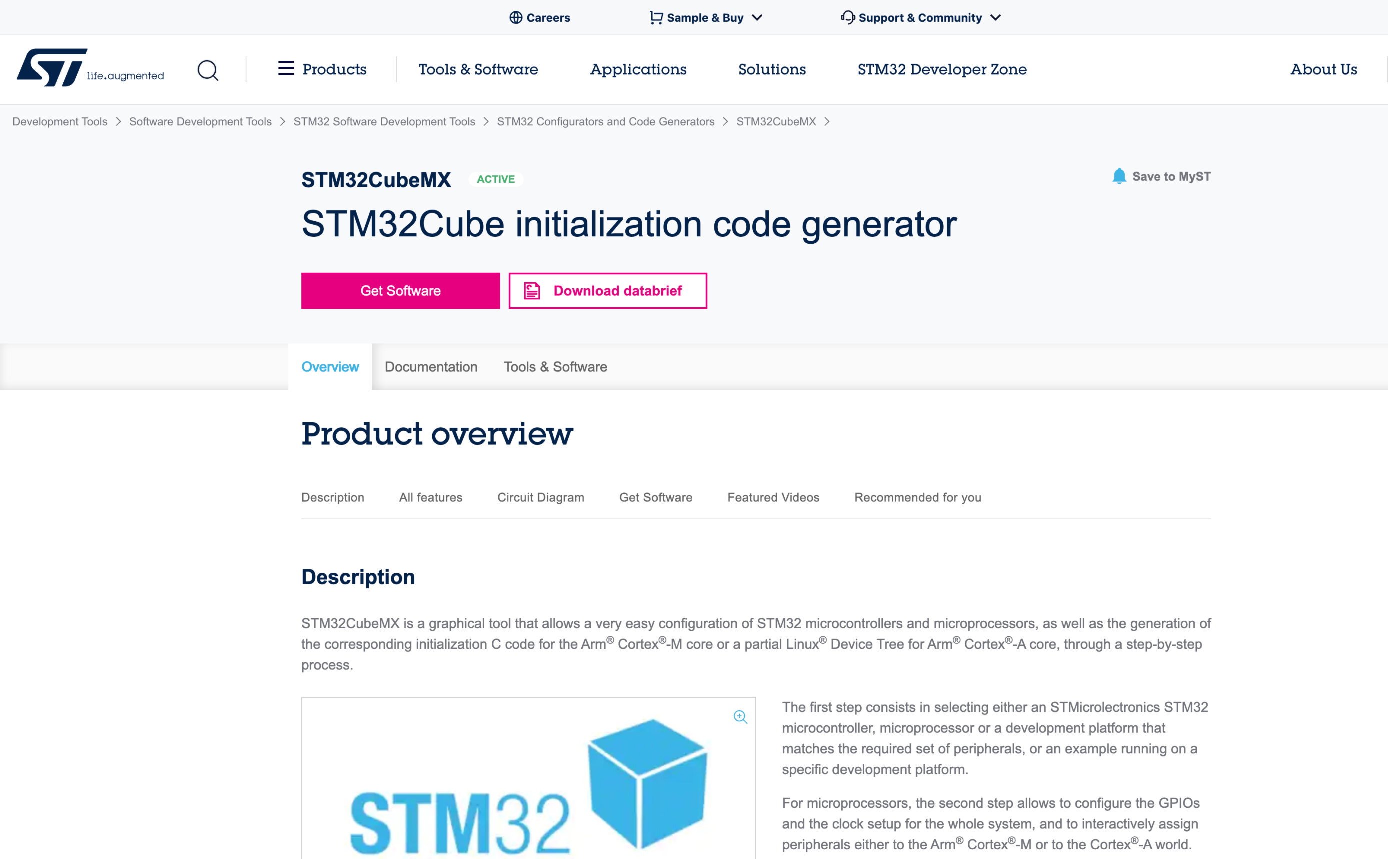
# Building and debugging instructions

## Prerequisites

> For Windows users, make sure the shell you are using is either PowerShell, Git Bash or WSL. The default Windows
> Command Prompt is not supported.

