Application Note

ARM GCC Development Environment Based on Windows Application Note

Contens

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 Contens** 8

4.1 Makefile 8
4.2 .s file 8
4.3 .ld file 8
4.4 Printing remapping 9
4.5 JLink script 9
4.6 Clearing Scripts 9

5. **Compile and Download** 10

5.1 Workspace 10
5.2 Working Directory 10
5.3 Code Compilation 10
5.4 Downloading Firmware 11
5.5 Clearing Intermediate Files 11

6. **Code Debugging** 12

6.1 VSCode Settings 12
6.2 Makefile Settings 13
6.3 Debugging Examples 13

7. **Configuration changes** 17

7.1 Chip Model 17
7.2 Firmware Download Algorithm 17
7.3 Using the SDK Algorithm Library 17
7.4 Debug Configuration 18
7.5 Optimization Level 18

8. **Version history** 19

9. **Disclaimer** 20

1. Overview

Using N32G430 series MCU as an example, this paper introduces the methods of setting up development environment, code compiling, firmware downloading and code debugging based on VSCode editor, GCC compilation tool chain and GDB debugging tool under Windows environment.

2. Development Tool

2.1 Software

- Editor Visual Studio Code 1.5x.x or above
- Compile tool chain arm-none-eabi-gcc 6.3.1 or above
- Make for Windows
- Download and debugging tool JLink_v6.40 or above (not exceeding hardware supported version)

2.2 Hardware

- Development board N32G430C8L7-STB V1.0
- JLink downloader V9.2 or above (not below the software supported version)

3. Development Environment Setup

3.1 Installing VSCode

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

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

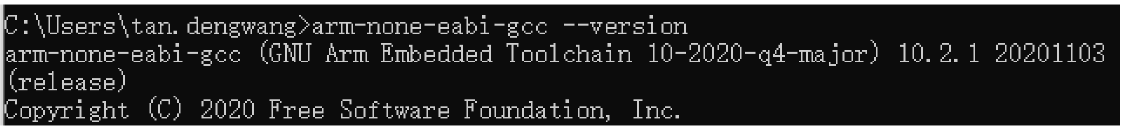
3.2 Installing the GCC Compilation Tool Chain

- **Download address:**
  https://launchpad.net/gcc-arm-embedded/+announcement/28093
  Example version: 10-2020-q4-major

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

The installation is successful if the content shown in the following figure appears:

Figure 3-1 Information About the Successful Installation of GCC



```
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 the installation is not successful, please do the following two checks.

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

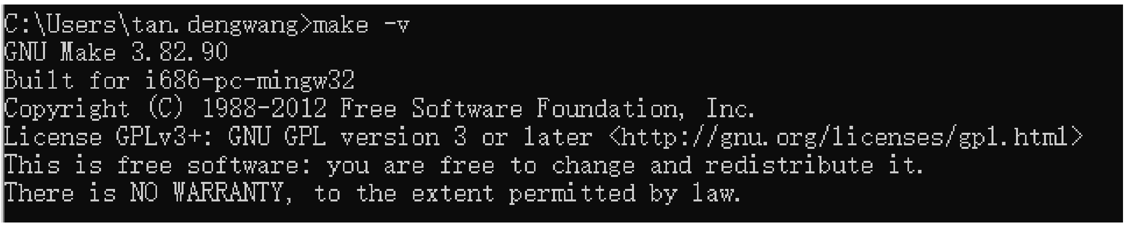
3.3 Installing Make for Windows

This tool is used to parse Makefile scripts. You can choose to install one of the following two software.

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

Check whether the installation is successful: Open a DOS command line window and type “make –v”. The installation is successful if the content shown in the following figure appears:

Figure 3-2 Information About the Successful Installation of Make for Windows



```
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 the installation is not successful, please do the following two checks

1. Check that the environment variables are properly added

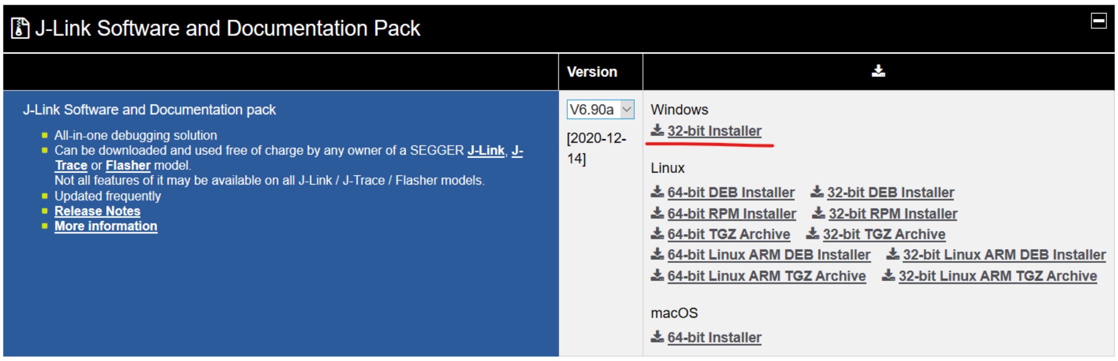
2. Go to the bin folder of the corresponding “make” installation directory to check whether the “make.exe” file name is correct

3.4 Installing the JLink Tool

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

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

Figure 3-3 Installation Interface



J-Link Software and Documentation Pack

Version

J-Link Software and Documentation pack

- All-in-one debugging solution
- Can be downloaded and used free of charge by any owner of a SEGGER J-Link, J-Trace or Flasher model.
  Not all features of it may be available on all J-Link / J-Trace / Flasher models.
- Updated frequently
- Release Notes
- More information

V6.90a

[2020-12-14]

Windows

32-bit Installer

Linux

64-bit DEB Installer 32-bit DEB Installer
64-bit RPM Installer 32-bit RPM Installer
64-bit TGZ Archive 32-bit TGZ Archive
64-bit Linux ARM DEB Installer 32-bit Linux ARM DEB Installer
64-bit Linux ARM TGZ Archive 32-bit Linux ARM TGZ Archive

macOS

64-bit Installer

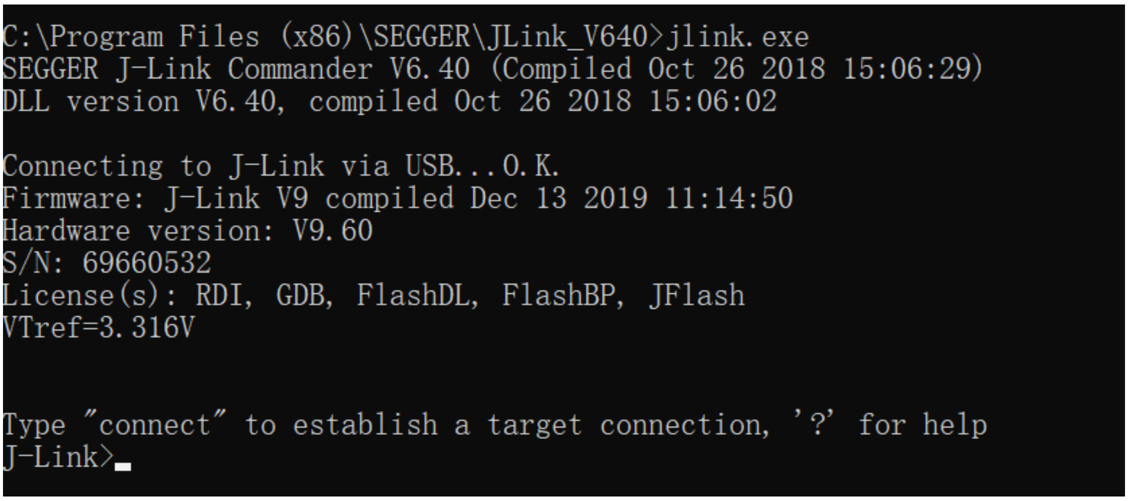
3.5 Adding Chip Support

After installing JLink, you need to add our company's chip patch package to JLink, so that the download algorithm can be correctly obtained during downloading and debugging.
For details, refer to <JLink Tool Adding Nsing Chip.7z>.

