

基于 Windows 的 ARM GCC 开发环境

目录

1. 概述.....	3
2. 开发工具.....	3
2.1 软件	3
2.2 硬件	3
3. 开发环境搭建.....	4
3.1 安装 VScode 软件.....	4
3.2 安装 gcc 编译工具链.....	4
3.3 安装 Make for Windows	4
3.4 安装 JLink 工具	5
3.5 添加芯片支持	5
3.6 JLink 下载测试	5
4. SDK 目录.....	7
4.1 Makefile.....	7
4.2 .s 文件	7
4.3 .ld 文件	7
4.4 打印重映射	8
4.5 J-Link 脚本.....	8
5. 编译和下载.....	9
5.1 工作区	9
5.2 工作目录	9
5.3 代码编译	9
5.4 固件下载	10
5.5 清除中间文件	10
6. 代码调试.....	11
6.1 VSCode 设置.....	11
6.2 Makefile 设置.....	12
6.3 调试示例	12
7. 配置修改.....	16
7.1 芯片型号	16
7.2 固件下载算法	16
7.3 使用 SDK 算法库	17
7.4 调试配置	17
7.5 优化等级	17
8. 版本历史.....	18
9. 声明.....	19

1.概述

本文以 N32H785 系列 MCU 为例，介绍了在 Windows 环境下基于 vscode 编辑器、GCC 编译工具链和 GDB 调试工具进行搭建开发环境，进行代码编译、固件下载和代码调试的方法。

2.开发工具

2.1 软件

- 1) 编辑器 Visual Studio Code 1.5x.x 或以上
- 2) 编译工具链 arm-none-eabi-gcc 6.3.1 或以上
- 3) Make for Windows
- 4) 下载调试工具 JLink_V6.40（需不高于硬件支持版本）或以上

2.2 硬件

- 1) 开发板 N32H785xIx7-STB V1.0
- 2) JLink 下载器 V9.2(需不低于软件支持版本)或以上

3. 开发环境搭建

3.1 安装 VScode 软件

- 下载软件: <https://code.visualstudio.com/>

VScode 用作代码查看和编辑, 它还提供了 powershell 和 bash 终端用于命令行操作, 我们的整个开发过程都要用到命令行终端。

3.2 安装 gcc 编译工具链

- 下载地址: <https://launchpad.net/gcc-arm-embedded/+announcement/28093>
示例版本: [10-2020-q4-major](#)

检查是否安装成功: 打开 dos 命令行窗口, 输入 `arm-none-eabi-gcc -v`, 如下表示安装成功:

```
C:\Users\tan.dengwang>arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.2.1 2021103
(release)
Copyright (C) 2020 Free Software Foundation, Inc.
```

若不成功

- 1, 请检查环境变量是否添加好
- 2, 进入“*C:\Program Files (x86)\GNU Arm Embedded Toolchain\10-2020-q4-major\bin*”安装目录下检查 `arm-none-eabi-gcc.exe` 文件名是否正确

3.3 安装 Make for Windows

此工具用于解析 Makefile 脚本, 可以选择下面两个软件其中一个进行安装。

- 安装 `cmake.exe` 工具
下载地址: <http://www.equation.com/servlet/equation.cmd?fa=make>
- 安装 MinGW 软件, 使用其自带的 `make` 工具。

检查是否安装成功: 打开 dos 命令行窗口, 输入 `make -v` 如下:

```
C:\Users\tan.dengwang>make -v
GNU Make 3.82.90
Built for i686-pc-mingw32
Copyright (C) 1988-2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

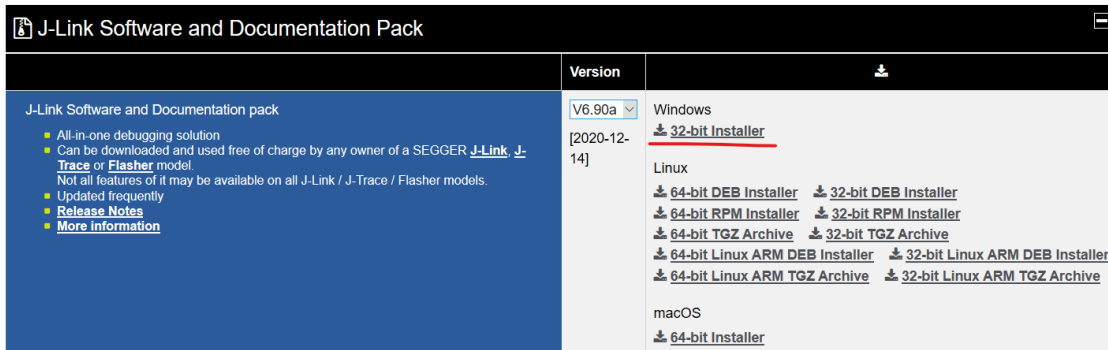
若不成功

- 1, 请检查环境变量是否添加好
- 2, 进入对应的 `make` 安装目录 `bin` 文件夹检查一下 `make.exe` 文件命名是否正确

3.4 安装 JLink 工具

- 下载 JLINK 安装包, V6.90a 或其他版本

<https://www.segger.com/downloads/jlink/#-LinkSoftwareAndDocumentationPack>



3.5 添加芯片支持

安装好 JLink 之后需要向 JLink 中添加我们公司的芯片补丁包, 以便在下载、调试时正确获取到下载算法。

具体请参考文档《jlink 工具添加 Nations 芯片.7z》

3.6 JLink 下载测试

- 测试 JLink 环境安装

- 1, 连接好 PC 和 J-Link 调试器, 连接好开发板, 上电;
- 2, 打开 cmd.exe 命令行工具, 进入 JLink 安装目录“C:\Program Files (x86)\SEGGER\JLink_V640”下, 输入“JLink.exe”;