### Project Management

Install stm32cubemx from here



STM32CubeMX ACTIVE

STM32Cube initialization code generator

Product overview

Description

STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code for the Arm® Cortex®-M core or a partial Linux® Device Tree for Arm® Cortex®-A core, through a step-by-step process.

The first step consists in selecting either an STMicrolectronics STM32 microcontroller, microprocessor or a development platform that matches the required set of peripherals, or an example running on a specific development platform.

For microprocessors, the second step allows to configure the GPIOs and the clock setup for the whole system, and to interactively assign peripherals either to the Arm® Cortex®-M or to the Cortex®-A world.

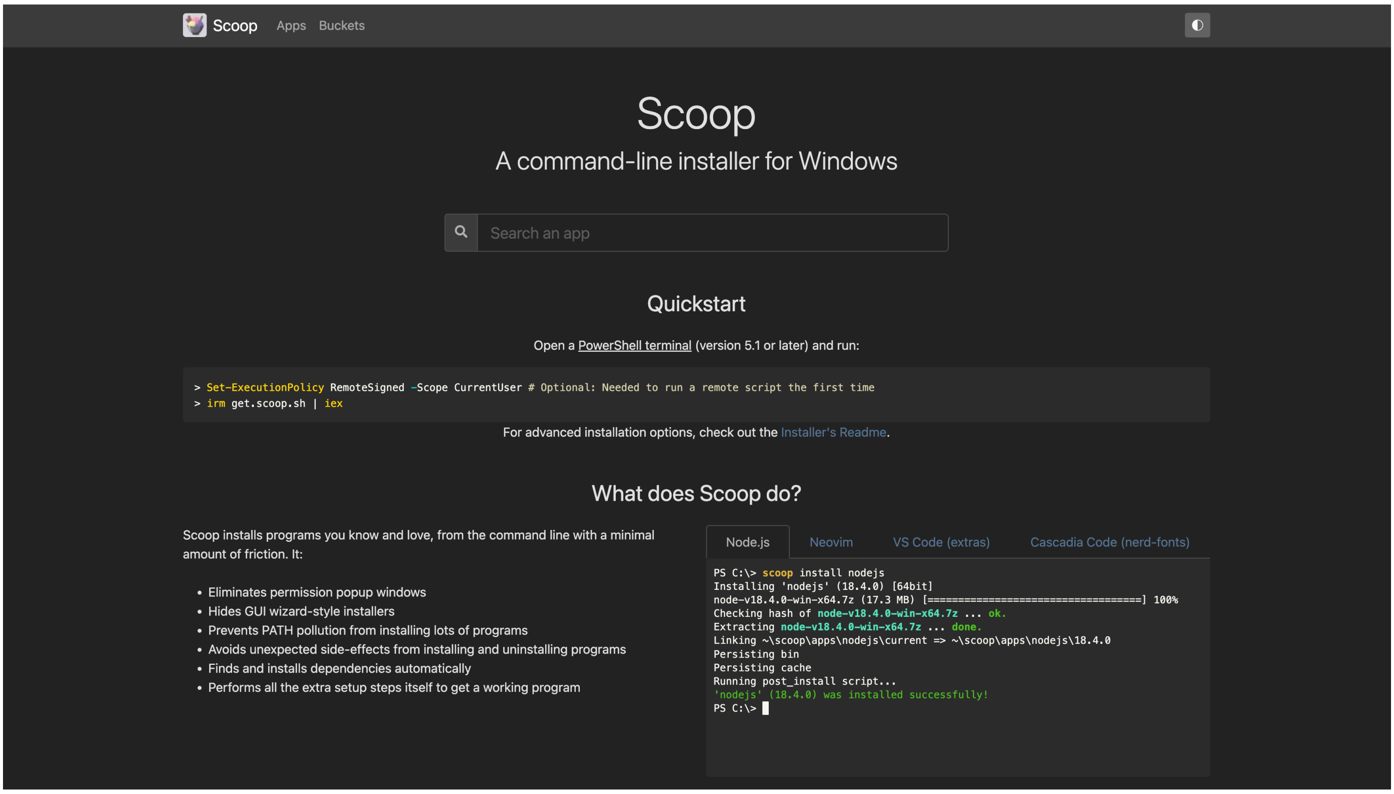
### Building tools

### CMake

The project is built using CMake. To install CMake, follow the instructions below:

Windows (make sure you have scoop installed):

```
scoop install cmake
```



Scoop

A command-line installer for Windows

Quickstart

Open a PowerShell terminal (version 5.1 or later) and run:

```
> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser # Optional: Needed to run a remote script the first time
> irm get.scoop.sh | iex
```

For advanced installation options, check out the Installer's Readme.

