WT32L064/032 Application Note

VSCODE 發展平台

(中文版)

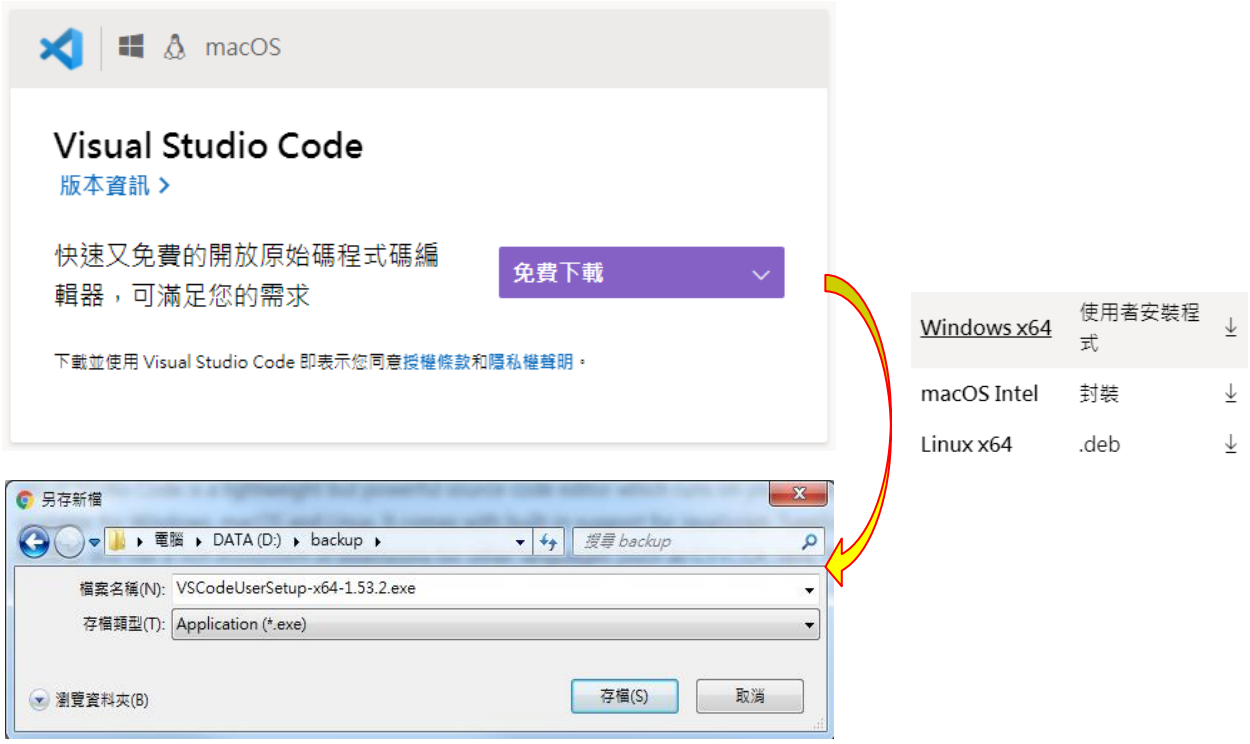
Rev. 0.3
March 2021

目 錄

1. VSCODE 安裝與環境設定

(Step 1) 下載 VSCODE

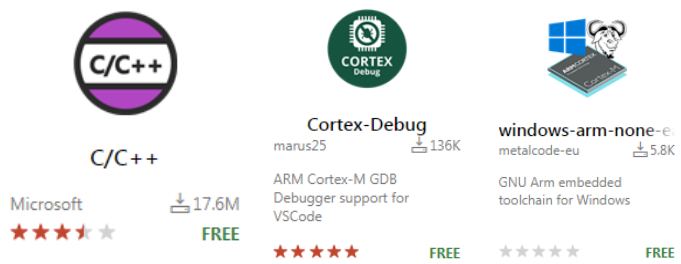
下載網址如右 *https://visualstudio.microsoft.com/zh-hant/downloads/*



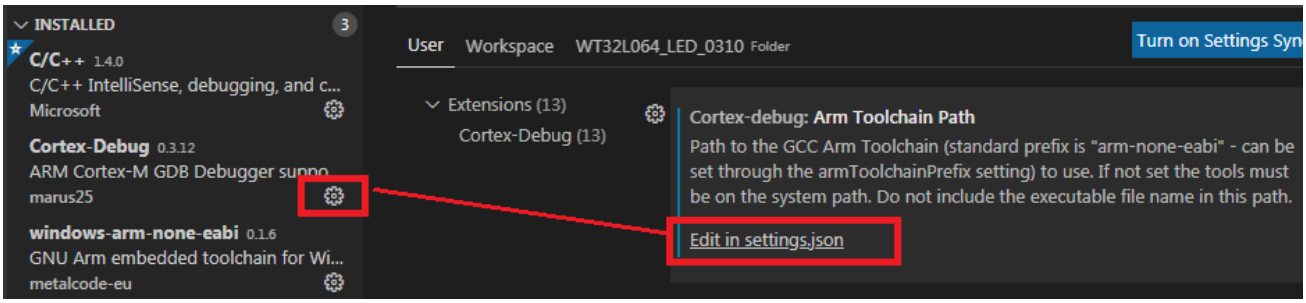
安裝完後出現下列 ICON

於網址 *https://marketplace.visualstudio.com/vscode* 安裝下列 Extensions 模組。

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



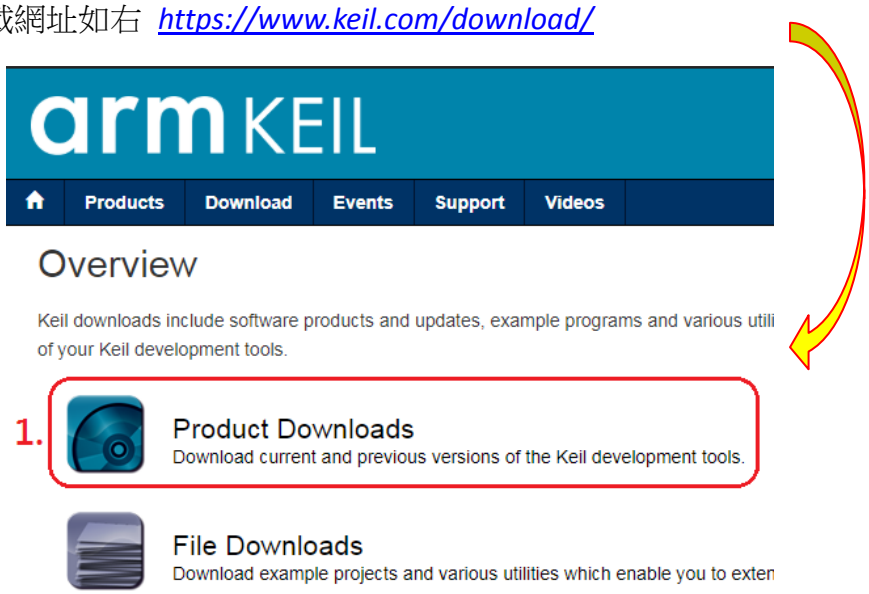
開啟 VSCODE 軟體左下設定 Extensions-> Settings，新增 Cortex-Debug 需要的路徑 "Cortex-debug.armToolchainPath"



