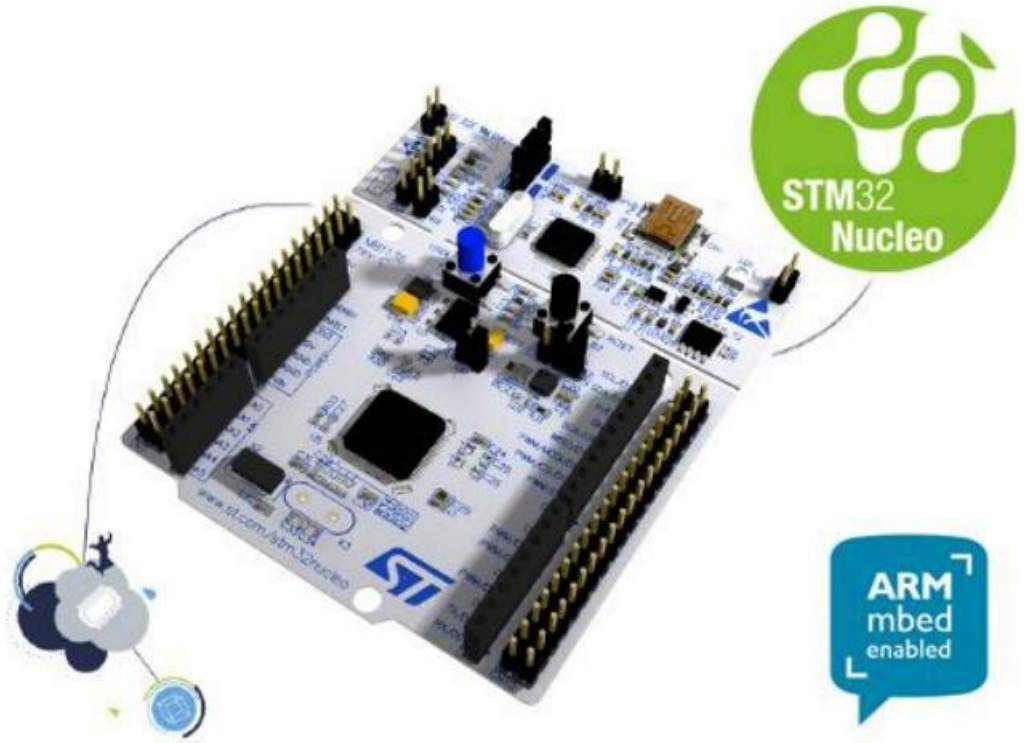
MPSL2019

Lab0

Lab hardware
實驗硬體

STM32 Nucleo Board

- An ARM Cortex-M4 development board
- Build in a ST-LINK as debugger
- Arduino pin compatible
- One user button
- One LED



Hardware Block

Figure 3. Top layout

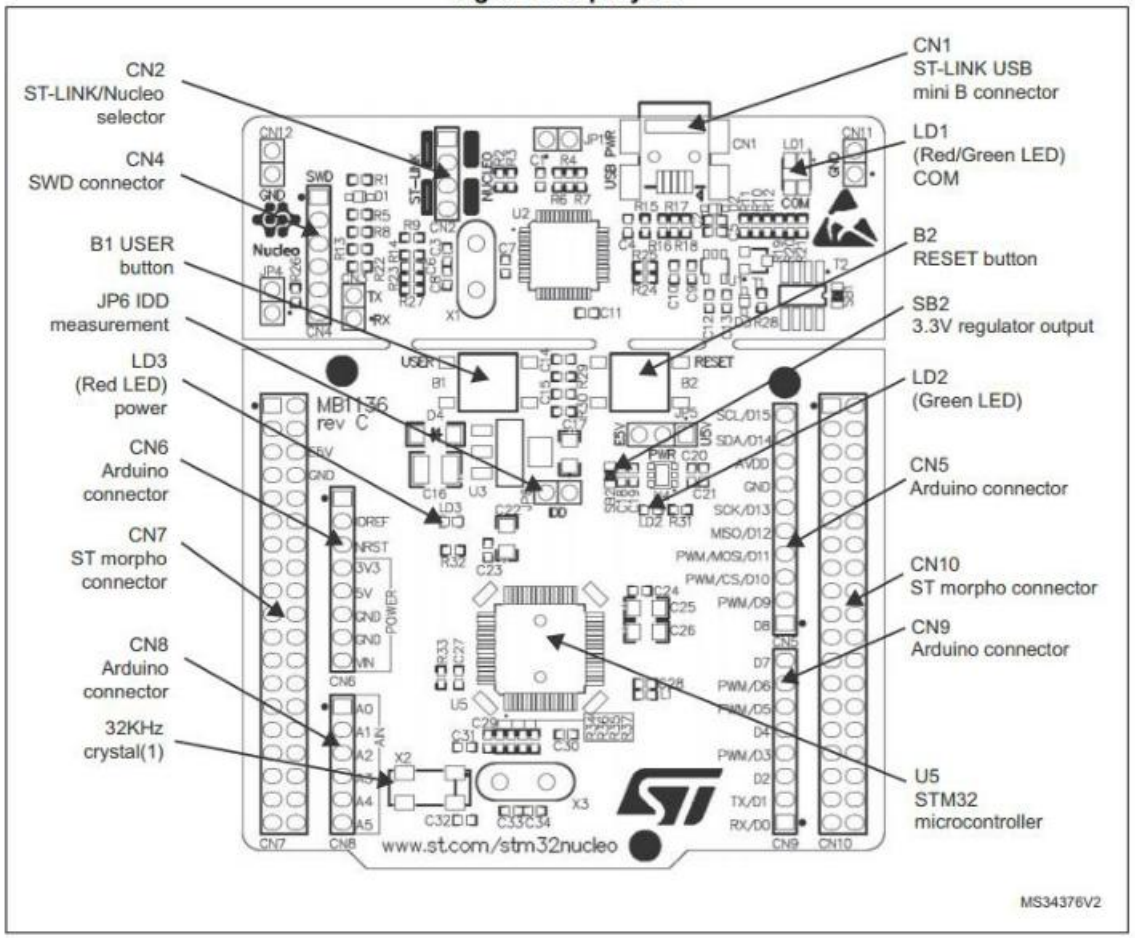
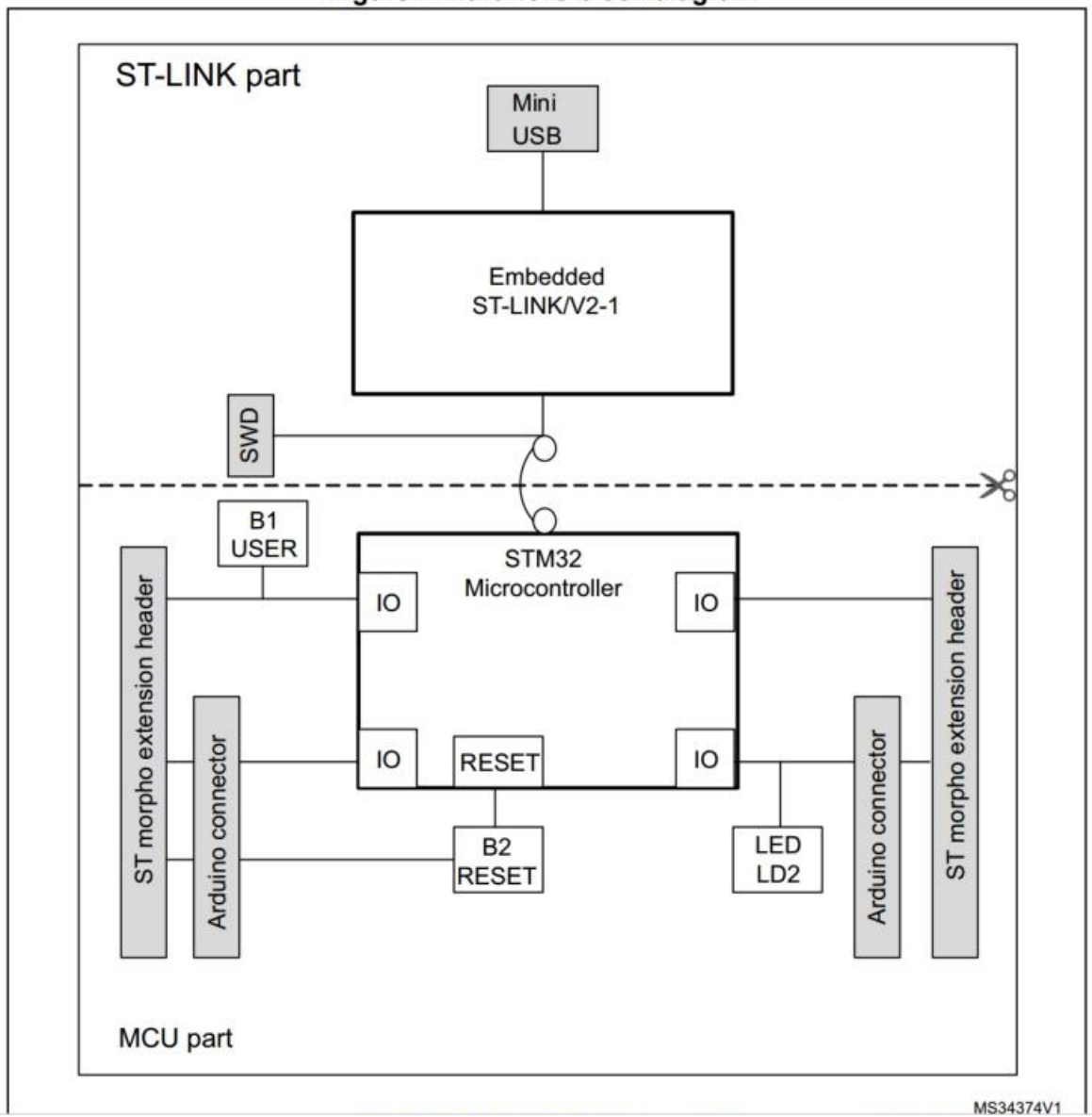


