

# 使用 QEMU + FreeRTOS + Visual

## Source Code 開發嵌入式系統

[Kevin Huang](#)

Dec 13, 2020

開發 Linux 核心與嵌入式系統已經一段時間，用的都是原廠的工具/編譯器，或者是廠商提供的開發版，但是平常在開發系統主要用的是 MacOS 和 Ubuntu。有些公司的開發工具都會侷限在 Windows 上面，雖然 Windows 10 已經有 Ubuntu 的子系統可以用，但總覺得不是很好用...所以開始在想建構自己的 OS 底層和工具鏈，抽空開始慢慢弄。

其實很少在開發應用的核心的時候用 QEMU，因為公司的設備與工具其實不錯，不太需要用模擬器... 不過如果人在公司外面，想開發一些自己興趣的應用 (Maker 本色)，的確使用 QEMU 可以節省很多時間，而且還可以任意定址硬體（如何設計 QEMU 的硬體模擬，請參閱另外一篇文章），在做設計組合研究的時候，其實是很好用的工具。

### 開始動手吧

首先，我的環境是使用 Ubuntu 20.04 LTS，安裝的時候可以使用一台實體機器，當然也可以使用 VirtualPC，我是使用 LXC container 將 Ubuntu 20.04 LTS 安裝起來。用 Ubuntu (Debian Package Manager) 的好處是，一些常用的工具你不需要自己從頭 Rebuild 出來（這個過程相當惱人...），可以用已經存在的套件先把環境架設起來，有需要修改環境或是要追版本的時候可以再下載 source code 自己慢慢 build。

Ubuntu 20.04 LTS 安裝完成之後，將系統更新到最新版本

```
$ sudo apt update; sudo apt upgrade
```

### 安裝開發工具與 QEMU 模擬器

當然 GNU 的開發工具是少不了的，還有一些附加的工具，像是編輯器 (vim)，或是代碼管理工具 (git)，打包工具 (make)，網路工具包 (net-tools) ... 等等，我們也一併安裝上去。

```
$ sudo apt install git vim make gcc gcc-arm-none-eabi gdb gdb-multiarch  
make net-tools qemu-system universal-ctags -y
```

經過一段時間的下載安裝的程序，全部正確安裝完成之後，你就有個基本的環境可以使用在開發工作上。

## 在 QEMU 上面使用 FreeRTOS

開始來編譯你的第一個 Kernel 吧，習慣在家目錄下建立一個名稱為 “work” 的工作目錄，將我和開發相關的代碼都放在那裡。

```
$ mkdir work; cd work
```

之後從 github 上面下載完整的 FreeRTOS

```
$ git clone https://github.com/FreeRTOS/FreeRTOS.git
```

有些 source code 是在 git 下面以 submodule 的方式存在，因此必須進入 FreeRTOS 目錄下，並且打入下面的指令把所有 submodule 的代碼全部拉下來。

```
$ git submodule update --init --recursive
```

把代碼全部拉下來之後，基本上的環境和測試用的代碼就準備齊全了

## 平台的選擇

這次用來測試的 CPU 架構選擇 ARM CORTEX-M3，而開發板選擇的是 STM32，選擇這個板子的原因是因為 FreeRTOS 本身有提供這個 CPU 的模擬，而 QEMU 也把這片板子的外圍週邊硬體模擬得很好。

[開發板的資料](#)

[CPU 的資料](#)

FreeRTOS 裡面，這片開發板的 Kernel 可以在目錄


FreeRTOS/FreeRTOS/Demo/CORTEX\_LM3S811\_GCC 內找到，入手不用再花什麼腦筋。

## 編譯 FreeRTOS

編譯這個開發板用的 FreeRTOS 也很簡單，因為 FreeRTOS 的源代碼樹裡面就有這片開發板用的代碼和 Makefile，進入相對應的目錄

(~/work/FreeRTOS/FreeRTOS/Demo/CORTEX\_LM3S811\_GCC)，直接打 make，可用的 FreeRTOS Kernel 就會被 Build 出來，很簡單。

