0 Flash 地址



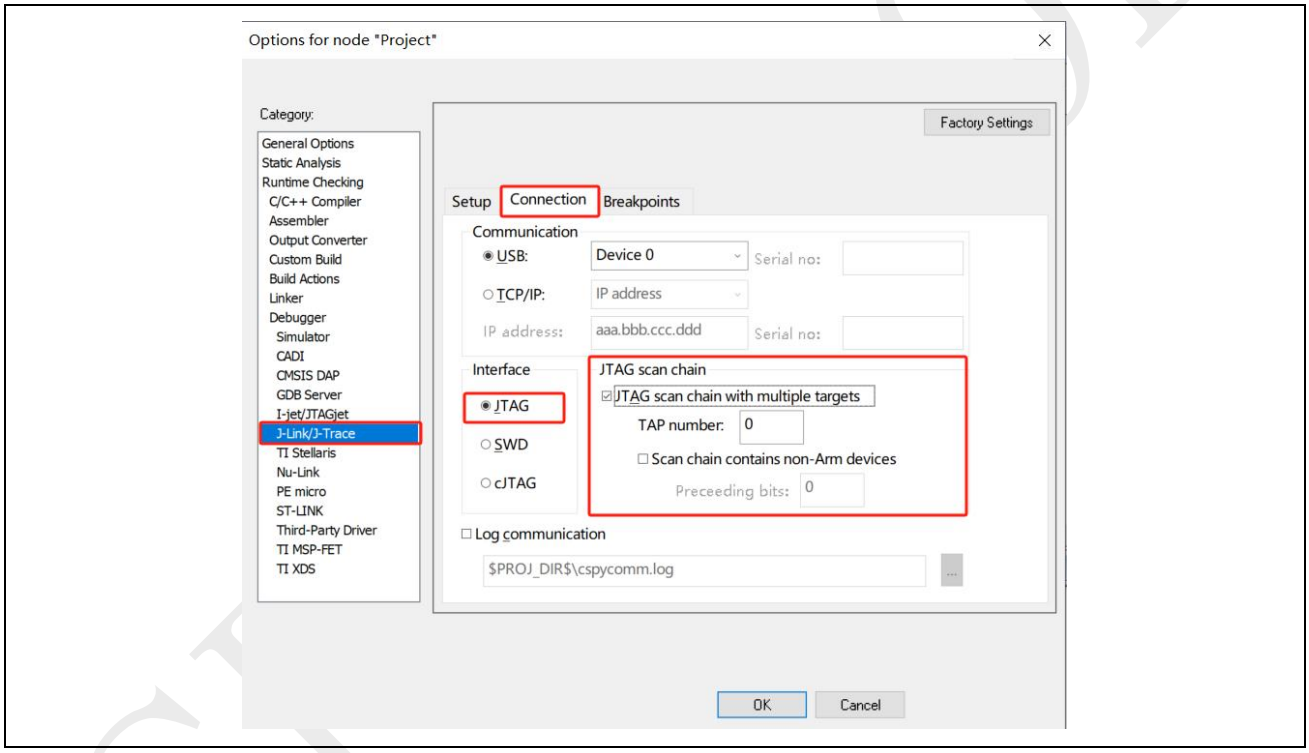
4.2.2 CPU1 工程配置

- 选择 CPU1 的设备节点

在 Project -> Options -> J-Link/J-Trace -> Connection 选项卡下，选择 JTAG 接口，在 JTAG scan chain 框下选中“JTAG scan chain with multiple targets”，“TAP number”选择 0，如图 4-1 所示。