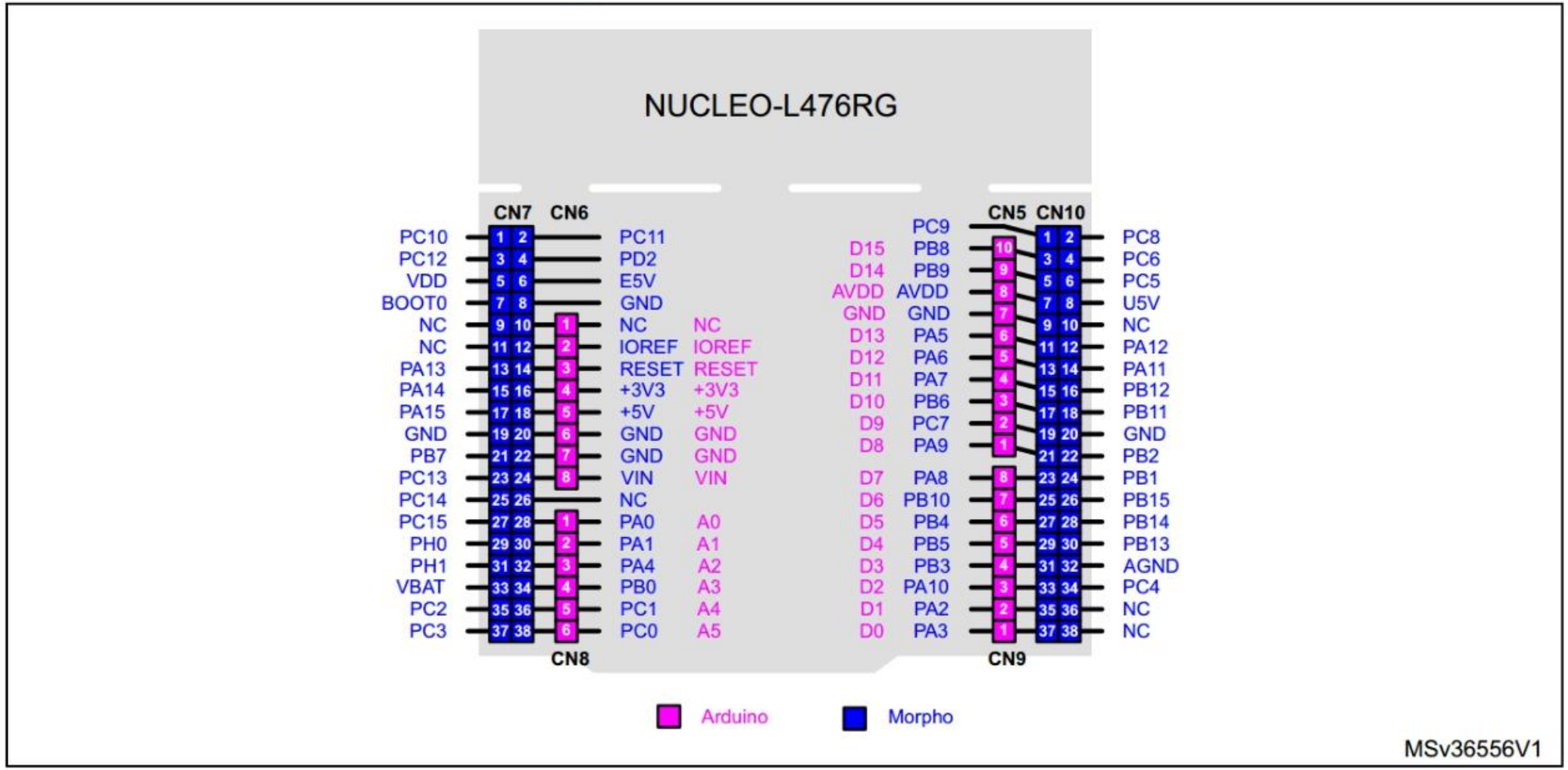
Figure 2. Hardware block diagram



From UM1724-STM32-NUCLEO-User-Manual.pdf

Pin Map

Figure 22. NUCLEO-L476RG



From UM1724-STM32-NUCLEO-User-Manual.pdf

Lab software
實驗軟體

Development Environment

- We use SW4STM32 which is a **eclipse based STM32 IDE tool**
  - STM32 Devices database and libraries
  - Source code editor
  - Linker script generator
  - Building tools (GCC-based cross compiler, assembler, linker)
  - Debugging tools (OpenOCD, GDB)
  - Flash programing tools
  - http://www.openstm32.org/HomePage