3.6 JLink Download Test

- **Test the JLink environment installation**
  1. Connect the PC and JLink 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”.
     The below figure shows that the PC successfully connected to the JLink debugger.

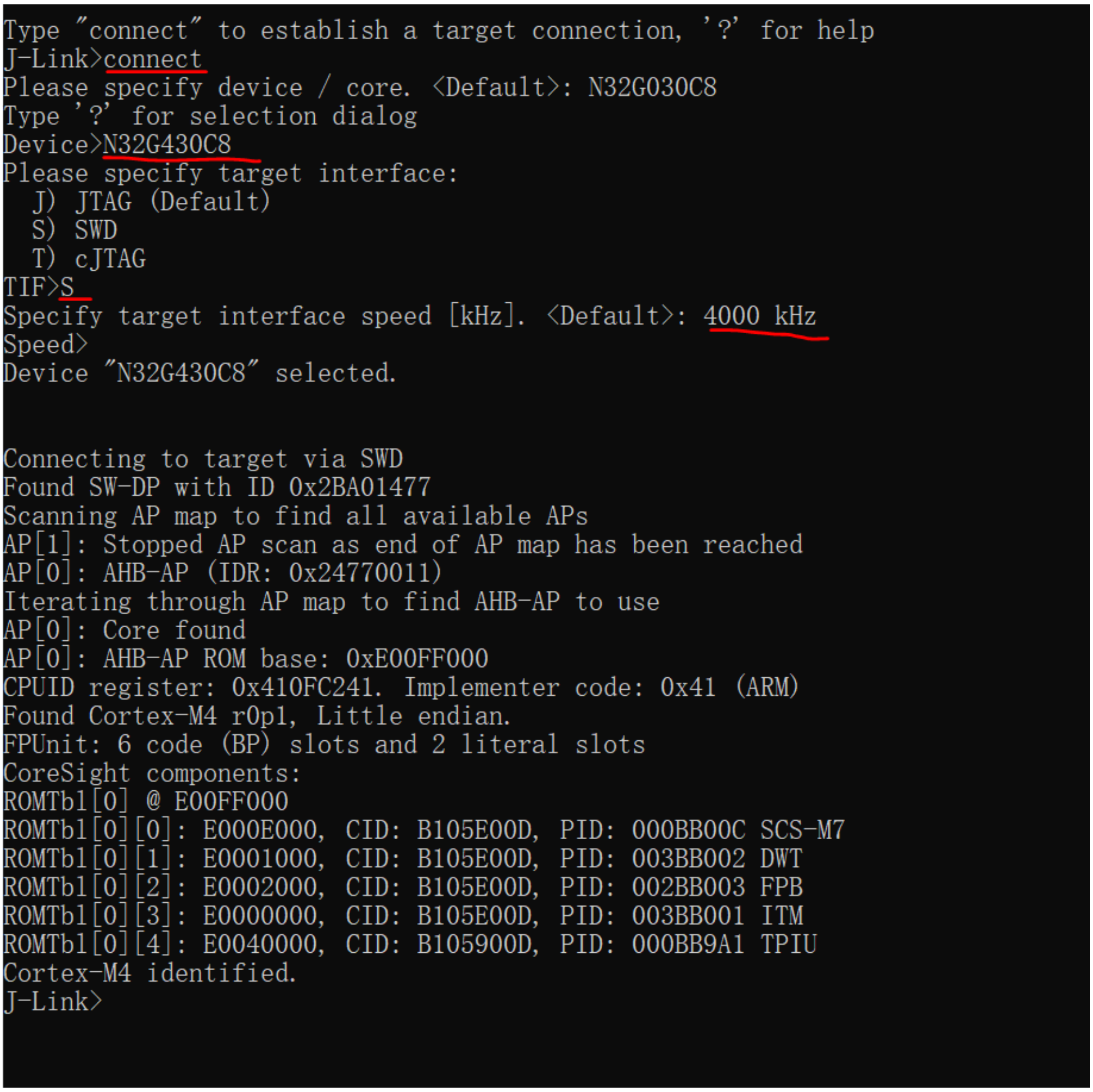
Figure 3-4 Information About PC Successfully Connected To The JLink Debugger.



```
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>
```

3. Then type the information sequentially as prompted: “connect”, “N32G430C8”, “SWD”, “4000”. If the previous operation is successful, you will see the following output information, JLink download debugging environment can be used normally.

Figure 3-5 Information About JLink Download Debugging Environment Can Be Used Normally



```
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: N32G030C8
Type '?' for selection dialog
Device>N32G430C8
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "N32G430C8" selected.


Connecting to target via SWD
Found SW-DP with ID 0x2BA01477
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: 0x24770011)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: 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
Cortex-M4 identified.
J-Link>
```

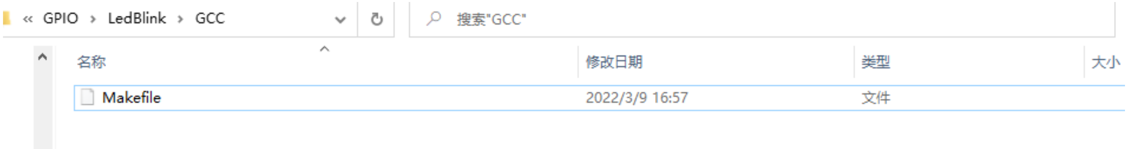
4. SDK Contens

SDK follows the issued SDK version, currently using V1.0.0. On this basis, the following modifications are made to adapt to GCC development environment.

4.1 Makefile

Add “GCC” folder under module routines directory in SDK package: (please copy "GCC" folder to each routine)

Figure 4-1 Add “GCC” Folder



« GPIO › LedBlink › GCC

搜索"GCC"

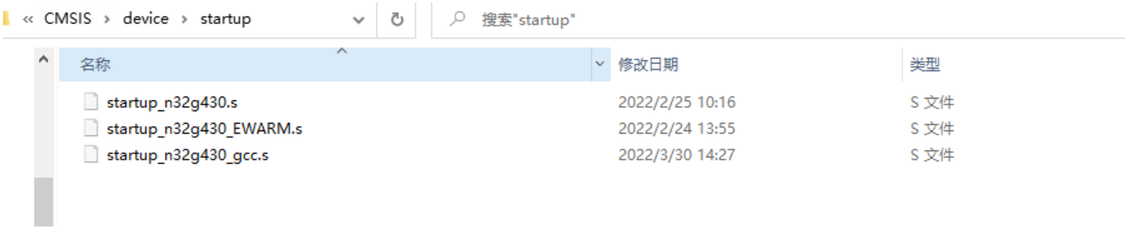
名称	修改日期	类型	大小
Makefile	2022/3/9 16:57	文件	

The “Makefile” file is the GCC compilation script file.