注意： TAP 的定义为：靠近 TDO 的设备编号为 0，远离 TDO 的设备编号依次加 1，所以 CPU1 的 TAP 编号为 0。

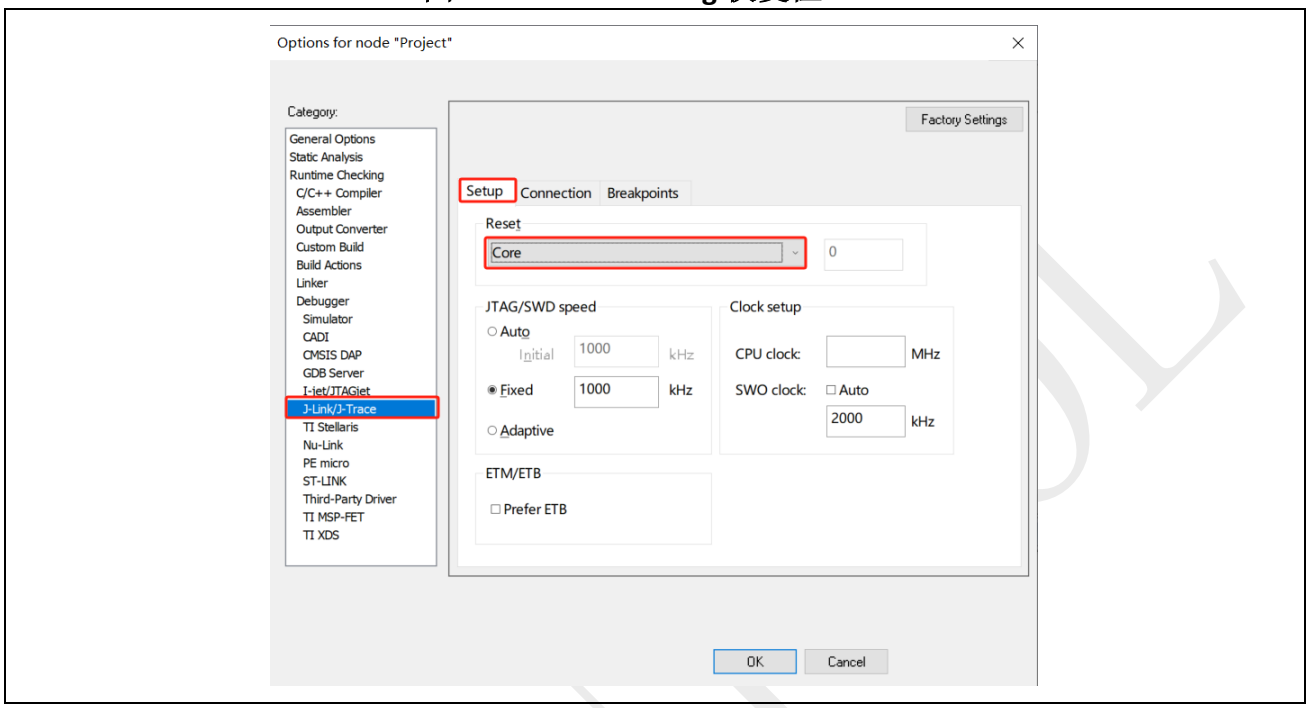
图 4-17：选择 CPU1 的设备节点



- **选择 Debug 时，仅复位 core**

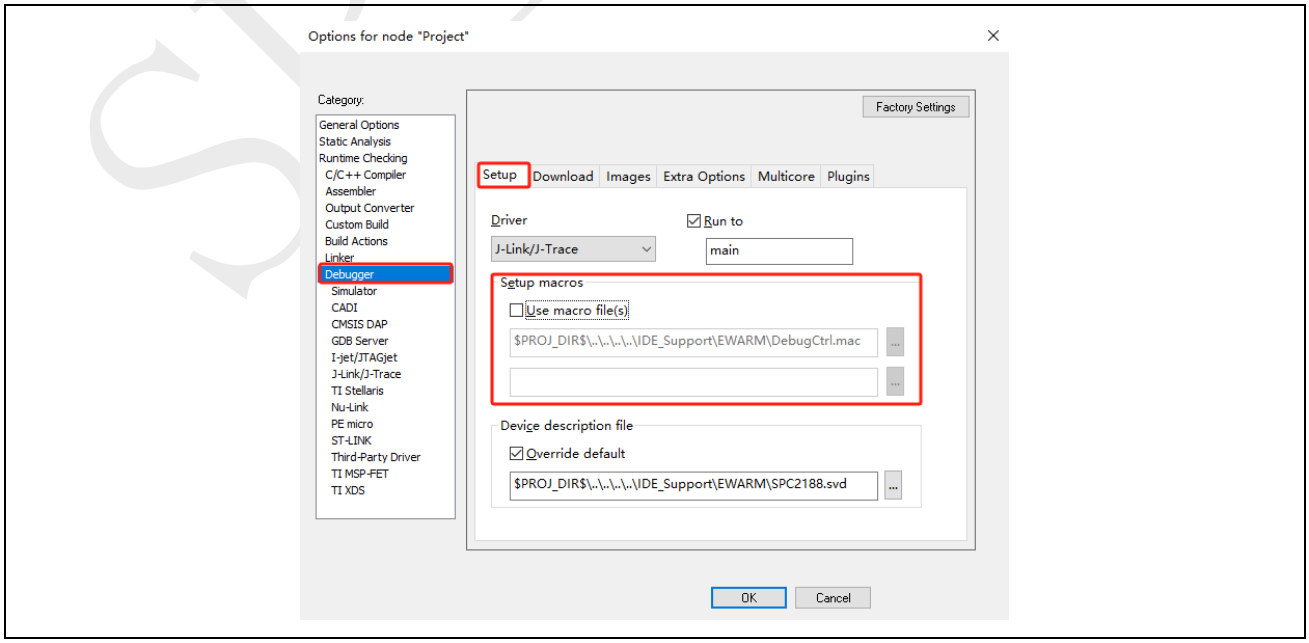
  在 Project -> Options -> J-Link/J-Trace -> Setup 选项卡下，选择仅复位 core。