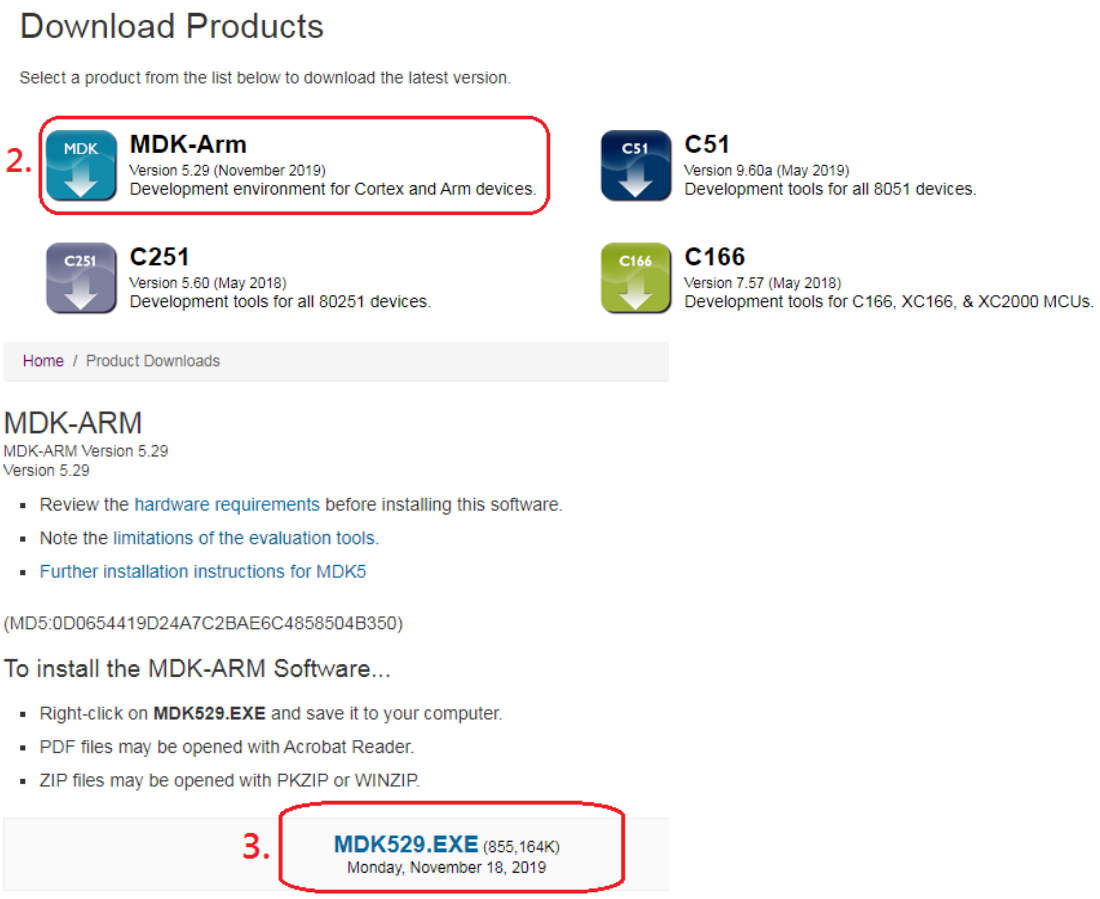

```
{
  "arm-none-eabi.bin": "${env:USERPROFILE}/.vscode/...../bin",
  "cortex-debug.armToolchainPath": "C:/ARM/gcc/bin"
}
```

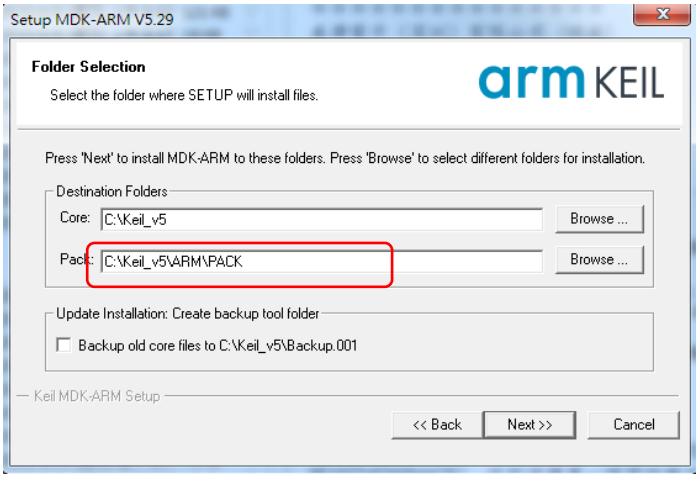
(Step 2) 下載 ARM-MDK

下載網址如右 *https://www.keil.com/download/*

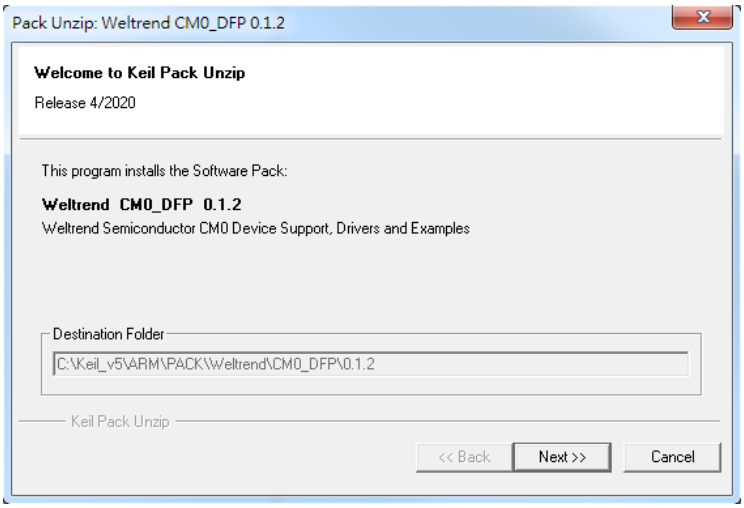


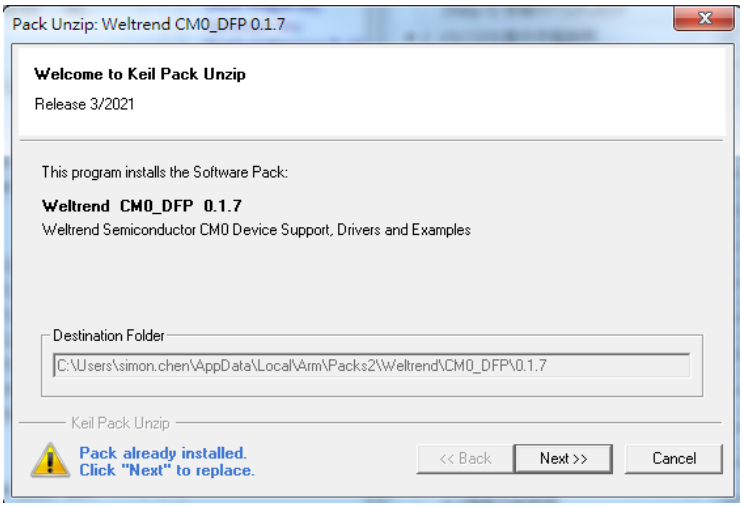
https://www.keil.com/download/product/





下載並安裝 MDK 後，請於 PC 端再安裝偉詮 PACK 檔案 *Weltrend.CM0_DFP.0.1.x.pack*
官網下載路徑 *http://www.weltrend.com.tw/zh-tw/support/detail/2/105/105*





(Step 3) 安裝 SEGGER/J-Link

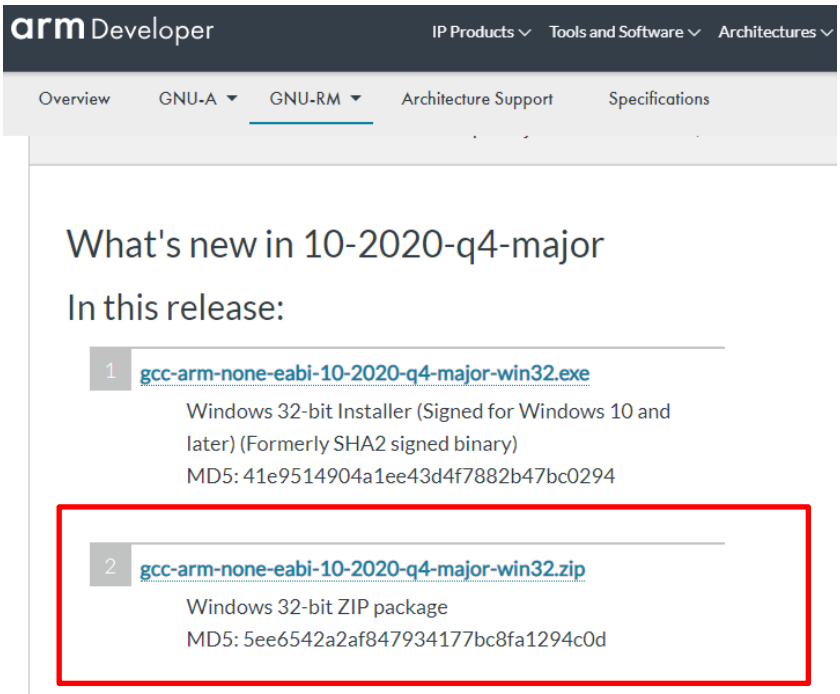
下載 SEGGER 安裝如下列網址，選用 V5.12 版本預設安裝路徑 C:/Program Files (x86) /SEGGER/JLink_V512