```
Microsoft Windows [版本 10.0.19045.5247]
(c) Microsoft Corporation. 保留所有权利。

C:\Program Files (x86)\SEGGER\JLink_V640>JLink.exe
SEGGER J-Link Commander V6.40 (Compiled Oct 26 2018 15:06:29)
DLL version V6.40, compiled Oct 26 2018 15:06:02

Connecting to J-Link via USB...O.K.
Firmware: J-Link V9 compiled May 7 2021 16:26:12
Hardware version: V9.40
S/N: 59417452
License(s): RDI, GDB, FlashDL, FlashBP, JFlash
VTref=3.304V

Type "connect" to establish a target connection, '?' for help
J-Link>
```

如上图表示 PC 连接 JLink 调试器成功。

- 1, 然后根据提示依次输入: “connect”, “N32H785XIX7:CM4”, “SWD”, “4000”, 如果前面的操作成功, 则会看到下面的输出信息, JLink 下载调试环境就可以正常使用了。

```
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: N32H785XIX7:CM4
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "N32H785XIX7:CM4" selected.

Connecting to target via SWD
Executing J-Link script file function: InitTarget()
*****
J-Link script: NationsTech Cortex-M4 J-Link script
*****
Found SW-DP with ID 0x2BA01477
AP map detection skipped. Manually configured AP map found.
AP[0]: AHB-AP (IDR: Not set)
AP[1]: AHB-AP (IDR: Not set)
AP[2]: AHB-AP (IDR: Not set)
AP[2]: Core found
AP[2]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
ROMTbl[0][4]: E0040000, CID: B105900D, PID: 000BB9A1 TPIU
ROMTbl[0][5]: E0041000, CID: B105900D, PID: 000BB925 ETM
Cortex-M4 identified.
J-Link>
```

4.SDK 目录

SDK 沿用已发的 SDK 版本，当前使用 v1.2.0，在此基础上做如下修改以适应 GCC 开发环境。

4.1 Makefile

在 SDK 包中的 H78x_GCC_In_Flash 例程目录下“GCC”文件夹:

Nations.N32H7xx_Library.1.2.0 > projects > n32h7xx_EVAL > applications > H78x_GCC_In_Flash > GCC			▼	🔄
名称	修改日期	类型		
Makefile	2026/1/16 14:13	文件		

其中“Makefile”文件是 GCC 编译脚本文件。

4.2 .s 文件

在 SDK 包中“[Nations.N32H7xx_Library.1.2.0\firmware\CMSIS\device\startup](#)”路径下有对应 gcc 编译器的.s 文件“[startup_n32h78x_cm4_gcc.s](#)”、“[startup_n32h78x_cm7_gcc.s](#)”

Nations.N32H7xx_Library.1.2.0 > firmware > CMSIS > device > startup			▼	🔄	在 startup 中搜索	🔍
名称	修改日期	类型				
startup_n32h73x_76x.s	2026/1/12 11:27	S 文件				
startup_n32h73x_76x_EWARM.s	2026/1/12 11:27	S 文件				
startup_n32h73x_76x_gcc.s	2026/1/19 9:19	S 文件				
startup_n32h76x_ITCM_gcc.s	2026/1/16 17:39	S 文件				
startup_n32h78x_cm4.s	2026/1/12 11:27	S 文件				
startup_n32h78x_cm4_EWARM.s	2026/1/12 11:27	S 文件				
startup_n32h78x_cm4_gcc.s	2025/4/25 12:04	S 文件				
startup_n32h78x_cm7.s	2026/1/12 11:27	S 文件				
startup_n32h78x_cm7_EWARM.s	2026/1/12 11:27	S 文件				
