

Application note

GCC development environment based on Windows Application Note

Contents

1. Overview.....	3
2. Development tools	3
2.1 software	3
2.2 hardware	3
3. Development environment setup	4
3.1 Installing VScode	4
3.2 Installing the GCC Compilation tool chain	4
3.3 Installing Make for Windows	4
3.4 Installing the JLink Tool	5
3.5 Adding Chip Support	5
3.6 JLink download test.....	5
4. SDK Contents	6
4.1 Makefile.....	7
4.2 .s file	7
4.3 .ld file.....	7
4.4 Printing remapping	7
4.5 J-Link script.....	8
5. Compile and download.....	9
5.1 Workspace	9
5.2 Working Directory	9
5.3 Code Compilation.....	9
5.4 Downloading Firmware	10
5.5 Clearing Intermediate Files.....	10
6. Code debugging	11
6.1 VSCode set	11
6.2 Makefile Settings	12
6.3 Debugging Examples	12
7. Configuration changes	15
7.1 Chip Models	15
7.2 Firmware Download Algorithm.....	15
7.3 Using the SDK algorithm library.....	16
7.4 DEBUG configuration	16
7.5 Optimization Grade	16
8. Version history.....	17
9. Notice	18

1. Overview

Taking N32G033 series MCU as an example, this paper introduces the methods of setting up development environment, compiling, firmware downloading and code debugging based on VScode editor, GCC compilation tool chain and GDB debugging tool under Windows environment.

2. Development tools

2.1 software

- 1) Editor Visual Studio Code 1.5x.x or above
- 2) Compile toolchain arm-none-eabi-gcc 6.3.1 or above
- 3) Make for Windows
- 4) Download and debugging tool JLink_v6.40(need to be no higher than the hardware support version) or above

2.2 hardware

- 1) Development board N32G033K8Q7-1-STB V1.0
- 2) JLink Downloader V9.2(need to be no lower than the software support version) or above

3. Development environment setup

3.1 Installing VScode

- **Download the software:** <https://code.visualstudio.com/>

VScode is used for code viewing and editing, and it also provides powershell and bash terminals for command-line operations, which will be used throughout our development process.

3.2 Installing the GCC Compilation tool chain

- **Download address:**
<https://launchpad.net/gcc-arm-embedded/+announcement/28093>
example version: [10-2020-q4-major](#)

Check whether the installation is successful: Open the DOS command line window, type `arm-none-eabi-gcc -v`,

The installation is successful if:

```
C:\Users\tan.dengwang>arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.2.1 20201103
(release)
Copyright (C) 2020 Free Software Foundation, Inc.
```

If you don't succeed

1. Check whether environment variables are properly added
2. Go to “*C:\Program Files (x86)\GNU Arm Embedded Toolchain\10-2020-q4-major\bin*” and check whether the `arm-none-eabi-gcc.exe` file name is correct

3.3 Installing Make for Windows

This tool is used to parse Makefile scripts and can be installed with either of the following software.

- **Install the cmake.exe tool**
Download address: <http://www.equation.com/servlet/equation.cmd?fa=make>
- **Install MinGW software and use its own make tool.**

Check whether the installation is successful: Open the DOS command line window and enter `make -v` as follows:

```
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.
```

If you don't succeed

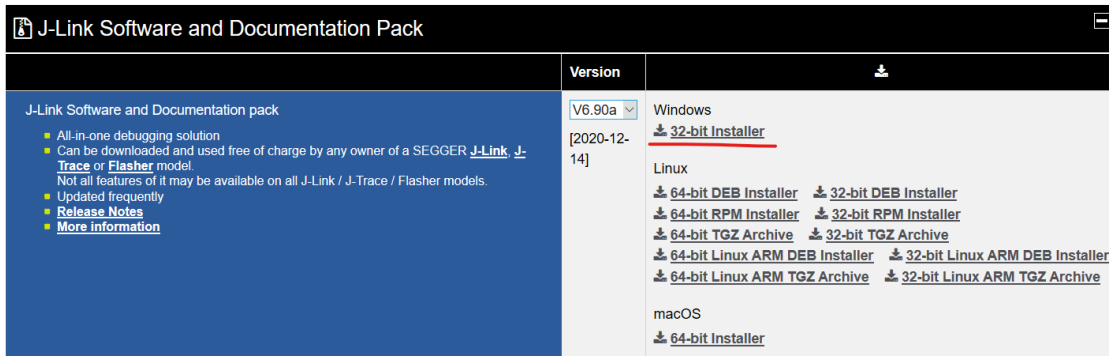
- 1, Check that the environment variables are properly added
- 2, Go to the bin folder of the corresponding `make` installation directory to check whether the

make.exe file is correctly named

3.4 Installing the JLink Tool

- Download the JLINK installation package, V6.90a or others version

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



3.5 Adding Chip Support

After installing JLink, we need to add our company's chip patch package to JLink, so that we can get the download algorithm correctly during downloading and debugging.