What does Scoop do?

Scoop installs programs you know and love, from the command line with a minimal amount of friction. It:

- Eliminates permission popup windows
- Hides GUI wizard-style installers
- Prevents PATH pollution from installing lots of programs
- Avoids unexpected side-effects from installing and uninstalling programs
- Finds and installs dependencies automatically
- Performs all the extra setup steps itself to get a working program

```
PS C:\> scoop install nodejs
Installing 'nodejs' (18.4.0) [64bit]
node-v18.4.0-win-x64.7z (17.3 MB) [=====================================] 100%
Checking hash of node-v18.4.0-win-x64.7z ... ok.
Extracting node-v18.4.0-win-x64.7z ... done.
Linking ~\scoop\apps\nodejs\current => ~\scoop\apps\nodejs\18.4.0
Persisting bin
Persisting cache
Running post_install script...
'nodejs' (18.4.0) was installed successfully!
PS C:\>
```

macOS (make sure you have brew installed):

```
brew install cmake
```

```
 ~/Documents/School/UW/ECE198/ECE198   main !2
brew install cmake
```

## Cross-compilation toolchain

To compile, build and debug the project, you also need to have `arm-none-eabi-gcc` and `openocd` installed with scoop(
and only scoop because this package doesn't exist in the package index of Chocolatey or winget):

### gcc-arm-none-eabi

Windows:

```
scoop bucket add extras
scoop install extras/gcc-arm-none-eabi
```

```
 ~/Documents/School/UW/ECE198/ECE198   main !2 ?1                    ✔  anaconda3   01:42:28
    scoop bucket add extras
scoop install extras/gcc-arm-none-eabi
```

macOS:

> **DON'T INSTALL gcc-arm-none-eabi DIRECTLY VIA BREW SINCE IT RESULTS IN BROKEN DEPENDENCIES**

```
brew install --cask gcc-arm-embedded
```

```
~/Documents/School/UW/ECE198/ECE198  main !2 ?1                ✓  anaconda3  01:43:31
brew install --cask gcc-arm-embedded
```

### openocd

The sourcecode of `openocd` is included as git submodule, so you can build it on your own with the [source code](source code)

openocd uses `make` as its building tool. Use the following command to install these prerequisites:

Windows:

```
scoop install gcc make autoconf automake libtool pkg-config
```

```
~/Documents/School/UW/ECE198/ECE198  main !2 ?1                ✓  anaconda3  01:43:31
scoop install gcc make autoconf automake libtool pkg-config
```

MacOS:

```
brew install automake libtool libusb wget pkg-config
```

Then, build with following command:

```
cd dependencies/openocd-esp32
./bootstrap
./configure
make
make install DESTDIR=$PWD/out
```