4.2 .s File

In “Nations.n32g430_Library.1.0.0\firmware\CMSIS\device\startup” path of the SDK package, there is a “startup_n32g430_gcc.s” file corresponding to the .s file.

Figure 4-2 The “startup_n32g430_gcc.s” File



« CMSIS › device › startup

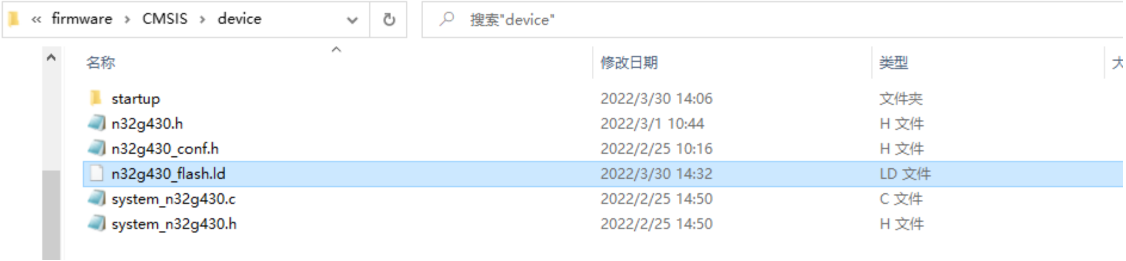
搜索"startup"

名称	修改日期	类型
startup_n32g430.s	2022/2/25 10:16	S 文件
startup_n32g430_EWARM.s	2022/2/24 13:55	S 文件
startup_n32g430_gcc.s	2022/3/30 14:27	S 文件

4.3 .ld File

In “Nations.N32G430_Library.1.0.0\firmware\CMSIS\device” path of the SDK package, there is a corresponding .s file for the GCC compiler named “n32g430_flash.ld”.

Figure 4-3 The “n32g430_flash.ld” File



« firmware › CMSIS › device

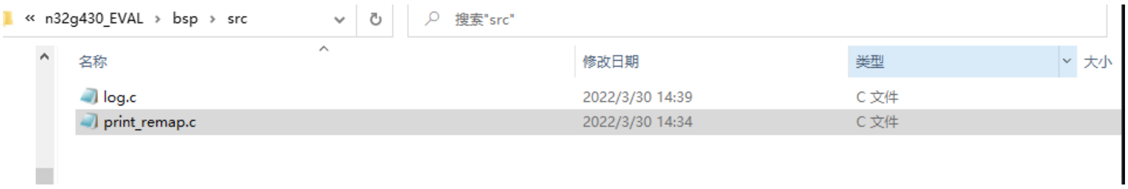
搜索"device"

名称	修改日期	类型	大
startup	2022/3/30 14:06	文件夹	
n32g430.h	2022/3/1 10:44	H 文件	
n32g430_conf.h	2022/2/25 10:16	H 文件	
n32g430_flash.ld	2022/3/30 14:32	LD 文件	
system_n32g430.c	2022/2/25 14:50	C 文件	
system_n32g430.h	2022/2/25 14:50	H 文件	

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.

Figure 4-4 The “print_remap.c” File



« n32g430_EVAL › bsp › src

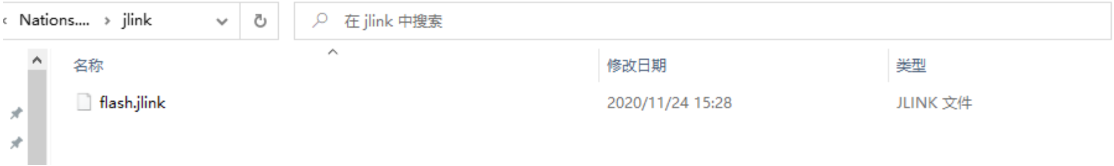
搜索"src"

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 JLink Script

Added the “jlink” folder in home directory of the SDK package, which contains a JLink download script for downloading firmware using the JLink tool.

Figure 4-5 Add The “jlink” Folder



‹ Nations.... › jlink

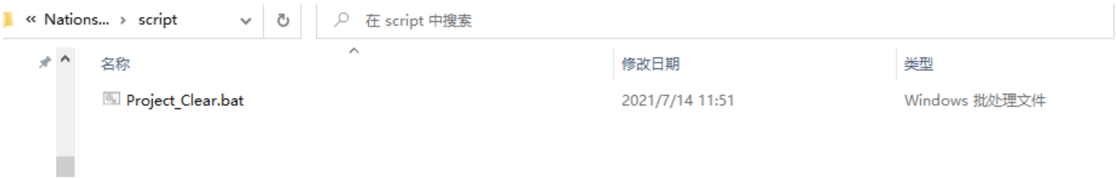
在 jlink 中搜索

名称	修改日期	类型
flash.jlink	2020/11/24 15:28	JLINK 文件

4.6 Clearing Scripts

The “script” folder is added in home directory of the SDK package, and there is a “.bat” script in this folder, which is used to clear intermediate files generated during compilation.

Figure 4-6 A “.bat” Script in “script” Folder



« Nations... › script

在 script 中搜索

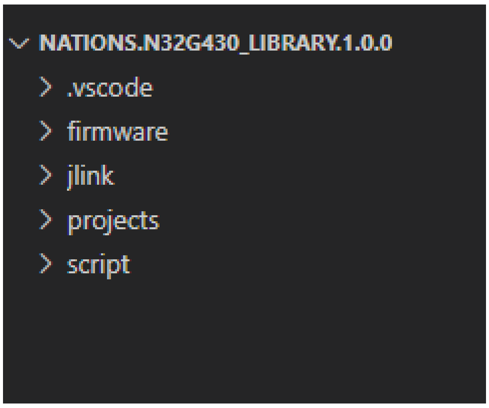
名称	修改日期	类型
Project_Clear.bat	2021/7/14 11:51	Windows 批处理文件

5. Compile and Download

5.1 Workspace

Open the SDK folder in VSCode and save it as a workspace. The “.vscode” folder will be generated under the SDK folder to place the workspace configuration file.

Figure 5-1 The “.vscode” Folder



```
∨ NATIONS.N32G430_LIBRARY.1.0.0
  > .vscode
  > firmware
  > jlink
  > projects
  > script
```

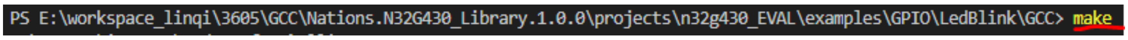
5.2 Working Directory

Using the GPIO routine LedBlink as an example to enter the project directory:
“Nations.N32G430_Library.1.0.0\projects\n32g430_EVAL\examples\GPIO\LedBlink”
KEIL project “MDK-ARM”
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, then type “make” to start compiling

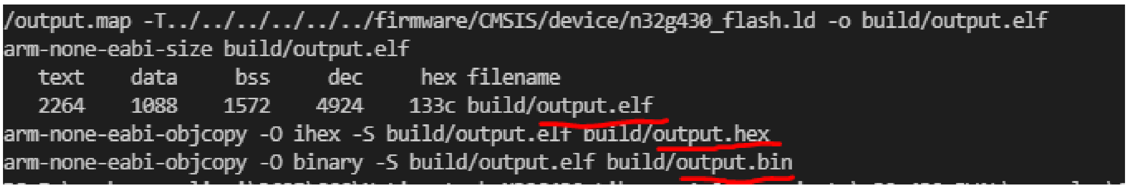
Figure 5-2 Type “make” in the “GCC” Folder Directory



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

The .elf, .bin and .hex files will be generated after successful compilation without errors.