startup_n32h78x_cm7_gcc.s	2025/4/25 11:25	S 文件				

4.3 .ld 文件

在 SDK 包中“[Nations.N32H7xx_Library.1.2.0\firmware\CMSIS\device](#)”路径下有对应的.ld 文件“[n32h78x_cm4_flash.ld](#)”、“[n32h78x_cm7_flash.ld](#)”

Nations.N32H7xx_Library.1.2.0 > firmware > CMSIS > device		在 dev
名称	修改日期	
startup	2026/1/19 9:19	
n32h7xx.h	2026/1/12 11:27	
n32h73x_76x_flash.ld	2026/1/16 11:32	
n32h76x_ITCM_flash.ld	2026/1/16 17:31	
<u>n32h78x_cm4_flash.ld</u>	2025/4/24 17:25	
<u>n32h78x_cm7_flash.ld</u>	2025/4/24 17:24	
system_n32h7xx.c	2026/1/12 11:27	
system_n32h7xx.h	2026/1/12 11:27	

4.4 打印重映射

在 SDK 包的“*bsp/src*”目录下增加了“*print_remap.c*”文件用于串口打印重定向。

Nations.N32H7xx_Library.1.2.0 > projects > n32h7xx_EVAL > bsp > src		在 src 中搜索
名称	修改日期	
bsp_sdram.c	2026/1/12 11:27	
delay.c	2026/1/12 11:27	
log.c	2026/1/14 10:02	
<u>print_remap.c</u>	2026/1/16 11:30	

4.5 J-Link 脚本

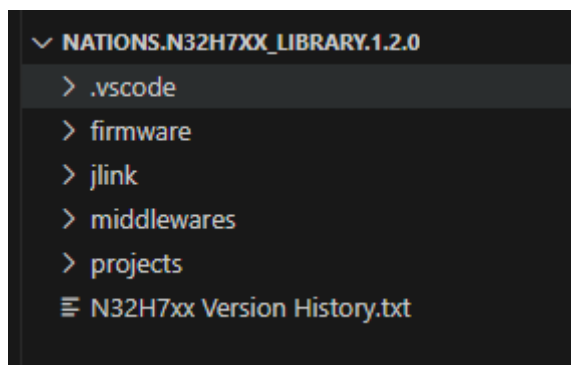
在 SDK 包主目录下增加了“jlink”文件夹，文件夹中有 jlink 下载脚本，用于通过 J-Link 工具下载固件。

开发套件 > SDK > Nations.N32H7xx_Library.1.2.0 > jlink		在 jlink
名称	修改日期	
flash.jlink	2023/1/4 16:28	
<u>flash_CM4.jlink</u>	2025/4/25 14:04	
<u>flash_CM7.jlink</u>	2025/4/25 14:04	

5. 编译和下载

5.1 工作区

在 VScode 中打开 SDK 文件夹，并另存为工作区。此时在 SDK 文件夹下会生成“.vscode”文件夹用于放置工作区配置文件。



5.2 工作目录

以 GPIO 例程 LedBlink 为例，进入到工程目录下：

"Nations.N32H7xx_Library.1.2.0\projects\n32h7xx_EVAL\applications\H78x_GCC_In_Flash"
GCC 工程"GCC"

源文件"CMx/src /xxx.c"

头文件 "CMx/inc/xxx.h"

Makefile 文件 "GCC/Makefile"

5.3 代码编译

在 VScode 编辑器的终端中，切换到“GCC”文件夹目录下，输入“make”开始 CM7 的编译

PS D:\desktop\GCC\Nations.N32H78x_Library.0.2.0\projects\n32h78x_EVAL\examples\GPIO\LedBlink\GCC> **make**
编译完成无错误会生成 CM7 的.elf、.bin 和.hex 文件。

```
-m7 -mthumb -Wl,--gc-sections --specs=nosys.specs -Xlinker -Map=build_CM7/output_CM7.map -T./../../../../../../../../firmware/PSIS/device/n32h76x_78x_flash_cm7.ld -o build_CM7/output_CM7.elf
arm-none-eabi-size build_CM7/output_CM7.elf
text      data      bss      dec      hex      filename
11040     2128     12296     25464     636c build_CM7/output_CM7.elf
arm-none-eabi-objcopy -O ihex -S build_CM7/output_CM7.elf build_CM7/output_CM7.hex
arm-none-eabi-objcopy -O binary -S build_CM7/output_CM7.elf build_CM7/output_CM7.bin
```

此时在“GCC”文件夹下面会创建“build_CM7”文件夹，编译的固件和中间文件都在此文件夹下。

输入“make DCORE=CM4”开始 CM4 的编译

PS D:\desktop\GCC\Nations.N32H78x_Library.0.2.0\projects\n32h78x_EVAL\examples\GPIO\LedBlink\GCC> **make DCORE=CM4**
编译完成无错误会生成 CM4 的.elf、.bin 和.hex 文件。