图 4-18：CPU1 Debug 仅复位 core



在 Project -> Options -> Debugger -> Setup 选项卡下，取消“Use macro file(s)”的选项，该文件会重新配置 Flash 的接口，因为菊花链调试只复位 Core，外设不会进行复位，所以不需要重新配置 Flash。

图 4-19：取消 Use macro file 选项

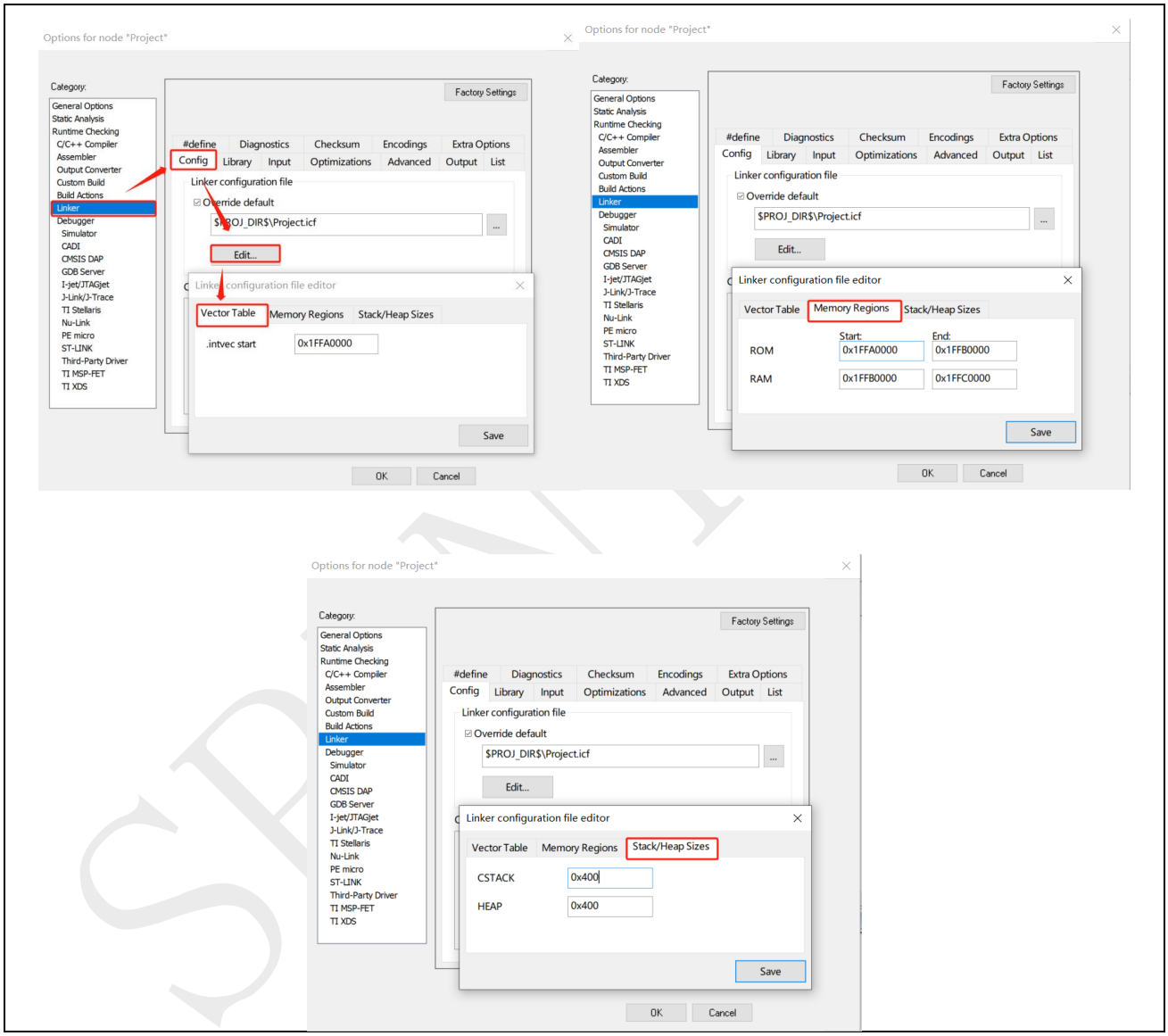


- 配置 CPU1 的 RAM 的下载地址

在 Project -> Options -> Linker -> Config 选项卡下，配置异常向量表地址、Flash 下载地址、RAM 的地址以及堆栈大小。