Figure 5-3 The .elf, .bin and .hex Files



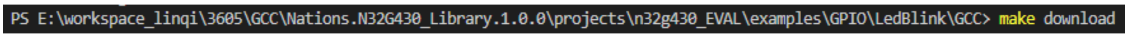
```
/output.map -T../../../../../../firmware/CMSIS/device/n32g430_flash.ld -o build/output.elf
arm-none-eabi-size build/output.elf
   text    data     bss     dec     hex filename
   2264    1088    1572    4924    133c build/output.elf
arm-none-eabi-objcopy -O ihex -S build/output.elf build/output.hex
arm-none-eabi-objcopy -O binary -S build/output.elf build/output.bin
```

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. In the terminal, type “make download”

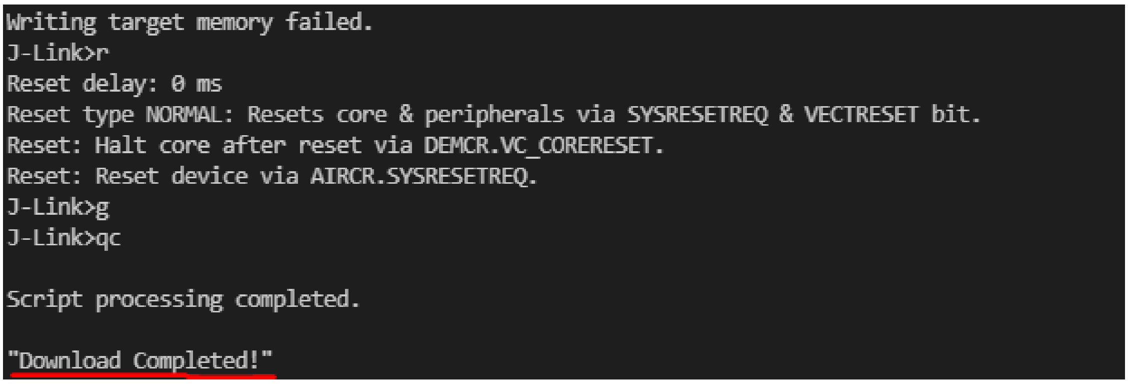
Figure 5-4 Type “make download” in The “GCC” Folder Directory



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

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

Figure 5-5 Information About Download Completed



```
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 download is complete, the system will automatically reset and start running
4. If the download fails, please check the JLink configuration

5.5 Clearing Intermediate Files

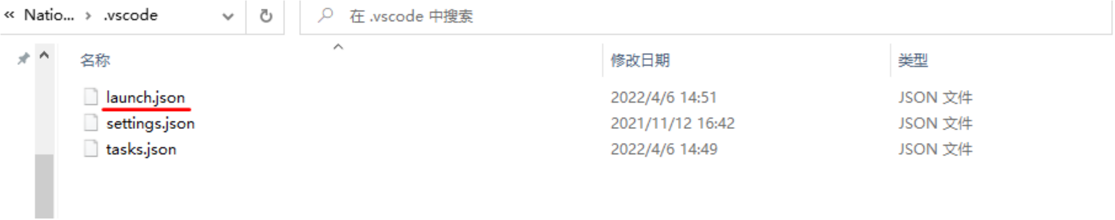
Type “make clean” on the terminal to clear the intermediate files generated by the compilation.

6. Code Debugging

6.1 VScode Settings

There is a “.vscode” folder in the SDK working path, which contains “launch.json” workspace configuration file that needs to be configured for code debugging:

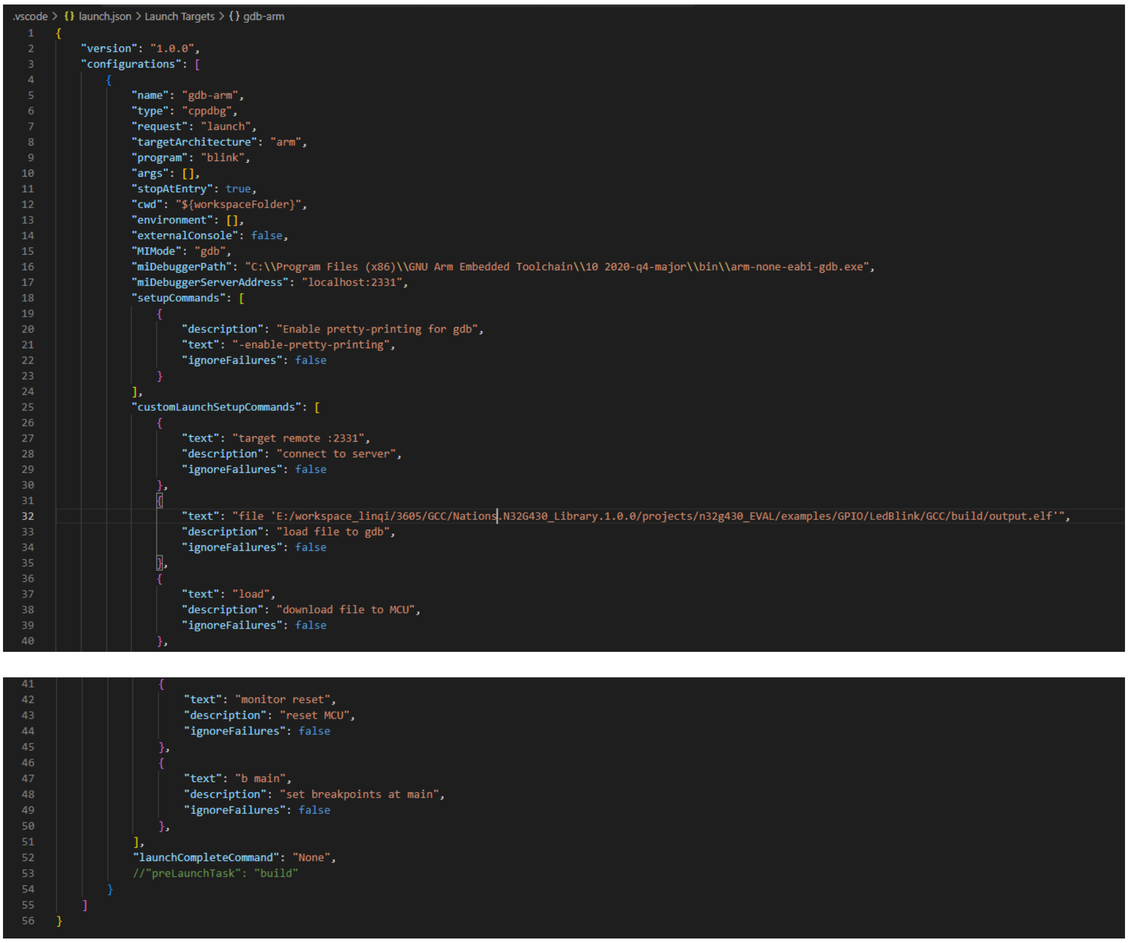
Figure 6-1 The “launch.json” Workspace Configuration File in The SDK Working Path



名称	修改日期	类型
launch.json	2022/4/6 14:51	JSON 文件
settings.json	2021/11/12 16:42	JSON 文件
tasks.json	2022/4/6 14:49	JSON 文件

launch.json:

Figure 6-2 The Code of the VScode Debugger Configuration File

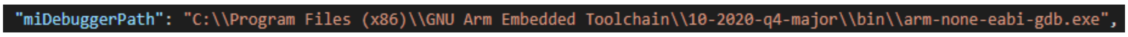


