

MPSL2020

Lab0

Lab Hardware

STM32 Nucleo Board L476RG

- STM32L476RG
- An ARM Cortex-M4 development board
- Build in a ST-LINK as debugger
- Arduino pin compatible
- One user button (B1)
- One LED (LD2)



Hardware Block

Figure 3. Top layout

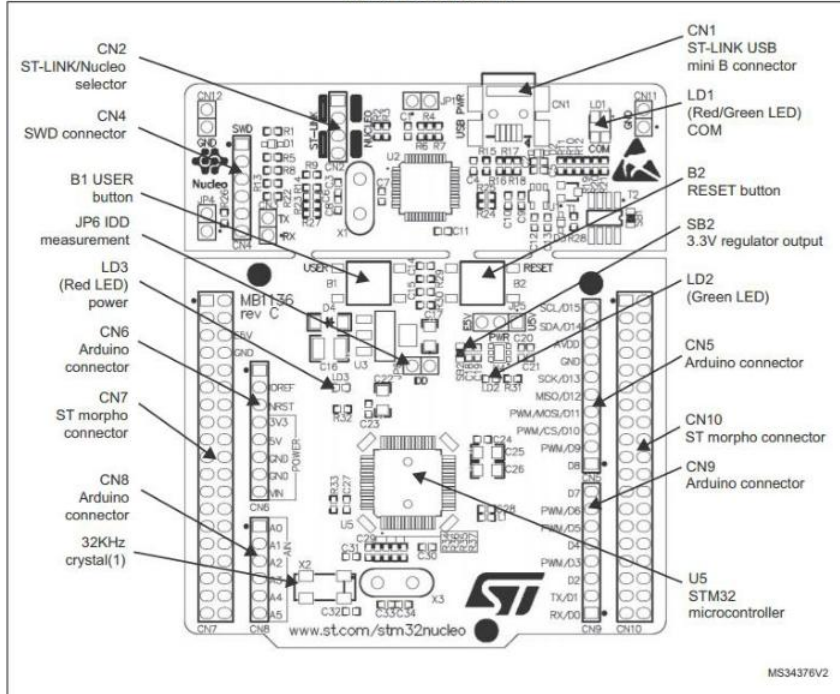
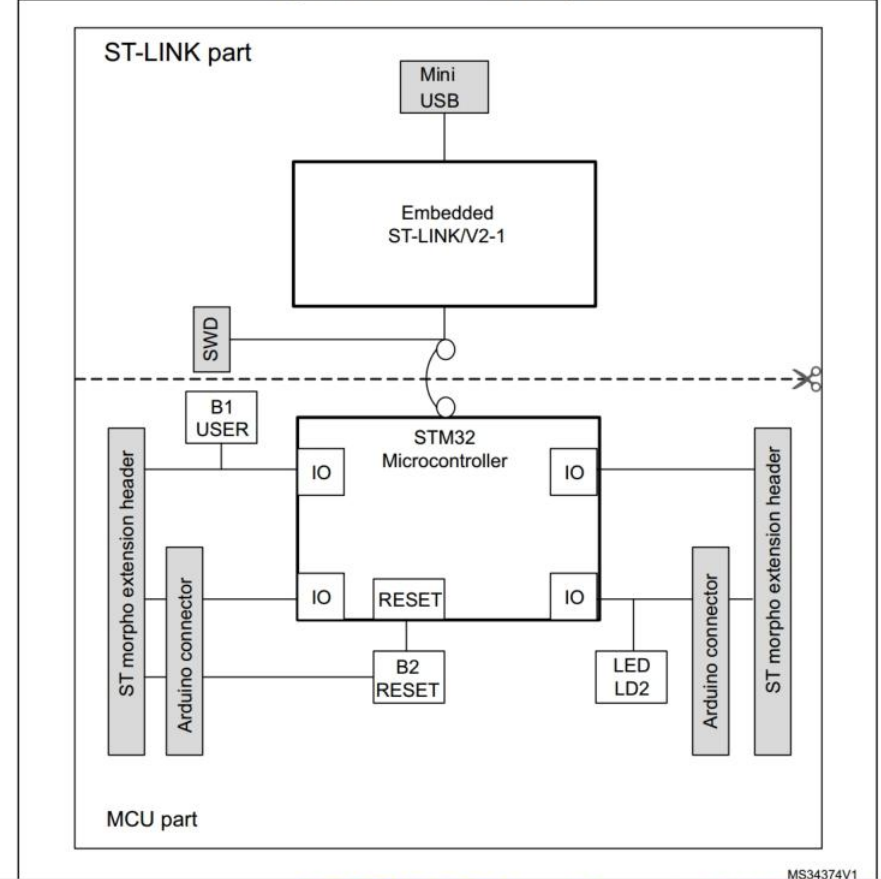
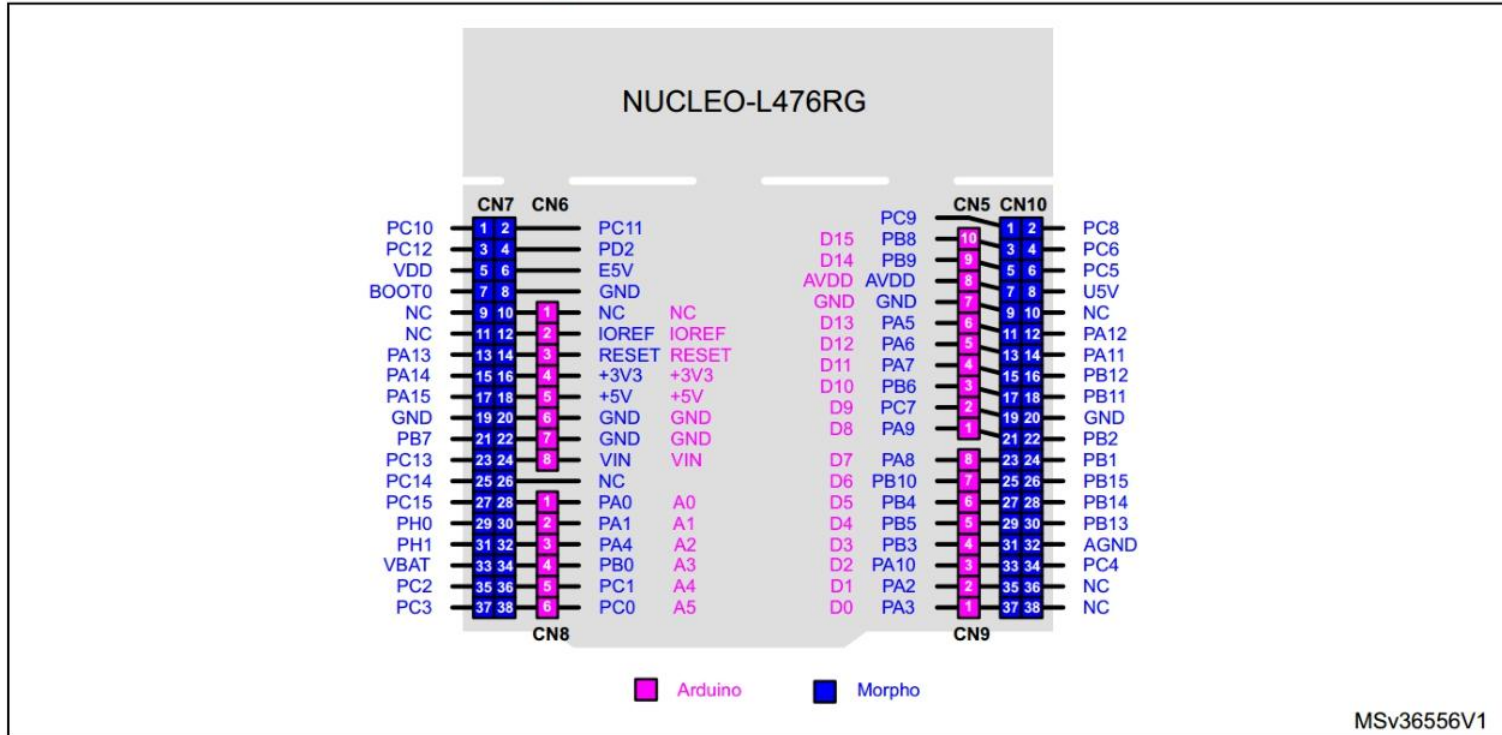