注意： CPU0 使能 CPU1 的时候需要配置 CPU1 的起始地址为 CPU1 的代码段的起始地址。

图 4-20：配置 CPU1 RAM 地址

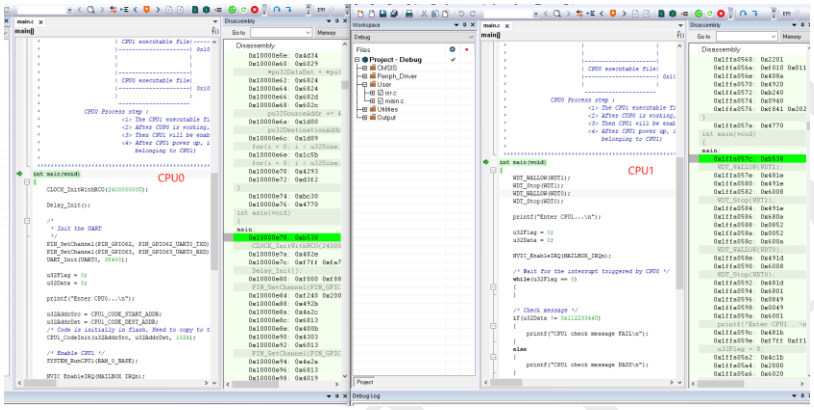


4.2.3 菊花链调试

- **开始调试**

根据上述方法进行完 CPU0 和 CPU1 的配置后，分别点击 CPU0 和 CPU1 工程的下载和仿真按钮，就可以开始进行调试了。

图 4-21：IAR 菊花链调试



4.3 Vscode 工程菊花链命令行调试（使用 JLink）

由于 SPC2188 暂时没有 Vscode 工程，本小节将以 SPD1179 为例，但同样适用于 SPC2188，说明如何使用 Vscode 工程进行菊花链调试。

