WT32L064/032
VSCODE Development Platform

User Guide

Rev. 0.3

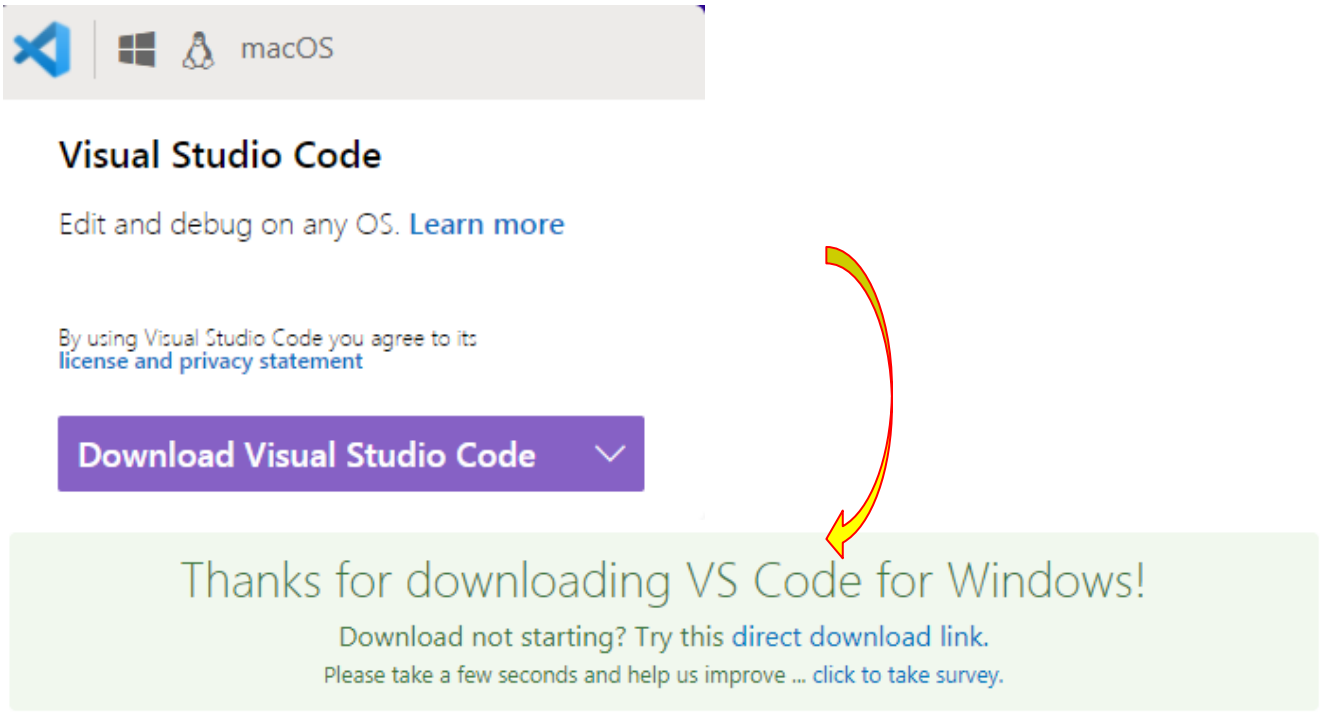
March 2021

Table of Contents

1. VSCODE Installation and Environment Setup

(Step 1) Download VSCODE

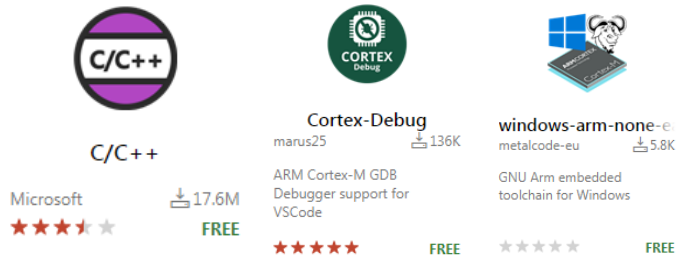
Download from https://visualstudio.microsoft.com/zh-hant/downloads/



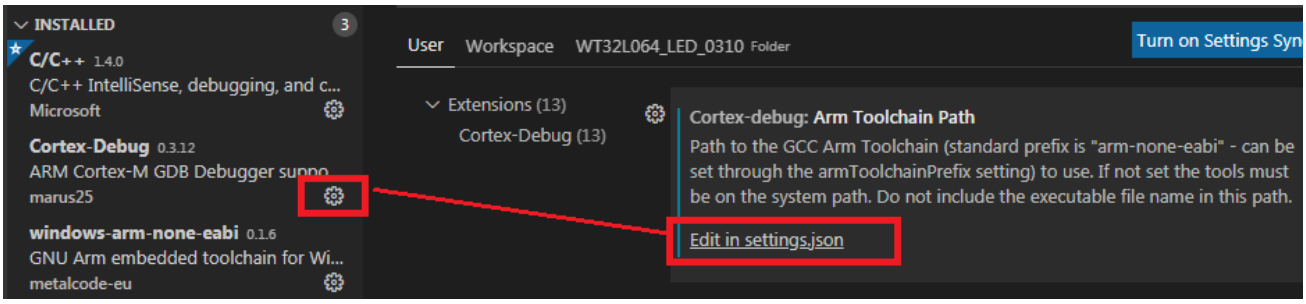
After installation, the following ICON will appear.

Install the following Extensions module from https://marketplace.visualstudio.com/vscode.

1. C/C++
2. Cortex Debug
3. Windows-arm-none-eabi



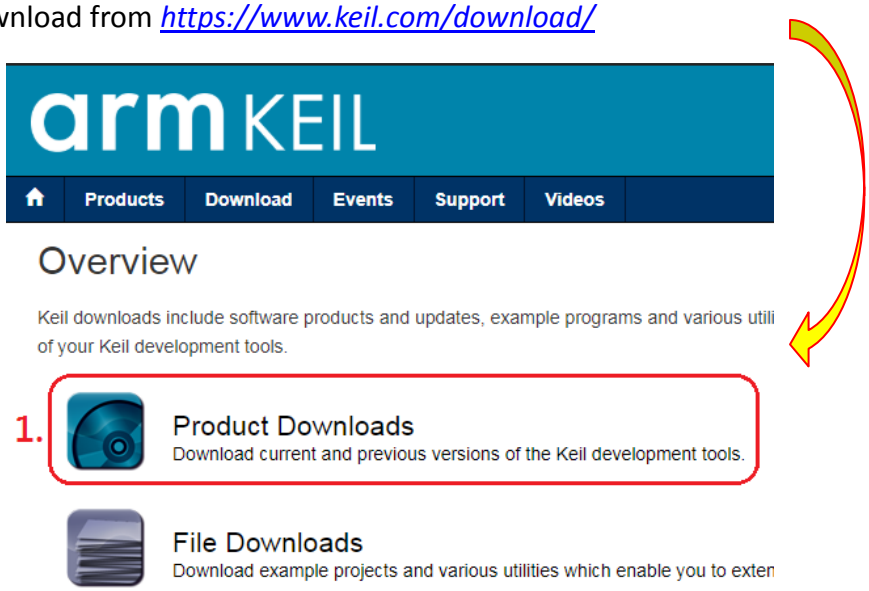
Open the lower left setting of VSCODE software, Extensions-> settings, add the path needed for cortex-debug "cortex-debug.armToolchainPath".