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

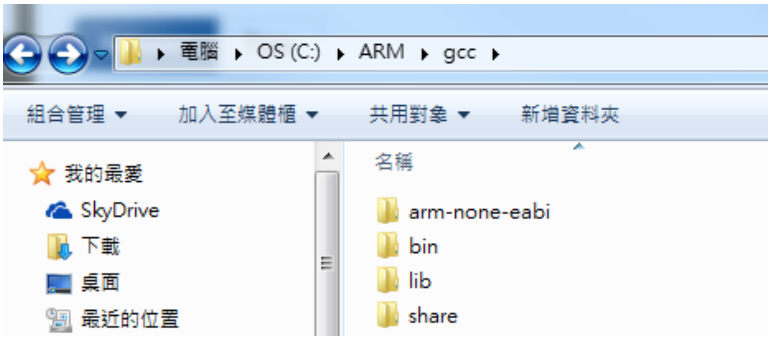
(Step 4) 安裝 GCC compiler

下載網址如下，請選擇下載 ZIP 檔，解壓縮後放置 **C:\ARM\gcc** 資料夾內

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



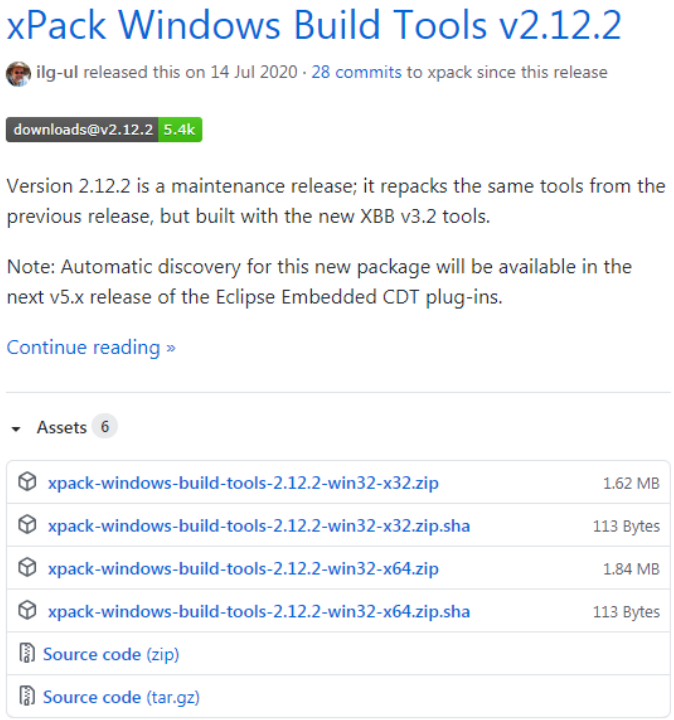
解壓縮後放置的位置如下:



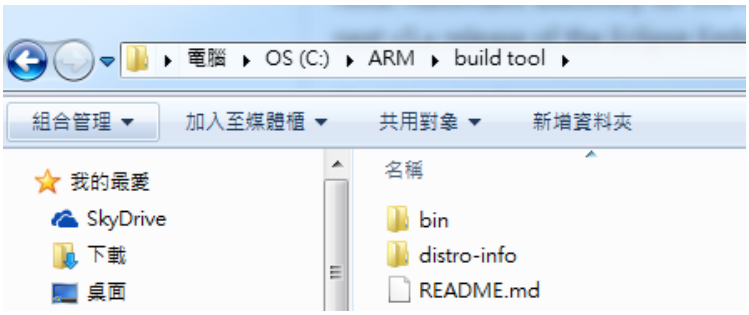
(Step 5) 安裝 windows-build-tool

下載網址如下，請選擇下載 ZIP 檔，解壓縮後放置 **C:\ARM\build tool** 資料夾內

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



解壓縮後放置的位置如下:

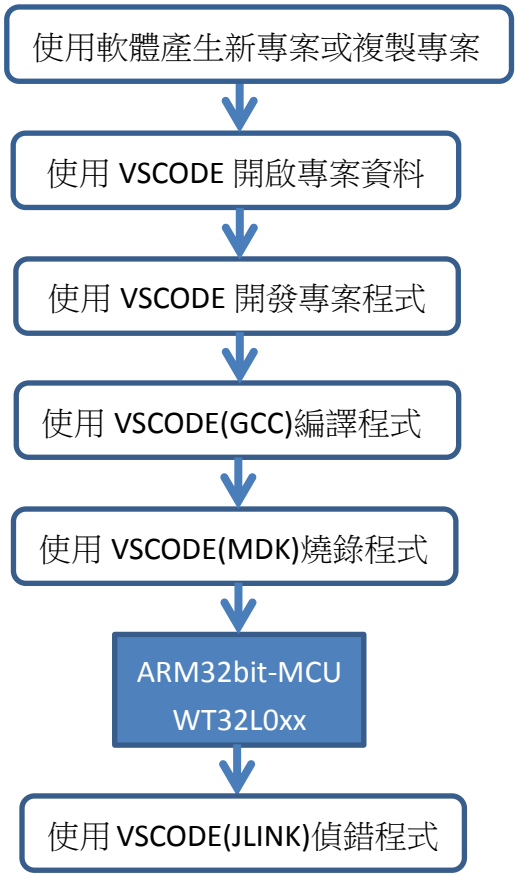


(Step 6) 安裝 WT32FLASH

執行安裝專案 WT32FLASH_setup.exe 後可將 c:\WT32FLASH\WT32_LED 複製到任意的工作資料夾並使用 VSCODE 開啟(Open folder)後可進行修改。

2. VSCODE 操作流程說明

建立應用方案可複製舊專案進行修改或使用 GUI 軟體產生新專案，使用 GUI 軟體自動產出對應方案的程式碼，範例程使可參考前章節 WT32FLASH 的路徑 C:\WT32FLASH\ WT32L064_LED 專案，上述兩個方式都支持 VSCODE 與 MDK 兩種類型專案，範例原始碼並可使用 VSCODE 或 MDK 進行編譯，任何在專案內 SRC 的 C 檔案都會進行編譯，將編譯後的 HEX 燒入至目標 IC，並可對該 IC 使用 VSCODE 進行驗證與偵錯工作，標準開發流程如下圖所示。



2.1 代碼產生器軟體安裝

於微軟作業系統 WIN7 以上的電腦端上安裝 WT32AutoGen_setup 後執行 WT32AutoGen_ V1xx.exe 即可，開啟軟體後可先載入參數檔 AutoGen_xxx.cfg (與執行檔放置同個資料夾內)。