Vscode 工程菊花链调试采用 GDB 命令行的形式，其常见的命令如表 4-1 所示。

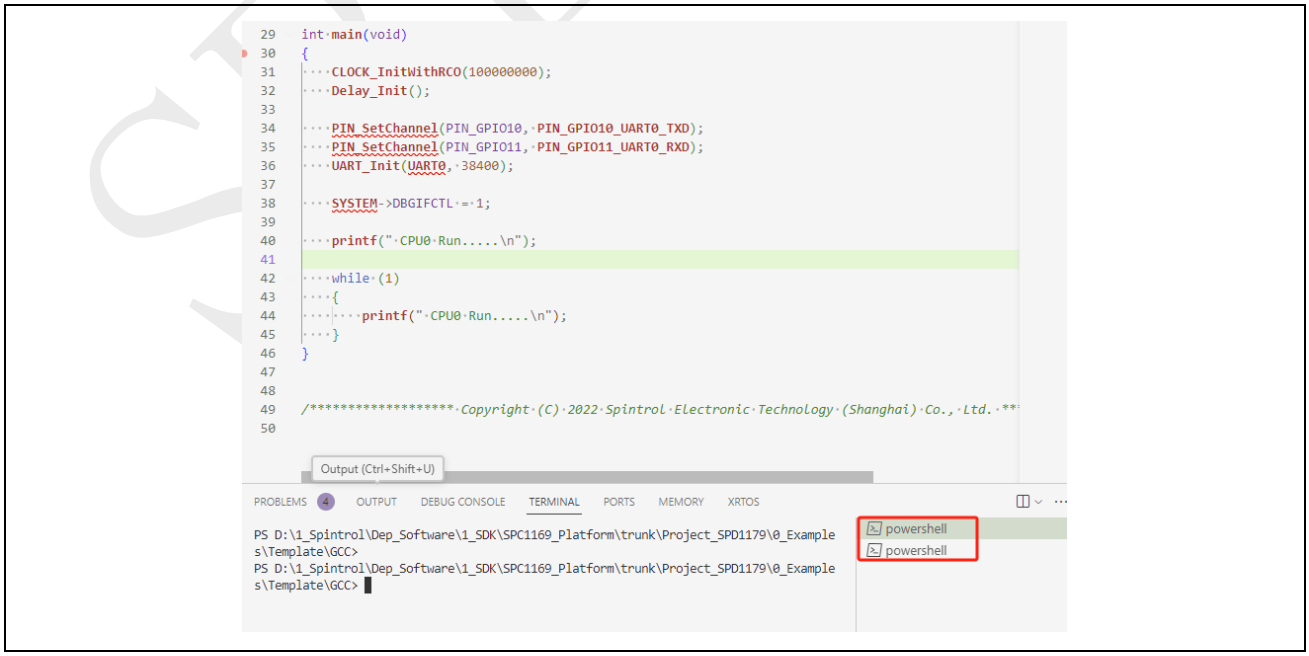
表 4-1：GDB 常见指令列表

调试命令	缩写	作用
list	l	显示源程序代码的内容，包含各行代码所在的行号
continue	c	全速执行被调试的程序
next	n	调试程序一行一行执行
step	s	如果有调用函数，进入调用函数内部；否则和 next 命令一样
finish	fi	结束当前正在执行的函数，并跳出函数调用处停止程序执行
return	return	结束当前函数调用到上一层函数调用处停止执行
break	b	在源程序指定位置（函数名/行号）设置软件断点
hardfault break	hb	在源程序指定位置（函数名/行号）设置硬件断点
print	p	打印指定变量的值

4.3.1 CPU0 工程菊花链调试

- 创建两个“TERMINAL”窗口

图 4-22：创建两个“TERMINAL”窗口



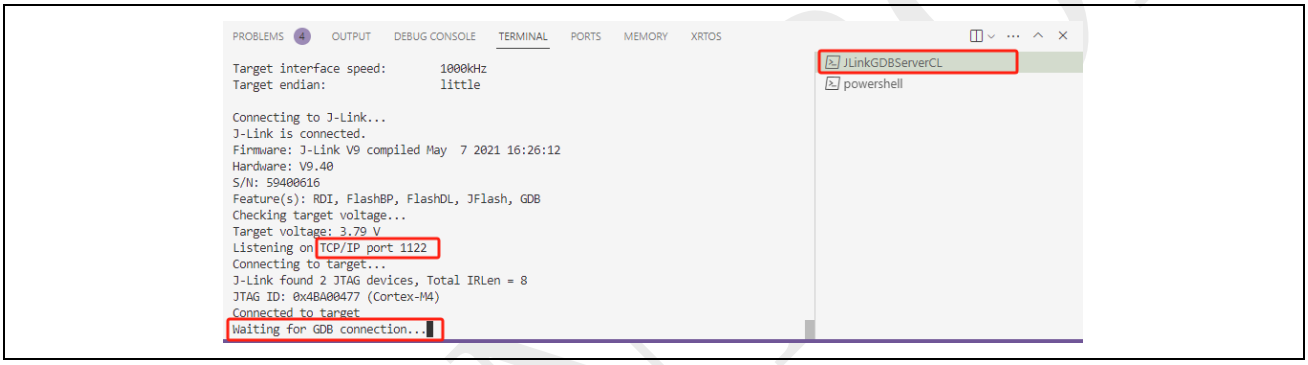
- **创建 JLink GDB 服务端**

执行以下命令，现象如图 4-23 所示。

```
JLinkGDBServerCL.exe -if jtag -device SPC1169_128K -jtagconf 4,1 -port 1122
```