For details, see <jlink Tool Adding Nsing Chip.7z>.

3.6 JLink download test

- Test the JLink environment installation

- 1, Connect the PC and j-Link debugger, connect the development board, and power on;
- 2, Open cmd.exe command line tool, go to JLink installation directory C:\Program Files (x86)\SEGGER\JLink_V640, type jlink.exe.

```
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 Dec 13 2019 11:14:50
Hardware version: V9.60
S/N: 69660532
License(s): RDI, GDB, FlashDL, FlashBP, JFlash
VTref=3.316V

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

The image above shows that the PC successfully connected to the JLink debugger.

- 3, Then according to the prompt input: "connect", "N32G033x8", "SWD", "4000", if the

previous operation is successful, you will see the following output information, JLink download debugging environment can be used normally.

```
Hardware version: V9.40
S/N: 59417452
License(s): RDI, GDB, FlashDL, FlashBP, JFlash
VTref=3.311V

Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: N32G033X8
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 "N32G033X8" selected.

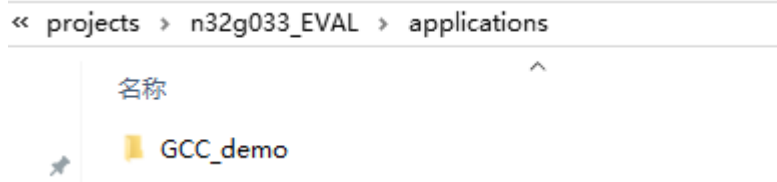
Connecting to target via SWD
Found SW-DP with ID 0x0BB11477
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x04770021)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410CC200. Implementer code: 0x41 (ARM)
Found Cortex-M0 r0p0, Little endian.
FPUnit: 2 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB008 SCS
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 000BB00A DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 000BB00B FPB
Cortex-M0 identified.
J-Link>
```

4.SDK Contens

SDK follows the issued SDK version, currently using V0.1.0, on this basis to make the following modifications to adapt to GCC development environment.

4.1 Makefile

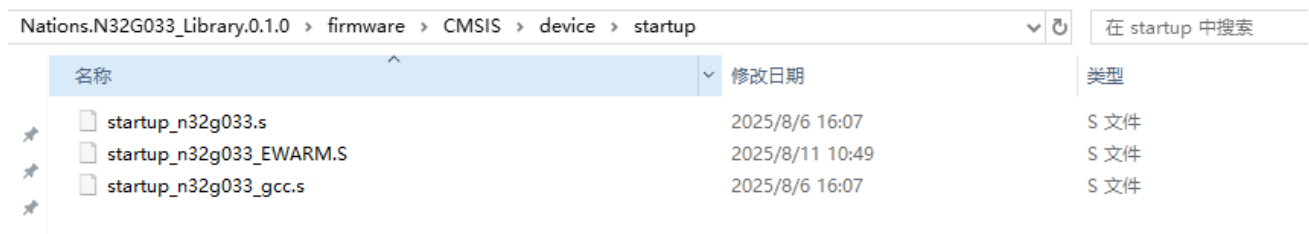
In the "GCC" folder of the GCC_demo routine directory in the SDK package:



The "Makefile" file is the GCC compilation script file.

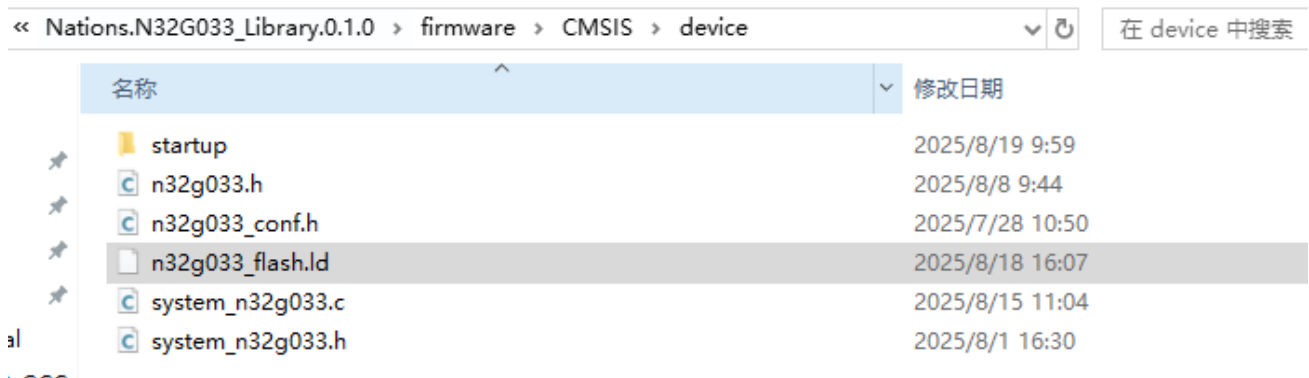
4.2 .s file

In the SDK package "[Nsing.n32g033_Library.0.1.0 \firmware\CMSIS\device\ startup](#)" there is a GCC compiler .S file “[startup_n32g033_gcc.s](#)” in the corresponding path.



4.3 .ld file

In the SDK package, "[Nsing.N32G033_Library.0.1.0\firmware\CMSIS\ device](#)" there is a .ld file "[n32g033_flash.ld](#)" in the corresponding path.



4.4 Printing remapping