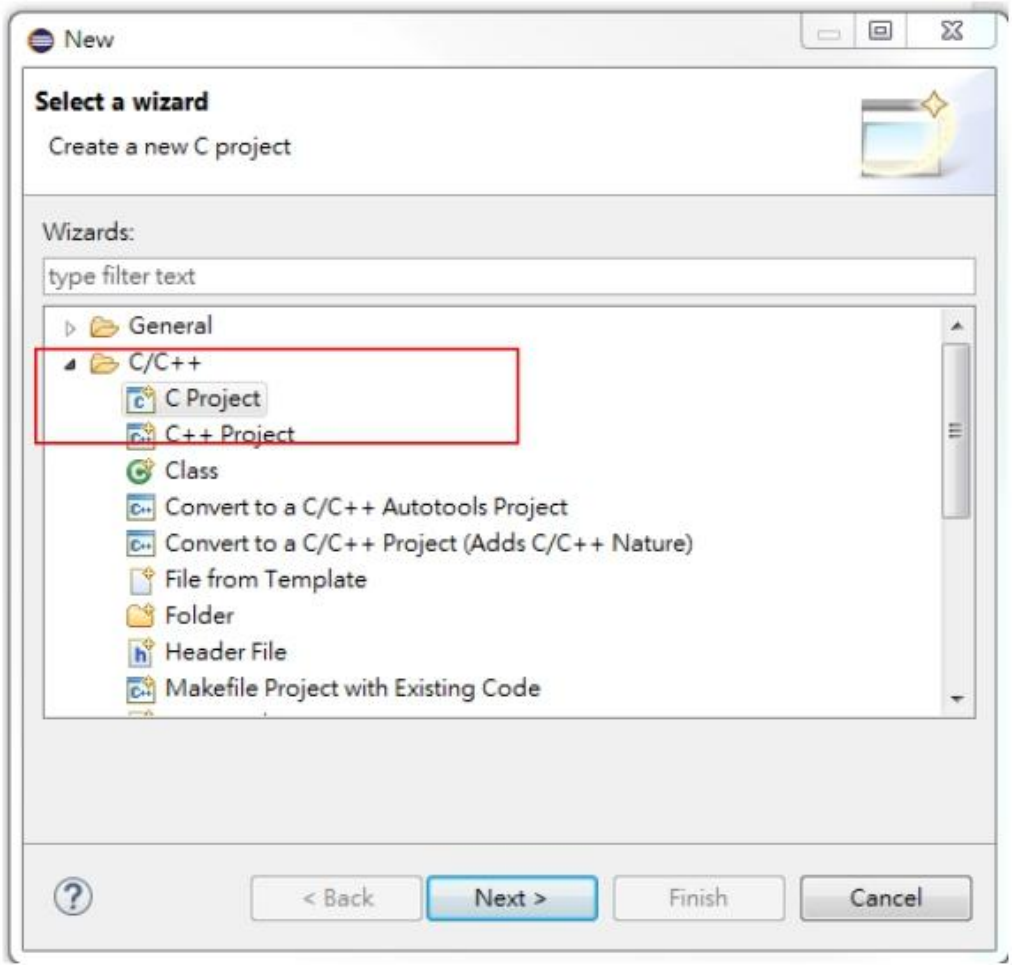
SW4STM32 IDE

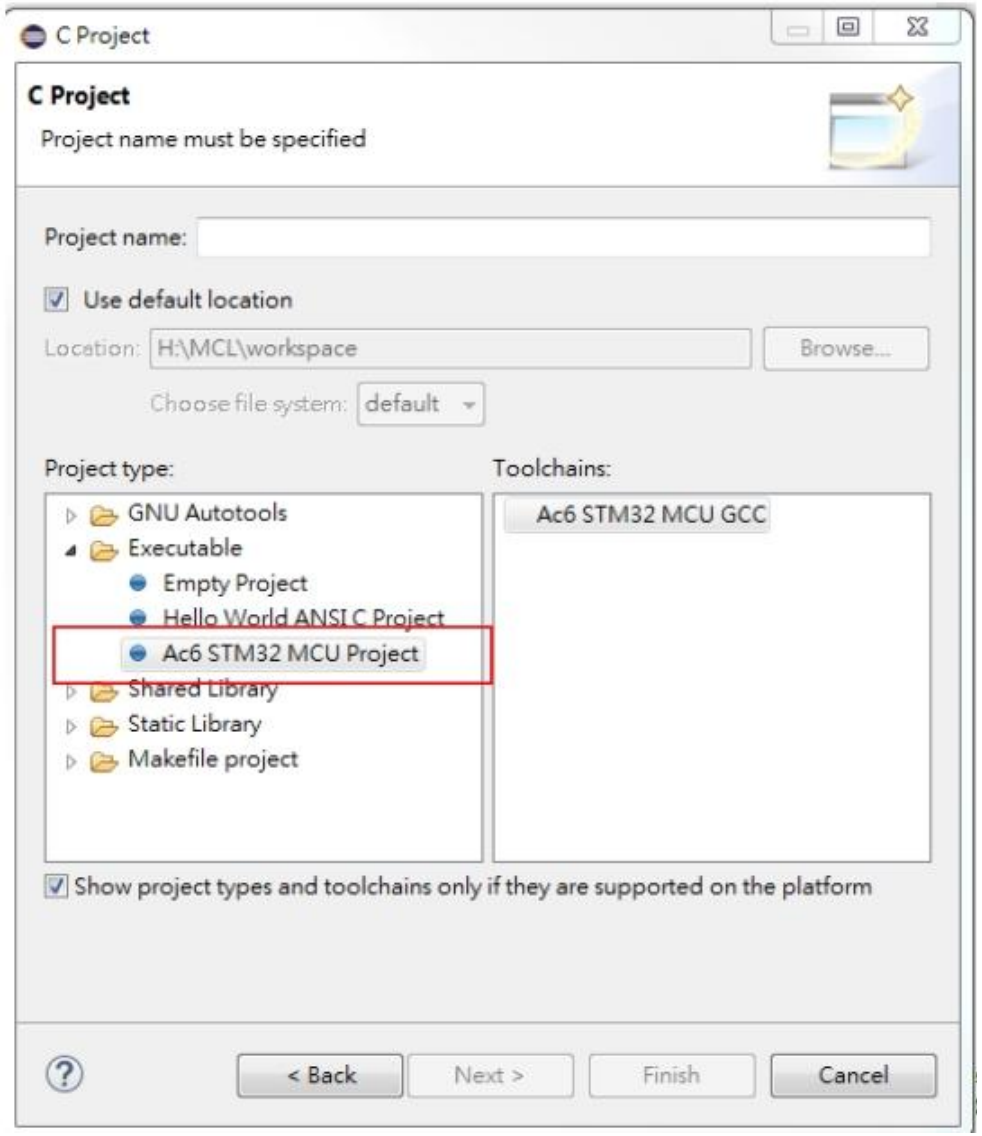
- Download_Page
- Windows 7 or Windows 10
  - http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_win_64bits-latest.exe
- Linux
  - http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_linux_64bits-latest.run
  - Dependence
    - JRE7 or later
    - Need some shared librarys
      - libc6:i386 lib32ncurses5
- MacOS
  - http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_macos_64bits-latest.run