Figure 2. Hardware block diagram



Pin Map

Figure 22. NUCLEO-L476RG



More Details

- User Manual
 - https://www.st.com/resource/en/user_manual/dm00105823-stm32-nucleo-64-boards-mb1136-stmicroelectronics.pdf

Lab Software

Development Environment

- We use **SW4STM32(AC6)** 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 programming tools
 - <http://www.openstm32.org/HomePage>

SW4STM32 IDE

- SW4STM32 Installation Guide

- <http://www.openstm32.org/Downloading+the+System+Workbench+for+STM32+installer>

- 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
- **Warning**
 - You may need some dependency on 64-bit system
 - libc6:i386, lib32ncurses5

- MacOS

- http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_macos_64bits-latest.run

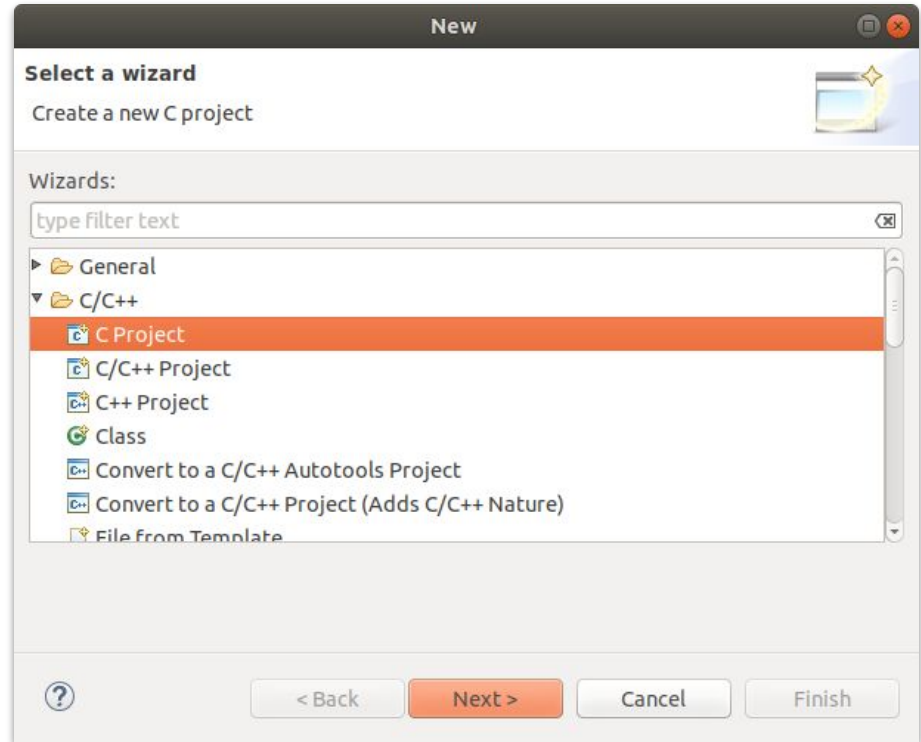
Install note for Linux & MacOS

- Open a terminal session and set permission to executable
e.g. \$ *chmod +x install_sw4stm32_linux_64bits-latest.run*
e.g. \$ *chmod +x install_sw4stm32_macos_64bits-latest.run*
- Execute the installer by running the installation file.
e.g. \$ *./install_sw4stm32_linux_64bits-latest.run*
e.g. \$ *./install_sw4stm32_macos_64bits-latest.run*