```
{
    "version": "1.0.0",
    "configurations": [
        {
            "name": "gdb-arm",
            "type": "cppdbg",
            "request": "launch",
            "targetArchitecture": "arm",
            "program": "blink",
            "args": [],
            "stopAtEntry": true,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\Program Files (x86)\\GNU Arm Embedded Toolchain\\10 2020-q4-major\\bin\\arm-none-eabi-gdb.exe",
            "miDebuggerServerAddress": "localhost:2331",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": false
                }
            ],
            "customLaunchSetupCommands": [
                {
                    "text": "target remote :2331",
                    "description": "connect to server",
                    "ignoreFailures": false
                },
                {
                    "text": "file 'E:/workspace_linqi/3605/GCC/Nations.N32G430_Library.1.0.0/projects/n32g430_EVAL/examples/GPIO/LedBlink/GCC/build/output.elf'",
                    "description": "load file to gdb",
                    "ignoreFailures": false
                },
                {
                    "text": "load",
                    "description": "download file to MCU",
                    "ignoreFailures": false
                },
                {
                    "text": "monitor reset",
                    "description": "reset MCU",
                    "ignoreFailures": false
                },
                {
                    "text": "b main",
                    "description": "set breakpoints at main",
                    "ignoreFailures": false
                },
            ],
            "launchCompleteCommand": "None",
            //"preLaunchTask": "build"
        }
    ]
}
```

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)

Figure 6-3 The Path To The GDB Debugger

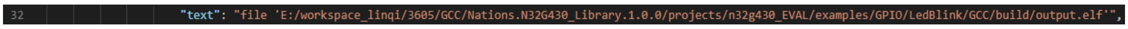


```
"miDebuggerPath": "C:\\Program Files (x86)\\GNU Arm Embedded Toolchain\\10-2020-q4-major\\bin\\arm-none-eabi-gdb.exe",
```

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 the debug code “xxx.elf” file path: (Note: path cannot be too long)

Figure 6-4 The Path of Debug Code “xxx.elf” File

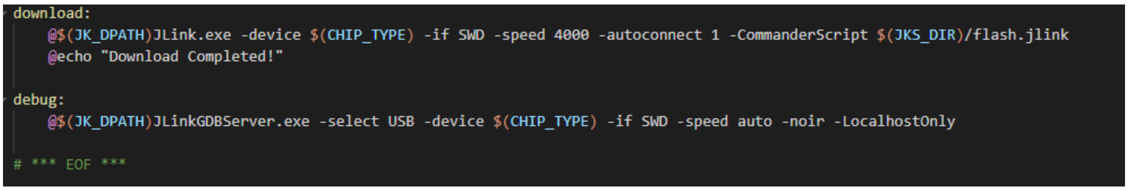


```
                "text": "file 'E:/workspace_linqi/3605/GCC/Nations.N32G430_Library.1.0.0/projects/n32g430_EVAL/examples/GPIO/LedBlink/GCC/build/output.elf'",
```

6.2 Makefile Settings

Open the routine “GCC/Makefile” file:

Figure 6-5 The Code from 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 ***
```

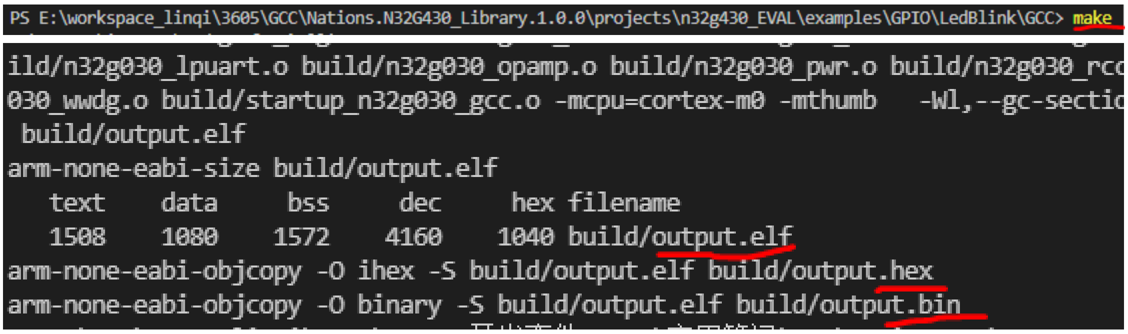
1. There is a debug startup configuration pointing to the JLinkGDBserver in the JLink installation directory.
2. The “make” command is in debug mode by default, with some debugging information. If you want to switch to the release version, you need to use the following command when compiling the code: “make release =y”

6.3 Debugging Examples

Using the GPIO LedBlink project as an example to show how to start code debugging:

1. Open SDK project in VScode, switch to “LedLink/GCC” directory in terminal, and type “make” to compile code

Figure 6-6 Information about Typing “make” To Compile Code



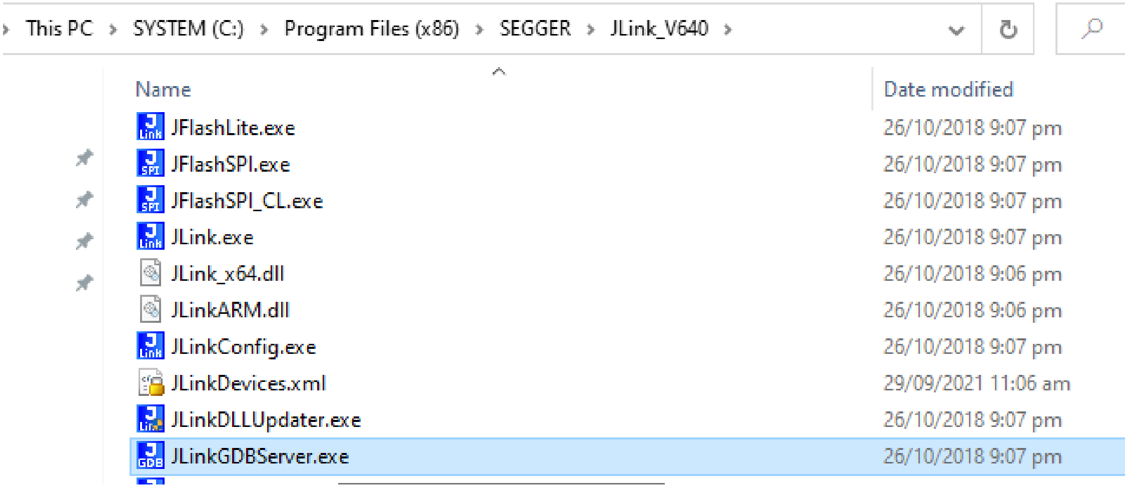
```
PS E:\workspace_linqi\3605\GCC\Nations.N32G430_Library.1.0.0\projects\n32g430_EVAL\examples\GPIO\LedBlink\GCC> make
```

```
ild/n32g030_lpuart.o build/n32g030_opamp.o build/n32g030_pwr.o build/n32g030_rcc
030_wwdg.o build/startup_n32g030_gcc.o -mcpu=cortex-m0 -mthumb   -Wl,--gc-sectio
 build/output.elf
arm-none-eabi-size build/output.elf
   text    data     bss     dec     hex filename
   1508    1080    1572    4160    1040 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
```