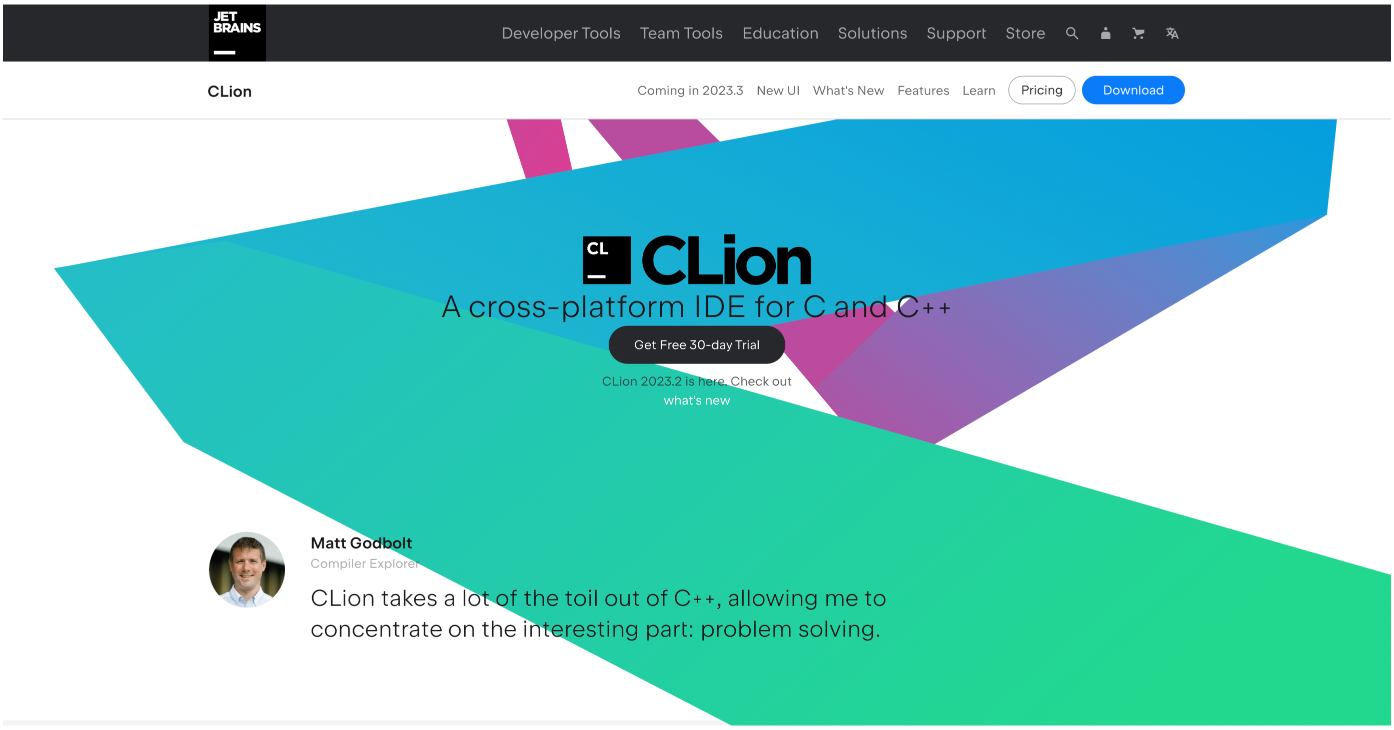
```
~/Documents/School/UW/ECE198/ECE198  main !2 ?1            INT ✘  anaconda3  01:46:32
  cd dependencies/openocd-esp32
./bootstrap
./configure
make
make install DESTDIR=$PWD/out
```

The openocd executable file will be in `dependencies/openocd-esp32/out/usr/local/bin/`, called `openocd`

# Building and debugging

The recommended building platform and IDE is [CLion](CLion).