SW4STM32 Quick Start

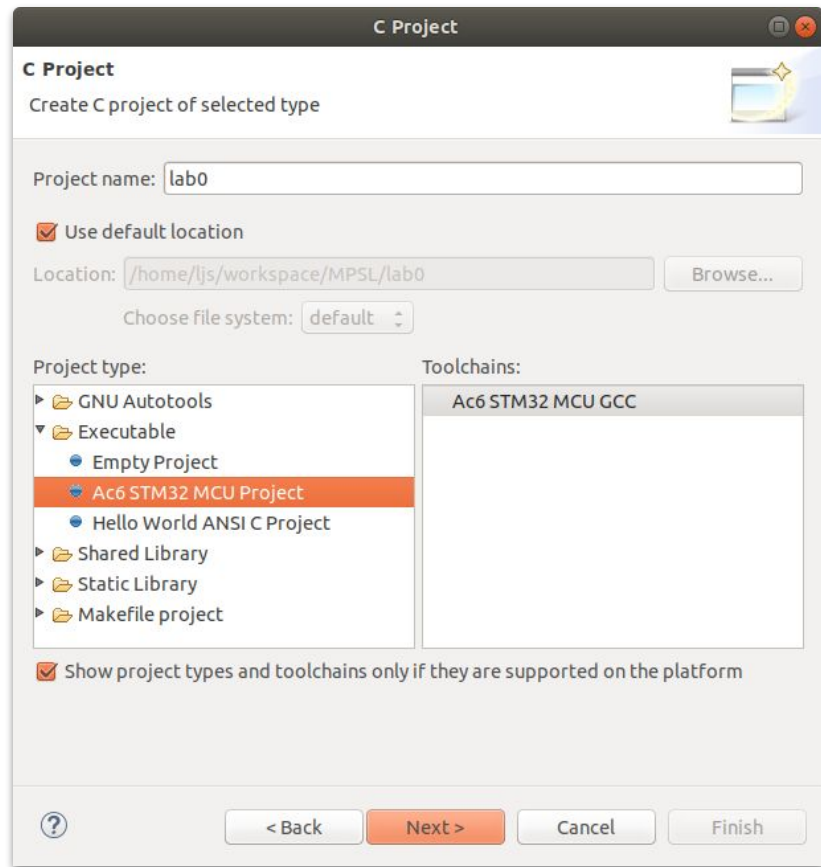
Create Project

- File → New → Other
- (Or File → New → C Project)



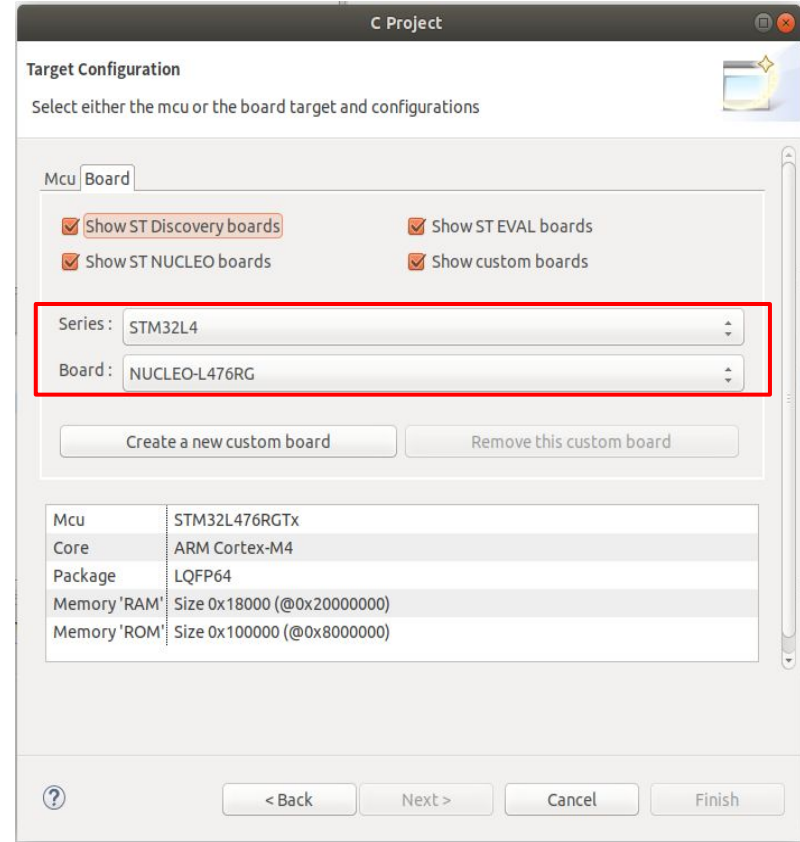
Create Project (cont.)