SW4STM32 Getting Started Guide

Create Project

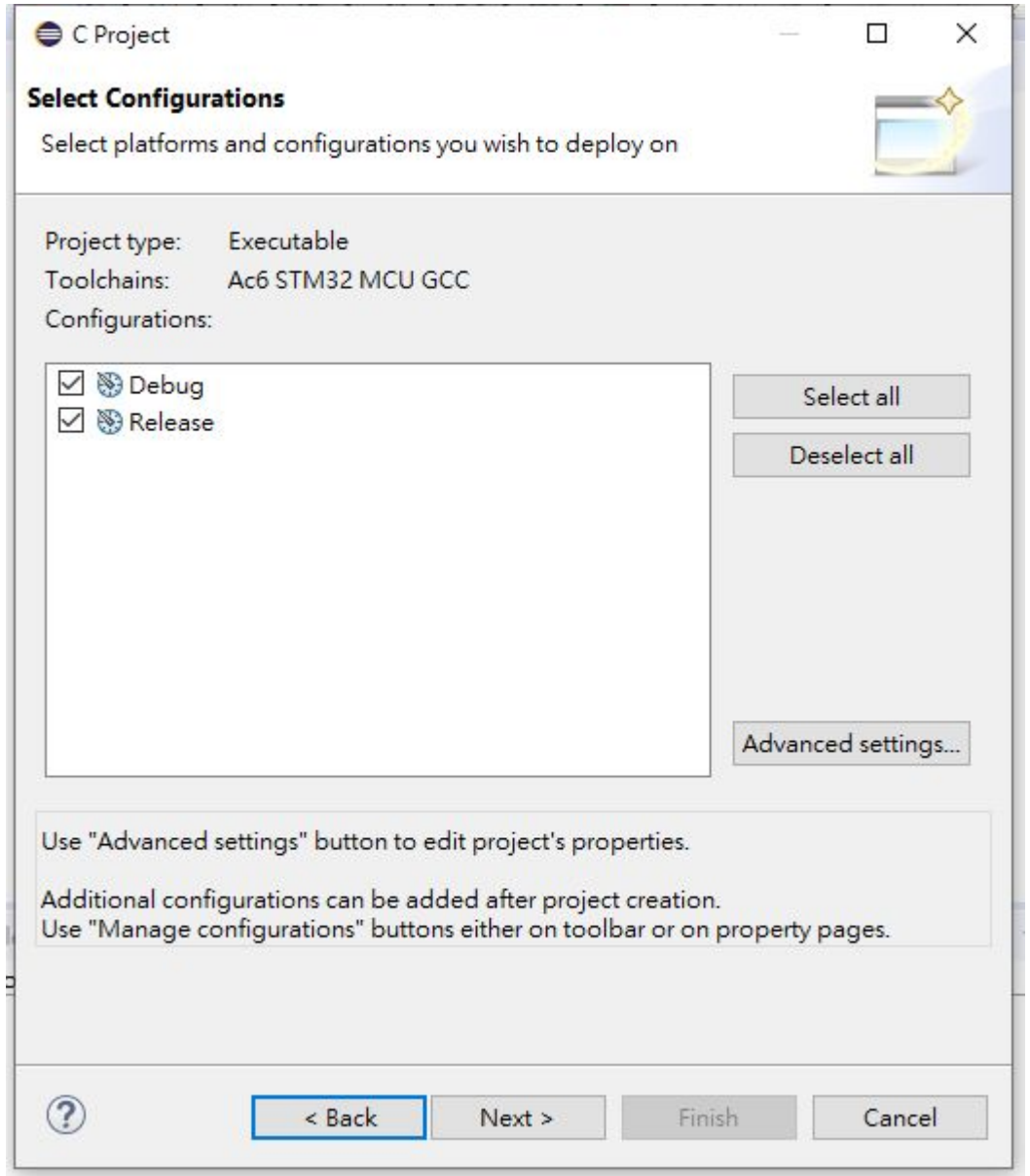
File -> New -> Other





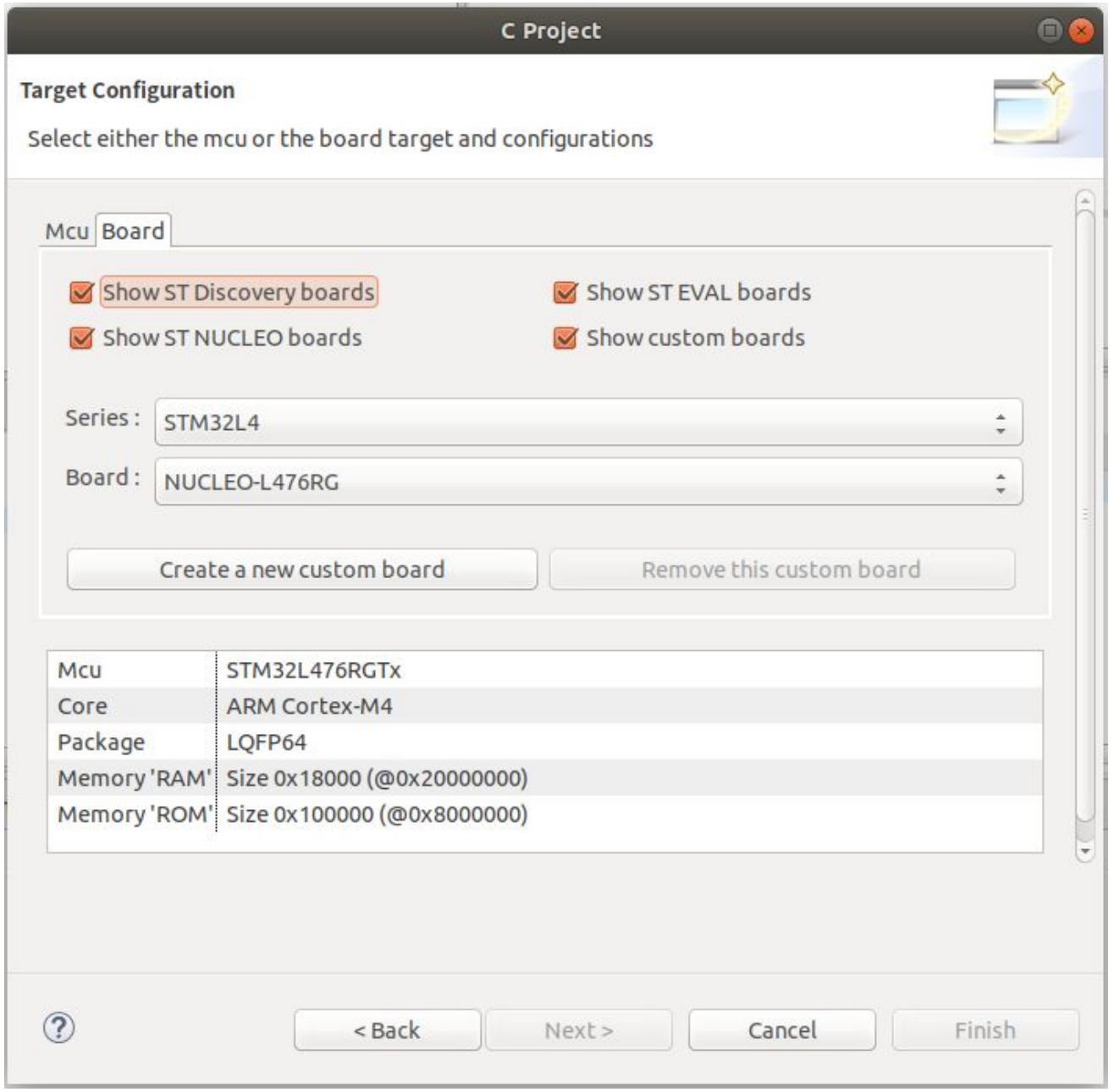
Create Project (cont.)