但是在 make 之前要注意的是，這次我們要用 gdb 來做 kernel source level 的 debug，需要把 symbol table 產生出來，因此要稍微修改一下 Makefile，把“-g”這個選項加到 CFLAGS 裡面，如下圖：



```
26 include makedefs
27
28 RTOS_SOURCE_DIR=../../Source
29 DEMO_SOURCE_DIR=../Common/Minimal
30
31 CFLAGS+=-g -I hw_include -I . -I ${RTOS_SOURCE_DIR}/include -I ${RTOS_SOURCE_DIR}/portable
32
33 VPATH=${RTOS_SOURCE_DIR}:${RTOS_SOURCE_DIR}/portable/MemMang:${RTOS_SOURCE_DIR}/portable/Queue
34
35 OBJS=${COMPILER}/main.o \
36       ${COMPILER}/list.o \
37       ${COMPILER}/queue.o \
```

A red arrow points to the `-g` option being added to the `CFLAGS` variable on line 31.

記得把 -g 選項加入，產生 kernel 的 symbol table

接下來就可以來編譯了

```
$ cd ~/work/FreeRTOS/FreeRTOS/Demo/CORTEX_LM3S811_GCC; make
```

編譯成功之後，在 gcc 目錄下會產生兩個檔案，一個叫做 RTOSDemo.axf，這個檔案是給 gdb 看的。另外一個叫做 RTOSDemo.bin，這個檔案是要丟進去 QEMU 來執行的。

可以先執行下面的命令看看，如果成功的話，核心就已經跑起來了，並且 QEMU 會在 port 1234 下開始等待 gdb 的連線。

```
$ qemu-system-arm -machine lm3s811evb -kernel gcc/RTOSDemo.bin -s -S -nographic
```

在還沒有設定 VSCode 之前，可以自己打下面的命令，看是不是可以連得上 QEMU 用的 gdb server。

```
$ gdb-multiarch -ex="target remote localhost:1234" gcc/RTOSDemo.axf
```

如果可以連上，那麼下 `gdb` 的 `list` 命令，你應該可看到下列的畫面，可以看到停在第一個指令上，恭喜你，你已經完成 FreeRTOS 在 QEMU 目標硬體模擬上的運作。

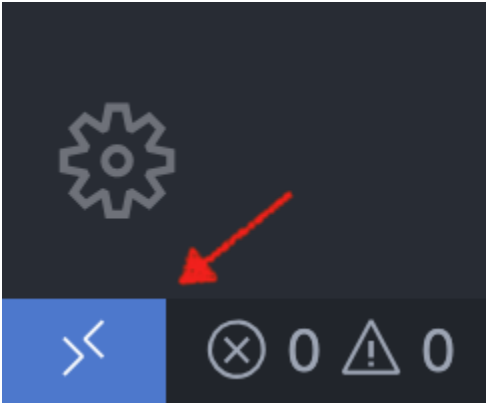
```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from gcc/RTOSDemo.axf...
Remote debugging using localhost:1234
ResetISR () at init/startup.c:151
151         for(pulDest = &_edata; pulDest < &_edata; )
(gdb) list
146
147         //
148         // Copy the data segment initializers from flash to SRAM.
149         //
150         pulSrc = &_etext;
151         for(pulDest = &_edata; pulDest < &_edata; )
152         {
153             *pulDest++ = *pulSrc++;
154         }
155
(gdb) █
```