- Project Name
 - Up to you
- Project Type
 - Ac6 STM32 MCU Project



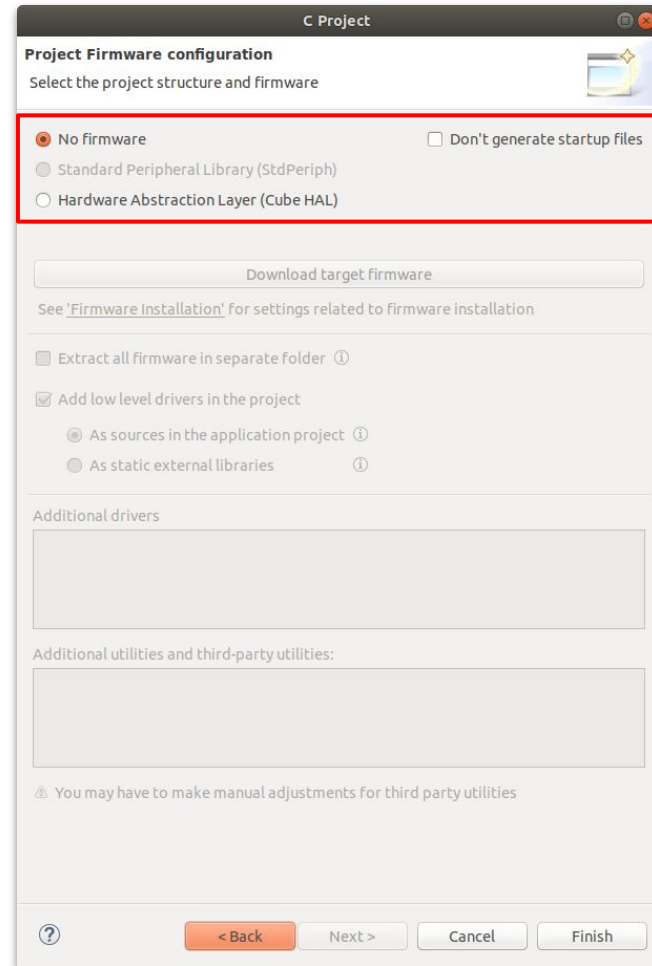
MCU Configuration

- Series
 - STM32L4
- Board
 - NUCLEO-L476RG



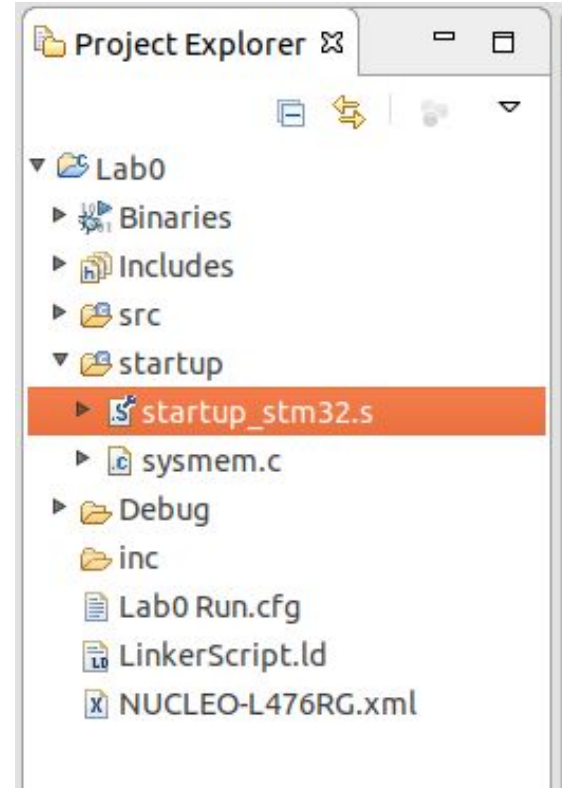
MCU Configuration

- Select
 - No firmware



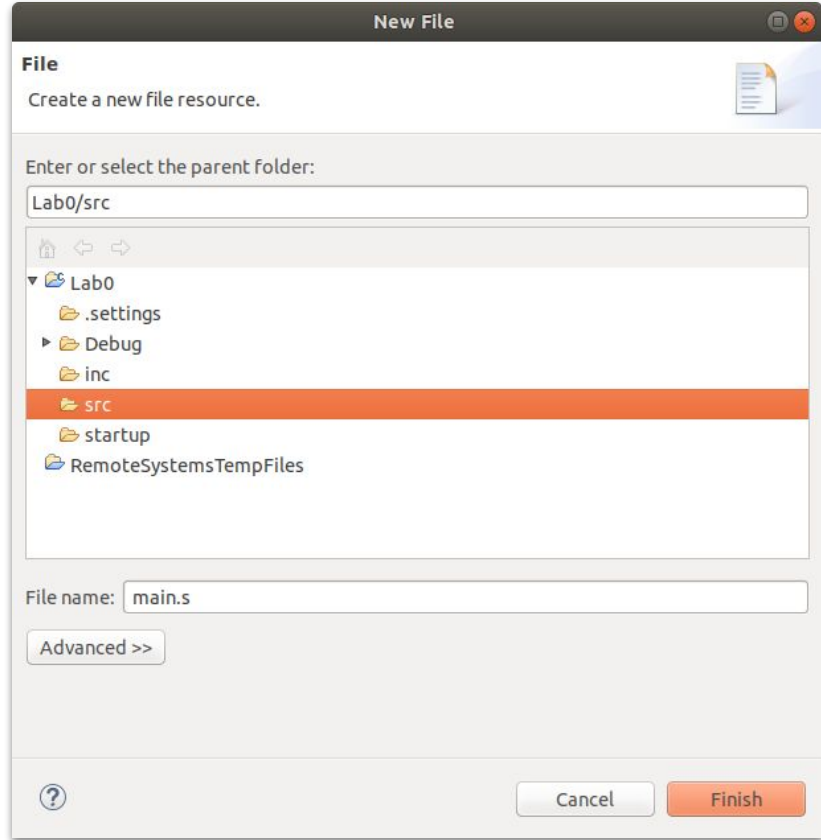
Project Files

- Then you can see the project in your Project Explorer list
- It contain the default startup(boot) code and linker script
 - **startup_stm32.s**
 - **LinkerScript.ld**

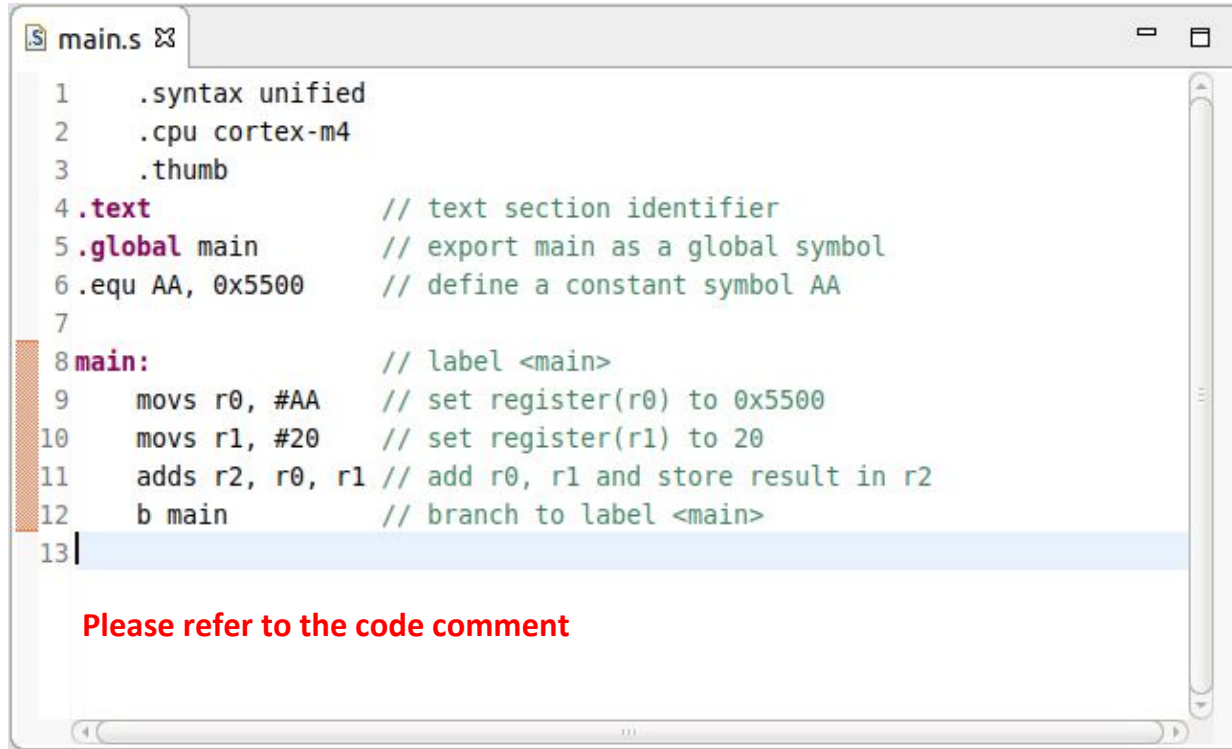


Create File

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




Write Your First Code



```
1  .syntax unified
2  .cpu cortex-m4
3  .thumb
4  .text           // text section identifier
5  .global main    // export main as a global symbol
6  .equ AA, 0x5500 // define a constant symbol AA
7
8  main:           // label <main>
9      movs r0, #AA // set register(r0) to 0x5500
10     movs r1, #20  // set register(r1) to 20
11     adds r2, r0, r1 // add r0, r1 and store result in r2
12     b main        // branch to label <main>
13
```