```
-m4 -mthumb -Wl,--gc-sections --specs=nosys.specs -Xlinker -Map=build_CM4/output_CM4.map -T../.../.../firmware/CHSIS/device/n32h76x_78x_flash_cm4.ld -o build_CM4/output_CM4.elf
arm-none-eabi-size build_CM4/output_CM4.elf
text      data      bss      dec      hex filename
3496      1888      12324      16980      4204 build_CM4/output_CM4.elf
arm-none-eabi-objcopy -O ihex -S build_CM4/output_CM4.elf build_CM4/output_CM4.hex
arm-none-eabi-objcopy -O binary -S build_CM4/output_CM4.elf build_CM4/output_CM4.bin
```

此时在“GCC”文件夹下面会创建“build_CM4”文件夹，编译的固件和中间文件都在此文件夹下。

5.4 固件下载

- 1, 连接好 PC→JLink→开发板
- 2, 在终端输入“[make download](#)”下载 CM7 的代码

```
PS E:\workspace_linqi\3605\GCC\Nations.N32G430_Library.1.0.0\projects\n32g430_EVAL\examples\GPIO\LedBlink\GCC> make download
```

中间会输出一些信息..., 最后下载完成

```
Writing target memory failed.
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>g
J-Link>qc

Script processing completed.

"Download Completed!"
```

- 3, 在终端输入“[make download DCORE=CM4](#)”下载 CM4 的代码，结果同 CM7

```
PS D:\desktop\GCC\Nations.N32H78x_Library.0.2.0\projects\n32h78x_EVAL\examples\GPIO\LedBlink\GCC> make download DCORE=CM4
```

- 4, 下载完成后按一下复位键，CM7&CM4 系统开始运行
- 5, 如果下载不成功，请检查 JLink 配置

5.5 清除中间文件

在终端输入“[make clean](#)”可以清除编译生成的中间文件。

6.代码调试

6.1 VSCode 设置

在 SDK 工作路径下有一个“.vscode”文件夹，里面有“launch.json”工作区配置文件，进行代码调试需要配置：

名称		修改日期	类型
<input checked="" type="checkbox"/>	launch.json	2022/4/6 14:51	JSON 文件
<input type="checkbox"/>	settings.json	2021/11/12 16:42	JSON 文件
<input type="checkbox"/>	tasks.json	2022/4/6 14:49	JSON 文件

launch.json:

```

vscode > {} launch.json > Launch Targets > {} gdb-arm
1 {
2   "version": "1.0.0",
3   "configurations": [
4     {
5       "name": "gdb-arm",
6       "type": "cppdbg",
7       "request": "launch",
8       "targetArchitecture": "arm",
9       "program": "output",
10      "args": [],
11      "stopAtEntry": true,
12      "cwd": "${workspaceFolder}",
13      "environment": [],
14      "externalConsole": false,
15      "MIMode": "gdb",
16      "miDebuggerPath": "C:\\Program Files (x86)\\GNU Arm Embedded Toolchain\\10 2020-q4-major\\bin\\arm-none-eabi-gdb.exe",
17      "miDebuggerServerAddress": "localhost:2331",
18      "setupCommands": [
19        {
20          "description": "Enable pretty-printing for gdb",
21          "text": "-enable-pretty-printing",
22          "ignoreFailures": false
23        }
24      ],
25      "customLaunchSetupCommands": [
26        {
27          "text": "target remote :2331",
28          "description": "connect to server",
29          "ignoreFailures": false
30        },
31        {
32          "text": "file 'D:/desktop/GCC/Nations.N32H78x_Library.0.2.0/projects/n32h78x_EVAL/examples/GPIO/LedBlink/GCC/build_CM4/output_CM4.elf'",
33          "description": "load file to gdb",
34          "ignoreFailures": false
35        },
36        {
37          "text": "load",
38          "description": "download file to MCU",
39          "ignoreFailures": false
40        },
41        {
42          "text": "monitor reset",
43          "description": "reset MCU",
44          "ignoreFailures": false
45        },
46        {
47          "text": "b main",
48          "description": "set breakpoints at main",
49          "ignoreFailures": true
50        }
51      ],
52      "launchCompleteCommand": "None",
53      //preLaunchTask: "build"
54    }
55  ]
56 }
57

```

这是 vscode 调试器配置文件，下面几个地方要根据自己的项目路径来修改：

1, 指定 gdb 调试器的路径: (绝对路径)

```
"miDebuggerPath": "C:\\Program Files (x86)\\GNU Arm Embedded Toolchain\\10-2020-q4-major\\bin\\arm-none-eabi-gdb.exe",
```

此 gdb 工具的版本一定要与编译器工具的版本相匹配, 否则会报误或部分功能不可用, 一般在我们的 arm-none-eabi-gcc.exe 工具同目录下会有 arm-none-eabi-gdb.exe 工具。

2, 指定调试代码 xxx.elf 文件路径: (注意: 路径不能太长)