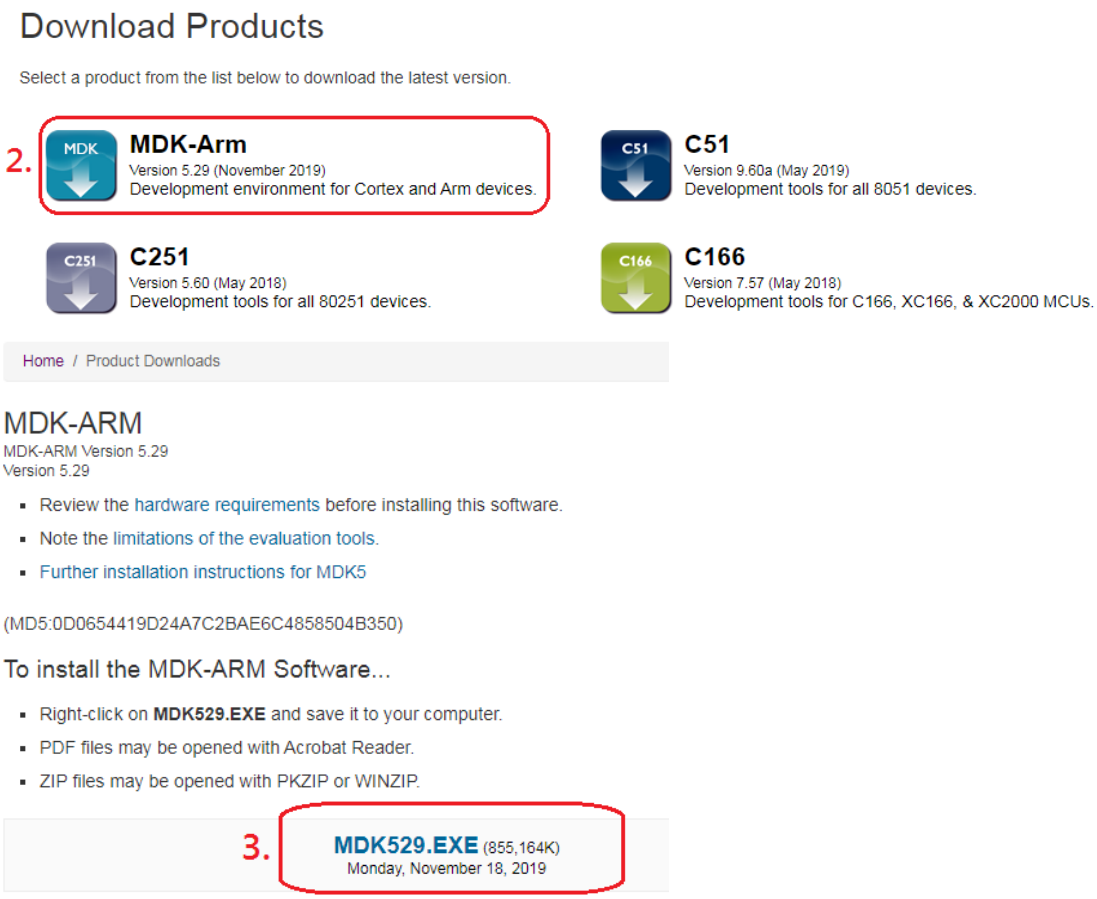

```
{
  "arm-none-eabi.bin": "${env:USERPROFILE}/.vscode/...../bin",
  "cortex-debug.armToolchainPath": "C:/ARM/gcc/bin"
}
```

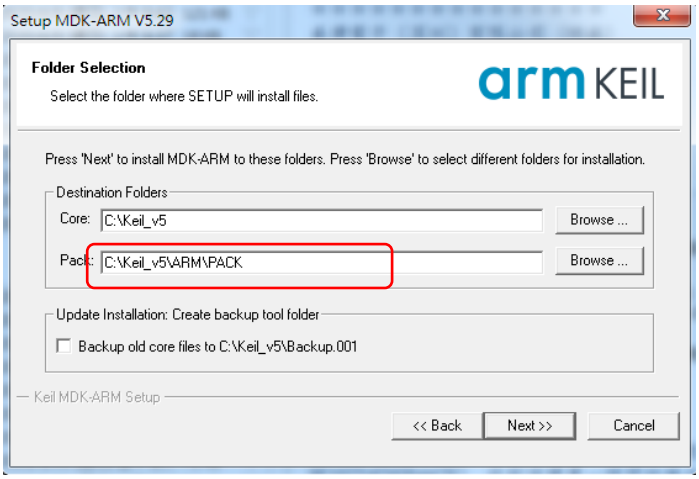
(Step 2) Download ARM-MDK

Download from https://www.keil.com/download/



https://www.keil.com/download/product/

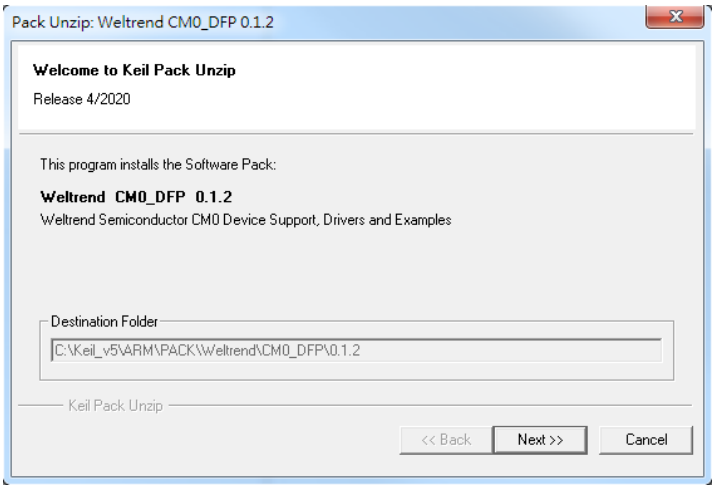


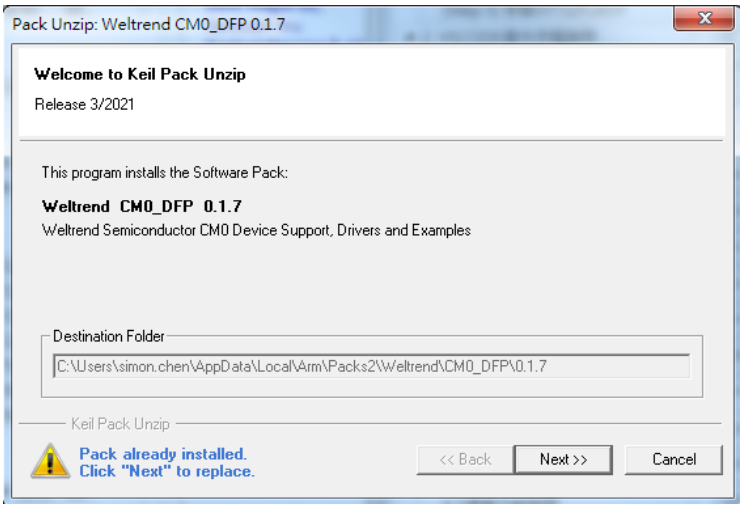


After downloading and installing the MDK, please install the Weltrend PACK file on the PC side.

Official *Weltrend.CM0_DFP.0.1.x.pack*

Official download path http://www.weltrend.com.tw/zh-tw/support/detail/2/105/105





(Step 3) Install SEGGER/J-Link

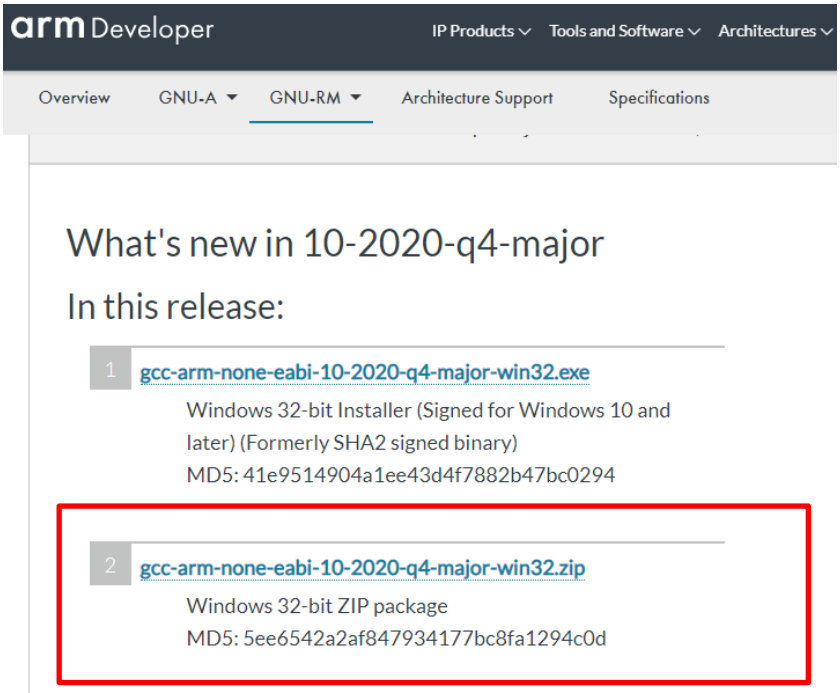
Download SEGGER and install it from the following website, select V5.12 default installation path as below:
C:/Program Files (x86) /SEGGER/JLink_V512