各参数意义如下：
-if jtag：选择 JTAG 接口连接
-device SPC1169_128K：连接的设备为 SPC1169_128K 需要与 Jlink 安装目录的 JLinkDevices.xml 文件定义的设备名一致
-jtagconf 4,1：创建菊花链中的“device1”也是 CPU0 的服务端，如图 4-1 所示。
-prot 1122：设备 JLink GDB 的服务端口号为 1122

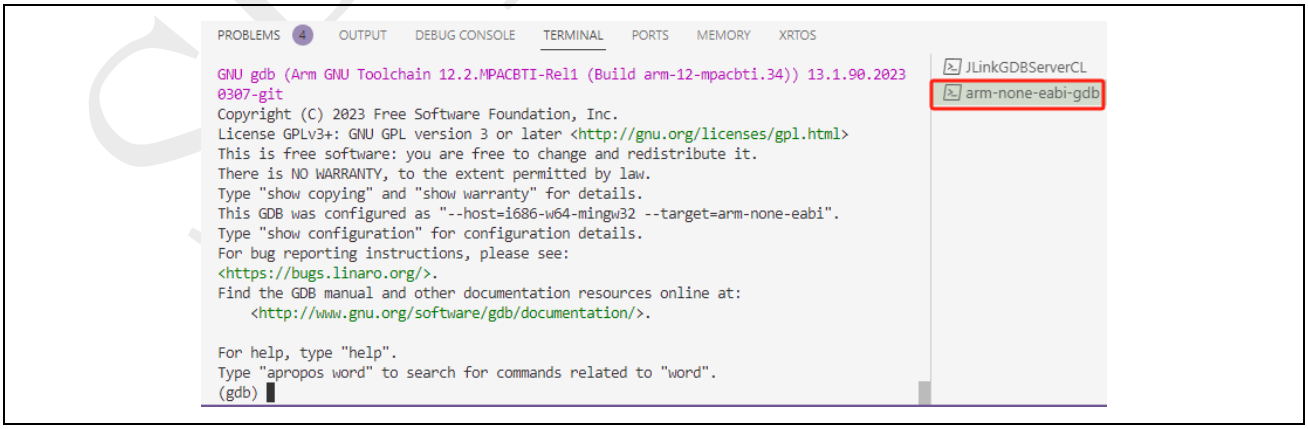
图 4-23：创建 JLink GDB 服务端



- **创建 GDB 客户端**

1. arm-none-eabi-gdb.exe // 运行 GDB 客户端程序

图 4-24：执行 GDB 客户端命令



2. target remote:1122 // 连接 device0 的 JLink GDB 服务端，连接的端口号 1122 需要与创建服务端指定的端口一致。

图 4-25：客户端接连服务端

```
(gdb) target remote:1122
Remote debugging using :1122
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x000000e0 in ?? ()
(gdb)
```

3. file ./project.elf //指定调试使用的符号表

图 4-26：指定调试符号表

```
(gdb) file ./project.elf
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from ./project.elf...
```

4. load ./project.elf // 加载调试的文件

图 4-27：加载调试文件

```
(gdb) load ./project.elf
Loading section .isr_vector, size 0x118 lma 0x10000000
Loading section .text, size 0x3e64 lma 0x10000120
Loading section .rodata, size 0x430 lma 0x10003f88
Loading section .ARM, size 0x8 lma 0x100043b8
Loading section .init_array, size 0x4 lma 0x100043c0
Loading section .fini_array, size 0x4 lma 0x100043c4
Loading section .data, size 0x1e8 lma 0x100043c8
Start address 0x10003d18, load size 17828
Transfer rate: 197 KB/sec, 2546 bytes/write.
(gdb)
```

5. hb main // 设置断点在 main 函数处

图 4-28：设置断点

```
(gdb) hb main
warning: could not convert 'main' from the host encoding (CP1252) to UTF-32.
This normally should not happen, please file a bug report.
Hardware assisted breakpoint 1 at 0x1000353c: file D:/1_Spintrol/Dep_Software/1_SDK/
SPC1169_Platform/trunk/Project_SPD1179/0_Examples/Template/main.c, line 31.
(gdb)
```