```
"text": "file 'D:\\desktop\\GCC\\Nations.N32H78x_Library.0.2.0\\projects\\n32h78x_EVAL\\examples\\GPIO\\LedBlink\\GCC\\build_CM4\\output_CM4.elf'",
```

CM4 和 CM7 生成文件的路径不一样, 所以在不同内核调试时这里需要修改。

6.2 Makefile 设置

打开例程中的“GCC/Makefile”文件:

```
download:
ifeq ($(DCORE), CM4)
    @$(JK_DPATH)JLink.exe -device $(CHIP_TYPE) -if SWD -speed 4000 -autoconnect 1 -CommanderScript $(JKS_DIR)/flash_CM4.jlink
else
    @$(JK_DPATH)JLink.exe -device $(CHIP_TYPE) -if SWD -speed 4000 -autoconnect 1 -CommanderScript $(JKS_DIR)/flash_CM7.jlink
endif
@echo "Download Completed!"

debug:
    @$(JK_DPATH)JLinkGDBServer.exe -select USB -device $(CHIP_TYPE) -if SWD -speed auto -noir -LocalhostOnly

# *** EOF ***
```

1, 可以看到有一个 debug 的启动配置, 指向 JLink 安装目录的 JLinkGDBserver 服务程序。
2, 编译命令 make 默认就是在调试模式编译 CM7, 会带有一些调试信息。如果想编译 CM4, 就用编译命令 make DCORE=CM4。如果想要切换到发布版本, 则编译代码时需要用如下命令: make release=y

6.3 调试示例

以 GPIO LedBlink 工程作为示例, 看看如何开始代码调试:

1, 在 vscode 中打开 SDK 工程, 在终端中切换到 LedLink/GCC 目录, 输入 make 编译 CM7 代码

```
PS D:\desktop\GCC\Nations.N32H78x_Library.0.2.0\projects\n32h78x_EVAL\examples\GPIO\LedBlink\GCC> make