The “[print_remap.c](#)” file is added in the “[bsp/src](#)” directory of the SDK package for serial port printing remapping.

« n32g430_EVAL » bsp » src				搜索"src"	
名称	修改日期	类型	大小		
log.c	2022/3/30 14:39	C 文件			
print_remap.c	2022/3/30 14:34	C 文件			

4.5 J-Link script

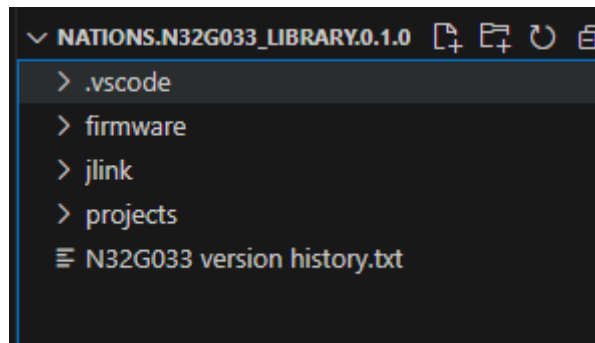
Added the jlink folder in the SDK home directory, which contains a Jlink download script for downloading firmware using the J-Link tool.

< Nations.... » jlink				在 jlink 中搜索	
名称	修改日期	类型	大小		
flashjlink	2020/11/24 15:28	JLINK 文件			

5. Compile and download

5.1 Workspace

Open the SDK folder in VScode and save it as a workspace. At this point, the ".vscode" folder will be generated under the SDK folder to place the workspace configuration file.



5.2 Working Directory

Take the GPIO routine LedBlink as an example to enter the project directory:

"Nsing.N32G033_Library.0.1.0\projects\n32g033_EVAL\applications\GCC_demo"

GCC project "GCC"

Project source file "src /xxx.c"

Project header file "inc/xxx.h"

Makefile file "GCC/Makefile"

5.3 Code Compilation

In the terminal of the VScode editor, switch to the "GCC" folder directory and type "make" to start compiling

```
PS D:\desktop\GCC\Nations.N32G033_Library.0.1.0\projects\n32g033_EVAL\applications\GCC_demo\GCC> make
```

And the .elf, .bin and .hex files are generated when compiled error-free

```
arm-none-eabi-size build/output.elf
text      data      bss      dec      hex filename
2912      1080      1572      5564      15bc build/output.elf
arm-none-eabi-objcopy -O ihex -S build/output.elf build/output.hex
arm-none-eabi-objcopy -O binary -S build/output.elf build/output.bin
```

In this case, the "build" folder is created under the "GCC" folder. The compiled firmware and intermediate files are stored in this folder.

5.4 Downloading Firmware

1. Connect PC->JLink->development board
2. On the terminal, type “[make download](#)”

```
PS D:\desktop\GCC\Nations.N32G033_Library.0.1.0\projects\n32g033_EVAL\applications\GCC_demo\GCC> make download
```

Some information will be printed in the process...Finally, the download is complete

```
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. After downloading, the system will automatically reset and start running
4. If the download fails, check the JLink configuration

5.5 Clearing Intermediate Files

Type "[make clean](#)" on the terminal to clear the intermediate files generated by the compilation.