Next

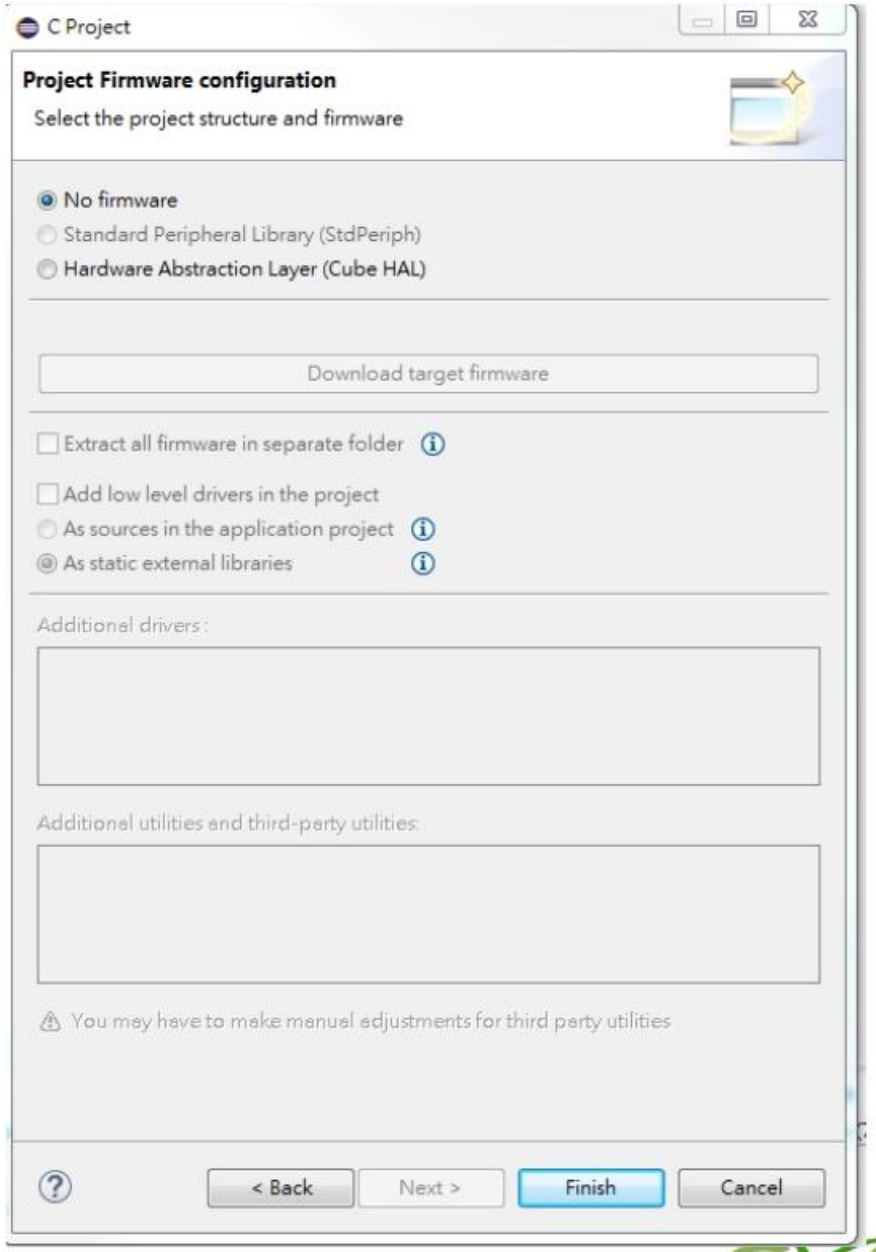


MCU Configuration

- Series select STM32L4
- Board select NUCLEO-L476RG

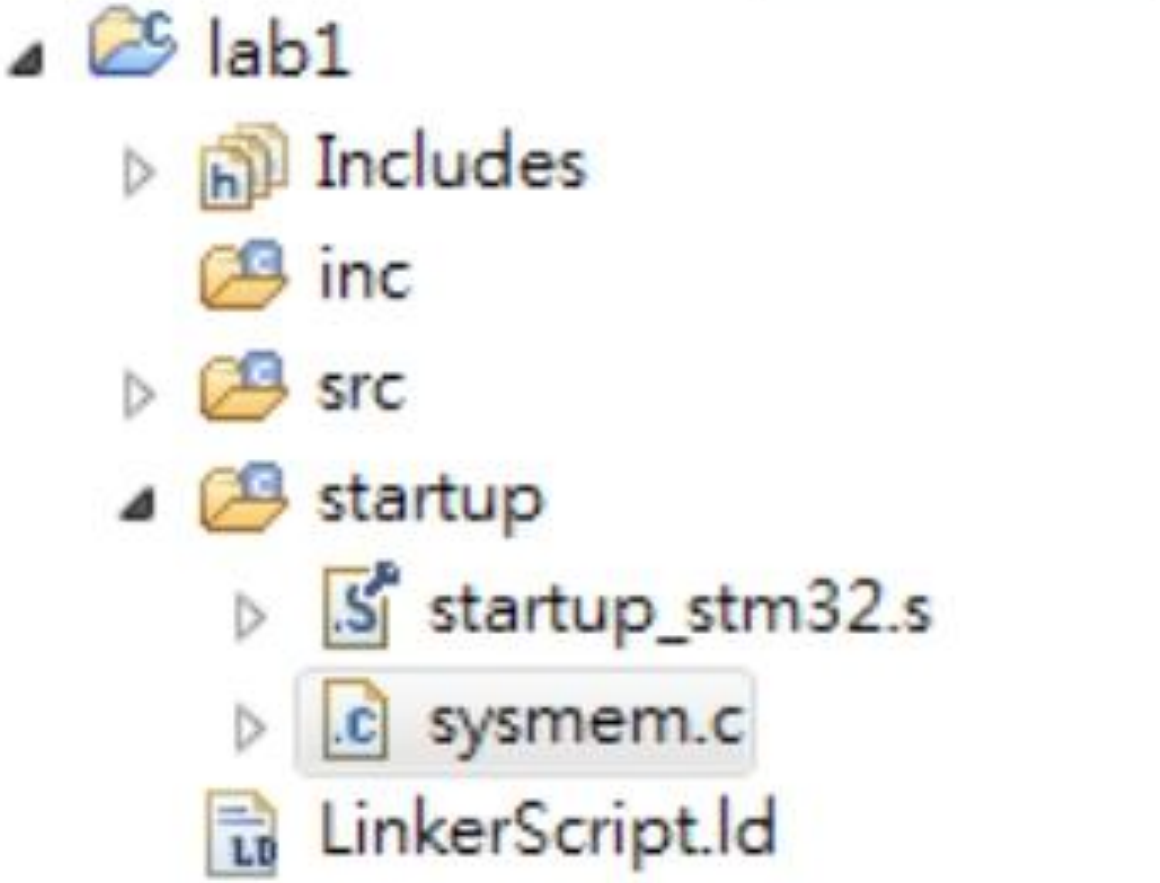


- Choose No firmware
- Then press Finish



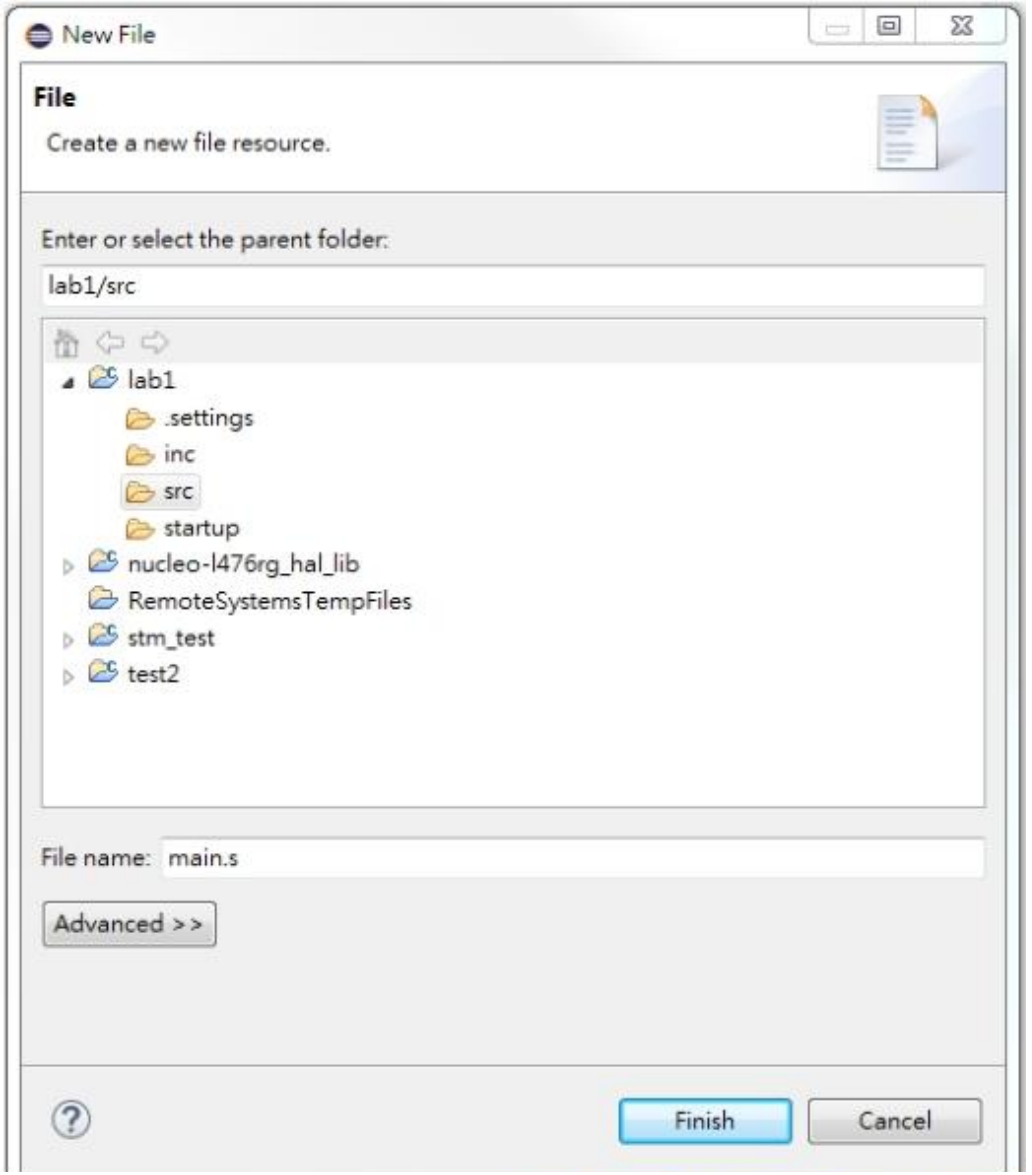
Project Files

- Then you can see the project files in the Project Explorer list
- It contain the board startup code **startup_stm32.s** and linker script **LinkerScript.ld**
- This about practice 1



Create File

- Right click the src folder
- Select New -> File
- Create a file call **main.s**



Write Your First Code

Use UAL syntax

Text section start point

Define global symbol

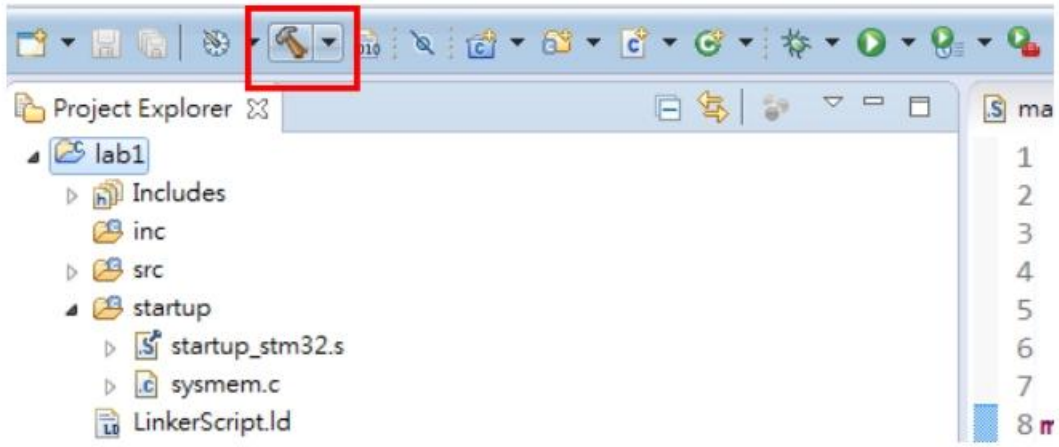
Define a constant symbol AA

```
1     .syntax unified
2     .cpu cortex-m4
3     .thumb
4 .text
5 .global main
6 .equ AA,0x5566 // How about 0x1000 ?
7
8 main:
9     movs r0, #AA
10    movs r1, #20
11    adds r2,r0,r1
12    b main
13
```