-m7 -mthumb -Wl,--gc-sections --specs=nosys.specs -Xlinker -Map=build_CM7/output_CM7.map -T.../.../.../firmware/OSIS/device/n32h78x_flash_CM7.ld -o build_CM7/output_CM7.elf
arm-none-eabi-size build_CM7/output_CM7.elf
text    data    bss     dec      hex filename
11840   2128   12396   26364   66fc build_CM7/output_CM7.elf
arm-none-eabi-objcopy -O ihex -S build_CM7/output_CM7.elf build_CM7/output_CM7.hex
arm-none-eabi-objcopy -O binary -S build_CM7/output_CM7.elf build_CM7/output_CM7.bin
```

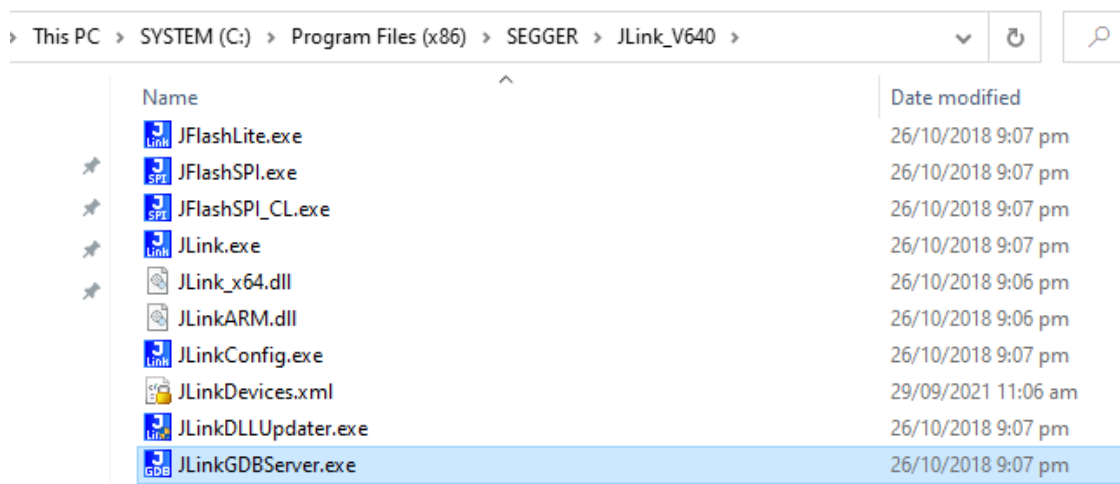