6.Code debugging

6.1 VSCode set

There is a ".vscode" folder in the SDK working path, which contains "launch.json" workspace configuration files that need to be configured for code debugging:

« Natio... > .vscode

在 .vscode 中搜索

名称

launch.json

settings.json

tasks.json

2022/4/6 14:51

2021/11/12 16:42

2022/4/6 14:49

JSON 文件

JSON 文件

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             {
33                 "text": "file 'D:/desktop/GCC/Nations.N32G033_Library.0.1.0/projects/n32g033_EVAL/applications/GCC_demo/GCC/build/output.elf'",
34                 "description": "load file to gdb",
35                 "ignoreFailures": false
36             },
37             {
38                 "text": "load",
39                 "description": "download file to MCU",
40                 "ignoreFailures": false
41             },
42             {
43                 "text": "monitor reset",
44                 "description": "reset MCU",
45                 "ignoreFailures": false
46             },
47             {
48                 "text": "b main",
49                 "description": "set breakpoints at main",
50                 "ignoreFailures": true
51             },
52         ],
53         "launchCompleteCommand": "None",
54         // "preLaunchTask": "build"
55     }
56 ]
57 }
```

This is the vscode debugger configuration file, and the following changes should be made according to your project path:

1, specify the path to the **GDB** debugger :(absolute path)

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

The version of the **GDB** tool must match the version of the compiler tool. Otherwise, errors will be reported or some functions will be unavailable. The **arm-none-eabi-gdb.exe** tool is usually in the same directory as the **arm-none-eabi-gcc.exe** tool.

2, specify debug code **xxx.elf** file path: (Note: path cannot be too long)

```
"text": "file 'D:/desktop/GCC/Nations.N32G033_Library.0.1.0/projects/n32g033_EVAL/applications/GCC_demo/GCC/build/output.elf'",
```

6.2 Makefile Settings

Open the routine "**GCC/Makefile**" file:

```
download:
    @$(JK_DPATH)JLink.exe -device $(CHIP_TYPE) -if SWD -speed 4000 -autoconnect 1 -CommanderScript $(JKS_DIR)/flash.jlink
    @echo "Download Completed!"

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

# *** EOF ***
```

1, you can see that there is a debug startup configuration pointing to the JLinkGDBserver server in the JLink installation directory.

2. The **make** command is in debug mode by default, with some debugging information. If you want to switch to the release version, compile the code with the following command: **make Release=y**

6.3 Debugging Examples

Using the GCC_demo project as an example, see how to start code debugging:

1. Open SDK project in vscode, switch to **GCC_demo/GCC** directory in terminal, and type **make** to compile code