6. c //全速运行

图 4-29：全速运行

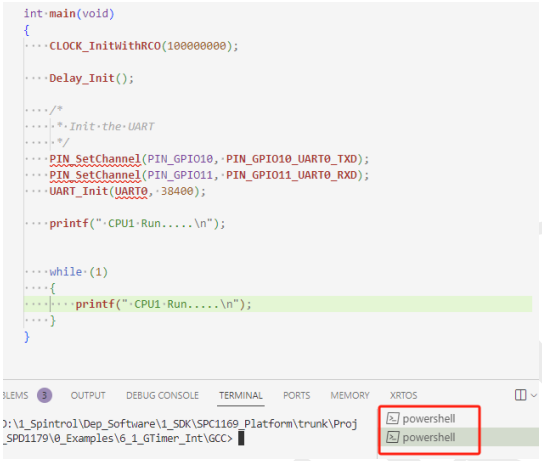
```
(gdb) c
Continuing.

Breakpoint 1, main ()
    at D:/1_Spintrol/Dep_Software/1_SDK/SPC1169_Platform/trunk/Project_SPD1179/0_Exa
mples/Template/main.c:31
31          CLOCK_InitWithRCO(100000000);
(gdb) c
Continuing.
```

4.3.2 CPU1 工程菊花链调试

- 创建两个“TERMINAL”窗口

图 4-30：创建两个“TERMINAL”窗口



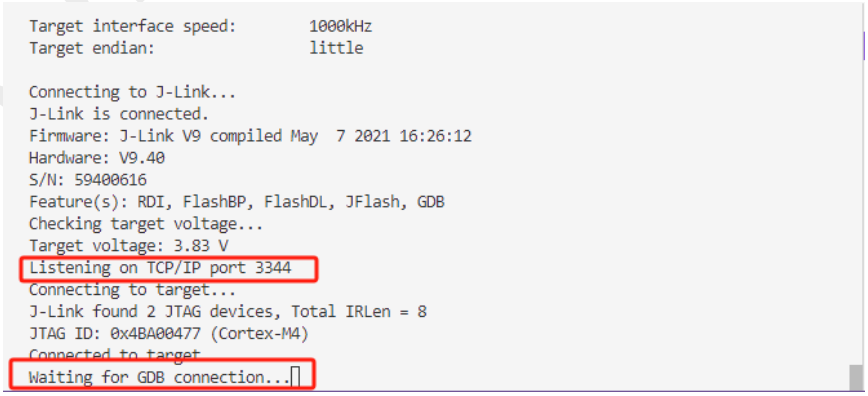
- 创建 JLink GDB 服务端

执行以下命令，现象如图 4-31 所示。

```
JLinkGDBServerCL.exe -if jtag -device SPC1169_128K -jtagconf 0,0 -port 3344
```

各参数意义如下：
-if jtag：选择 JTAG 接口连接
-device SPC1169_128K：连接的设备为 SPC1169_128K 需要与 Jlink 安装目录的 JLinkDevices.xml 文件定义的设备名一致
-jtagconf 0,0：创建菊花链中的“device0”也就是 CPU1 的服务端，如图 4-1 所示。
-prot 1122：设备 JLink GDB 的服务端口号为 3344