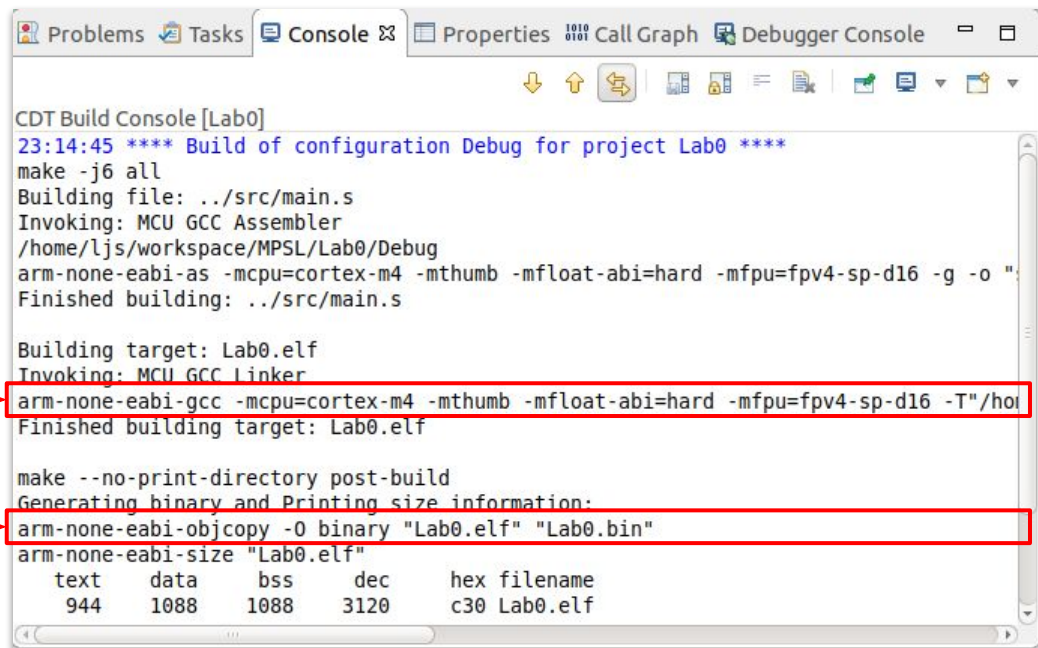
Please refer to the code comment

Build Code

- Project->Build all (Ctrl+B) or Press  button

Compiling

Generating target
binary




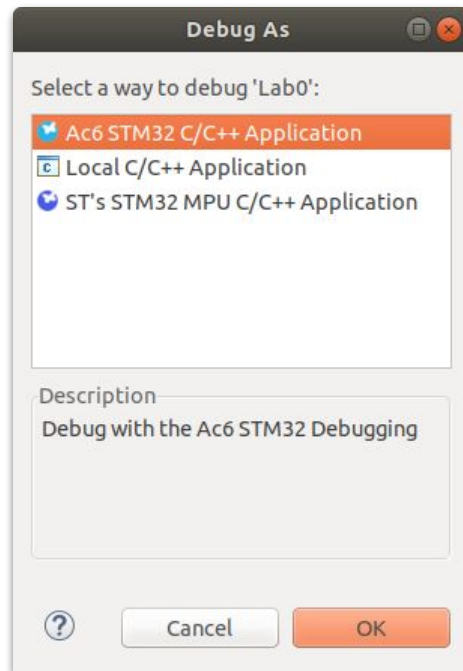
```
CDT Build Console [Lab0]
23:14:45 **** Build of configuration Debug for project Lab0 ****
make -j6 all
Building file: ../src/main.s
Invoking: MCU GCC Assembler
/home/ljs/workspace/MPSL/Lab0/Debug
arm-none-eabi-as -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpv4-sp-d16 -g -o ".
Finished building: ../src/main.s

Building target: Lab0.elf
Invoking: MCU GCC Linker
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpv4-sp-d16 -T"/ho
Finished building target: Lab0.elf

make --no-print-directory post-build
Generating binary and Printing size information:
arm-none-eabi-objcopy -O binary "Lab0.elf" "Lab0.bin"
arm-none-eabi-size "Lab0.elf"
text    data    bss     dec     hex filename
944     1088     1088    3120    c30 Lab0.elf
```





Debug your Code on board

- Run->Debug(F11)
or press  button
- Debug As
 - AC6 STM32 C/C++ Application



Debug Guide

Break points

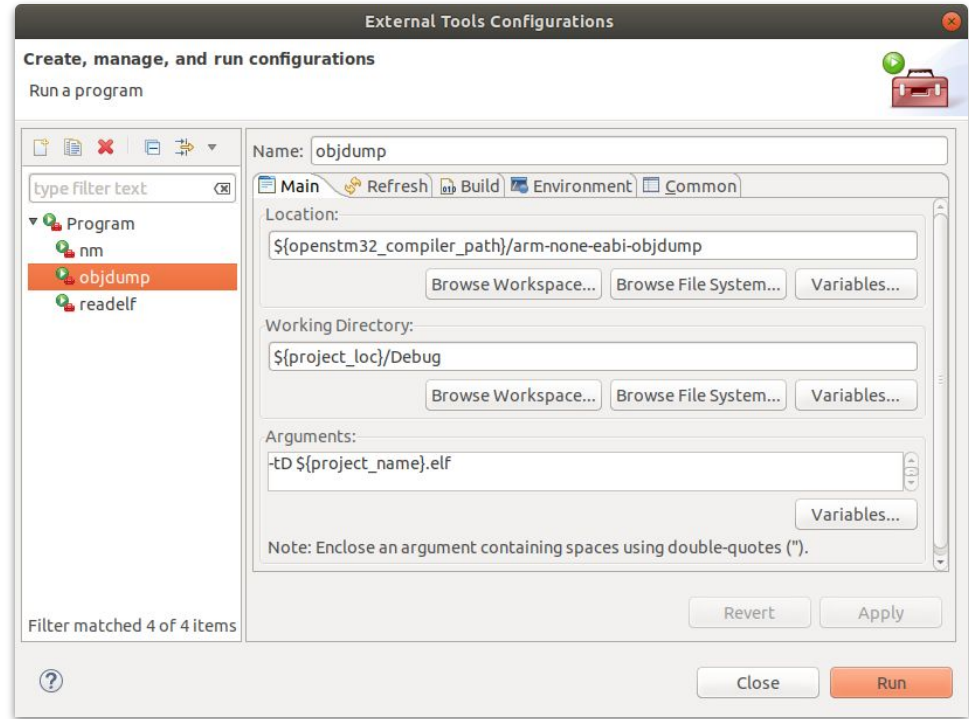
-  Step Into (F5)
 - will stop at the first line of subroutine
-  Step Over (F6)
 - will stop at next line of current routine
-  Step Return (F7)
 - will directly return to parent routine
-  Resume (F8)
 - Stop at next breakpoint