## CLion



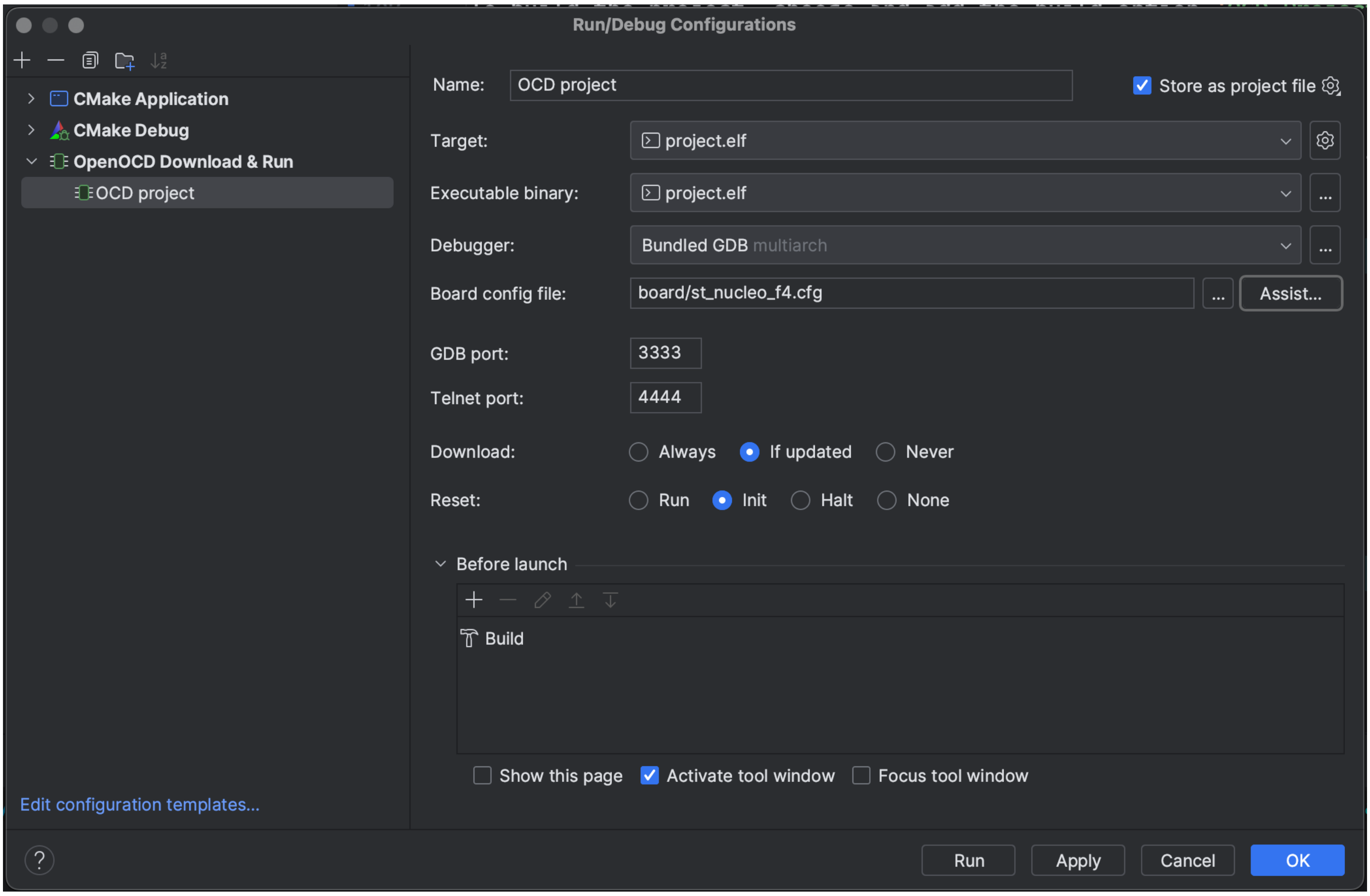
A cross-platform IDE for C and C++

CLion 2023.2 is here. Check out what's new

Matt Godbolt
Compiler Explorer

CLion takes a lot of the toil out of C++, allowing me to concentrate on the interesting part: problem solving.

To build the project, choose and add the build option `OCD Project` and relaunch the project to make the predefined options to appear



Name: OCD project

☑ Store as project file

Target: project.elf

Executable binary: project.elf

Debugger: Bundled GDB multiarch

Board config file: board/st_nucleo_f4.cfg

GDB port: 3333

Telnet port: 4444

Download: ○ Always ● If updated ○ Never

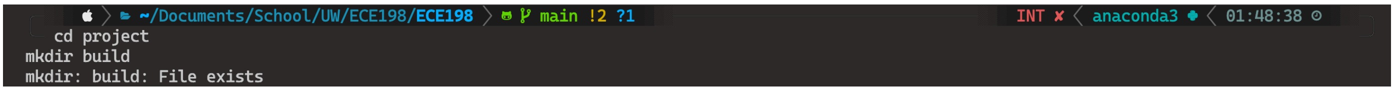
Reset: ○ Run ● Init ○ Halt ○ None

□ Show this page ☑ Activate tool window □ Focus tool window

## Manually build with CMake:

If this is the first time build, make a directory called `build` in the project directory (`./project/`)