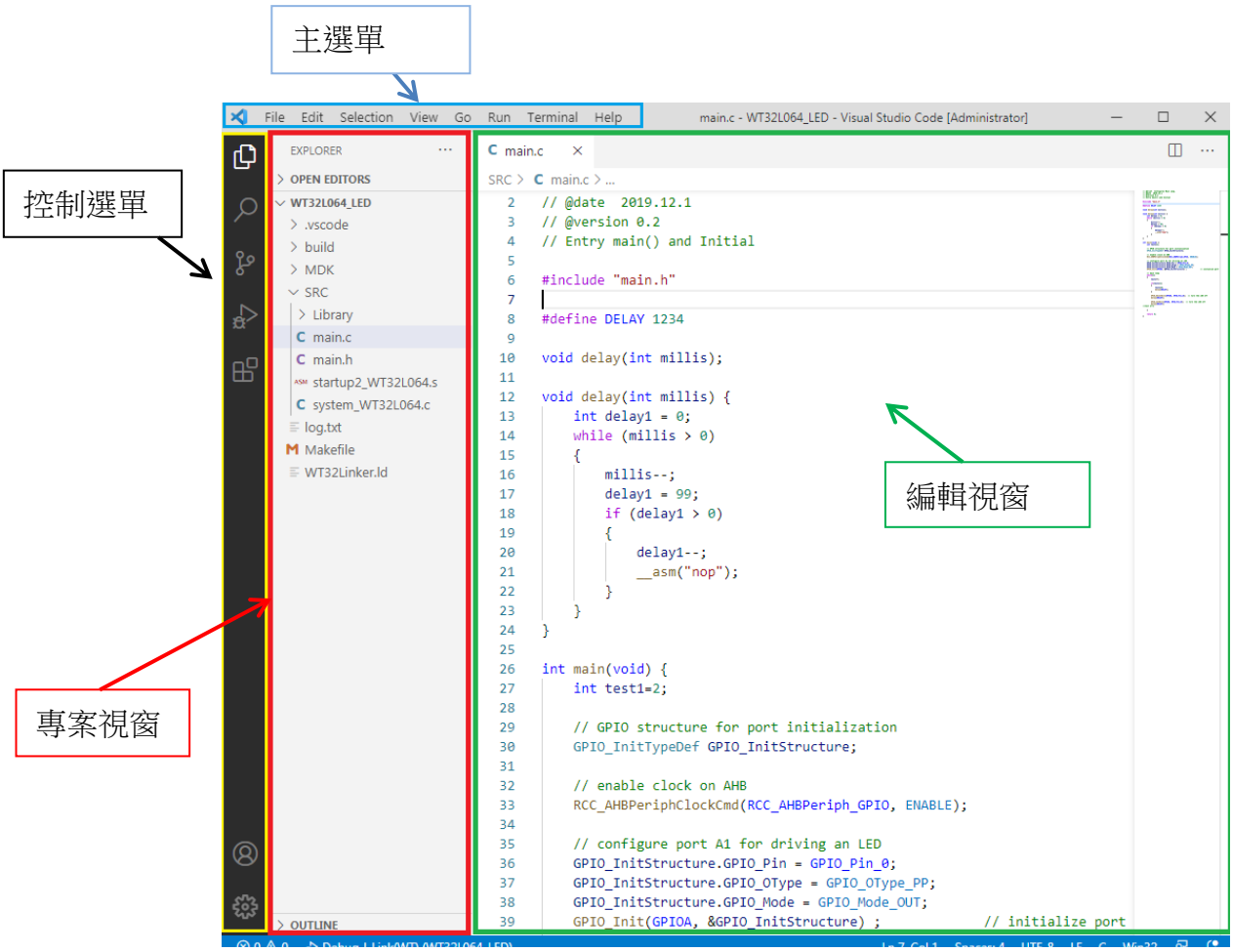
2.2 代碼產生器軟體說明

WT32AutoGen 可依照使用者設定產出基礎範例程序，若不使用 GUI 軟體產出程式，使用者可複製原本的範例專案並自行修改 SRC 資料夾內的程式，或任意增減 SRC 資料夾內的 C 檔案(*.c)後可進行 GCC 編譯，詳細請參閱應用文件-泛用型代碼產生器說明。

3. VSCODE 平台環境說明

開啟 VSCODE 軟體後，有四個主要區域分別為主選單、控制選單、專案視窗、編輯視窗，個別功能概述如下，常用功能於下個章節有細部說明。

主選單: 開檔、編輯、選擇、檢視、跳行、執行、任務、協助
控制選單: 專案瀏覽、查詢、源檔控制、偵錯、擴充
專案視窗: 對應控制選單進行切換任務，與瀏覽檔案
編輯視窗: 進行文字編輯的區域



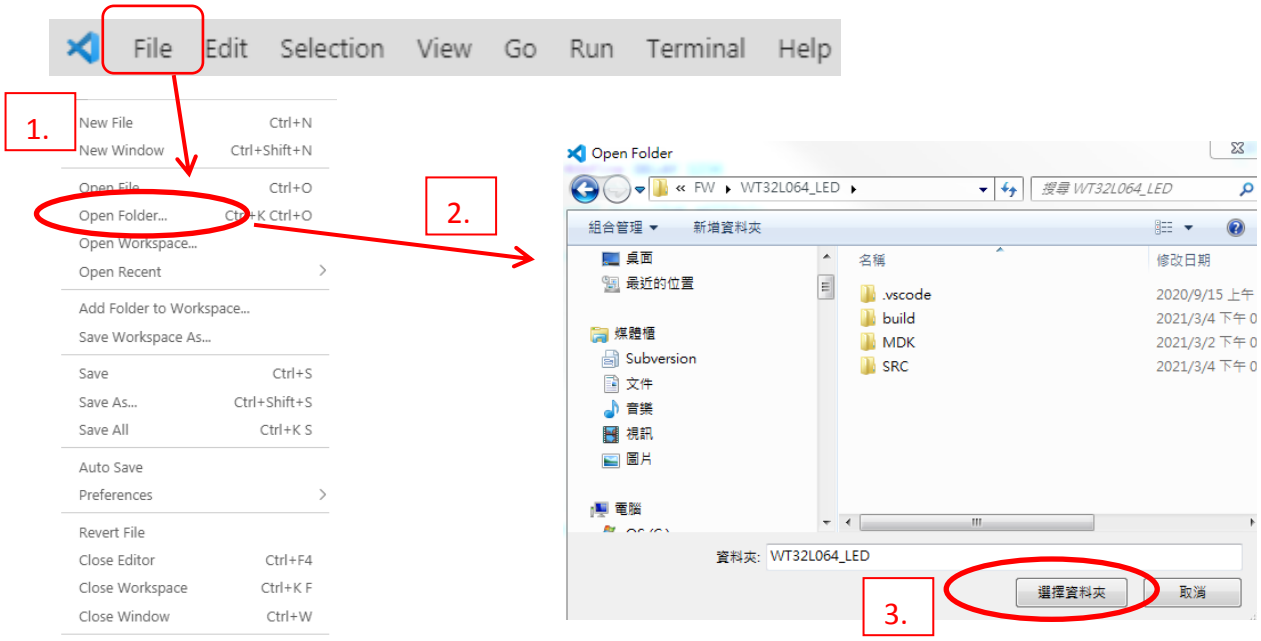
所有的編輯工作都在編輯視窗內完成，所有關鍵字會自動反藍凸顯，可使用滑鼠右鍵(Go To Define)跳躍至來源文件，點選左上 ICON  專案視窗會顯示所有工作資料夾內的檔案清單。

3.1 主選單

如下圖所示其常用選單功能為 File 與 Terminal，File 功能可開啟檔案、專案資料夾，Terminal 可執行編譯、燒錄、清除等任務，下列章節進行範例與說明。