生成的 output_CM7.elf、output_CM7.bin、output_CM7.hex 文件在 GCC/build_CM7 文件夹中。

如果编译 CM4 代码, 同上参考 5.3 章节。

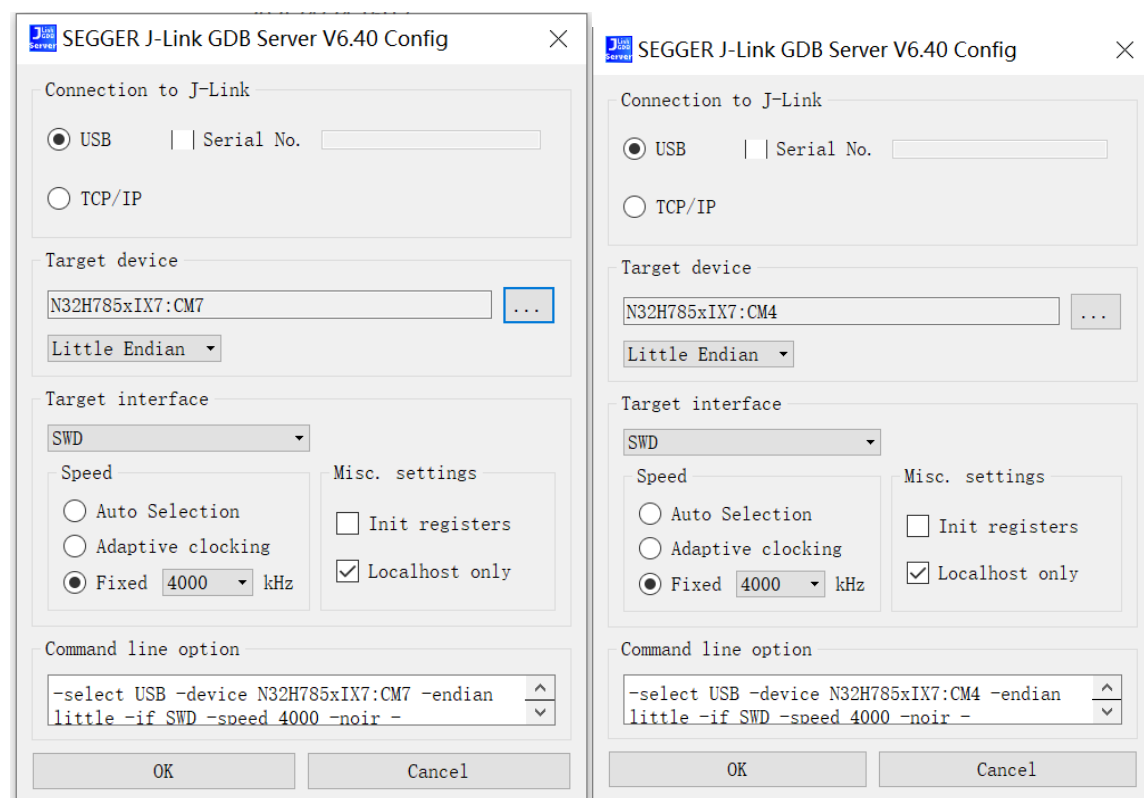
2, 请参考 6.1、6.2 章节配置好 launch.json 文件中的路径。

3, 连接 JLink 调试器到开发板, 上电准备好。

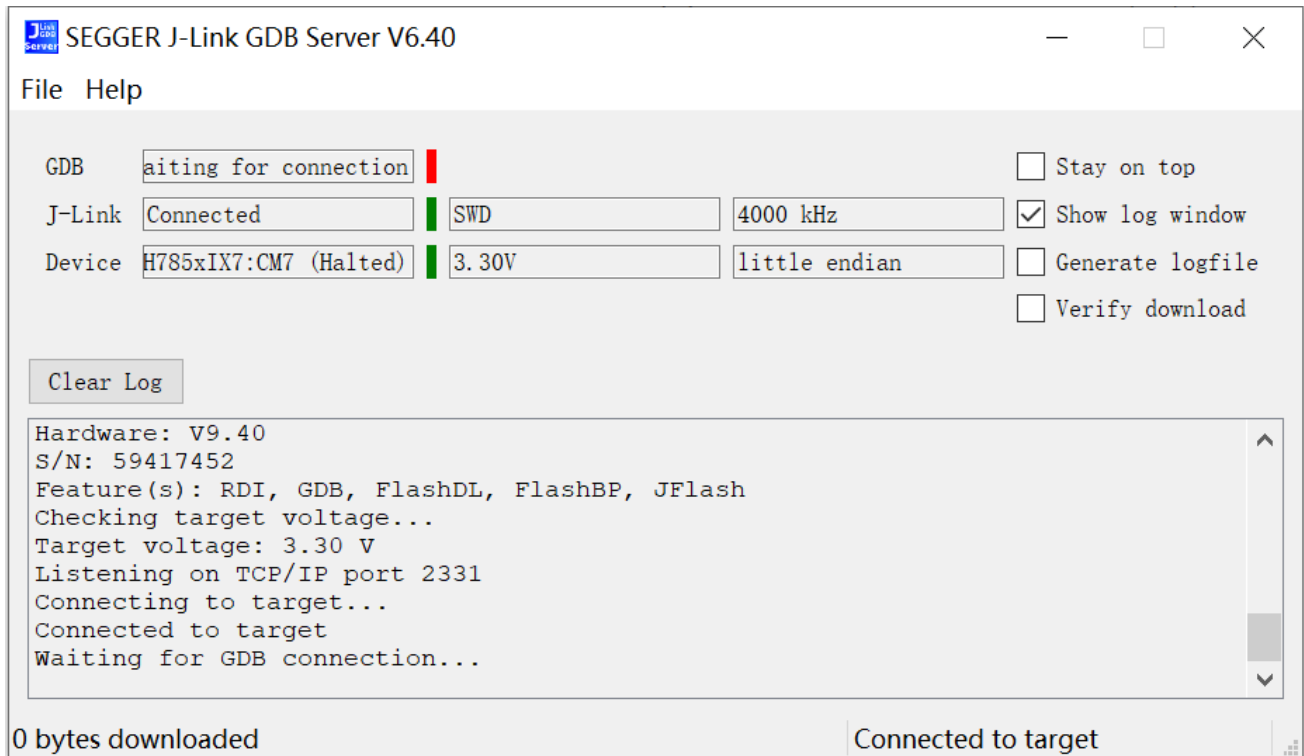
4, 找到你的 JLink 安装目录, 双击运行 JLinkGDBServer.exe 程序



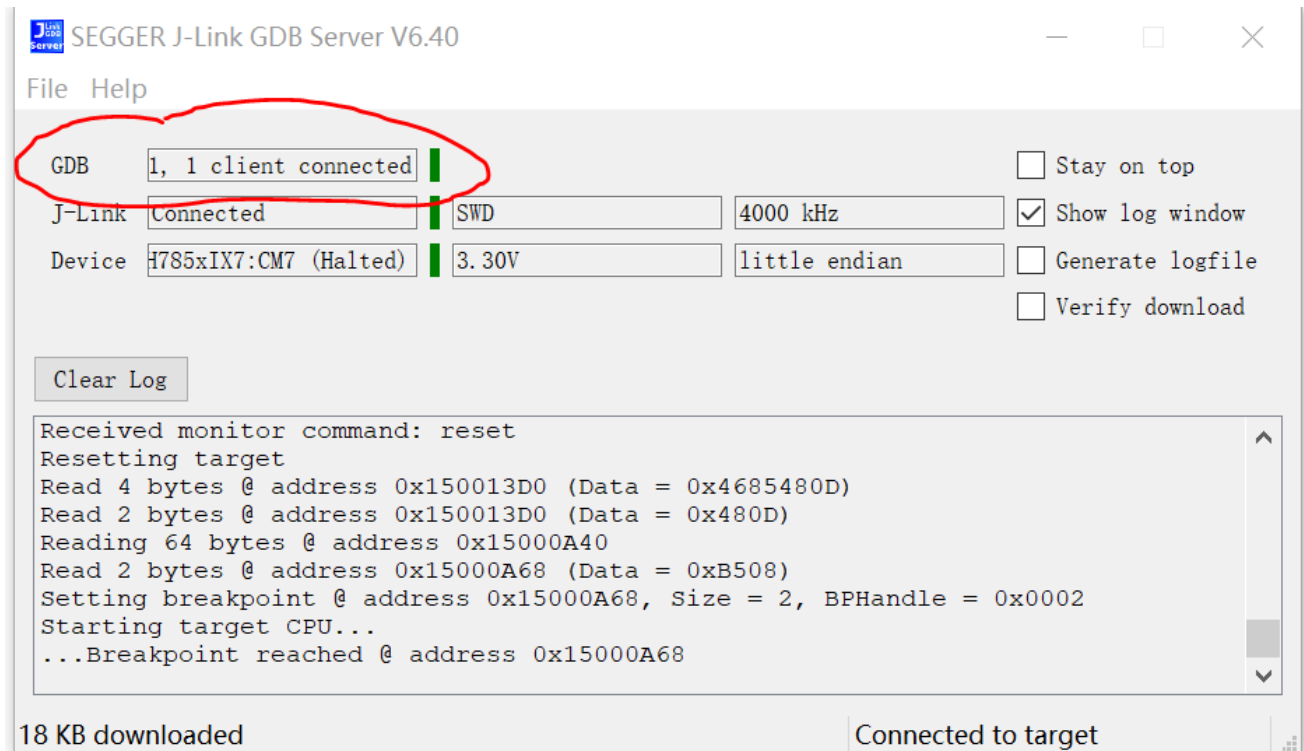
配置端口、协议、芯片型号等, 调试 CM7 就选择 CM7, 调试 CM4 就选择 CM4, 点击 OK



如下, 表示 JLink 调试器连接到芯片成功:



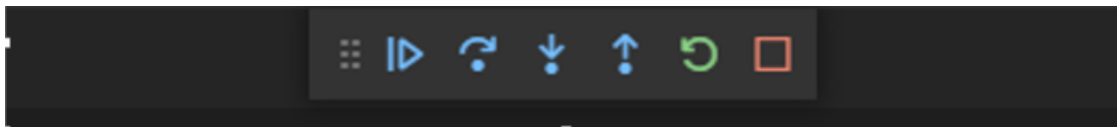
5, 在 vscode 工作环境下, 按“F5”或点击“运行”->“启动调试”, 此时可以看到下图标签变为了绿色, 表示 gdb 工具连接 JLinkGDBserver 成功。



6, 此时在 vscode 自动切换到了调试窗口