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

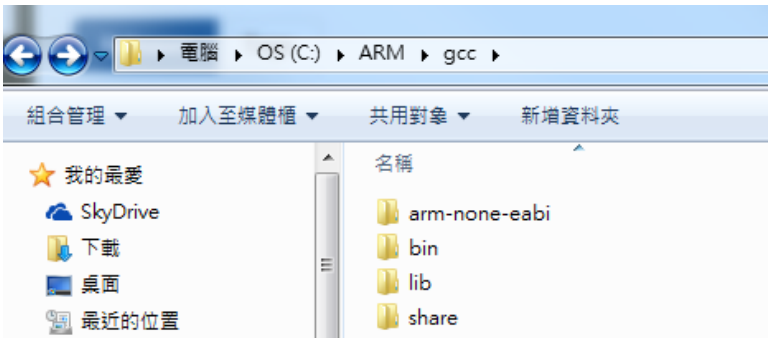
(Step 4) Install GCC compiler

Please download ZIP files from the following path and place it in the C:\ARM\gcc folder after decompression.

https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads



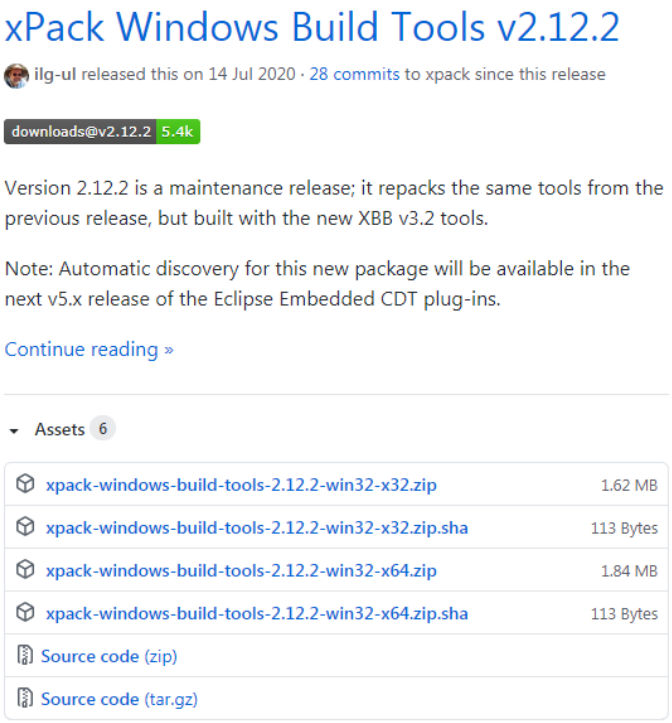
After decompression, place it in the following location.



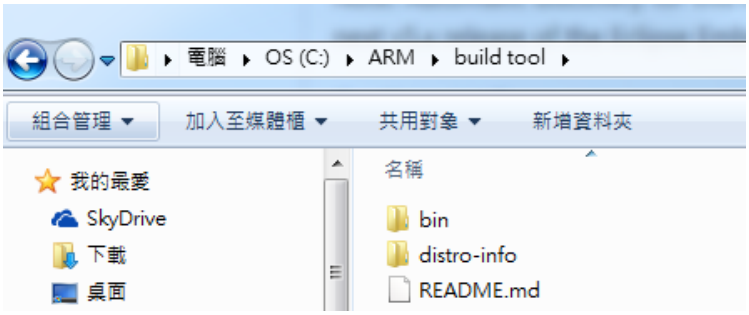
(Step 5) Install windows-build-tool

Please download ZIP files from the following path and place it in the **C:\ARM\build tool** after decompression.

https://github.com/xpack-dev-tools/windows-build-tools-xpack/releases/tag/v2.12.2/



After decompression, place it in the following location.

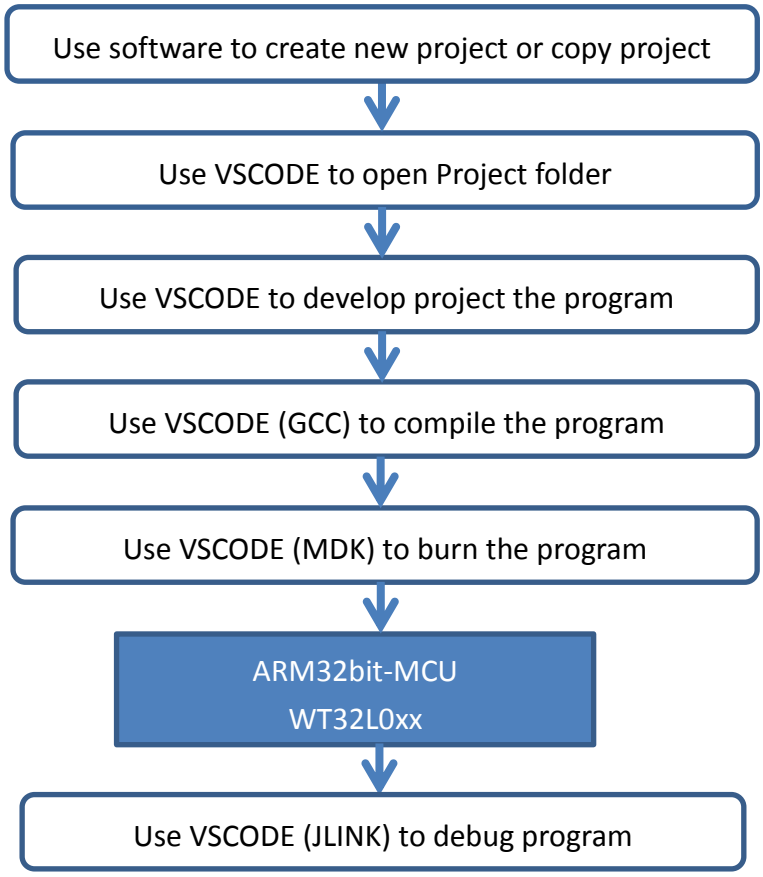


(Step 6) Install WT32FLASH

After running the project WT32FLASH_setup.exe, you can copy c:\WT32FLASH\WT32_LED to any working folder and open it with VSCODE (Open folder) and then you can modify it.

2. VSCODE Operating Procedures

To create application solutions, you can copy old projects for modification or use GUI software to generate new projects, and use GUI software to automatically generate the code for the corresponding solution. The source code can be compiled using VSCODE or MDK, any SRC C file in the project will be compiled, the compiled HEX will be programmed into the target IC, and the IC can be verified and debugged using VSCODE, the standard development flow is shown in the figure below.



2.1 Code Generator Software Installation

After installing WT32AutoGen_setup on the Microsoft OS WIN7 or above, run WT32AutoGen_ V1xx.exe. After opening the software, please load the parameter file AutoGen_xxx.cfg (placed in the same folder as the execution file).

2.2 Code Generator Software

The WT32AutoGen can generate basic example programs according to user's settings. If GUI software is not used to generate programs, users can copy the original example projects and modify the programs in the SRC folder by themselves, or add or remove C files (*.c) in the SRC folder for GCC compilation. Please refer to Application Files - Generic Code Generator for more information.