main.s

Build Code

- Write your first code
- Project->Build all



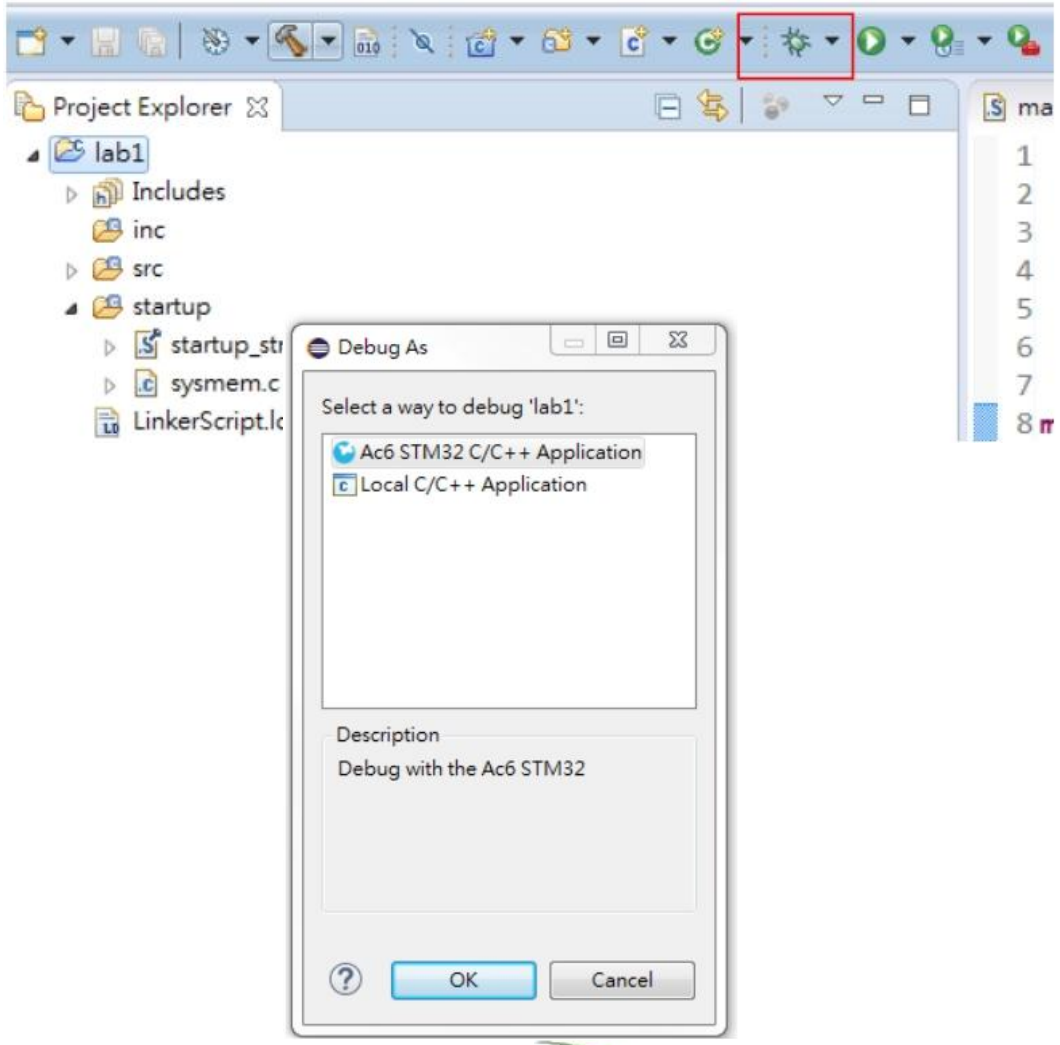
Build result

```
'Building target: lab1.elf'
'Invoking: MCU GCC Linker'
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16
'Finished building target: lab1.elf'
' '
make --no-print-directory post-build
'Generating binary and Printing size information:'
arm-none-eabi-objcopy -O binary "lab1.elf" "lab1.bin"
arm-none-eabi-size "lab1.elf"
   text    data     bss     dec     hex filename
    992    1080    1056    3128     c38 lab1.elf
' '
```

Create the target image file

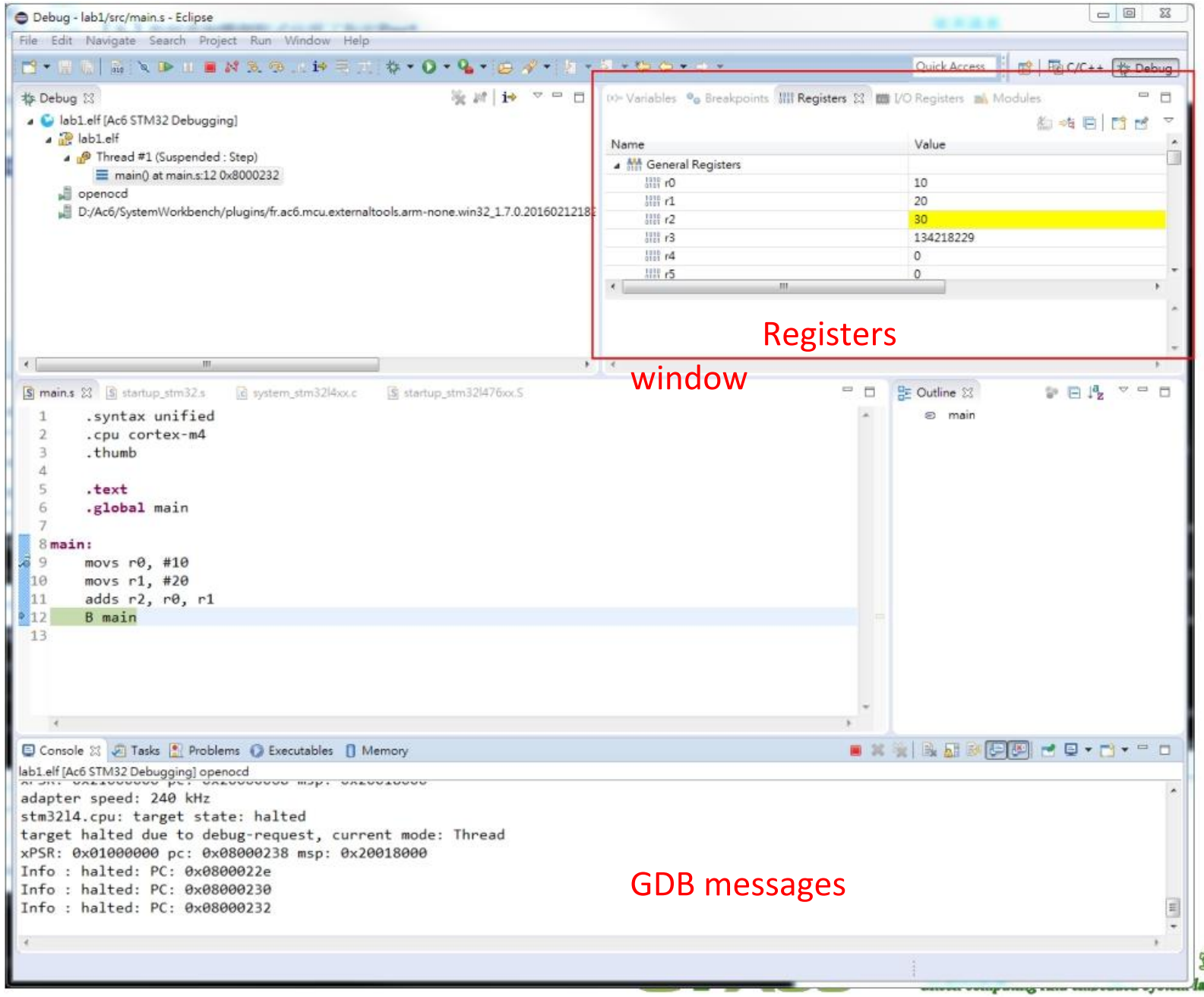
Debug your Code on board

- Run->Debug
- Debug as AC6 STM32 C/C++ Application



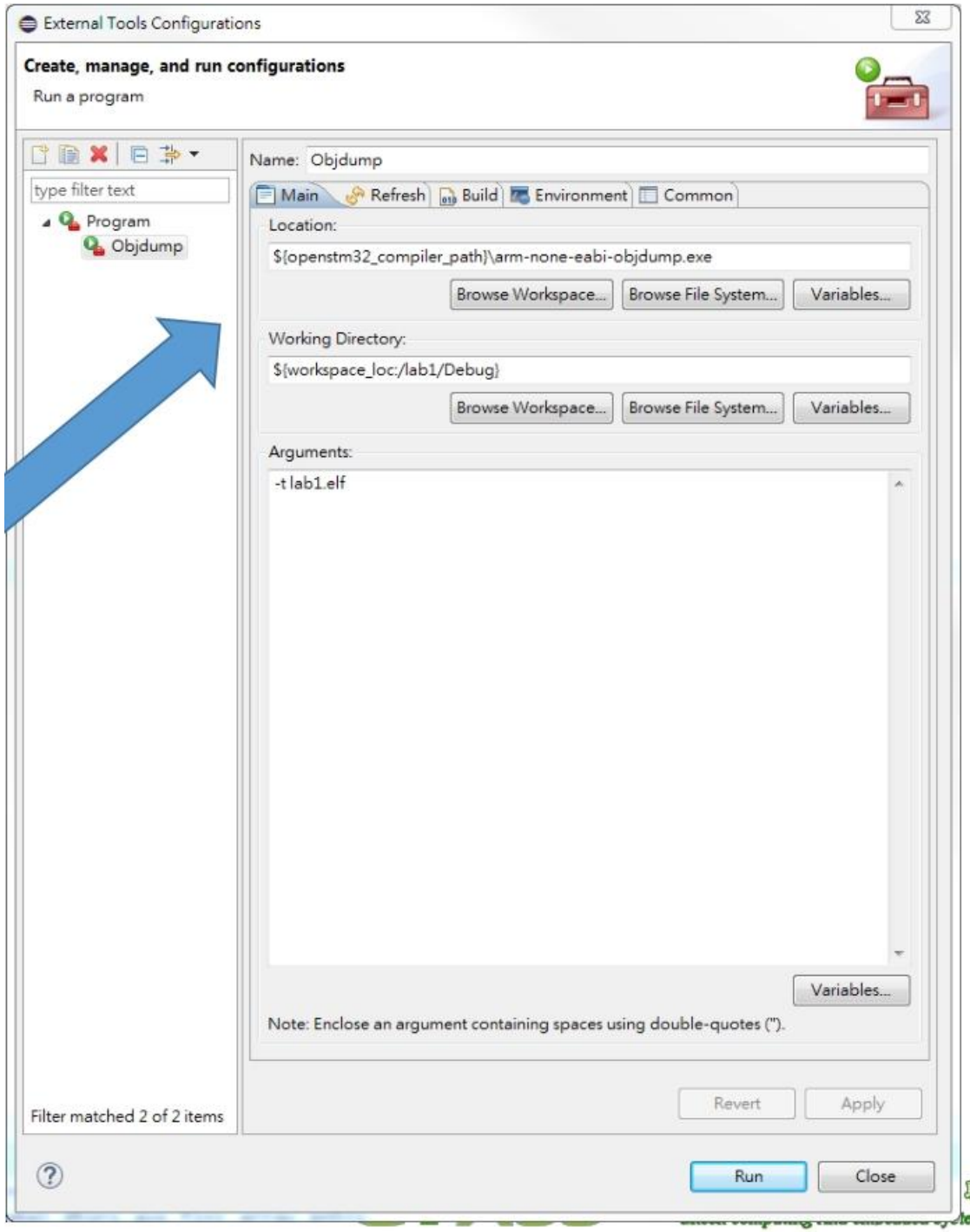
Debug Guide

- By default the GDB will set the first breakpoint at main
- Press Step into button or F5 will debug your code step by step
- You can use IDE to inspect registers, I/O registers, memory on board