```
PS D:\desktop\GCC\Nations.N32G033_Library.0.1.0\projects\n32g033_EVAL\applications\GCC_demo\GCC> make
```

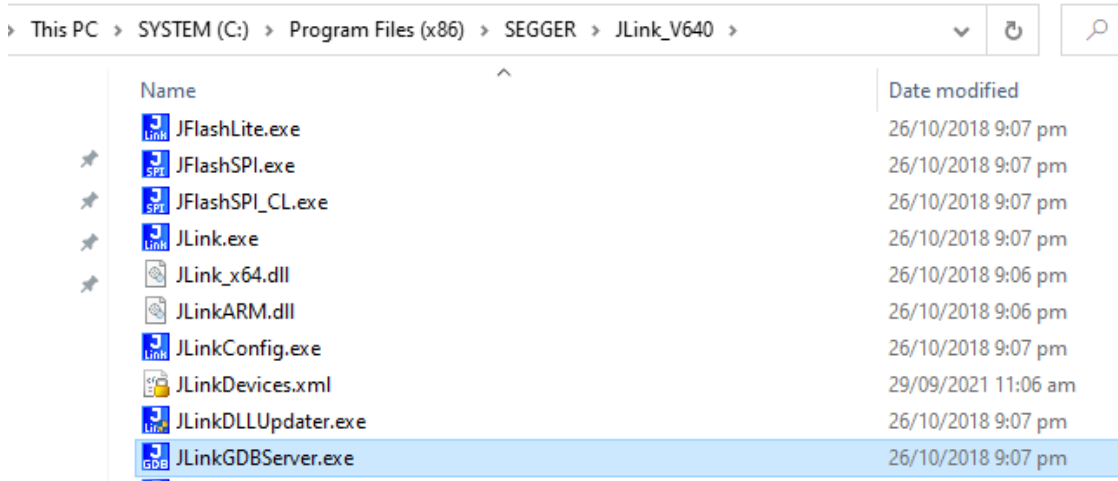
```
arm-none-eabi-size build/output.elf
   text    data     bss     dec     hex filename
   2912    1080     1572    5564    15bc build/output.elf
arm-none-eabi-objcopy -O ihex -S build/output.elf build/output.hex
arm-none-eabi-objcopy -O binary -S build/output.elf build/output.bin
```

output.elf, output.bin, output.hex files are generated in **GCC/build** folder.

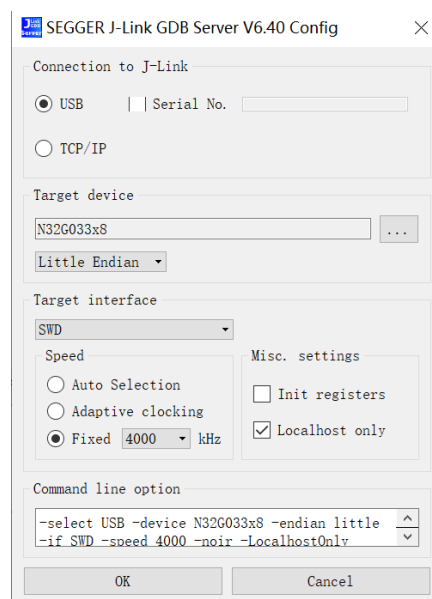
2. Refer to 6.1 and 6.2 section to configure the path in the launch.json files.

3, connect the JLink debugger to the development board, power on and prepare.

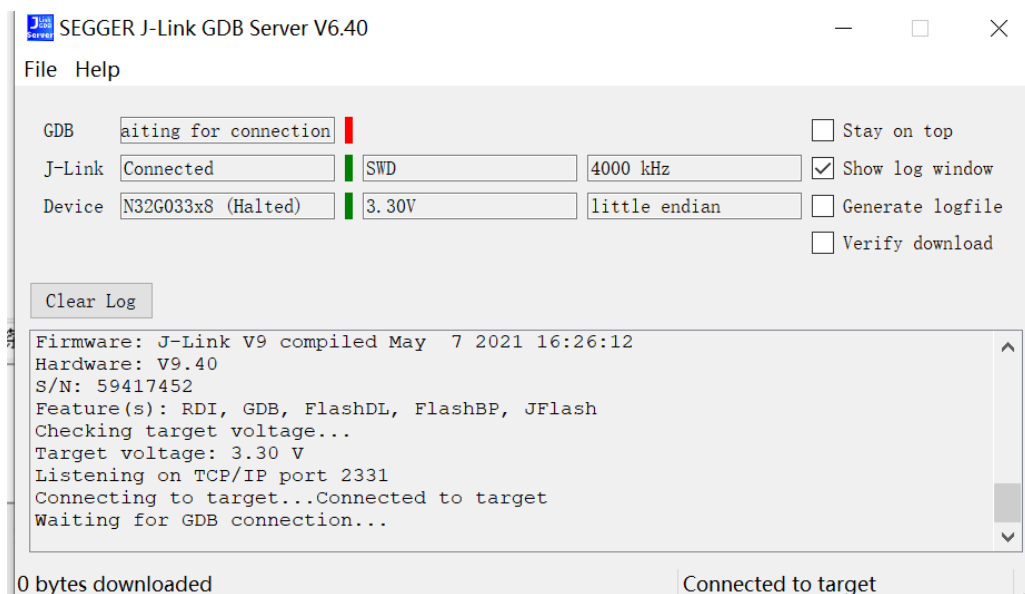
4, Go to your JLink installation directory and double-click **JLinkGDBServer.exe**