3. VSCODE Platform Environment

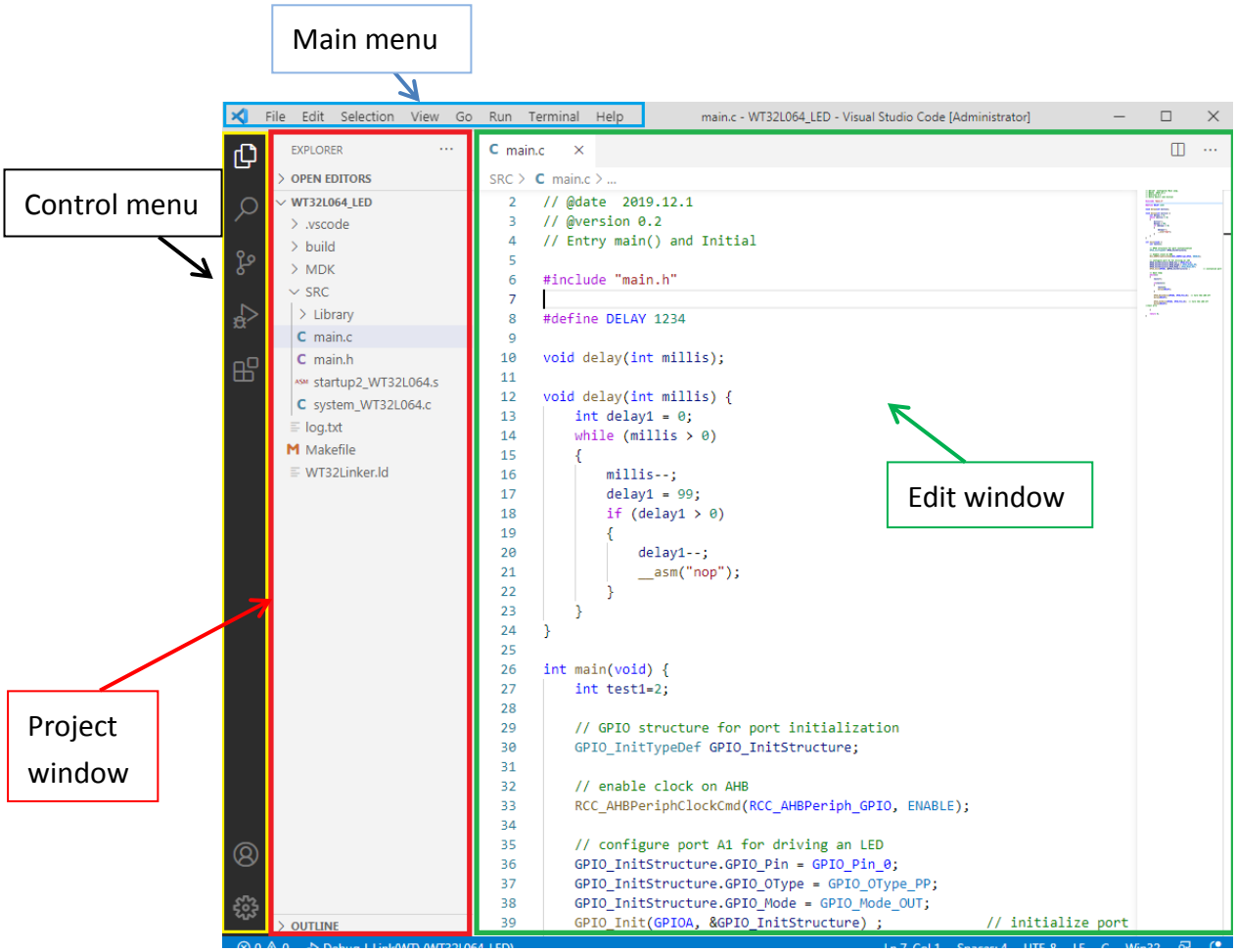
After opening the VSCODE software, there are four main areas: Main Menu, Control Menu, Project Window, and Edit Window.

Main Menu: Open File, Edit, Select, View, Skip Row, Execute, Task, and Help

Control Menu: Project browsing, Querying, Source File Control, Debugging, and Expansion

Project Window: Switch tasks and browse files corresponding to the Control menu

Editing window: Areas for text editing



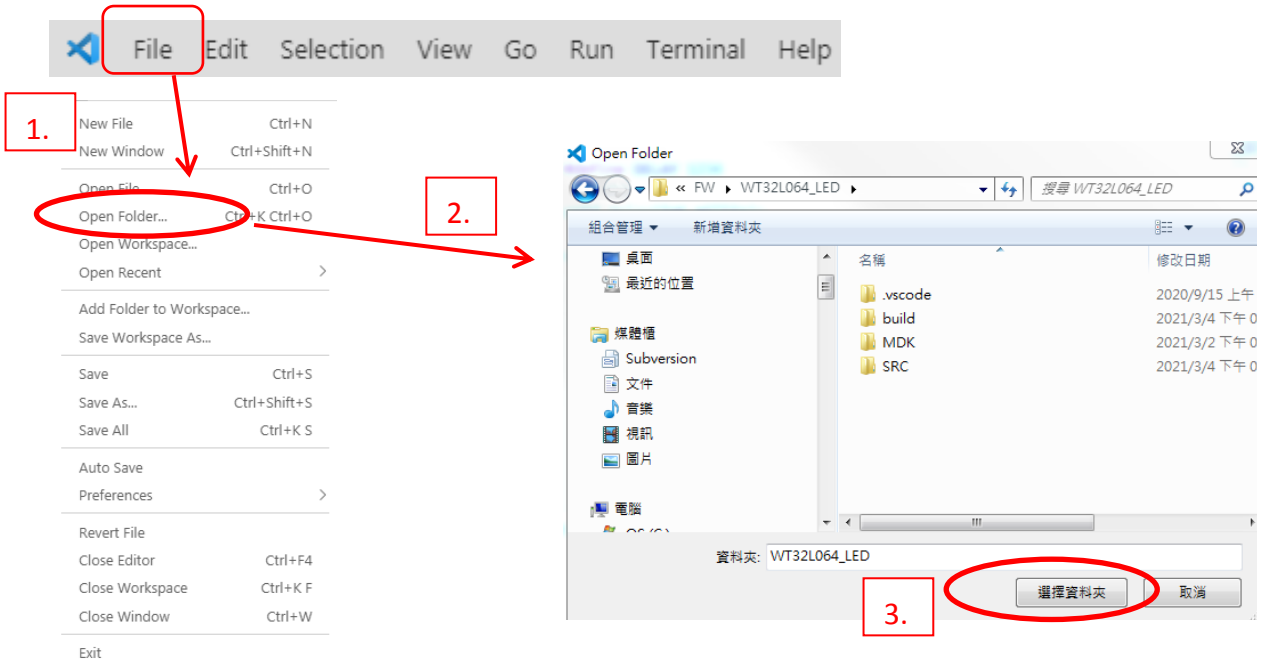
All editing is done in the Edit window, all keywords are automatically highlighted in blue, right-click the mouse button (Go To Define) to jump to the source file, and then click on the top left ICON  Project window will display a list of all files in the working folder.

3.1 Main menu

As illustrated below, the frequently used menus are File and Terminal functions. File function can open file and project folders, and Terminal function performs tasks such as compiling, programming, and erasing.

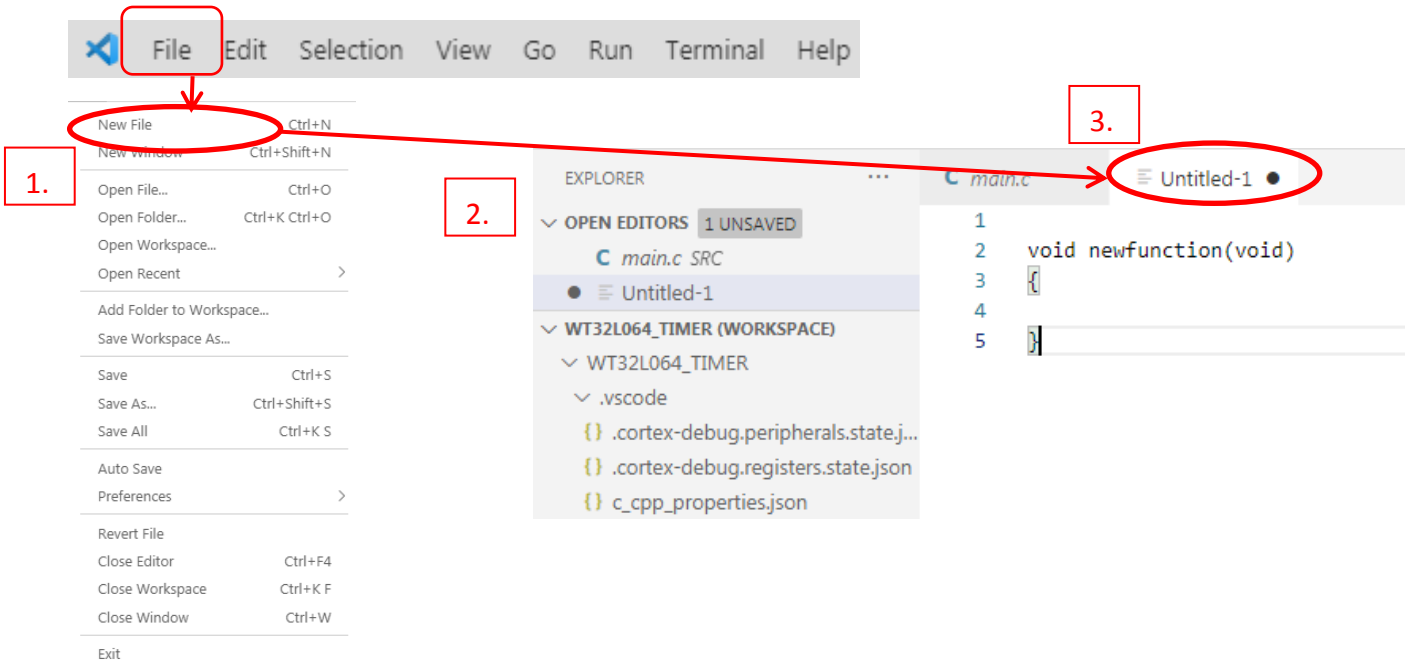
3.1.1 Examples of Opening Project Operation:

Select File -> Select Open Folder-> Select the target project folder



3.1.2 Examples of Creating New C File Operation:

Select File -> Select New File-> New file Untitled-1 appeared and it can be saved as xxx.c in the SRC folder to be compiled together.

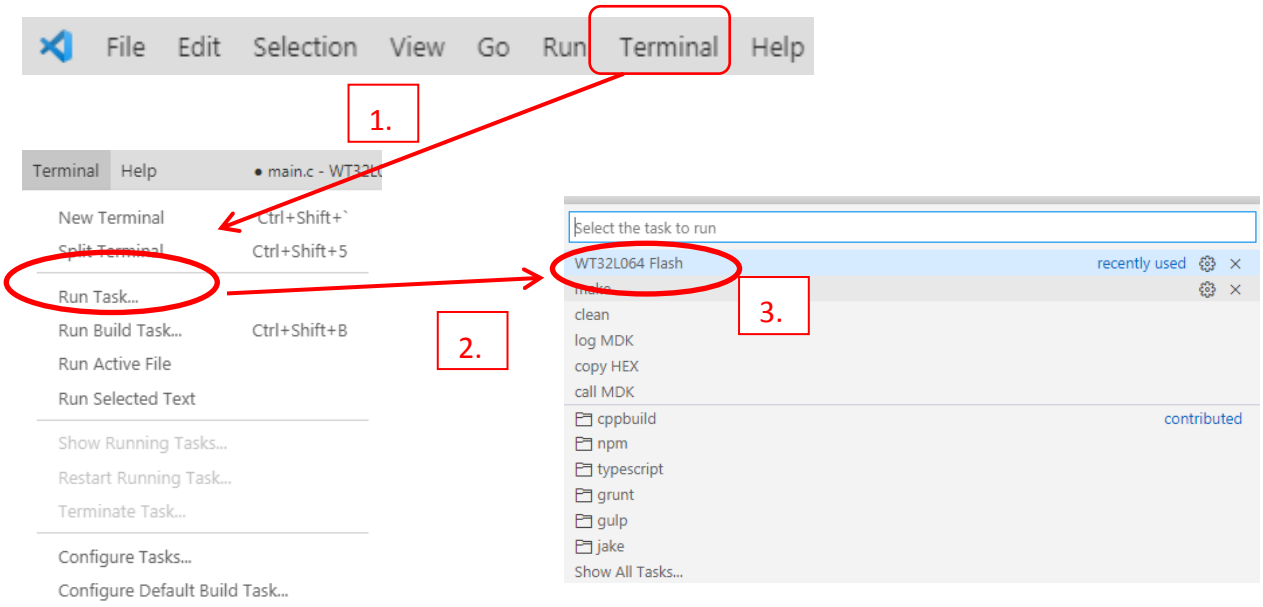


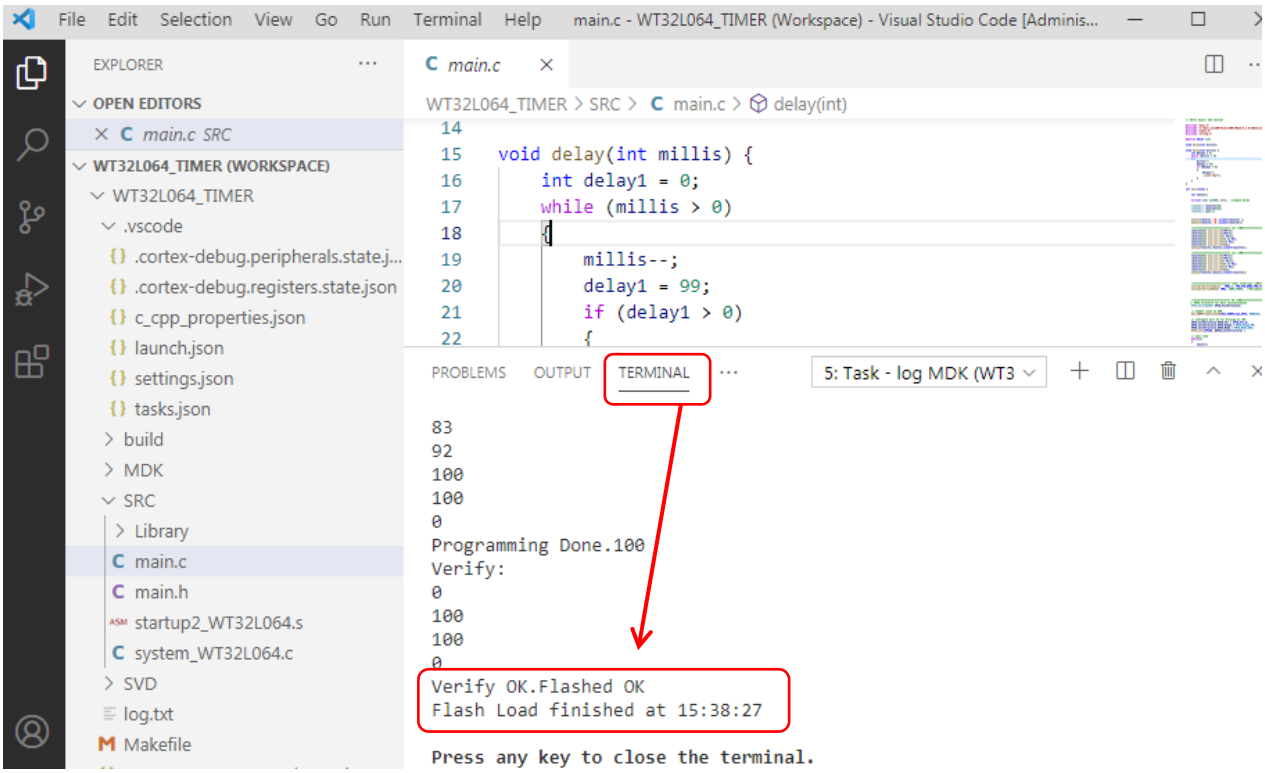
3.1.3 Examples of Programming C compiling operation:

Select Terminal-> Select Run Build Task-> The compilation results and the debugging (ELF) and programming (HEX) files are displayed, as shown below.

3.1.4 Examples of Opening Terminal Operation:

Select Terminal-> Select Run Task-> WT32L064 Flash programming (HEX) file. After successful programming, the result will be displayed in the Terminal Window as below figure.