The output.elf, output.bin, output.hex files are generated in “GCC/build” folder.

2. Refer to Section 6.1 and 6.2 to configure the path in the launch.json file.
3. Connect the JLink debugger to the development board and power on.
4. Go to JLink installation directory and double-click JlinkGDBServer.exe

Figure 6-7 The JlinkGDBServer.exe in JLink Installation Directory

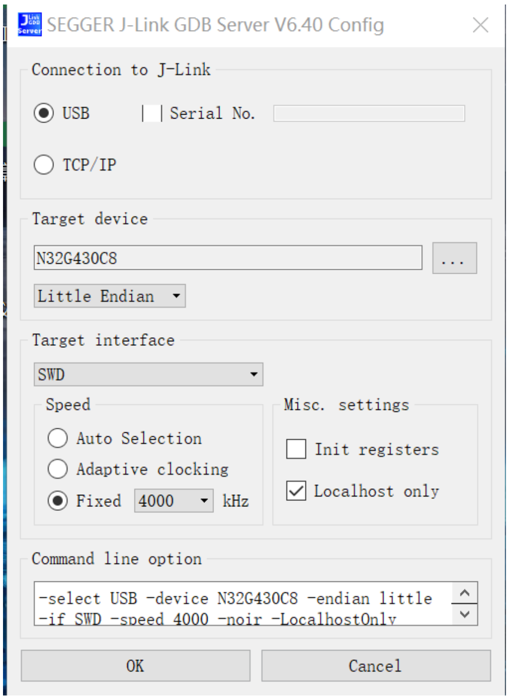


This PC › SYSTEM (C:) › Program Files (x86) › SEGGER › JLink_V640 ›

Name	Date modified
JFlashLite.exe	26/10/2018 9:07 pm
JFlashSPI.exe	26/10/2018 9:07 pm
JFlashSPI_CL.exe	26/10/2018 9:07 pm
JLink.exe	26/10/2018 9:07 pm
JLink_x64.dll	26/10/2018 9:06 pm
JLinkARM.dll	26/10/2018 9:06 pm
JLinkConfig.exe	26/10/2018 9:07 pm
JLinkDevices.xml	29/09/2021 11:06 am
JLinkDLLUpdater.exe	26/10/2018 9:07 pm
JLinkGDBServer.exe	26/10/2018 9:07 pm

Configure port, protocol, and chip model, click “OK”

Figure 6-8 Configure Information



SEGGER J-Link GDB Server V6.40 Config

Connection to J-Link

● USB □ Serial No.

○ TCP/IP

Target device

N32G430C8

Little Endian

Target interface

SWD

Speed

○ Auto Selection

○ Adaptive clocking

● Fixed 4000 kHz

Misc. settings

□ Init registers

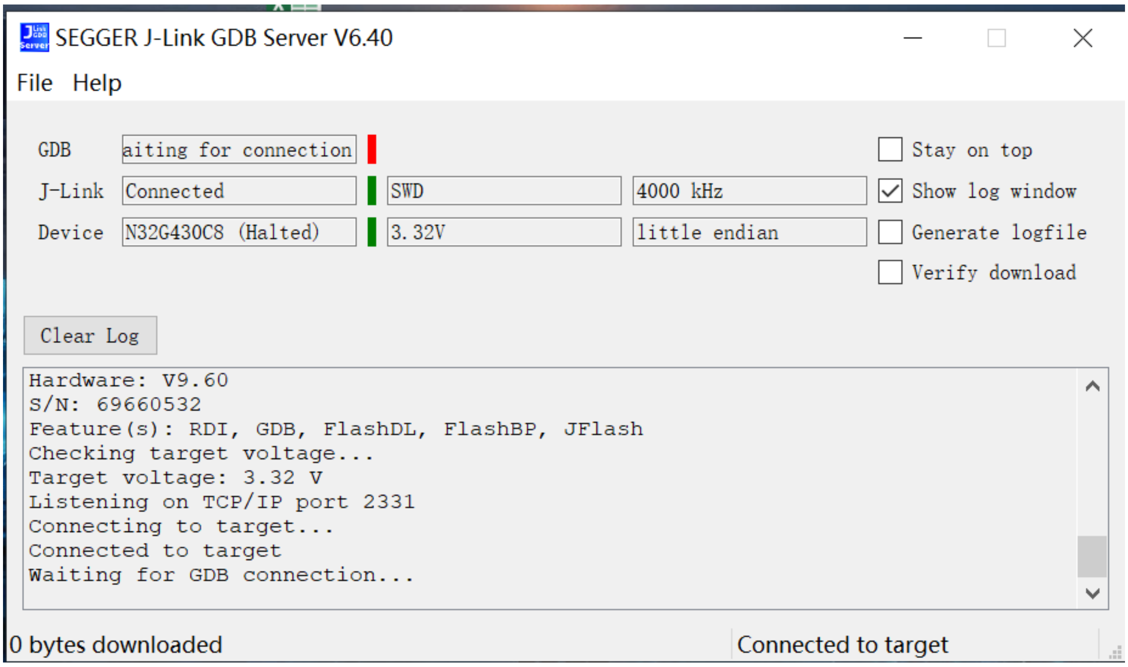
☑ Localhost only

Command line option

```
-select USB -device N32G430C8 -endian little
-if SWD -speed 4000 -noir -LocalhostOnly
```

If the JLink debugger is successfully connected to the chip:

Figure 6-9 The Interface That JLink Debugger Successfully Connected To The Chip


SEGGER J-Link GDB Server V6.40

GDB: aiting for connection 🟥
J-Link: Connected 🟩 SWD 4000 kHz
Device: N32G430C8 (Halted) 🟩 3.32V little endian