Memory Browser

- Add memory browser to debug layouts (**Not** in the default layout)
 - Window -> Show View ->Memory Browser
- To change memory rendering
 - Right-click to check the menu

External Tools

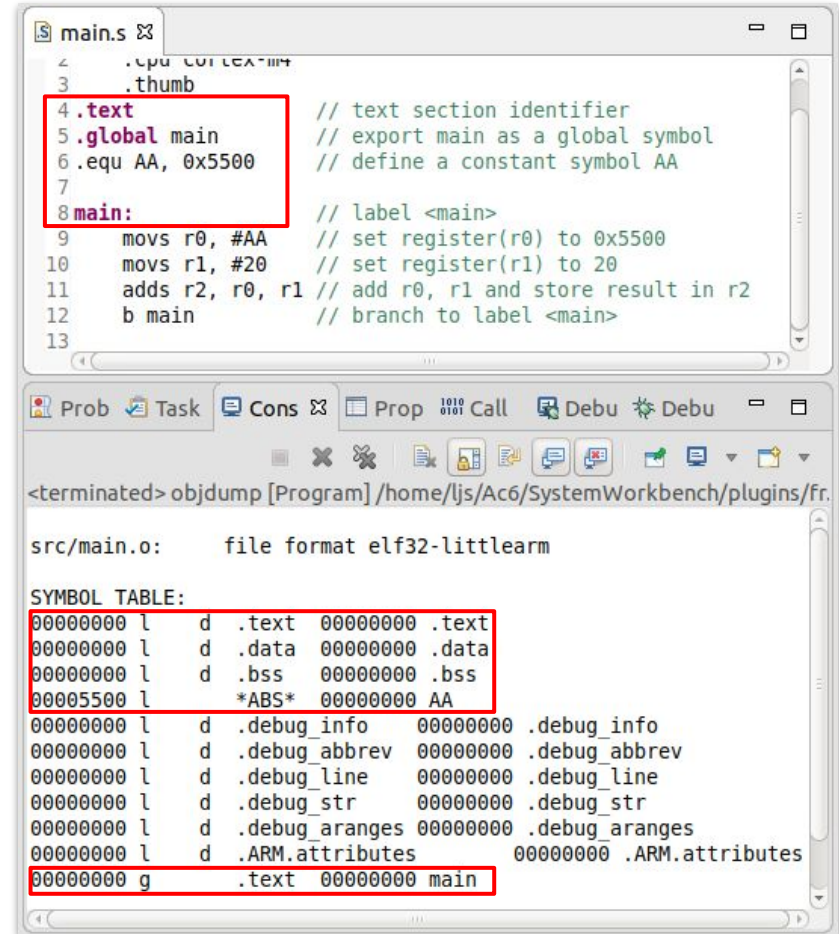
- Run -> External Tools -> External Tools Configuration
- Program -> New
- Location
 - Just copy from debug configuration and revise `gdb` to `objdump`
- Working Directory
 - `${project_loc}/Debug`
- Arguments
 - `-tD ${project_name}.elf`
 - You can change `-D t` to any option you want
- Note
 - Here we use the dynamic variable. Thus, **you have to select the target project** at the project explorer before press the button.



Do not press run directly or it will jump out error message

External Tools (cont.)

- Use **objdump** with **[-t|--syms]** flag to observe the symbol table of `Debug/src/main.o`
- Note: If you follow the setup in previous slide you might see tons of symbols defined by `startup_stm32.s`



The screenshot shows an IDE with two windows. The top window, titled 'main.s', displays assembly code for an ARM Cortex-M4. Lines 4 through 8 are highlighted with a red box: `4 .text`, `5 .global main`, `6 .equ AA, 0x5500`, and `8 main:`. The bottom window shows the output of the `objdump` command, displaying the symbol table for `src/main.o`. The symbol table lists various sections and symbols, with the first four entries highlighted by a red box: `00000000 l d .text 00000000 .text`, `00000000 l d .data 00000000 .data`, `00000000 l d .bss 00000000 .bss`, and `00005500 l *ABS* 00000000 AA`. The last entry, `00000000 g .text 00000000 main`, is also highlighted.

```
main.s
4 .text
5 .global main
6 .equ AA, 0x5500
7
8 main:
9     movs r0, #AA
10    movs r1, #20
11    adds r2, r0, r1
12    b main
13
```

```
<terminated> objdump [Program] /home/ljs/Ac6/SystemWorkbench/plugins/fr.
src/main.o:      file format elf32-littlearm

SYMBOL TABLE:
00000000 l d .text 00000000 .text
00000000 l d .data 00000000 .data
00000000 l d .bss 00000000 .bss
00005500 l *ABS* 00000000 AA
00000000 l d .debug_info 00000000 .debug_info
00000000 l d .debug_abbrev 00000000 .debug_abbrev
00000000 l d .debug_line 00000000 .debug_line
00000000 l d .debug_str 00000000 .debug_str
00000000 l d .debug_aranges 00000000 .debug_aranges
00000000 l d .ARM.attributes 00000000 .ARM.attributes
00000000 g .text 00000000 main
```

External Tools (cont.)

- Objdump Man Page
<https://linux.die.net/man/1/objdump>
 - To find the column information
 - Ctrl + F : **--syms**

```
<terminated> objdump [Program] /home/ljs/Ac6/SystemWorkbench/plugins/fr.  
  
src/main.o:      file format elf32-littlearm  
  
SYMBOL TABLE:  
00000000 l d .text 00000000 .text  
00000000 l d .data 00000000 .data  
00000000 l d .bss 00000000 .bss  
00005500 l *ABS* 00000000 AA  
00000000 l d .debug_info 00000000 .debug_info  
00000000 l d .debug_abbrev 00000000 .debug_abbrev  
00000000 l d .debug_line 00000000 .debug_line  
00000000 l d .debug_str 00000000 .debug_str  
00000000 l d .debug_aranges 00000000 .debug_aranges  
00000000 l d .ARM.attributes 00000000 .ARM.attributes  
00000000 g .text 00000000 main
```

value (or address)