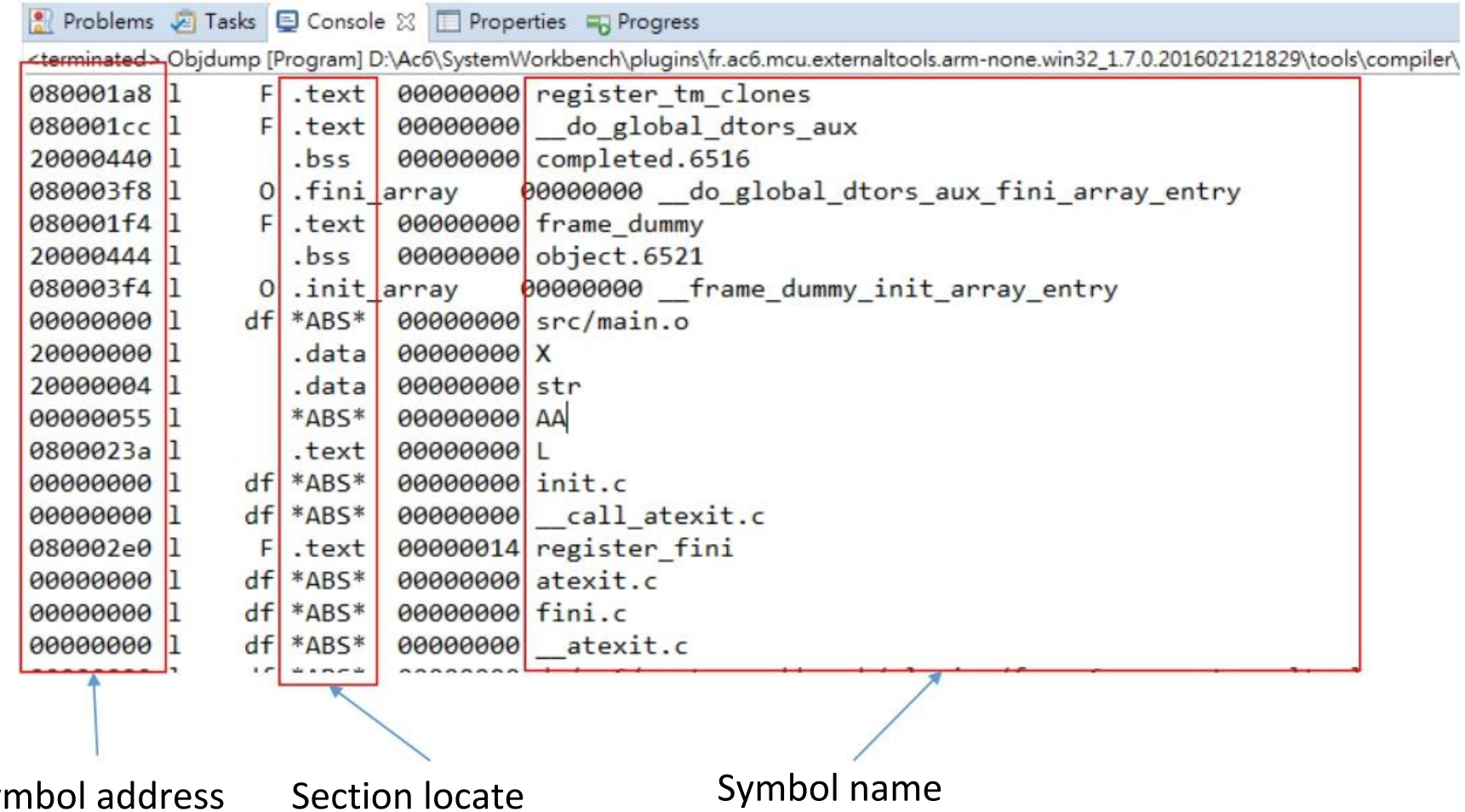


Object Dump

- This tool can help you show the program's symbol table
- Run -> External Tool -> External Tool Configurations
  - Set a new program Objdump with the same settings
- Objdump usage guide
  - https://sourceware.org/binutils/docs/binutils/objdump.html



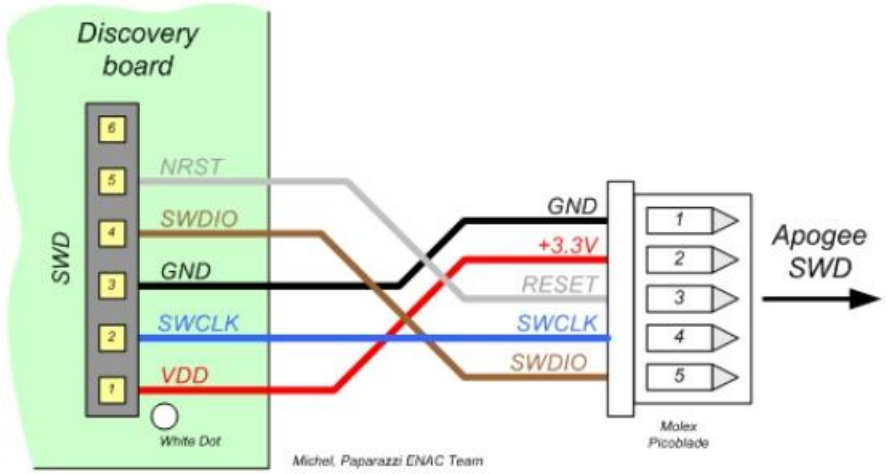
Symbol Table

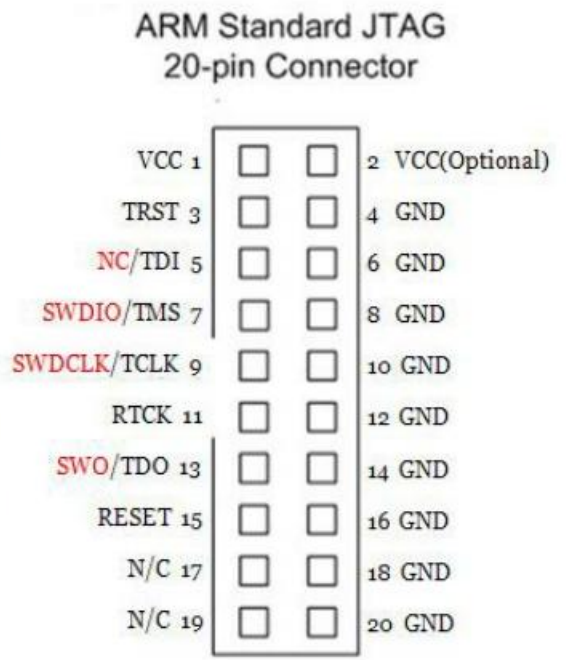


Debug architecture

Debug Interface

- JTAG(Joint Test Action Group)
  - A standard ASICs hardware debug interface
- SWD(Serial Wire Debug)
  - Only use 5 wires from part of JTAG interface