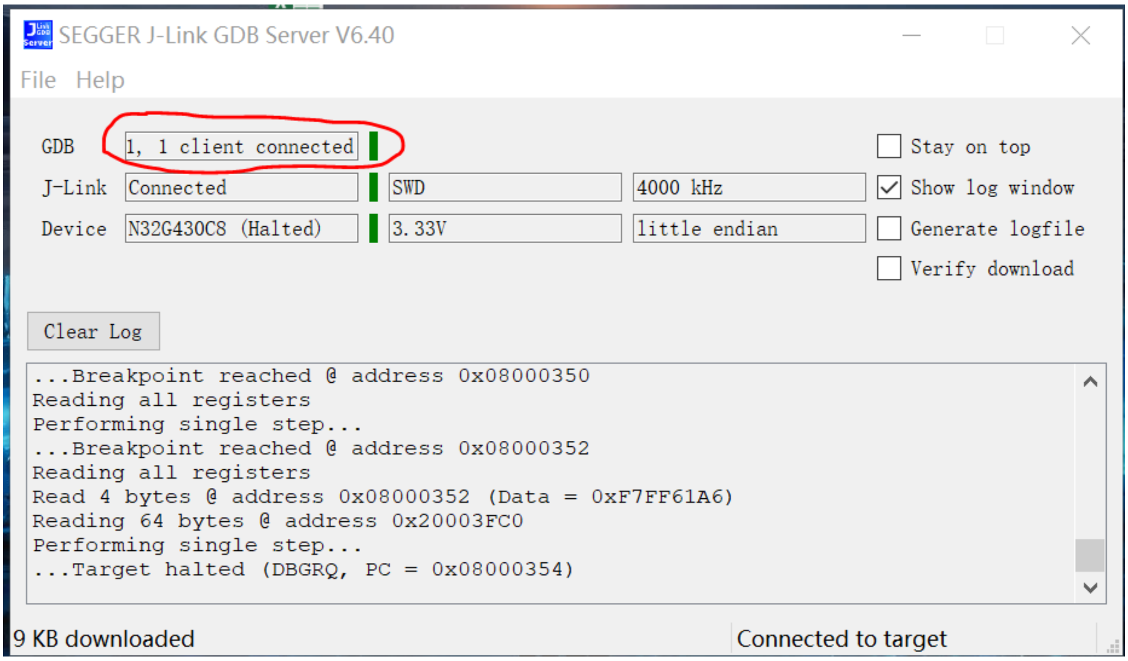
□ Stay on top
☑ Show log window
□ Generate logfile
□ Verify download

```
Hardware: V9.60
S/N: 69660532
Feature(s): RDI, GDB, FlashDL, FlashBP, JFlash
Checking target voltage...
Target voltage: 3.32 V
Listening on TCP/IP port 2331
Connecting to target...
Connected to target
Waiting for GDB connection...
```

0 bytes downloaded | Connected to target

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.

Figure 6-10 The Interface That GDB Tool Successfully Connect to JLinkGDBserver


SEGGER J-Link GDB Server V6.40

GDB: 1, 1 client connected 🟩
J-Link: Connected 🟩 SWD 4000 kHz
Device: N32G430C8 (Halted) 🟩 3.33V little endian

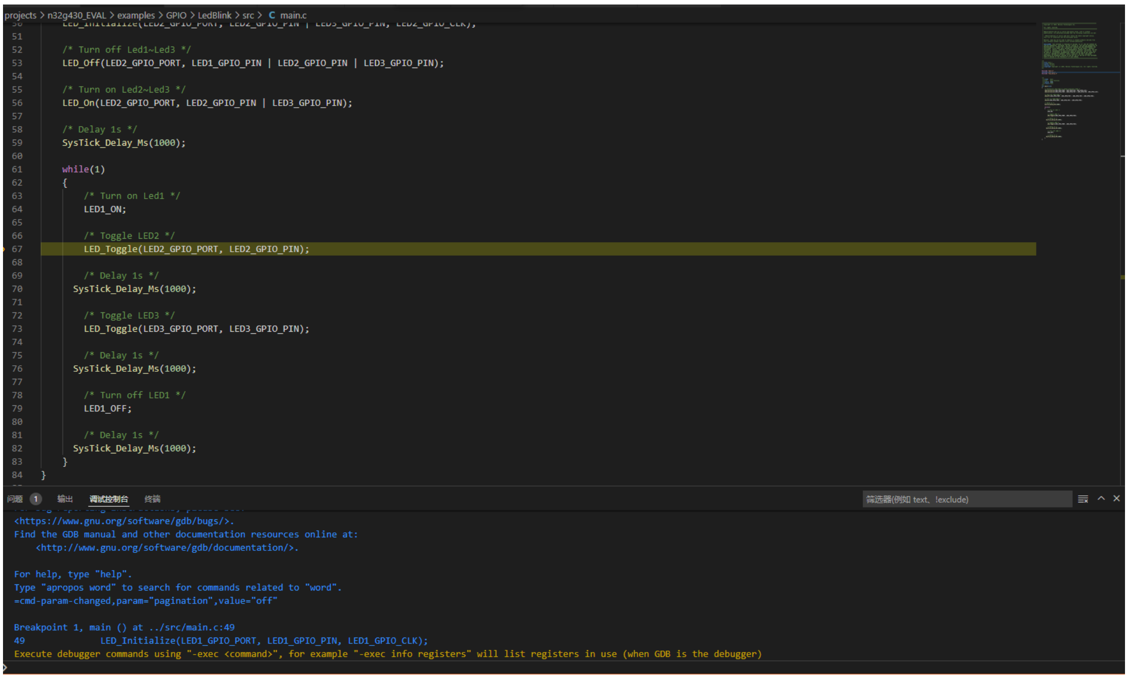
□ Stay on top
☑ Show log window
□ Generate logfile
□ Verify download

```
...Breakpoint reached @ address 0x08000350
Reading all registers
Performing single step...
...Breakpoint reached @ address 0x08000352
Reading all registers
Read 4 bytes @ address 0x08000352 (Data = 0xF7FF61A6)
Reading 64 bytes @ address 0x20003FC0
Performing single step...
...Target halted (DBGRQ, PC = 0x08000354)
```

9 KB downloaded | Connected to target

6. VSCode automatically switches to the debug window.

Figure 6-11 The Debug Interface

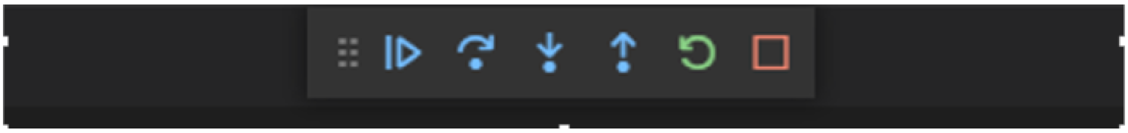

projects > n32g430_EVAL > examples > GPIO > LedBlink > src > C main.c
问题 1
输出
调试控制台
终端
筛选器(例如 text、!exclude)

```c
    LED_Initialize(LED2_GPIO_PORT, LED2_GPIO_PIN | LED3_GPIO_PIN, LED2_GPIO_CLK);

    /* Turn off Led1~Led3 */
    LED_Off(LED2_GPIO_PORT, LED1_GPIO_PIN | LED2_GPIO_PIN | LED3_GPIO_PIN);

    /* Turn on Led2~Led3 */
    LED_On(LED2_GPIO_PORT, LED2_GPIO_PIN | LED3_GPIO_PIN);

    /* Delay 1s */
    SysTick_Delay_Ms(1000);

    while(1)
    {
        /* Turn on Led1 */
        LED1_ON;

        /* Toggle LED2 */
        LED_Toggle(LED2_GPIO_PORT, LED2_GPIO_PIN);

        /* Delay 1s */
      SysTick_Delay_Ms(1000);

        /* Toggle LED3 */
        LED_Toggle(LED3_GPIO_PORT, LED3_GPIO_PIN);

        /* Delay 1s */
      SysTick_Delay_Ms(1000);

        /* Turn off LED1 */
        LED1_OFF;

        /* Delay 1s */
      SysTick_Delay_Ms(1000);
    }
}
```

```
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
=cmd-param-changed,param="pagination",value="off"

Breakpoint 1, main () at ../src/main.c:49
49              LED_Initialize(LED1_GPIO_PORT, LED1_GPIO_PIN, LED1_GPIO_CLK);
Execute debugger commands using "-exec <command>", for example "-exec info registers" will list registers in use (when GDB is the debugger)
```