section_identifier

symbol_name

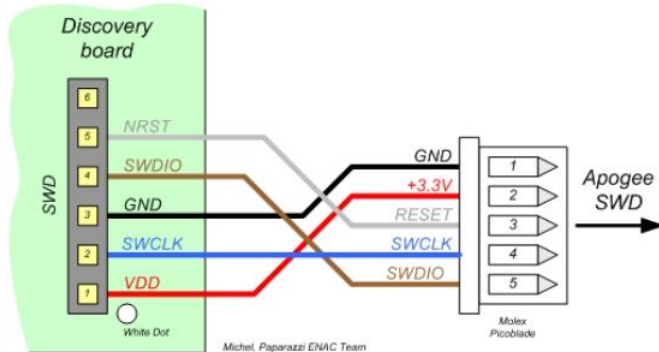
More Details

- SW4STM3 User Guide
 - <https://www.openstm32.org/User%2BGuide>

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 interf

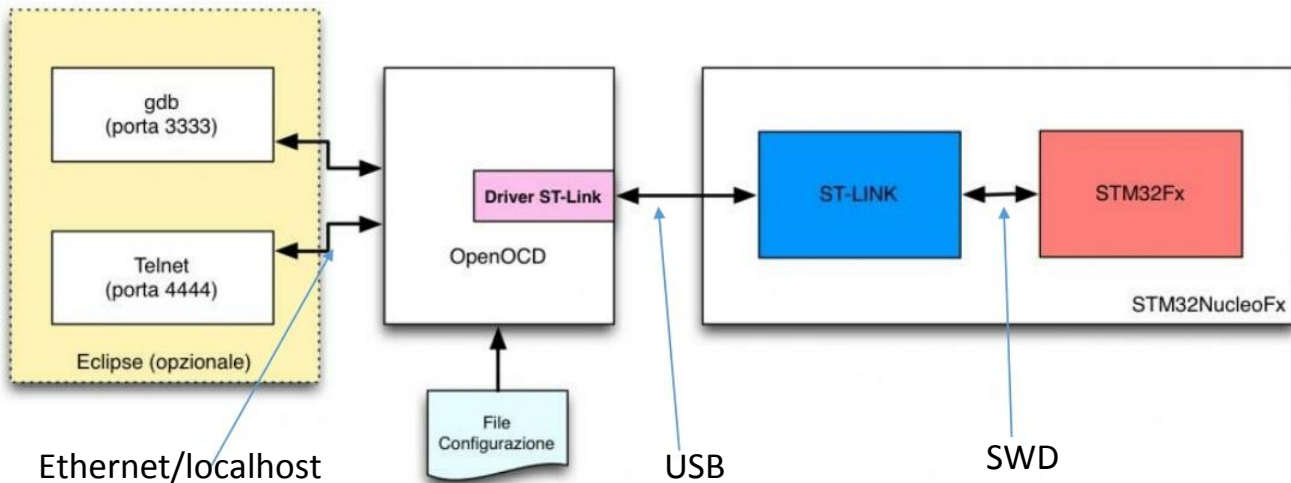


ARM Standard JTAG
20-pin Connector

VCC	1			2	VCC(Optional)
TRST	3			4	GND
NC/TDI	5			6	GND
SWDIO/TMS	7			8	GND
SWDCLK/TCLK	9			10	GND
RTCK	11			12	GND
SWO/TDO	13			14	GND
RESET	15			16	GND
N/C	17			18	GND
N/C	19			20	GND

Debug on board

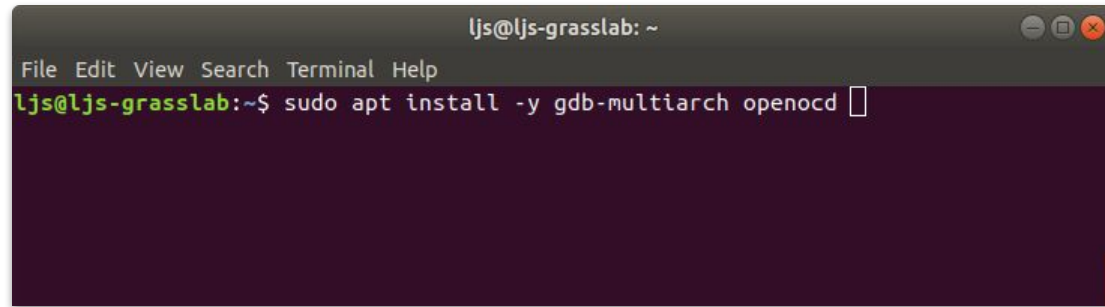
- ST-Link: A STM32 hardware flasher and debugger
- OpenOCD: An open source GDB server
- Note: Make sure your **port 3333 and 4444** no bind any network service!



Appendix (debug without IDE)

Install required toolkits

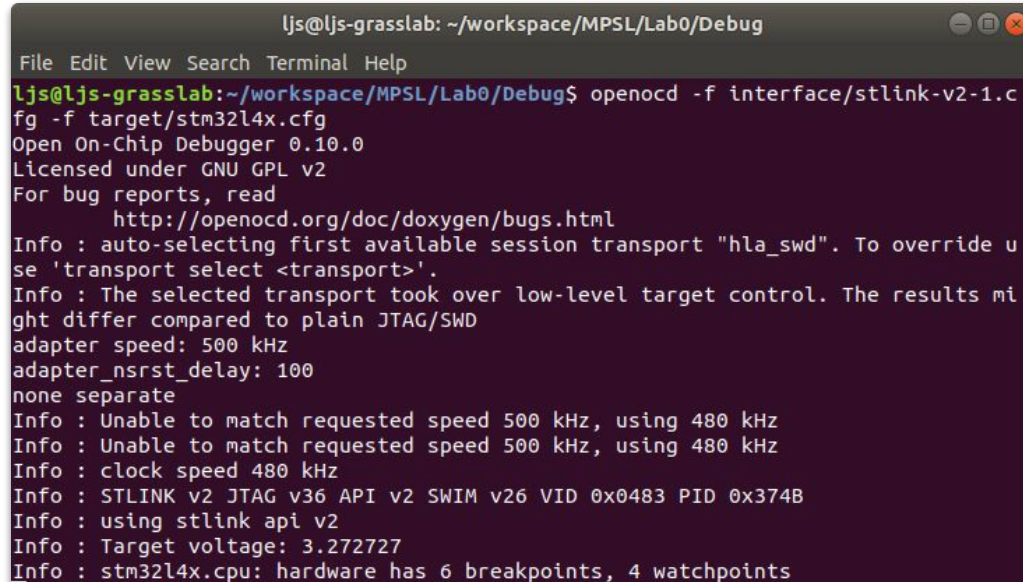
- Required toolkits
 - gdb-multiarch
 - `$ sudo apt install gdb-multiarch`
 - openOCD
 - `$ sudo apt install openocd`