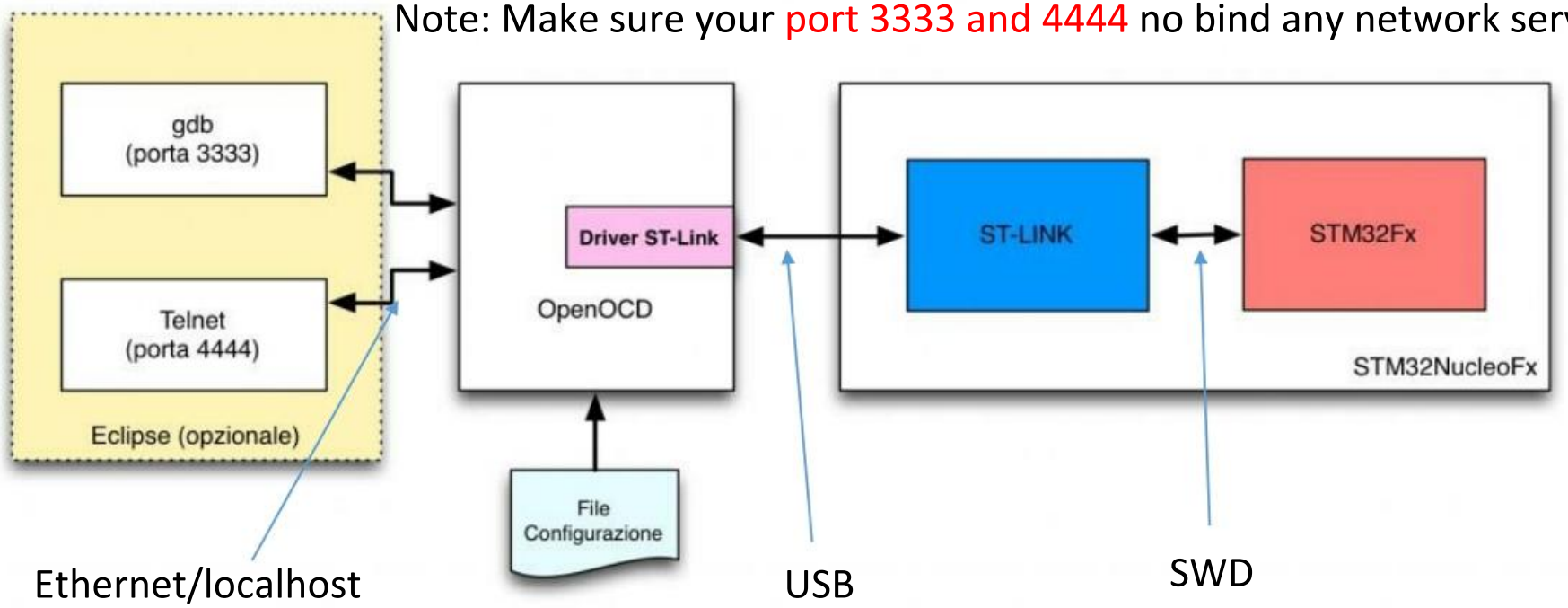




Debug on board

- ST-Link: A STM32 hardware flasher and debugger
- OpenOCD: An open source GDB server

Note: Make sure your port 3333 and 4444 no bind any network service!



Memory Guide

Memory Access

- Define data variable
- Direct access
- Indirect read access

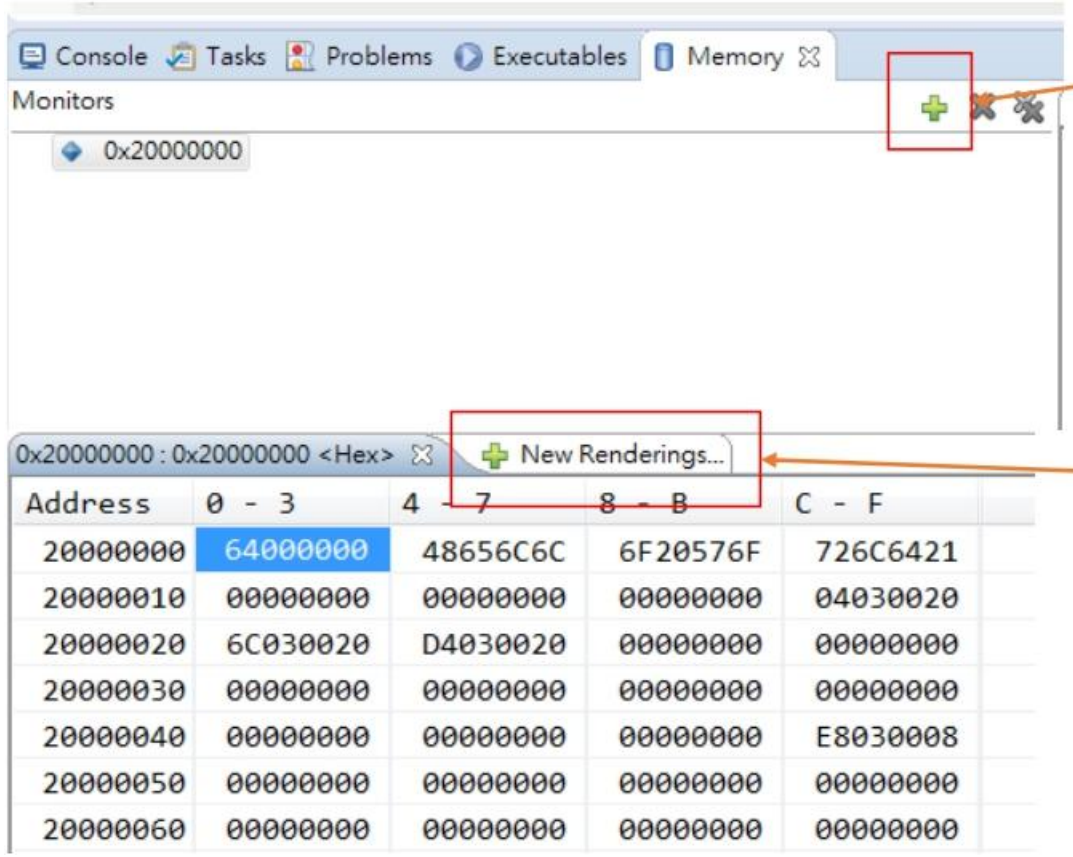
```
1   .syntax unified
2   .cpu cortex-m4
3   .thumb
4
5   .data
6       X: .word 100
7       Y: .asciz "Hello World!"
8   .text
9   .global main
10  .equ AA, 0x55
11
12  main:
13      ldr r1, =X
14      ldr r0, [r1]
15      movs r2, #AA
16      adds r2, r2, r0
17      str  r2, [r1]
18
19      ldr  r1, =Y
20      ldr  r2, [r1]
21
22  L:  B L
```

Data section start point

Write the data register into memory

Memory Monitors

- That can help you watch the memory content



Press it to add a memory monitor

Press “New Renderings” can change the display format

Practice
練習

Practice 1

Create a STM32 eclipse project according to Getting started guide. Add a "main.s" code as right and observe program execution result through debugger.

請依照 Getting started guide，建立一個 STM32 eclipse project，新增一個程式碼如右的 main.s 並透過 debugger 觀察程式執行結果。

Q: What is the R2 value after the program is executed ? How to observe ?

問: 程式執行結束後 R2 值為多少？如何觀察？

```
.syntax unified
.cpu cortex-m4
.thumb

.text
.global main
.equ AA, 0x55

main:
   movs r0, #AA
   movs r1, #20
   adds r2, r0, r1

L: B L
```

Practice 2

Q1: Where is the initial value of the variables X and str initialized by whom?

問1: 變數 X 與 Y 的初始值是由誰在何處初始化的？

Q2: What happens the program execution result if I change the X declaration to the text section?

問2: 若將 X 宣告改在 text section 對其程式執行結果會有何改變？

Q3: What is the difference between the r2 content and the str string in the first 4 bytes of memory after the program is executed?

問3: 執行完畢後 r2 內容與 Y 字串在 memory 前4個byte呈現內容有何差異？

Q4: The variable str "Hello World!" Is there any other way to declare? If there is one, please explain one of them.

問4: 變數 Y “Hello World!” 有無其他種宣告方式？

```
.syntax unified
.cpu cortex-m4
.thumb

.data
    X: .word 100
    Y: .asciz "Hello
World!"
.text
.global main
.equ AA, 0x55

main:
    ldr r1, =X
    ldr r0, [r1]
    movs r2, #AA
    adds r2, r2, r0
    str  r2, [r1]

    ldr  r1, =Y
    ldr  r2, [r1]

L:  B L
```

Practice 3

This part of the practice requires students to declare three X, Y, and Z variables of length 4 bytes in the data section and calculate the following formula using the ARM assembly language. Find the memory address of these variables and observe the program execution results.

這部分練習需要在 data section 中宣告三個 X, Y, Z 長度為 4 byte 的變數並利用 ARM 組合語言計算以下式子。找出這些變數的memory address並觀察程式執行結果。

$$X = 5$$

$$Y = 10$$

$$X = X * 10 + Y$$

$$Z = Y - X$$

Appendix

Reference

- Getting started with STM32 Nucleo board software development tools
  - http://www.st.com/content/ccc/resource/technical/document/user_manual/1b/03/1b/b4/88/20/4e/cd/DM00105928.pdf/files/DM00105928.pdf/jcr:content/translations/en.DM00105928.pdf

- STM32 Nucleo-64 boards user manual
  - https://www.st.com/content/ccc/resource/technical/document/user_manual/1b/03/1b/b4/88/20/4e/cd/DM00105928.pdf/files/DM00105928.pdf/jcr:content/translations/en.DM00105928.pdf

Linker Script

- https://www.math.utah.edu/docs/info/ld_toc.html#SEC4