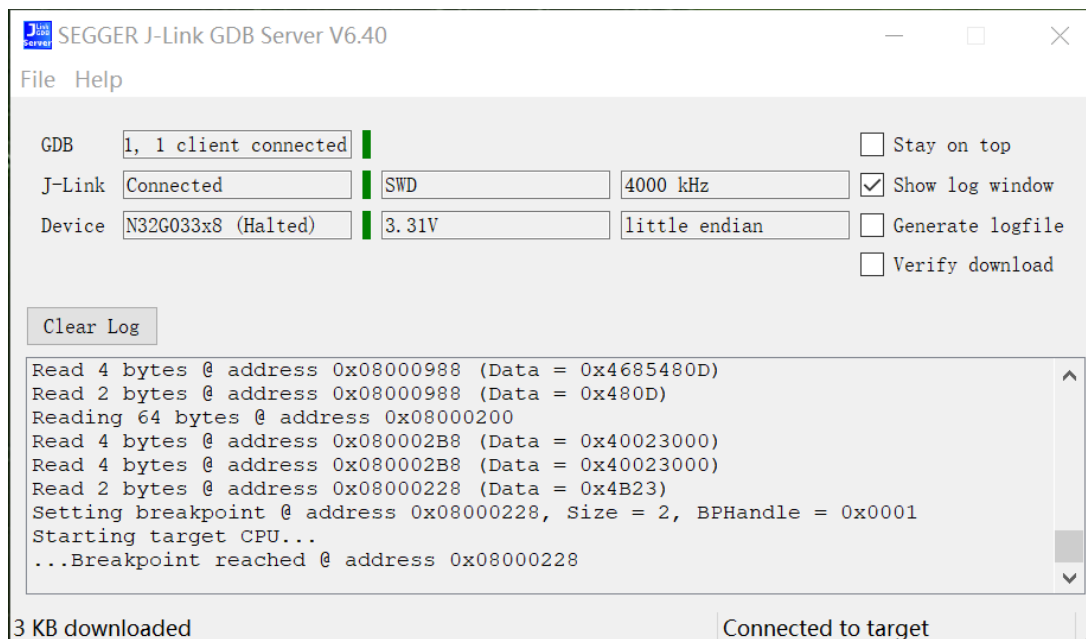
To configure ports, protocols, and chip models, click [OK](#)



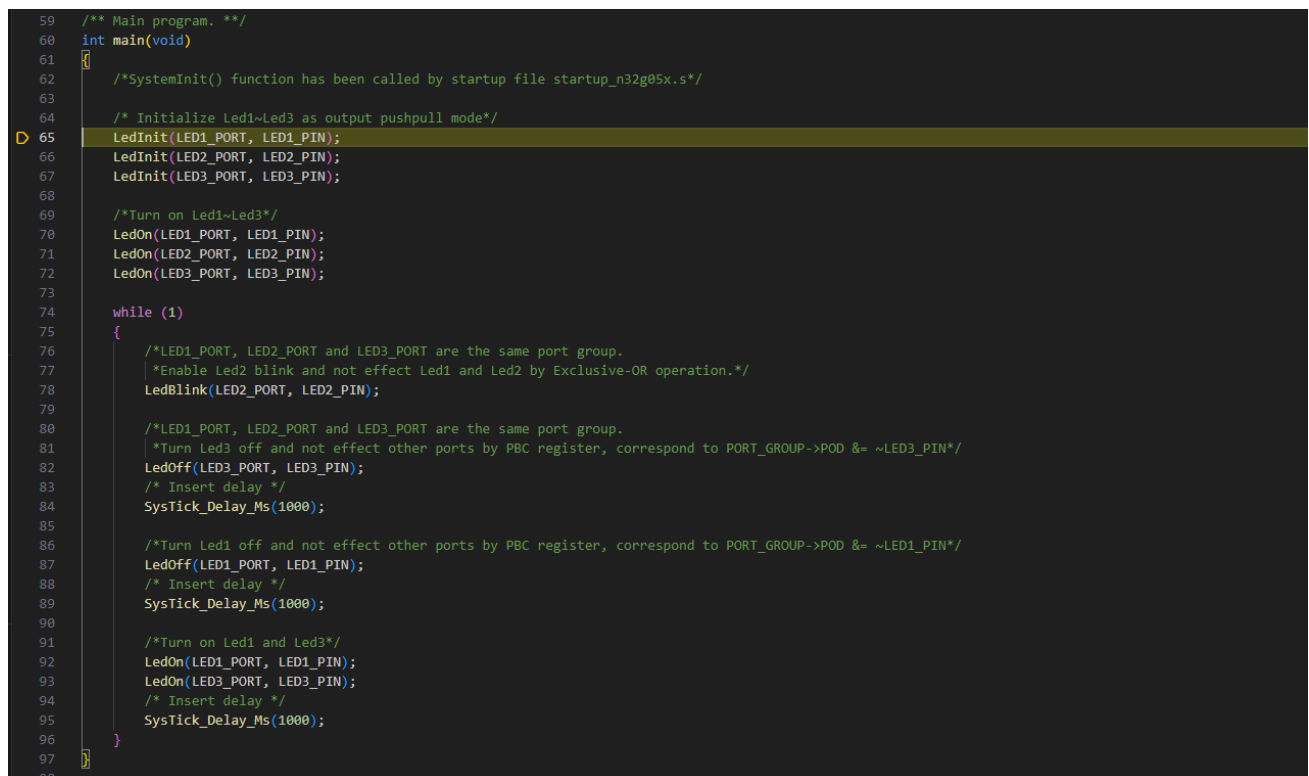
If the JLink debugger is successfully connected to the chip:



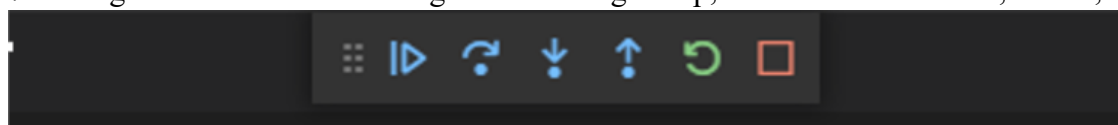
5. Under vscode working environment, press "F5" or click "Run" -> "Start debugging". At this time, it can be seen that the label below turns green, indicating that gdb tool successfully connects to JLinkGDBserver.



6, vscode automatically switches to the debug window



7. Debug buttons above the debug window: single step, continuous execution, restart, stop, etc



8. Now you can step and run at full speed

7. Configuration changes

7.1 Chip Models

If you are using chips other than the N32G033 family, you need to modify the variables "`TARGET_PLATFORM`" and "`DEFS`" in the makefile.