图 4-31：创建 JLink GDB 服务端



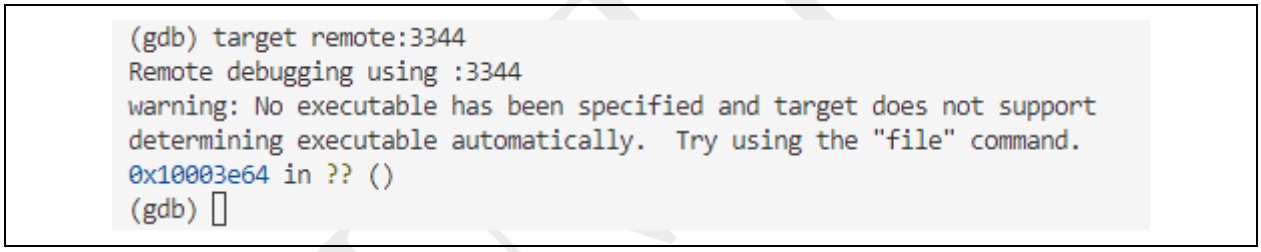
- 创建 GDB 客户端

1. arm-none-eabi-gdb.exe // 运行 GDB 客户端程序

图 4-32：执行 GDB 客户端命令

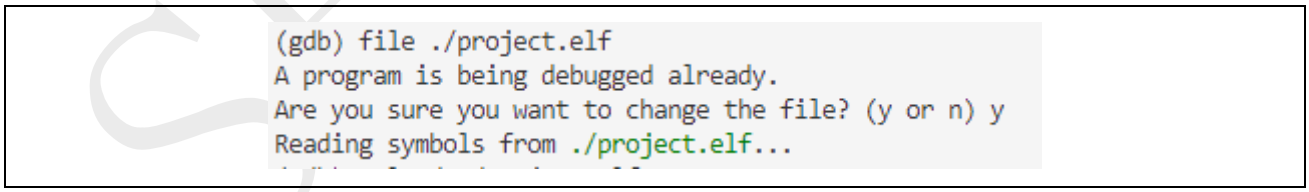
2. target remote:3344 // 连接 device1 的 JLink GDB 服务端，连接的端口号 3344 需要与创建服务端指定的端口一致。

图 4-33：客户端接连服务端



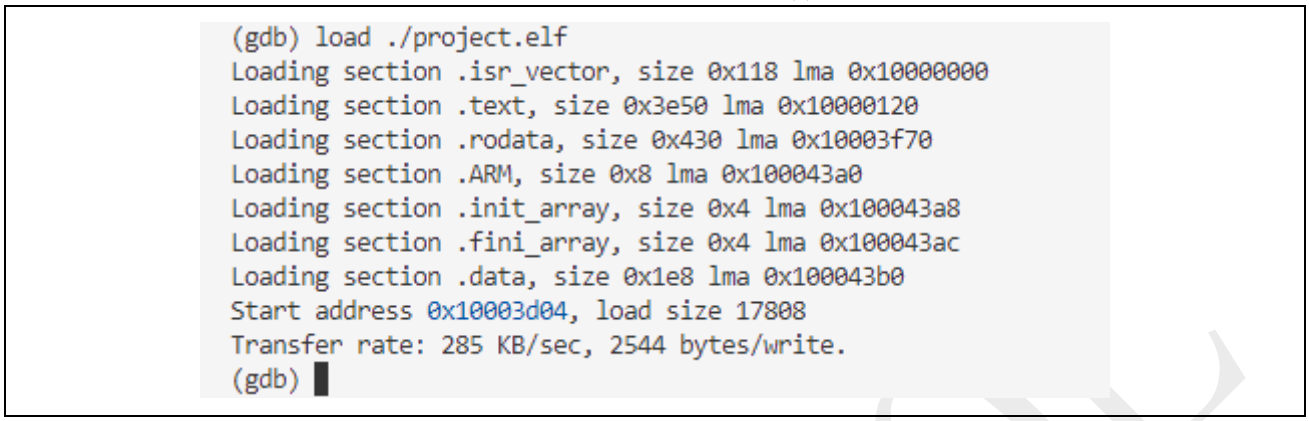
3. file ./project.elf //指定调试使用的符号表

图 4-34：指定调试符号表



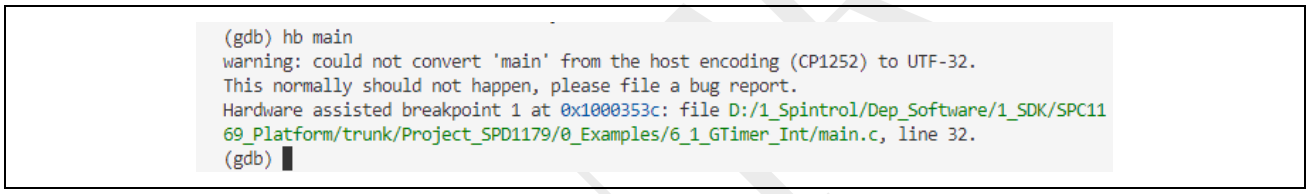
4. load ./project.elf // 加载调试的文件

图 4-35：加载调试文件



5. hb main // 设置断点在 main 函数处

图 4-36：设置断点



6. c //全速运行

图 4-37：全速运行