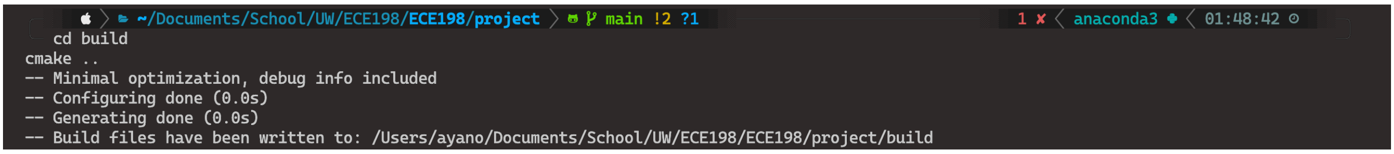
```
cd project
mkdir build
```



```
 ~/Documents/School/UW/ECE198/ECE198 〉 main !2 ?1                    INT ✘ 〈 anaconda3 〈 01:48:38
    cd project
mkdir build
mkdir: build: File exists
```

then generate CMake build files:

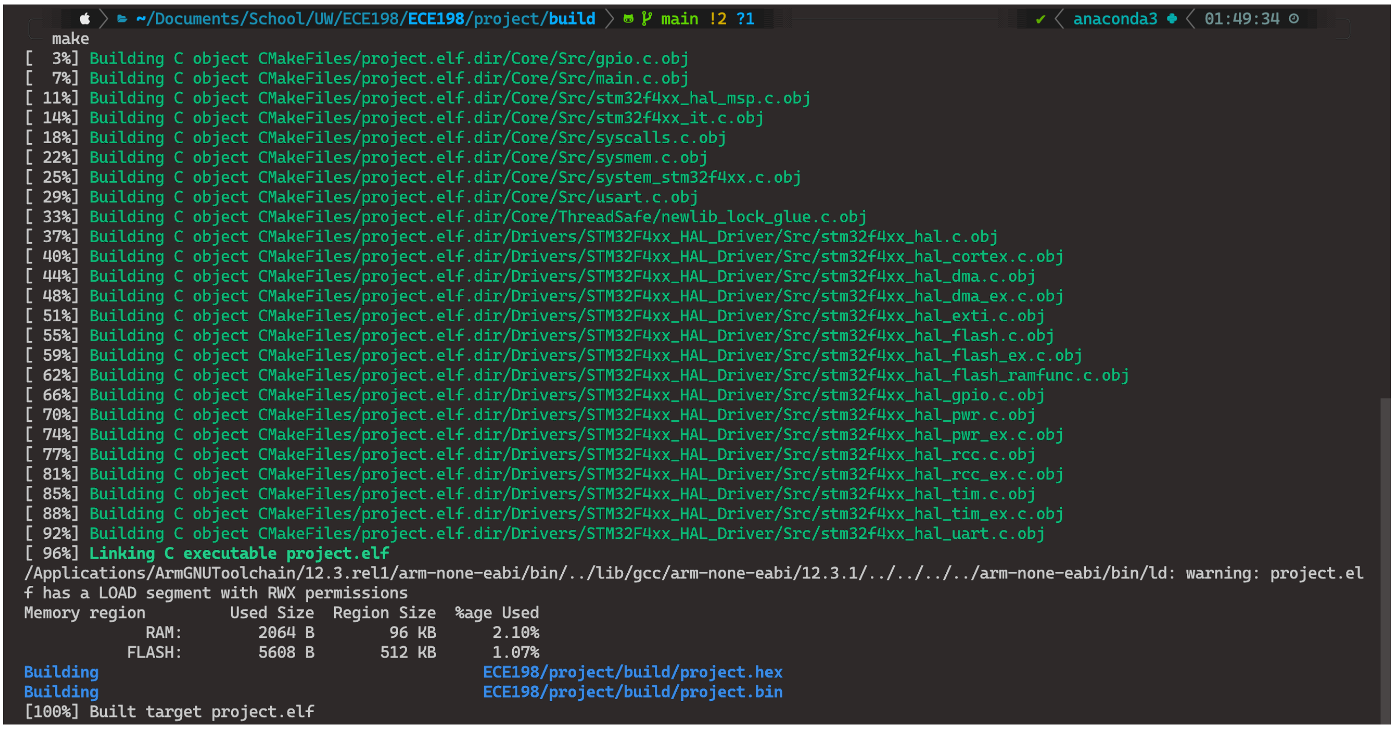
```
cd build
cmake ..
```



```
 ~/Documents/School/UW/ECE198/ECE198/project 〉 main !2 ?1                 1 ✘ 〈 anaconda3 〈 01:48:42
    cd build
cmake ..
-- Minimal optimization, debug info included
-- Configuring done (0.0s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/ayano/Documents/School/UW/ECE198/ECE198/project/build
```