```

32
33 #####
34 # chip platform info
35 #####
36 TARGET_PLATFORM := n32g033
37 DEFS += -DN32G033
38 DEFS += -DUSE_STDPERIPH_DRIVER
39

```

Different chip models need to change the Flash&SRAM capacity, stack top position, and stack size in the .ld file according to the data manual.

```

51 /* Highest address of the user mode stack */
52 _estack = 0x20001800; /* end of RAM */
53 /* Generate a link error if heap and stack don't fit into RAM */
54 _Min_Heap_Size = 0x200; /* required amount of heap */
55 _Min_Stack_Size = 0x400; /* required amount of stack */
56
57 /* Specify the memory areas */
58 MEMORY
59 {
60 RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 0x1800 /* 6K */
61 FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 0x10000 /* 64K */
62 }
63

```

7.2 Firmware Download Algorithm

You need to type the full chip model so that JLink can properly match the download algorithm.

```

172 CHIP_TYPE = N32G033x8

```

Configure the path to download the tool: configure it according to your installation directory

```

#####
# 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 Using the SDK algorithm library

By default, the library is not used. Please modify the variable `USELIB = 1` to use the library.

```
36 #####
37 # Algo libs
38 #####
39 USELIB = 0
```

7.4 DEBUG configuration

The default "`make`" compilation is with "`-g`" debugging information. If you want to build a release version, use "`make release=y`".

7.5 Optimization Grade

Default use of '`-O0`' optimization level, no code optimization done.

8. Version history

Date	Version	Modify
2022/03/30	V1.0.0	The initial release
2025/8/6	V1.1.0	1. Chapter 7.1 adds a description of modifying the .ld file according to different models
2025.8.19	V1.2.0	1. Move the example project to the path "Nsing. N32G033_Library. 0.1.0 \ projects \ n32G033-EVAL \ Templates" 2. Example explanation: Change from "N32G430" to "N32G033"

9. Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections, enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.