A terminal window with a dark purple background and a grey title bar. The title bar contains the text 'ljs@ljs-grasslab: ~' and three window control buttons (minimize, maximize, close). The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The command prompt shows 'ljs@ljs-grasslab:~\$' followed by the command 'sudo apt install -y gdb-multiarch openocd' and a cursor. The terminal is currently empty, indicating the command has been entered but not yet executed.

```
ljs@ljs-grasslab: ~  
File Edit View Search Terminal Help  
ljs@ljs-grasslab:~$ sudo apt install -y gdb-multiarch openocd
```

Start an openocd program

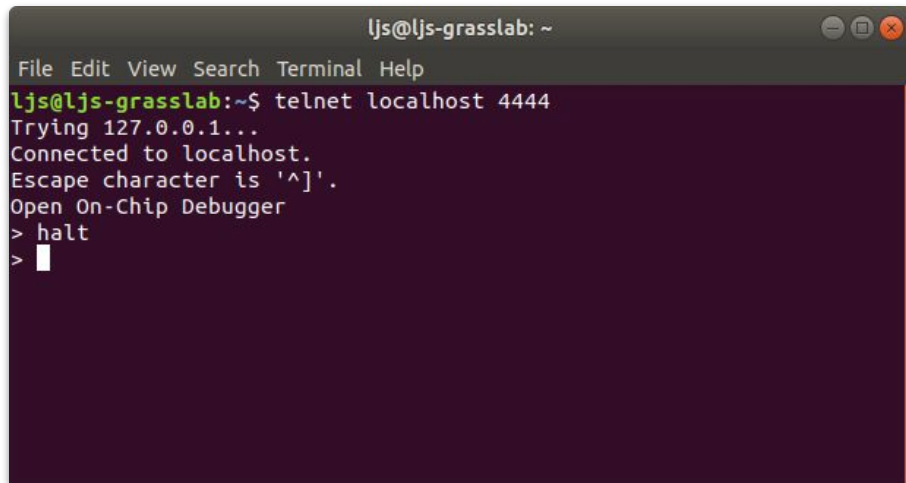
- Start an openocd program and pass two configuration file that describe the JTAG programmer with flag "-f"
 - `$ openocd -f interface/stlink-v2-1.cfg -f target/stm32l4x.cfg &`
 - "&" will make this process running in the background
- You can check these two file under the following path
 - `/usr/share/openocd/scripts/`
- Use `lsusb` to check the usb ID is equal to that in the `stlink-v2-1.cfg`.
 - For me, it is `0483:374b`

A terminal window titled 'ljs@ljs-grasslab: ~/workspace/MPSL/Lab0/Debug' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'openocd -f interface/stlink-v2-1.cfg -f target/stm32l4x.cfg' being executed. The output includes version information (0.10.0), license (GNU GPL v2), a link to bug reports, and detailed status messages about session transport selection (hla_swd), adapter speed (500 kHz), and target identification (STLINK v2 JTAG v36 API v2 SWIM v26, VID 0x0483, PID 0x374B).

```
ljs@ljs-grasslab: ~/workspace/MPSL/Lab0/Debug
File Edit View Search Terminal Help
ljs@ljs-grasslab:~/workspace/MPSL/Lab0/Debug$ openocd -f interface/stlink-v2-1.c
fg -f target/stm32l4x.cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "hla_swd". To override u
se 'transport select <transport>'.
Info : The selected transport took over low-level target control. The results mi
ght differ compared to plain JTAG/SWD
adapter speed: 500 kHz
adapter_nsrst_delay: 100
none separate
Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : clock speed 480 kHz
Info : STLINK v2 JTAG v36 API v2 SWIM v26 VID 0x0483 PID 0x374B
Info : using stlink api v2
Info : Target voltage: 3.272727
Info : stm32l4x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Connect to OpenOCD

- After start openocd, by default it will create two socket with port number 3333 and 4444. The former is for user connection and the later is for gdb connection.
- Now we can connect to openocd by "telnet"
 - `$ telnet localhost 4444`

A terminal window titled 'ljs@ljs-grasslab: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows a telnet session to localhost 4444. The output includes the connection attempt, confirmation of connection, escape character, and the Open On-Chip Debugger prompt. The user has entered 'halt' and is waiting for a response.

```
ljs@ljs-grasslab: ~  
File Edit View Search Terminal Help  
ljs@ljs-grasslab:~$ telnet localhost 4444  
Trying 127.0.0.1...  
Connected to localhost.  
Escape character is '^]'.  
Open On-Chip Debugger  
> halt  
> 
```

Download Image into device

- After connect to openOCD, we can
 - halt our device by "> halt"
 - and copy our image by "> flash write_image erase Labx.bin"
 - then reset our device by "> reset"

```
> pwd
/usr/share/openocd/scripts/interface
> flash write_image erase path_to_image.bin
```


Connect by GDB(cont.)

- To set a breakpoint
 - (gdb) b <line_number>
- To continue process
 - (gdb) c
- To dump register content
 - (gdb) info reg
 - (gdb) p \$<reg_name>
- To dump memory content
 - (gdb) x /<nr_words> <address>

Connect by GDB(cont.)

- You can also use python extension to get a more convenience layout
- GDB Enhanced Feature
 - <https://gef.readthedocs.io/en/master/>

```
ljs@ljs-grasslab: ~
File Edit View Search Terminal Help
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
$0 : 0x00005500 → 0x603b2300 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$1 : 0x00000014 → 0x00000225 → 0x700000e7 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$2 : 0x00005514 → 0xf7ffff49 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$3 : 0x000001ad → 0x1b4b03b5 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$4 : 0x20000470 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$5 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$6 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$7 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$8 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$9 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$r10 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$r11 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$r12 : 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$sp : 0x20018000 → 0x00000000 → 0x20018000 → [loop detected]
$lr : 0x0000020b → 0x018000e7 → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
$pc : 0x000001d0 → 0xbffaf7ff → 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]

stack
0x20018000 +0x0000: 0x00000000 → 0x20018000 → [loop detected] ← $sp
0x20018004 +0x0004: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x20018008 +0x0008: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x2001800c +0x000c: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x20018010 +0x0010: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x20018014 +0x0014: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x20018018 +0x0018: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]
0x2001801c +0x001c: 0x00000000 → 0x20018000 → 0x00000000 → [loop detected]

[!] Command 'context' failed to execute properly, reason: unsupported operand type(s) for &: 'NoneType' and 'int'
L () at ./src/main.s:12
12 L: b L // branch to label <main>
gef> 
```