then build the project:

```
make
```



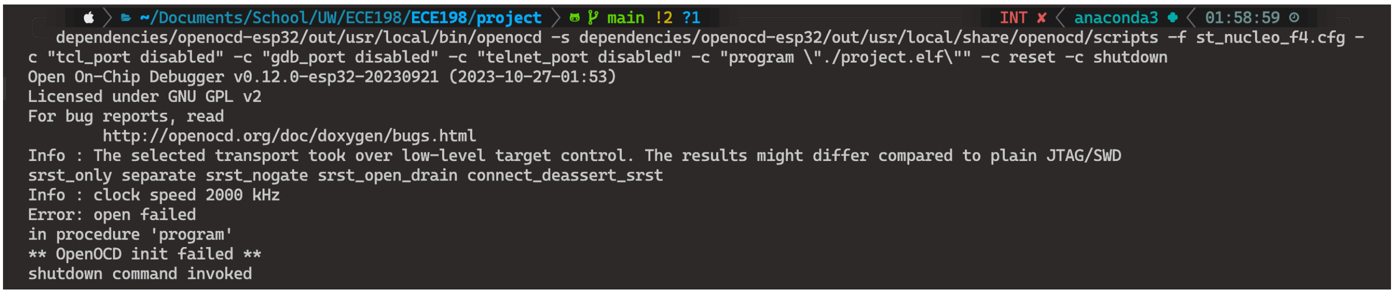
```
 ~/Documents/School/UW/ECE198/ECE198/project/build 〉 main !2 ?1              ✓ 〈 anaconda3 〈 01:49:34
    make
[  3%] Building C object CMakeFiles/project.elf.dir/Core/Src/gpio.c.obj
[  7%] Building C object CMakeFiles/project.elf.dir/Core/Src/main.c.obj
[ 11%] Building C object CMakeFiles/project.elf.dir/Core/Src/stm32f4xx_hal_msp.c.obj
[ 14%] Building C object CMakeFiles/project.elf.dir/Core/Src/stm32f4xx_it.c.obj
[ 18%] Building C object CMakeFiles/project.elf.dir/Core/Src/syscalls.c.obj
[ 22%] Building C object CMakeFiles/project.elf.dir/Core/Src/sysmem.c.obj
[ 25%] Building C object CMakeFiles/project.elf.dir/Core/Src/system_stm32f4xx.c.obj
[ 29%] Building C object CMakeFiles/project.elf.dir/Core/Src/usart.c.obj
[ 33%] Building C object CMakeFiles/project.elf.dir/Core/ThreadSafe/newlib_lock_glue.c.obj
[ 37%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal.c.obj
[ 40%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_cortex.c.obj
[ 44%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_dma.c.obj
[ 48%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_dma_ex.c.obj
[ 51%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_exti.c.obj
[ 55%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_flash.c.obj
[ 59%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_flash_ex.c.obj
[ 62%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_flash_ramfunc.c.obj
[ 66%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c.obj
[ 70%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_pwr.c.obj
[ 74%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_pwr_ex.c.obj
[ 77%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c.obj
[ 81%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc_ex.c.obj
[ 85%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim.c.obj
[ 88%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim_ex.c.obj
[ 92%] Building C object CMakeFiles/project.elf.dir/Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_uart.c.obj
[ 96%] Linking C executable project.elf
/Applications/ArmGNUToolchain/12.3.rel1/arm-none-eabi/bin/../lib/gcc/arm-none-eabi/12.3.1/../../../../arm-none-eabi/bin/ld: warning: project.el
f has a LOAD segment with RWX permissions
Memory region         Used Size  Region Size  %age Used
             RAM:        2064 B        96 KB      2.10%
           FLASH:        5608 B       512 KB      1.07%
Building                                         ECE198/project/build/project.hex
Building                                         ECE198/project/build/project.bin
[100%] Built target project.elf
```