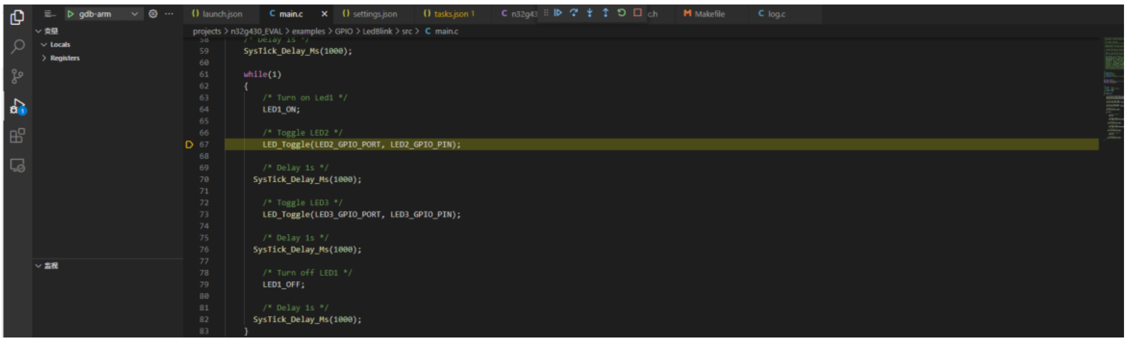
7. Debug buttons in the debug window: single step, continuous execution, restart, stop, etc.

Figure 6-12 The Debug Button



8. Now the program can run in single step or continuous execution mode

Figure 6-13 The Program Running in Single Step or Continuous Execution Mode



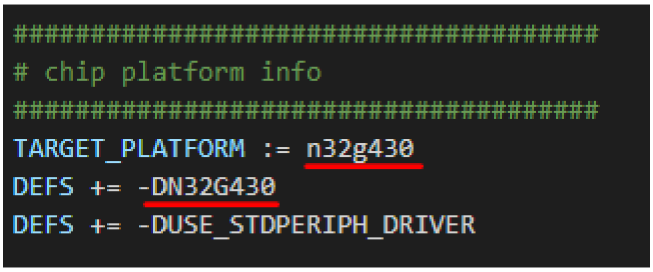
```c
    SysTick_Delay_Ms(1000);

    while(1)
    {
        /* Turn on Led1 */
        LED1_ON;

        /* Toggle LED2 */
        LED_Toggle(LED2_GPIO_PORT, LED2_GPIO_PIN);

        /* Delay 1s */
      SysTick_Delay_Ms(1000);

        /* Toggle LED3 */
        LED_Toggle(LED3_GPIO_PORT, LED3_GPIO_PIN);

        /* Delay 1s */
      SysTick_Delay_Ms(1000);

        /* Turn off LED1 */
        LED1_OFF;

        /* Delay 1s */
      SysTick_Delay_Ms(1000);
    }
```

7. Configuration Changes

7.1 Chip Model

If the chip used is not the N32G430 series, you need to modify the variables “TARGET_PLATFORM” and “DEFS” in the “Makefile” file.

Figure 7-1 The Variables “TARGET_PLATFORM” And “DEFS” in the “Makefile” File

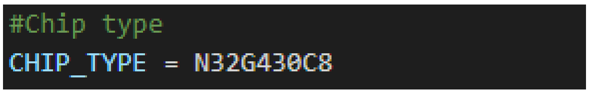


```
######################################
# chip platform info
######################################
TARGET_PLATFORM := n32g430
DEFS += -DN32G430
DEFS += -DUSE_STDPERIPH_DRIVER
```

7.2 Firmware Download Algorithm

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

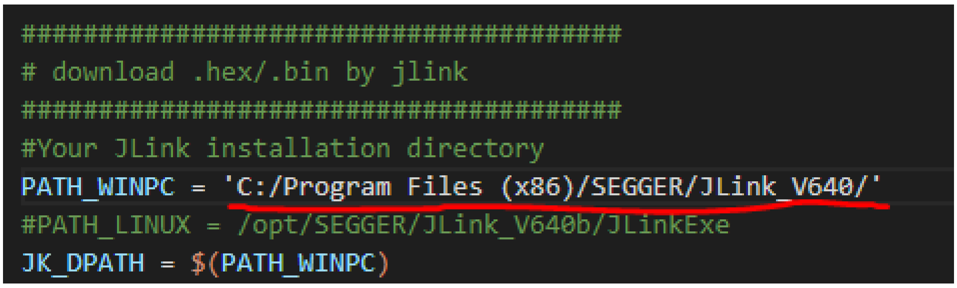
Figure 7-2 ChipType in the “Makefile” File



```
#Chip type
CHIP_TYPE = N32G430C8
```

Configure the download tool path according to the installation directory

Figure 7-3 The Download Tool Path in the “Makefile” File

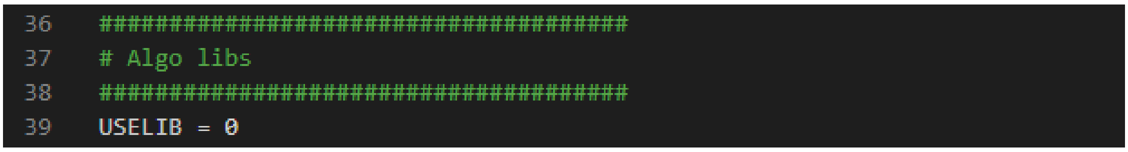


```
######################################
# download .hex/.bin by jlink
######################################
#Your JLink installation directory
PATH_WINPC = 'C:/Program Files (x86)/SEGGER/JLink_V640/'
#PATH_LINUX = /opt/SEGGER/JLink_V640b/JLinkExe
JK_DPATH = $(PATH_WINPC)
```

7.3 Using the SDK Algorithm Library

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

Figure 7-4 The Variables “USELIB” in the “Makefile” File



```
######################################
# Algo libs
######################################
USELIB = 0
```

7.4 Debug Configuration

The default “make” compilation is with “-g” debugging information. To compile a release version, use “make release =y”.

7.5 Optimization Level

The default optimization level is set to “-Os”, which takes into account both code size and execution speed.

8. Version History

Version	Date	Changes
V1.0	2022.05.16	Initial release

9. Disclaimer

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 Nations Technologies Inc. and Nations 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 NATIONS has attempted to provide accurate and reliable information, NATIONS 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 NATIONS 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.

NATIONS 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 NATIONS and hold NATIONS 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 NATIONS, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.