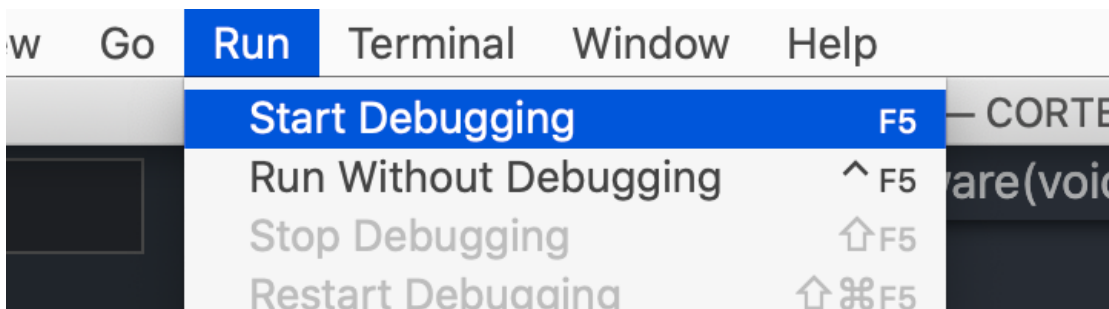
## 設定 VSCode

雖然用 `vim` + `ctags` 也可以方便地開發應用程式，但是別人看到我用 `vim` 的表情都好像我是摩登原始人似的，另一方面自己也覺得 VSCode 整合得愈來愈好，一些開發用的工具 `gdb` / `git` / ... 和 VSCode 都整合得相當好（有很多很厲害的 `extension` 可以用），所以接下來就來談一下如何使用 VSCode 來做 FreeRTOS 的 `source level debug`。

我的 VSCode 是跑在 MacBook Pro 上面，而開發環境（`host`）使用 PVE 的 LXC 跑在家裡的一台機器上面的 Ubuntu 20.04 LTS，需要先測試好 MacBook 的 `ssh` 連線和開發用的 `host` 機器之間是可以相連的。確定好之後，打開 VSCode，左下角的這個按鈕可以讓你將本機的 VSCode 透過 `ssh` 連線到遠端去進行開發。

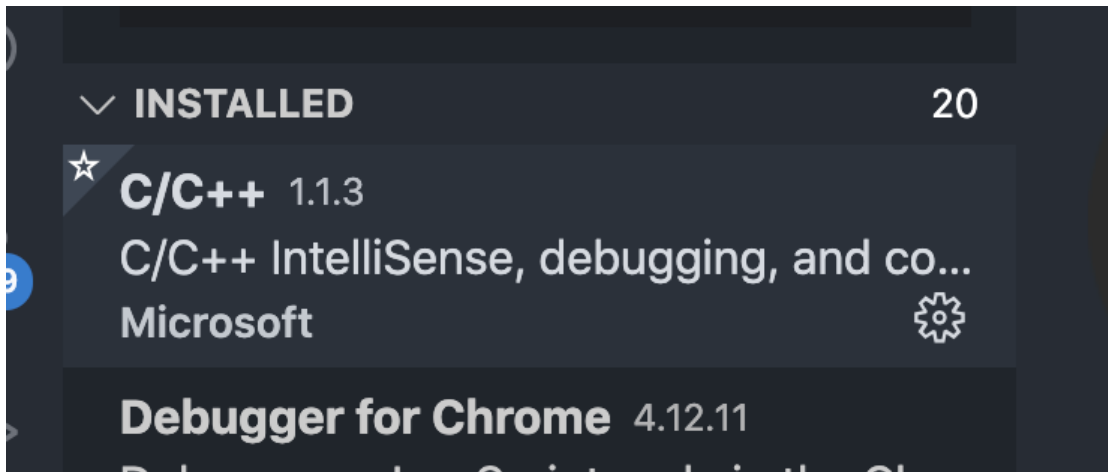


連線成功之後，選擇目錄到 `~/work/FreeRTOS/FreeRTOS/Demo/CORTEX_LM3S811_GCC`，如果你已經編譯好的話，直接選取 `Run → Start Debugging` 如下圖。