The output file is `project.elf`, then you will use `openocd` you compiled yourself to flash the program to the board.

to flash the program to the board, run:

> **MAKE SURE YOU ARE UNDER `project` DIRECTORY**

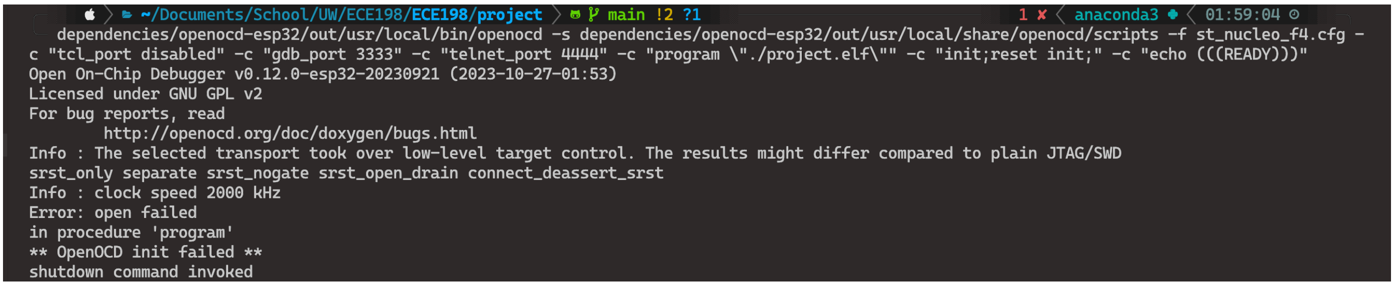
```
dependencies/openocd-esp32/out/usr/local/bin/openocd -s dependencies/openocd-esp32/out/usr/local/share/openocd/scripts -f st_nucleo_f4.cfg -c "tcl_port disabled" -c "gdb_port disabled" -c "telnet_port disabled" -c "program \"./project.elf\"" -c reset -c shutdown
```



```
 ~/Documents/School/UW/ECE198/ECE198/project  main !2 ?1                INT ✘  anaconda3  01:58:59
  dependencies/openocd-esp32/out/usr/local/bin/openocd -s dependencies/openocd-esp32/out/usr/local/share/openocd/scripts -f st_nucleo_f4.cfg -
c "tcl_port disabled" -c "gdb_port disabled" -c "telnet_port disabled" -c "program \"./project.elf\"" -c reset -c shutdown
Open On-Chip Debugger v0.12.0-esp32-20230921 (2023-10-27-01:53)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
srst_only separate srst_nogate srst_open_drain connect_deassert_srst
Info : clock speed 2000 kHz
Error: open failed
in procedure 'program'
** OpenOCD init failed **
shutdown command invoked
```

to debug the program, run:

```
dependencies/openocd-esp32/out/usr/local/bin/openocd -s dependencies/openocd-esp32/out/usr/local/share/openocd/scripts -f st_nucleo_f4.cfg -c "tcl_port disabled" -c "gdb_port 3333" -c "telnet_port 4444" -c "program \"./project.elf\"" -c "init;reset init;" -c "echo (((READY)))"
```



```
 ~/Documents/School/UW/ECE198/ECE198/project  main !2 ?1                  1 ✘  anaconda3  01:59:04
  dependencies/openocd-esp32/out/usr/local/bin/openocd -s dependencies/openocd-esp32/out/usr/local/share/openocd/scripts -f st_nucleo_f4.cfg -
c "tcl_port disabled" -c "gdb_port 3333" -c "telnet_port 4444" -c "program \"./project.elf\"" -c "init;reset init;" -c "echo (((READY)))"
Open On-Chip Debugger v0.12.0-esp32-20230921 (2023-10-27-01:53)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
srst_only separate srst_nogate srst_open_drain connect_deassert_srst
Info : clock speed 2000 kHz
Error: open failed
in procedure 'program'
** OpenOCD init failed **
shutdown command invoked
```

then you can connect to the program with `telnet` with port `4444` or `gdb` with port `3333`