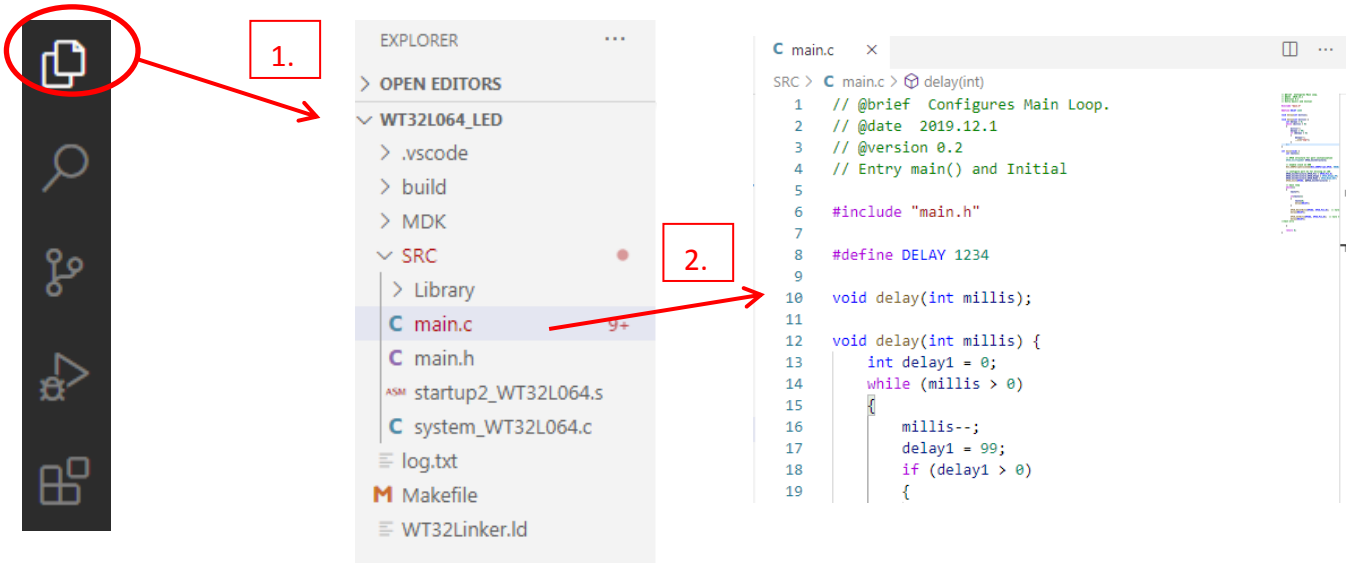


3.2 Control menu

The Control menu is usually on the left side of the software screen, and the common functions are browsing folders and debugging. Extension functions are only available after the first installation, and the following sections provide examples and explanations.

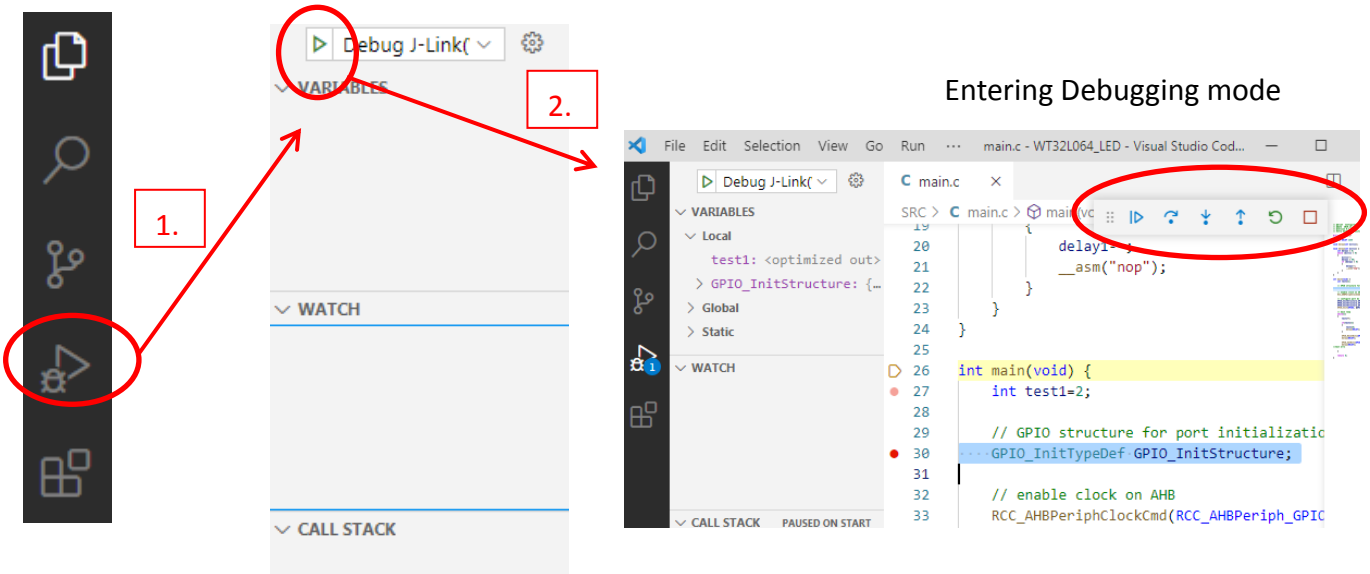
3.2.1 Examples of Opening Project Documents list:

Select the file icon on the left side -> a list of current file contents will be displayed.



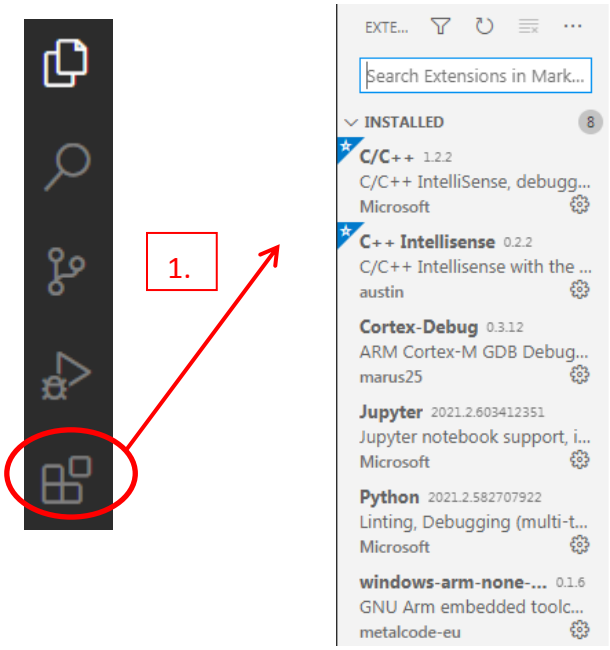
3.2.2 Examples of Opening Debugging Operation:

Select the Debugging icon on the left side-> a list of current file contents will be displayed -> Select the small green triangle icon above to start debugging.



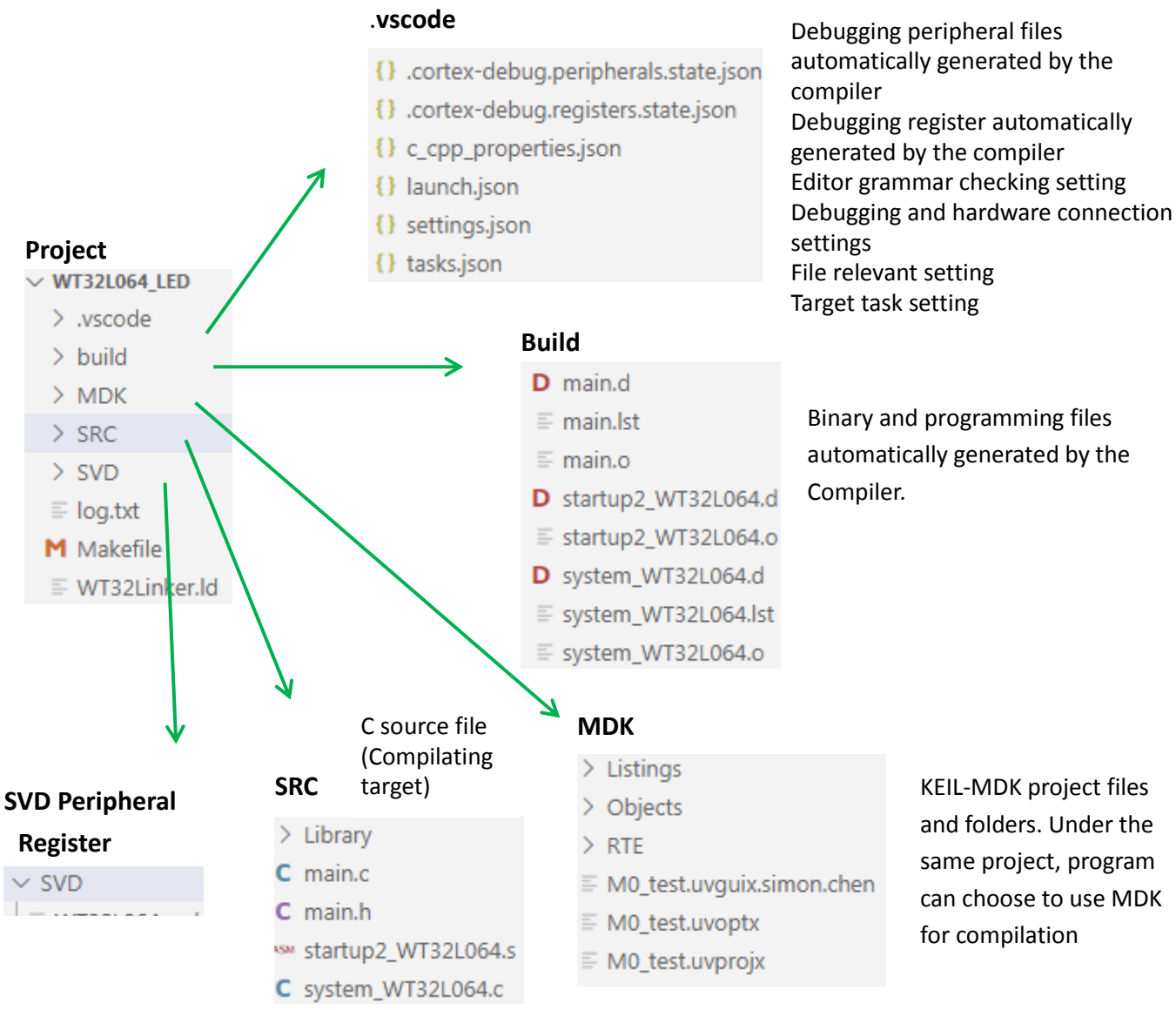
3.2.3 Examples of Opening Extension Components:

Select the block icon on the left side -> it will show the currently installed extensions, which was only used at the beginning of the installation. The four main necessary components are C/C++, C++ Intellisense, Cortex-Debug, and windows-arm-none-eabi.



3.3 Contents of Project Folder

Open the project folder to browse the contents, which are mainly divided into four folders, namely .vscode, build, MDK, SRC, and the outermost three files are log.txt, Makefile, and WT32Link.ld. The functions and icons are illustrated as below.



Makefile: Settings for the GCC compiler
WT32Linker.ld: Link and programmed files for GCC generated OBJ
startup2_wt32l064.s: boot initialization file for GCC
startup_wt32l064.s: boot initialization file for MDK

4. VSCODE Program development & Debugging

If there are writing errors in the development process, the compilation will fail and the ELF and HEX files will not be generated, and error detection will not be available. There are two main types of errors: grammatical errors and logical errors. The hardware uses J-Link to transmit SWD signals and execute debug commands.

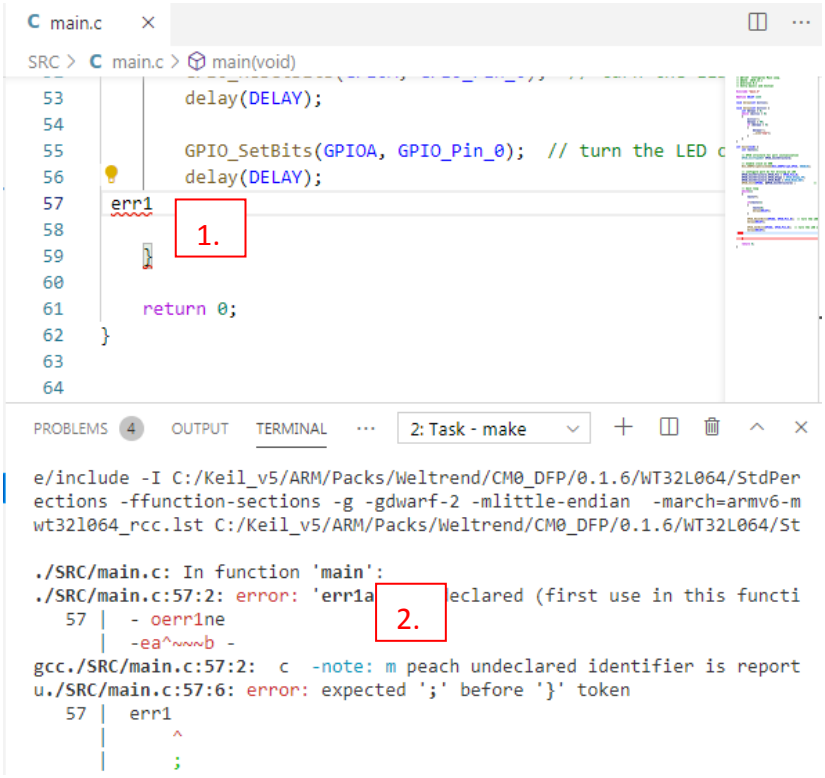
4.1 Compiler Function

After the program is finished, you can select Terminal->Run Build Task in the main menu above, or press the quick key CTRL+SHIFT+B, the successful compilation will appear in the terminal window, and will indicate the HEX file and its path, the default path is the project name \build, where the ELF file is used for debugging DEBUG.

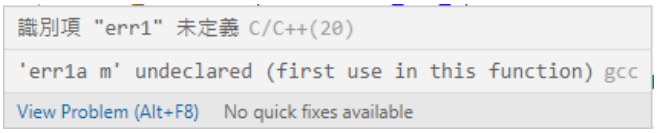
```
thumb   -specs=nano.specs -TWT32Linker.ld  -lc -lm -lnosys  -Wl,-Map=build/WT32L064_Sample.map,
cref -Wl,--gc-sections -o build/WT32L064_Sample.elf
arm-none-eabi-size build/WT32L064_Sample.elf
   text    data     bss     dec     hex filename
   1888       8    1316    3212     c8c build/WT32L064_Sample.elf
arm-none-eabi-objcopy -O ihex build/WT32L064_Sample.elf build/WT32L064_Sample.hex
arm-none-eabi-objcopy -O binary -S build/WT32L064_Sample.elf build/WT32L064_Sample.bin
```

4.2 Grammar Error Exclusion

When a general writing error occurs, a red wave will appear immediately as shown in label 1 below, and when compiling, red characters will appear on the output screen as shown in label 2 below.



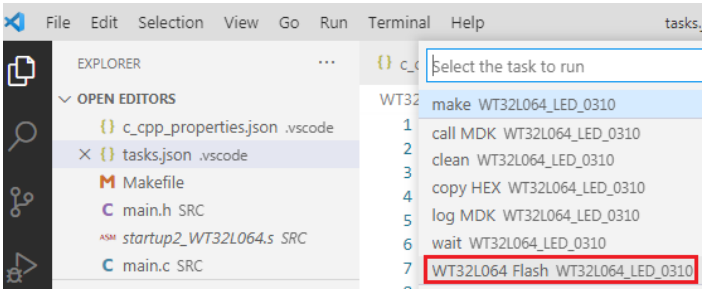
For the error message labeled 1, move the cursor over the wave and the instructions will appear immediately.



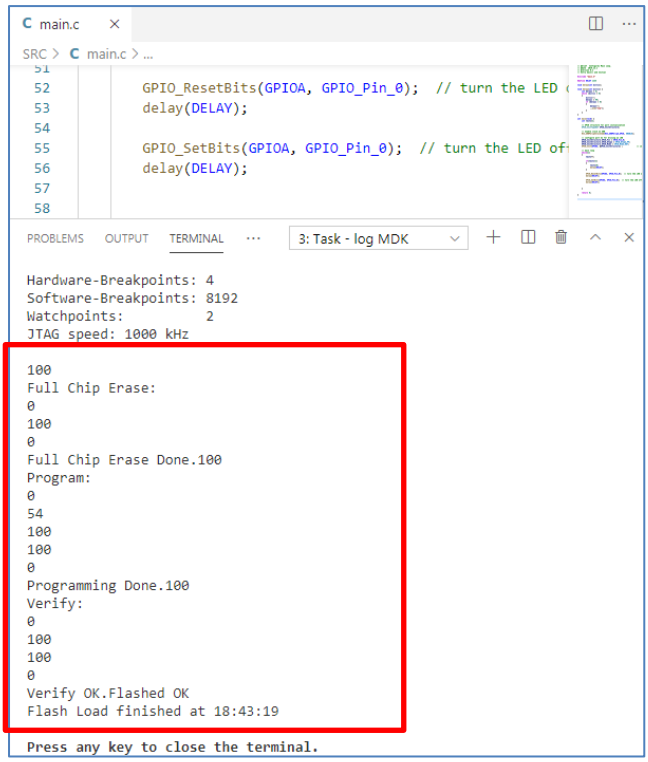
For the error message labelled 2, the key file name and the number of rows and columns will appear. Press CTRL+left mouse button, the editor will help you jump to the error point and you can proceed to edit the error.