```
projects > n32h78x_EVAL > examples > GPIO > LedBlink > CM7 > src > C main.c > main(void)
62
63 /**
64  * \name    main.
65  * \fun     Main program.
66  * \param   none
67  * \return  none
68  */
69 int main(void)
70 {
71     /* Initialize system clock */
72     RCC_SetSysClkToMode0();
73     /* Enable Cortex-M4 boot*/
74     RCC_EnableCM4(0x15080000);
75     /* Add Cortex-M7 user application code here */
76
77     /* RCC configuration -----*/
78     RCC_Configuration();
79     /* LOG configuration -----*/
80     log_init();
81     /* GPIO configuration -----*/
82     GPIO_Configuration();
83
84     log_info("This is led blink demo\r\n");
85
86     while (1)
87     {
88         GPIO_SetBits(LED1_PORT, LED1_PIN);
89         systick_delay_ms(500);
90         GPIO_ResetBits(LED1_PORT, LED1_PIN);
91         systick_delay_ms(500);
92     }
93 }
94
```

7, 调试窗口上方的调试按钮：单步、连续执行、重启、停止等



8, 现在就可以单步和全速运行了

```
8  */
9  int main(void)
10 {
11     /* Initialize system clock */
12     RCC_SetSysClkToMode0();
13     /* Enable Cortex-M4 boot*/
14     RCC_EnableCM4(0x15080000);
15     /* Add Cortex-M7 user application code here */
16
17     /* RCC configuration -----*/
18     RCC_Configuration();
19     /* LOG configuration -----*/
20     log_init();
21     /* GPIO configuration -----*/
22     GPIO_Configuration();
23
24     log_info("This is led blink demo\r\n");
25
26     while (1)
27     {
28         GPIO_SetBits(LED1_PORT, LED1_PIN);
29         systick_delay_ms(500);
30         GPIO_ResetBits(LED1_PORT, LED1_PIN);
31         systick_delay_ms(500);
32     }
33 }
```

7. 配置修改

7.1 芯片型号

如果使用的芯片不是 N32H785 系列,则需要修改 makefile 文件中的变量“**TARGET_PLATFORM**”和“**DEFS**”

```
#####  
# chip platform info  
#####  
TARGET_PLATFORM := n32h7xx  
ifeq ($(DCORE), CM4)  
TARGET_STARTUP := n32h78x_cm4  
DEFS += -DCORE_CM4  
else  
TARGET_STARTUP := n32h78x_cm7  
DEFS += -DCORE_CM7  
endif  
DEFS += -DN32H78x  
DEFS += -DUSE_STDPERIPH_DRIVER
```

7.2 固件下载算法

需要输入完整的芯片型号,以便 JLink 可以正确匹配下载算法。

```
241 #Chip type  
242 ifeq ($(DCORE), CM4)  
243 CHIP_TYPE = N32H785xIx7:CM4  
244 else  
245 CHIP_TYPE = N32H785xIx7:CM7  
246 endif
```

配置下载工具路径: 根据你的安装目录来配置

```
#####  
# download .hex/.bin by jlink  
#####  
#Your JLink installation directory  
PATH_WINPC = 'C:/Program Files (x86)/SEGGER/JLink_V640/'  
#PATH_LINUX = /opt/SEGGER/JLink_V640b/JLinkExe  
JK_DPATH = $(PATH_WINPC)
```


7.3 使用 SDK 算法库

默认不使用算法库，使用算法库请修改变量“USELIB = 1”

```
40 #####  
41 # Algo libs  
42 #####  
43 USELIB = 0
```

7.4 调试配置

默认的“make”编译是带有“-g”调试信息的，如果要编译 release 版本，请使用“make release=y”。

7.5 优化等级

默认使用“-O0”优化等级，不开启优化。

8.版本历史

日期	版本	修改
2025/04/27	V1.0	初始版本
2025/08/20	V1.1	1, 修改页眉的 logo
2026/01/19	V1.2	1. 将示例工程移至 “Nations.N32H7xx_Library.1.2.0\projects\n32h7xx_EVAL\applications\H78x_GCC_In_Flash”路径下 2. 更新页眉/页脚

9. 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用者在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。