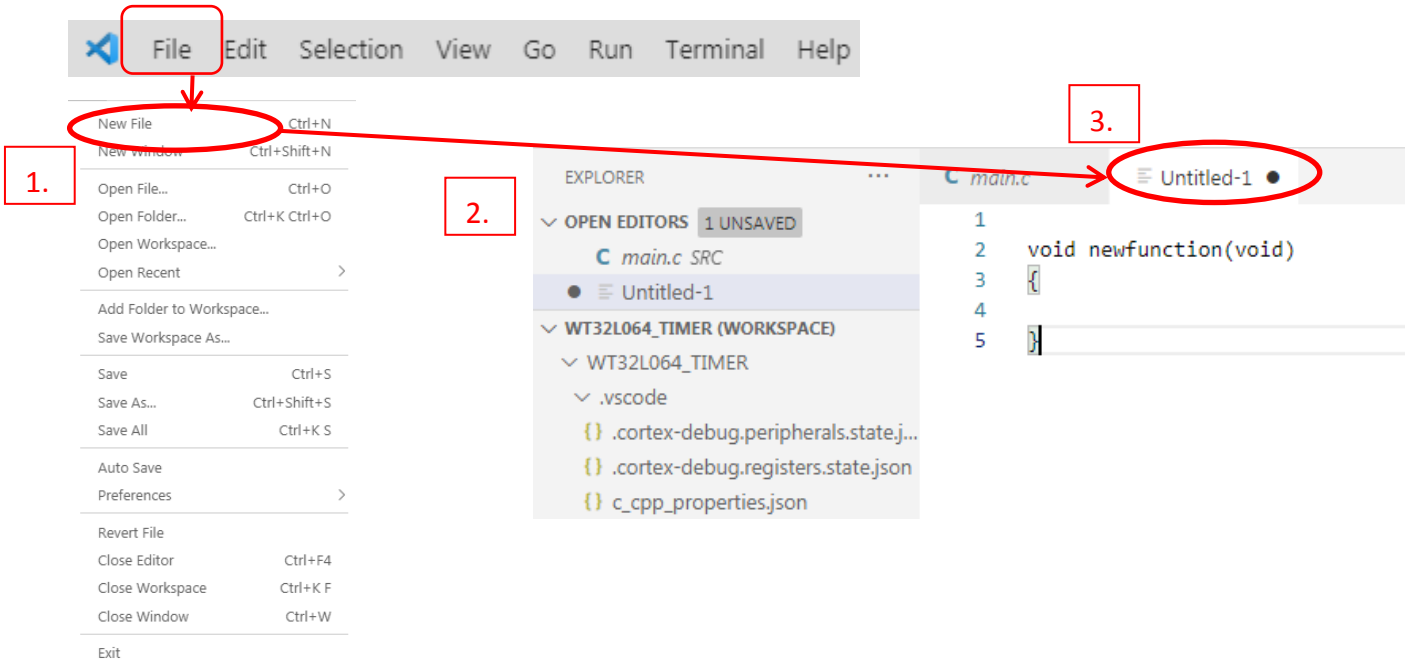
3.1.1 開啟專案操作範例:

選擇 File -> 選擇 Open Folder-> 選擇目標專案的資料夾。



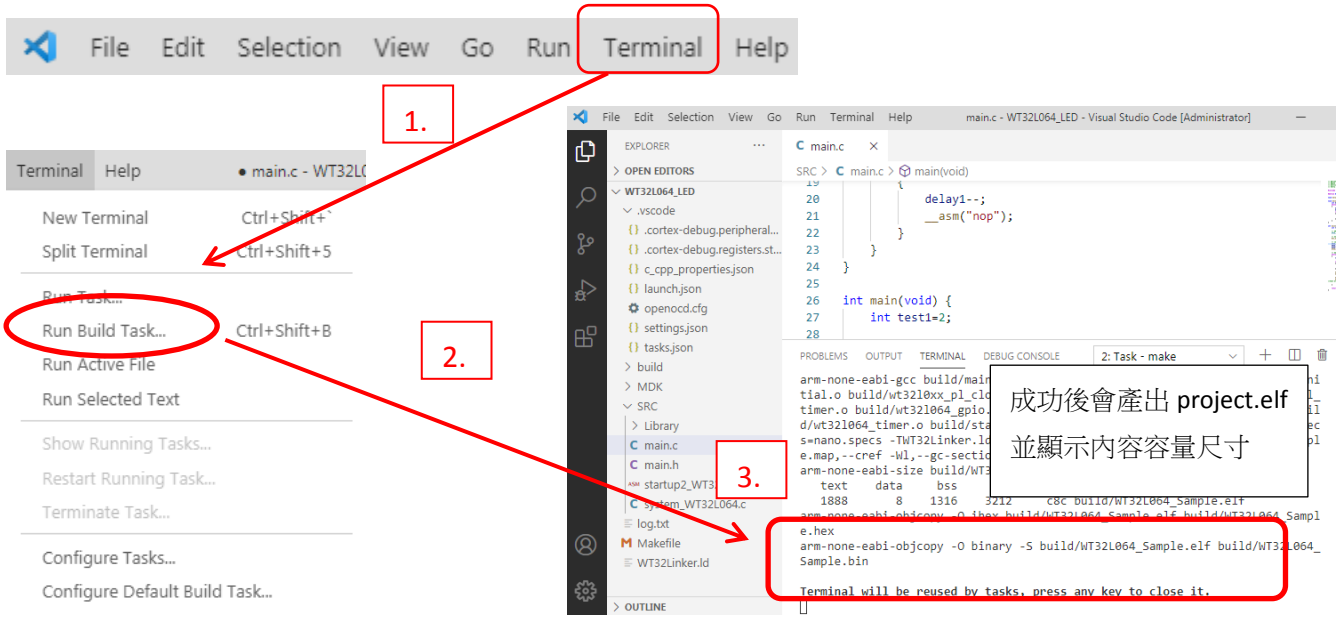
3.1.2 建立新 C 檔案操作範例:

選擇 File -> 選擇 New File-> 出現新檔 Untitled-1，可另存成 xxx.c 於 SRC 資料夾內方可一同編譯。



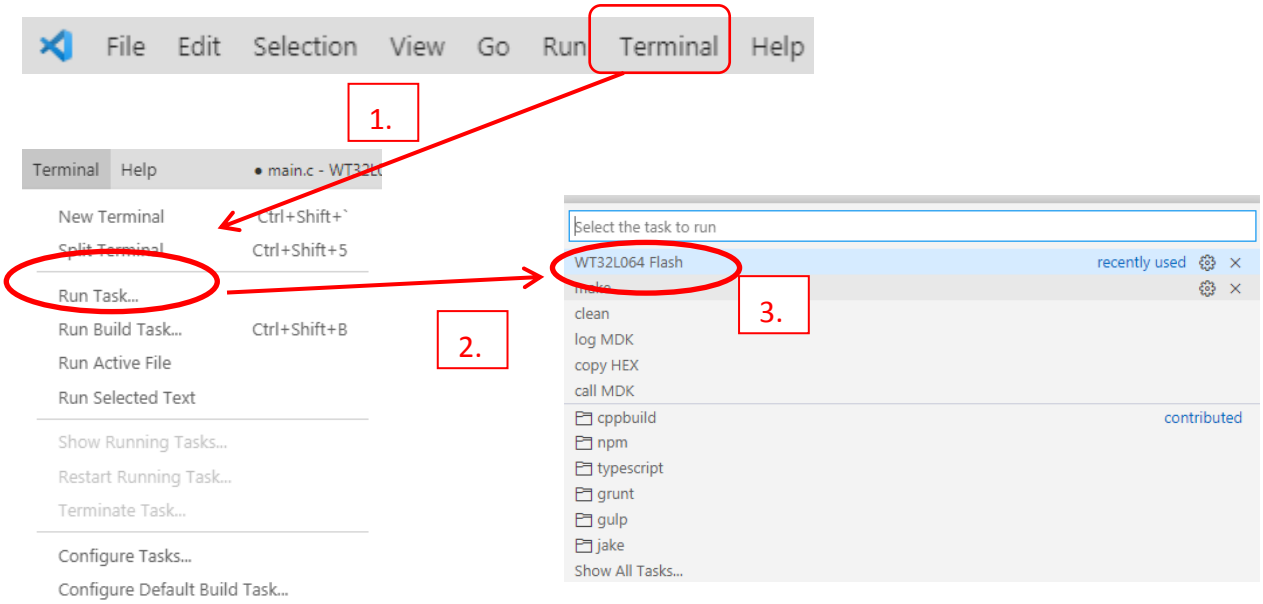
3.1.3 進行 C 程式編譯操作範例:

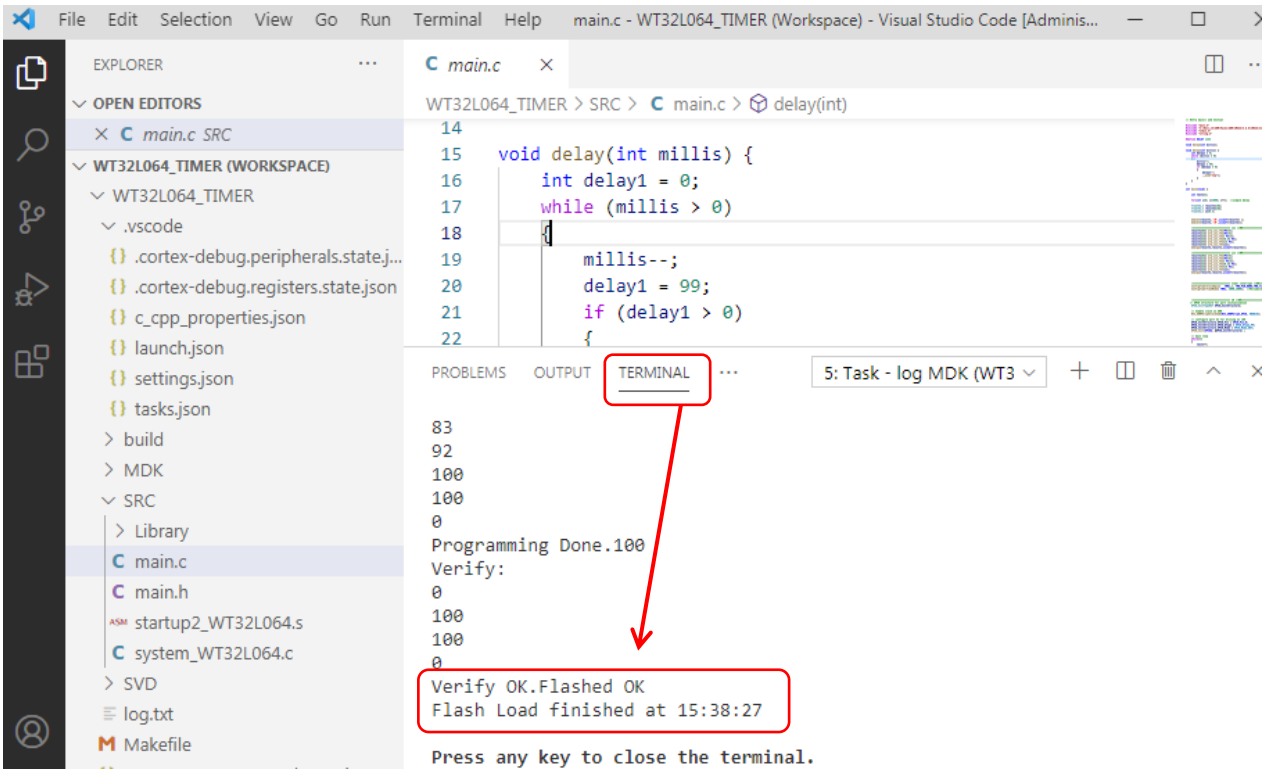
選擇 Terminal-> 選擇 Run Build Task-> 顯示編譯結果與偵錯(ELF)併燒錄(HEX)用檔案，如下圖所示。



3.1.4 開啟 Terminal 操作燒錄範例:

選擇 Terminal-> 選擇 Run Task-> WT32L064 Flash 進行燒錄(HEX)檔案，燒錄成功後結果會顯示在終端機窗口如下圖所示。



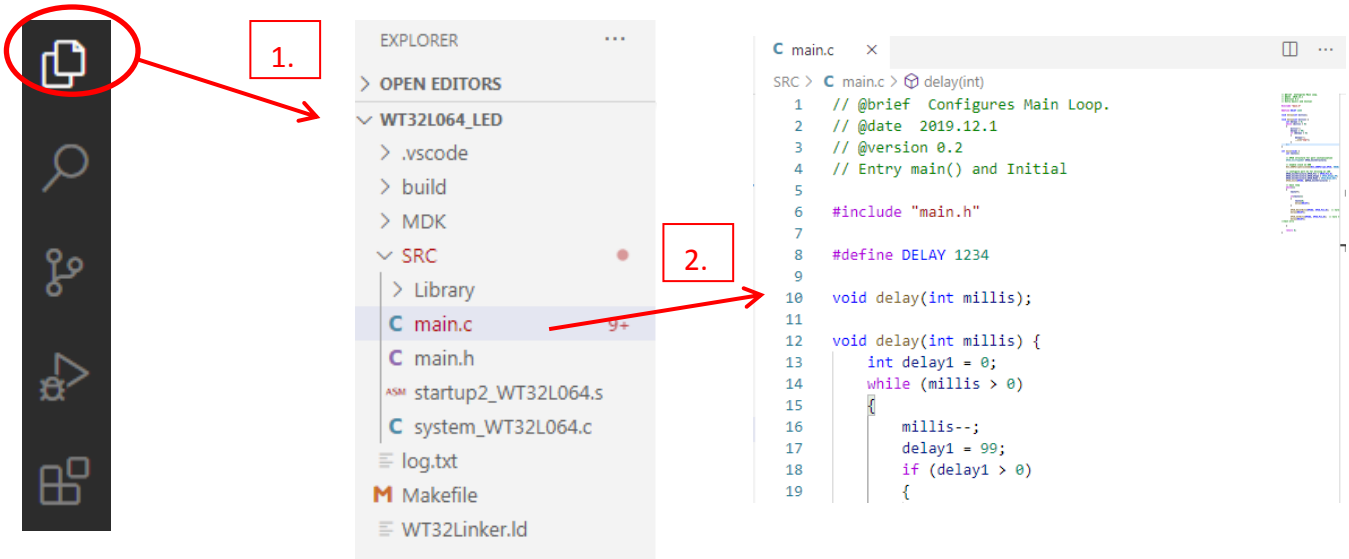


3.2 控制選單

控制選單一般在軟體畫面左側，常用功能有瀏覽資料夾、偵測，擴充功能一般僅在第一次安裝後會使用，下列章節進行範例與說明。

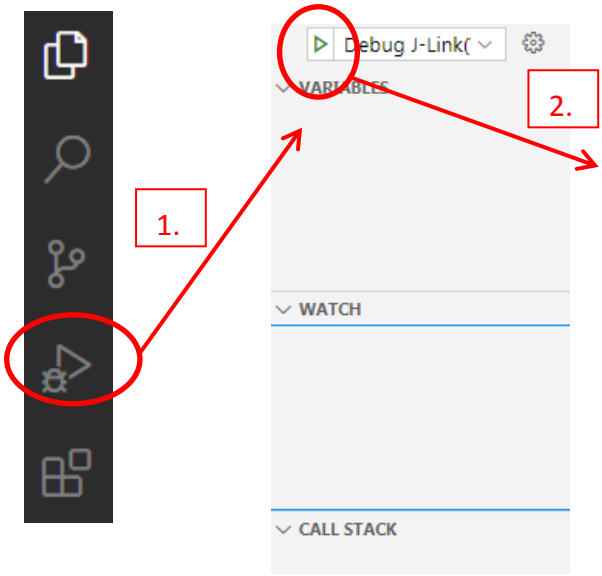
3.2.1 開啟專案文件清單範例:

選擇左側邊文件圖示->會顯示出目前檔案內容清單

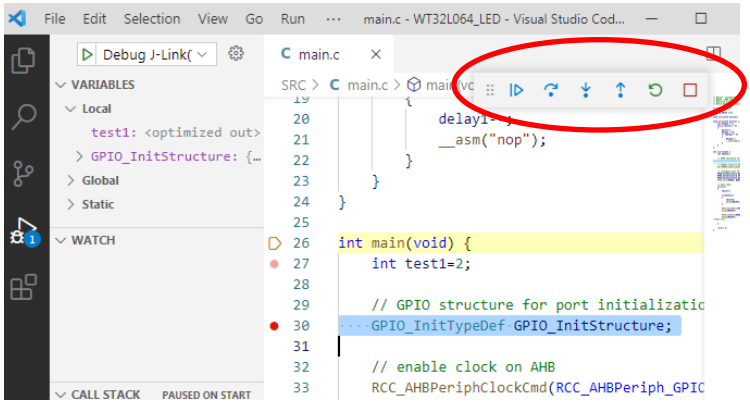


3.2.2 開啟偵錯(Debug)操作範例:

選擇左側邊偵錯三角圖示->會顯示出偵錯控制清單-> 選擇上方小綠色三角圖示，啟動偵錯

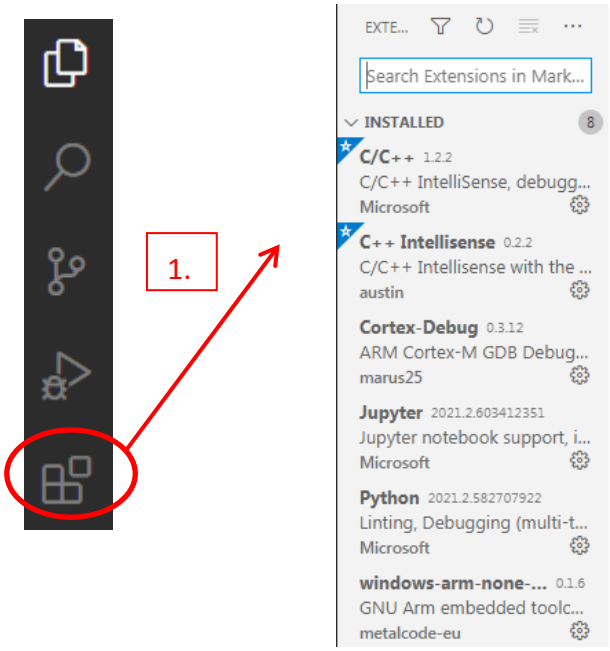


進入啟動偵錯模式



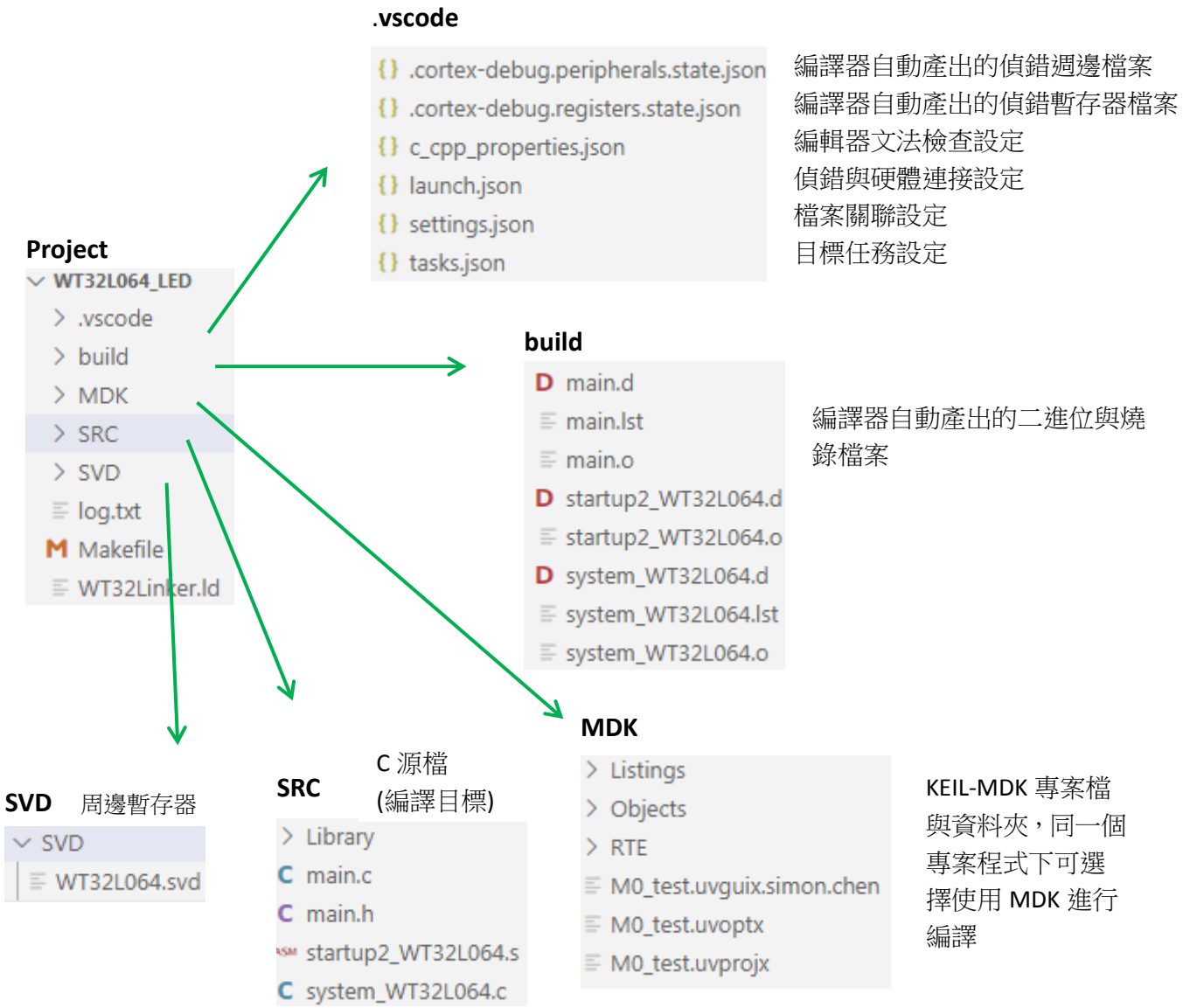
3.2.3 開啟擴充元件操作範例:

選擇左側邊方塊圖示->會顯示出目前已經安裝的擴充功能，一般只會在安裝初期使用，主要四個必要元件為 C/C++、C++ Intellisense、Cortex-Debug、windows-arm-none-eabi



3.3 專案資料夾內容

開啟專案資料夾進行瀏覽，其內容主要分四個資料夾，分別為.vscode、build、MDK、SRC，最外層為 log.txt、Makefile、WT32Link.ld 三個檔案，功能與圖示如下依序說明。



Makefile: 針對 GCC 編譯器的設定
WT32Linker.ld: 針對 GCC 產出的 OBJ 進行鏈結與產出燒錄檔
startup2_wt32l064.s: 針對 GCC 使用的開機初始化檔
startup_wt32l064.s: 針對 MDK 使用的開機初始化檔

4. VSCODE 程式開發與除錯

進行程式開發若有寫法錯誤會出現編譯失敗並無法產生 ELF 與 HEX 燒錄檔，也無法進行偵錯，錯誤主要分兩類為文法錯誤和邏輯錯誤，硬體使用 J-Link 傳輸 SWD 訊號並執行除錯命令，下列說明如何進行錯誤的程式寫法的排除。

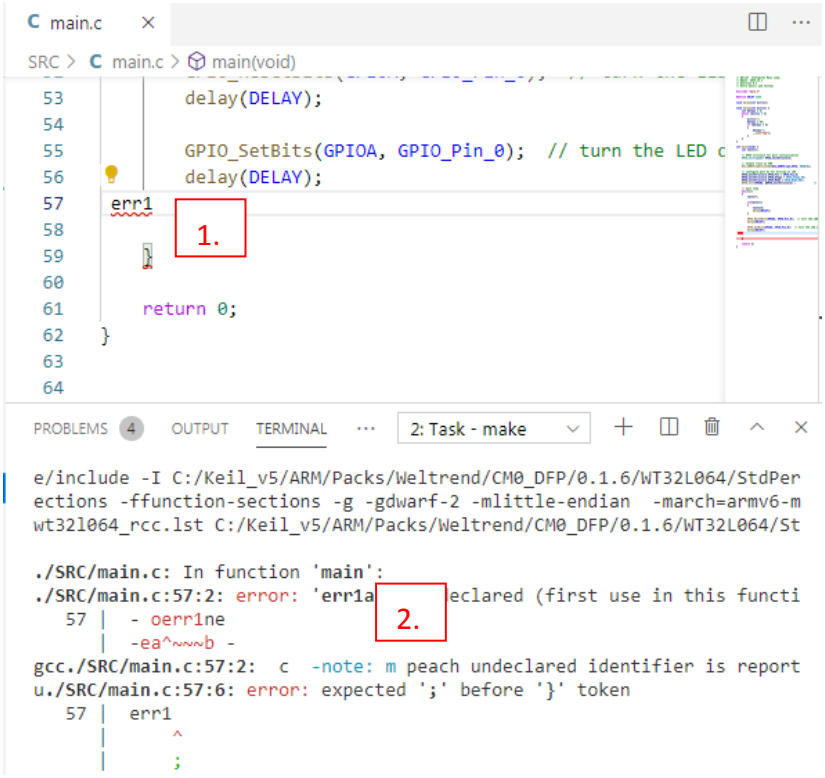
4.1 編譯功能說明

程式編寫完成後，可於上方主選單選擇 Terminal->Run Build Task，或按下快速鍵 CTRL+SHIFT+B，編譯成功會將出現下方提示文字於終端機視窗，並告知產出 HEX 檔案與其路徑，預設路徑為專案名稱\build，其中 ELF 檔案為偵錯 DEBUG 使用