4.3 Programming functions

After the compilation is completed and the HEX file is generated, please refer to 3.1.2 to operate the program, click on the WT32L064 FLASH xxx (your project) menu shown below, and then a successful programming message will appear in the Terminal window as shown below.



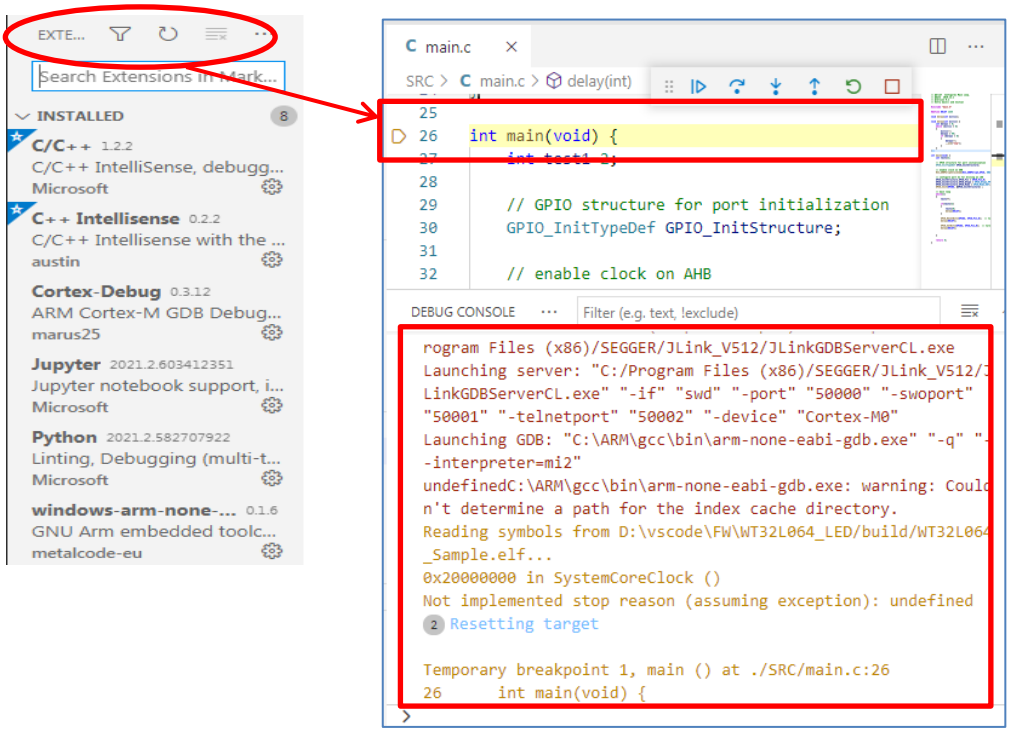
The message of successful programming is as follows.



4.4 Logic debugging function

General logic errors are debugged by using ICE simulation, and HEX files will be generated after the compilation is completed. Please refer to Section 3.2.2 for more information. Click on the green triangle below Debug-Jiink(WT), the editing window will appear with a red arrow to the right and stop at the first line of the main.c program, the terminal below will appear calling arm-none-eabi- The word gdb will appear at the bottom of the terminal and it will indicate that it is currently stopped at int main(void) as shown below.

The message of successful programming is as follows.



You can double-click the red dot breakpoint in the column number field, as shown in the left red dot position, or you can click the floating function task  to execute or stop the function.

```
36      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
37      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
38      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
39      GPIO_Init(GPIOA, &GPIO_InitStructure) ;
40
41      // main loop
42      while(1)
43      {
44          test1++;
```

5. Appendix

The following are additional descriptions of the main platform settings and compiler configuration files used in the VSCODE platform

5.1 tasks.json Function

The tasks available under the Main menu Terminal->Run Task, are in the order of make compiler, clean old files, and WT32L064 Flash programming tool. If no special requirements are needed, please use the current default tasks, shown as below.

```
"version": "2.0.0",
"options": {
    "env": {
        "path": "c:/ARM/gcc/bin/;c:/ARM/build tool/bin/;c:/ARM/OCD/bin/;"
    },
    "shell": {
        "executable": "${env:windir}/System32/WindowsPowerShell/v1.0/powershell.exe",
        "args": [ "-NoProfile", "-ExecutionPolicy", "Bypass", "-Command" ]
    }
},
"tasks": [
    {
        "label": "make",
        "type": "shell",
        "command": "make -j",    //-j: cpu core unlimited    -?: explain
        "group": {
            "kind": "build",
            "isDefault": true
        },
        "presentation": {
            "focus": true
        },
        "problemMatcher": [
            "$gcc"
        ]
    },
    {
        "label": "clean",
        "type": "shell",
        "command": "make clean",
        "group": "build",
        "presentation": {
            "focus": true
        },
        "problemMatcher": [
            "$gcc"
        ]
    },
```

5.2 launch.json Function

The link file used for debugging can be set to call a specific path of the driver to connect to VSCODE for synchronous debugging, and only need to modify if there are changes to the bridge, part of which is shown below.

```
"version": "0.2.0",
"configurations": [
    {
        "name": "Debug J-Link(WT)",
        "type": "cortex-debug",
        "request": "launch",
        "cwd": "${workspaceRoot}",
        "executable": "${workspaceRoot}/build/WT32L064_Sample.elf",
        //"serverpath": "C:/Program Files (x86)/SEGGER/JLink/JLinkGDBServerCL.exe",
        "serverpath": "C:/Program Files (x86)/SEGGER/JLink_V512/JLinkGDBServerCL.exe",
        "servertype": "jlink",
        "device":  "Cortex-M0",
        "interface": "swd",

        "serialNumber": "", //If you have more than one J-Link probe, add the serial n
       "runToMain": true,
    }
]
```

5.3 Makefile Function

The configuration file for GCC compilation contains the INCLUDE path and all the target source files for compilation. You can observe whether the current CMSIS is the latest version in the Makefile file as shown in red. The data in the SRC folder is automatically compiled.

```
#######################################
# target
#######################################
#TARGET = WT32L064_Sample
TARGET := $(notdir $(CURDIR))

#######################################
# path to Cortex-M0 standard peripheral library
#######################################
CMSIS_LIBS ?=       C:/Keil_v5/ARM/Packs/Weltrend/CM0_DFP/0.1.6/WT32L064
CMSIS_CORE ?=       C:/Keil_v5/ARM/Packs/ARM/CMSIS/5.6.0/CMSIS
```

5.4 Linker Function

Please refer to GCC and CMSIS usage for Link settings. It mainly sets the program start address and the length of RAM and ROM, specifies the program entry function and arranges the interrupt and data area, and replaces or modifies the target IC only if there is a change.

```
ENTRY(Reset_Handler)
MEMORY
{
  FLASH (rx) : ORIGIN = 0x10000000, LENGTH = 0x10000   /*  64k */
  RAM (rwx)  : ORIGIN = 0x20000000, LENGTH = 0x02000   /*  8k */
}
_ram_stack = 0x20002000;    /* end of RAM, simon.c */
_Min_Heap_Size = 0x100;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */
SECTIONS
{
    .isr_vector :
    {
        . = ALIGN(4);
        KEEP(*(.isr_vector)) /* Startup code */
        . = ALIGN(4);
    } >FLASH
```

6. Revision History:

Version	Contents of Change	Date
V0.1	Initial issue	March 02, 2021
V0.2	Add gcc/build tools installation descriptions	March 18, 2021
V0.3	Add Extension & GCC default path	March 31, 2021