開始做 `source level debug`

如果是第一次用，記得要安裝 `C/C++ IntelliSense` 這個外掛外掛，才能開始用 `gdb debug`。



安裝 `C/C++ IntelliSense`

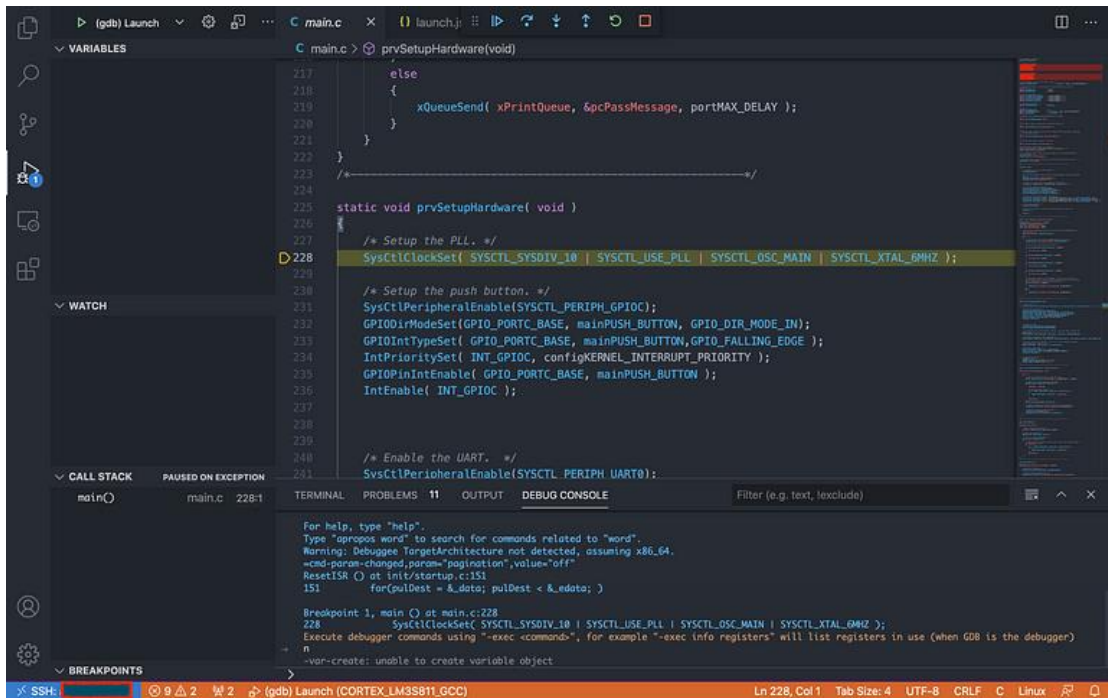
`Start Debugging` 之後，就會啟動 `source level debug`，如果你是第一次做，VSCode 會彈出一個視窗，讓你填寫 `launch.json`，依照下方的文字填寫進去，告訴 VSCode 這個東西要怎麼 `debug`，`symbol table` 在哪裡...等等這些資訊。

```

{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit:
https://go.microsoft.com/fwlink/?linkid=830387
  {
    "version": "0.2.0",
    "configurations": [
      {
        "name": "(gdb) Launch",
        "type": "cppdbg",
        "request": "launch",
        "program": "${workspaceFolder}/gcc/RTOSDemo.axf",
        "miDebuggerServerAddress": "localhost:1234",
        "args": [],
        "stopAtEntry": false,
        "cwd": "${workspaceFolder}",
        "environment": [],
        "externalConsole": false,
        "MIMode": "gdb",
        "setupCommands": [
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        ]
      }
    ],
    "miDebuggerPath": "/usr/bin/gdb-multiarch",
    "miDebuggerServerAddress": "localhost:1234"
  }
]
}

```

接下來重新 **Run → Start Debugging**，如果出現下面的畫面，那恭喜你，你已經可以在 **VSCode + QEMU** 模擬平台上面 **debug** 你的 **FreeRTOS kernel** 和應用程式了。



FreeRTOS + QEMU + VSCode 設定成功

## MacOS 也可以當 Host 用嗎？

可以的，但是 MacOS 在 XCode 把 gdb 從開發環境內移除之後，就只能自己使用 brew 去安裝 gdb。而 brew 裡面又沒有 cross-compiler 的環境可以用，還是都需要自己去安裝，雖然麻煩一點。

## 安裝 MacOS 版本的 gcc-arm-none-eabi

安裝 MacOS 版本的 cross-compiler。可以到[這裡](#)去下載相對應的 toolchain，我自己是直接下載 [gcc-arm-none-eabi-10-2020-q4-major-mac.pkg](#)，下載完成之後點擊幾下就安裝完成，如果沒有變更安裝目錄的話，他的 toolchain 的執行檔會放在 `/Applications/ARM/arm-none-eabi/bin` 下面，記得更改你的 `.zshrc` 將這個目錄放進去 `$PATH` 變數內。這樣就完成了安裝。

## 安裝 MacOS 版本的 QEMU

這個部分比較簡單，打下面的指令安裝 MacOS 版本的 QEMU

```
$ brew install qemu-system
```

接下來，就按照前面的步驟就可以了。