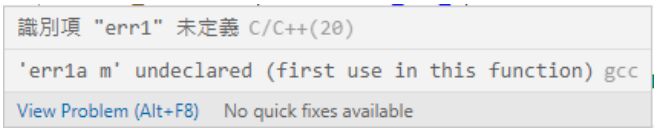
```
thumb   -specs=nano.specs -TWT32Linker.ld  -lc -lm -lnosys  -Wl,-Map=build/WT32L064_Sample.map,
cref -Wl,--gc-sections -o build/WT32L064_Sample.elf
arm-none-eabi-size build/WT32L064_Sample.elf
   text    data     bss     dec     hex filename
   1888       8    1316    3212     c8c build/WT32L064_Sample.elf
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 文法錯誤排除

一般錯誤寫法出現時會立即出現紅色波浪如下圖標示 1，進行編譯時下方輸出畫面則會出現紅色字樣如下圖標示 2。



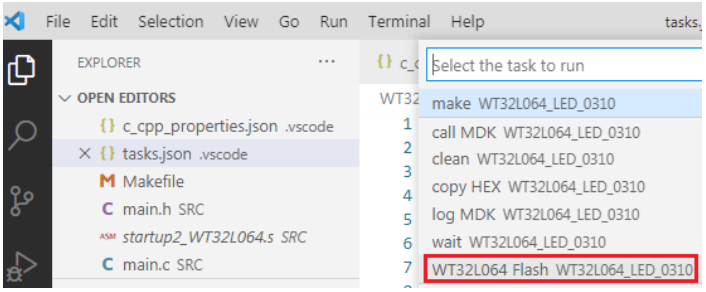
標示 1 的錯誤提示，將滑鼠移動倒到波浪處，會立即出現說明



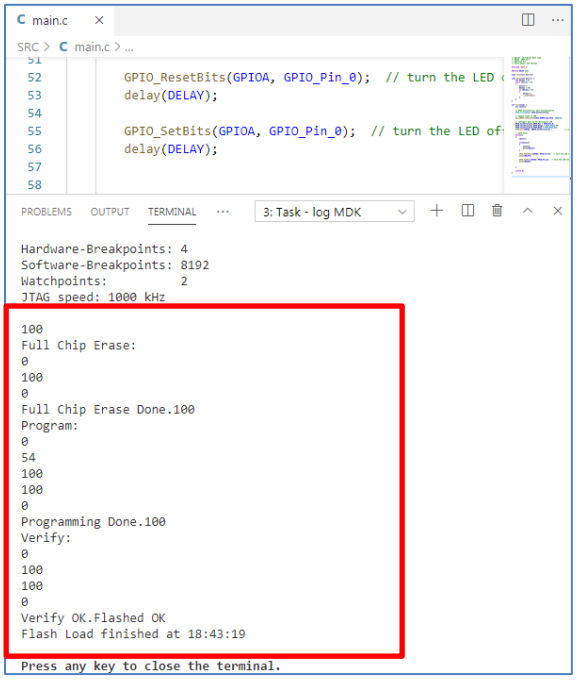
標示 2 的錯誤提示，則會出現關鍵檔名與行數列數，按下 CTRL+滑鼠左鍵，編輯器會協助跳至錯誤處，可進行程式編輯除錯。

4.3 燒錄功能說明

編譯完成後並產出 HEX 檔案，可參考 3.1.2 說明操作燒錄程序，點選下方圖示 WT32L064 FLASH xxx (your project)選單，之後會於下方終端機視窗出現燒錄成功訊息，如下圖所示。

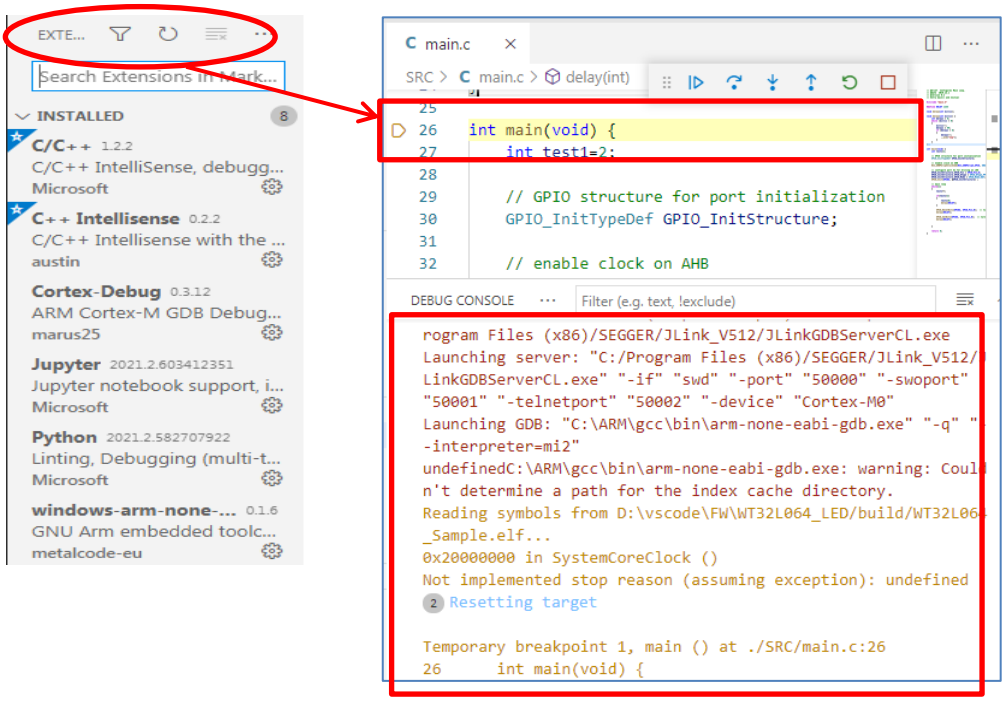


燒錄成功的訊息如下。

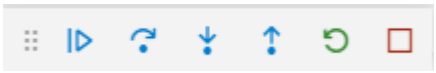


4.4 邏輯除錯功能

一般邏輯錯誤會使用 ICE 仿真模擬進行除錯，編譯完成後並產出 HEX 檔案，參考 3.2.2 章節操作，點選下方綠色三角圖示Debug-Jiink(WT)，編輯視窗會出現向右紅框箭頭並停在main.c 程式起始第一行，下方終端機會出現呼叫 arm-none-eabi-gdb 字樣並提出目前已停止在 int main(void) 如下圖所示。



此時可以在行列數欄位點兩下下紅點 斷點，如下圖左側紅點位置，或可點選浮動功能列



進行單步執行或停止功能。

```
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. 附錄

下列補充說明於 VSCODE 平台下主要使用的平台設定與編譯器設定檔案。

5.1 tasks.json 功能說明

在主選單 Terminal->Run Task 下所可執行的任務，依序為 make 編譯程式、clean 清除舊檔、WT32L064 Flash 燒錄程式，若無特殊任務需求可使用目前預設任務即可，部分內如下所示。

```
"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 功能說明

偵錯使用的連結檔，可設定呼叫特定路徑的驅動連接至 VSCODE 進行同步偵錯，若有變更橋接器才需修改，部分內如下所示。

```
"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 功能說明

進行 GCC 程式編譯的設定檔案，包含 INCLUDE 路徑與所有的目標源始檔進行編譯，可觀察 Makefile 文件內容中目前 CMSIS 是否為最新版本如圖紅色標示，目前在 SRC 資料夾內都會自動編譯。

```
########################################
# 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 功能說明

參照 GCC 與 CMSIS 用法進行 Link 設定，主要設定程式起始位址與 RAM、ROM 長度，指定程式入口函式並安排中斷與資料區域，若有變更目標 IC 才需更換或修改，部分內如下所示。

```
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. 版本更改紀錄:

版本	內容	時間
V0.1	初版	2021.03.02
V0.2	補充說明 gcc/build tools 安裝	2021.03.18
V0.3	補充說明 Extension 與 GCC 預設路徑	2021.03.31