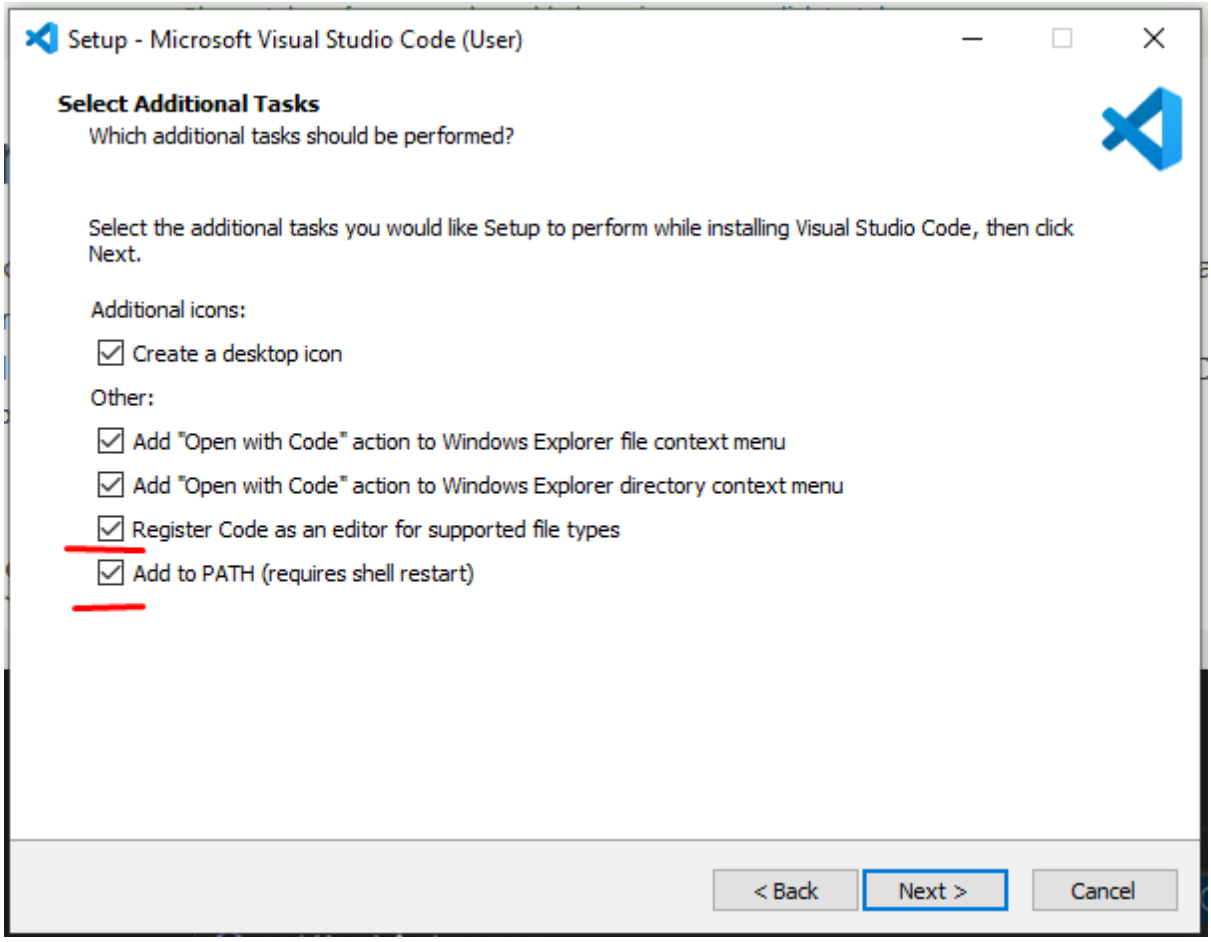
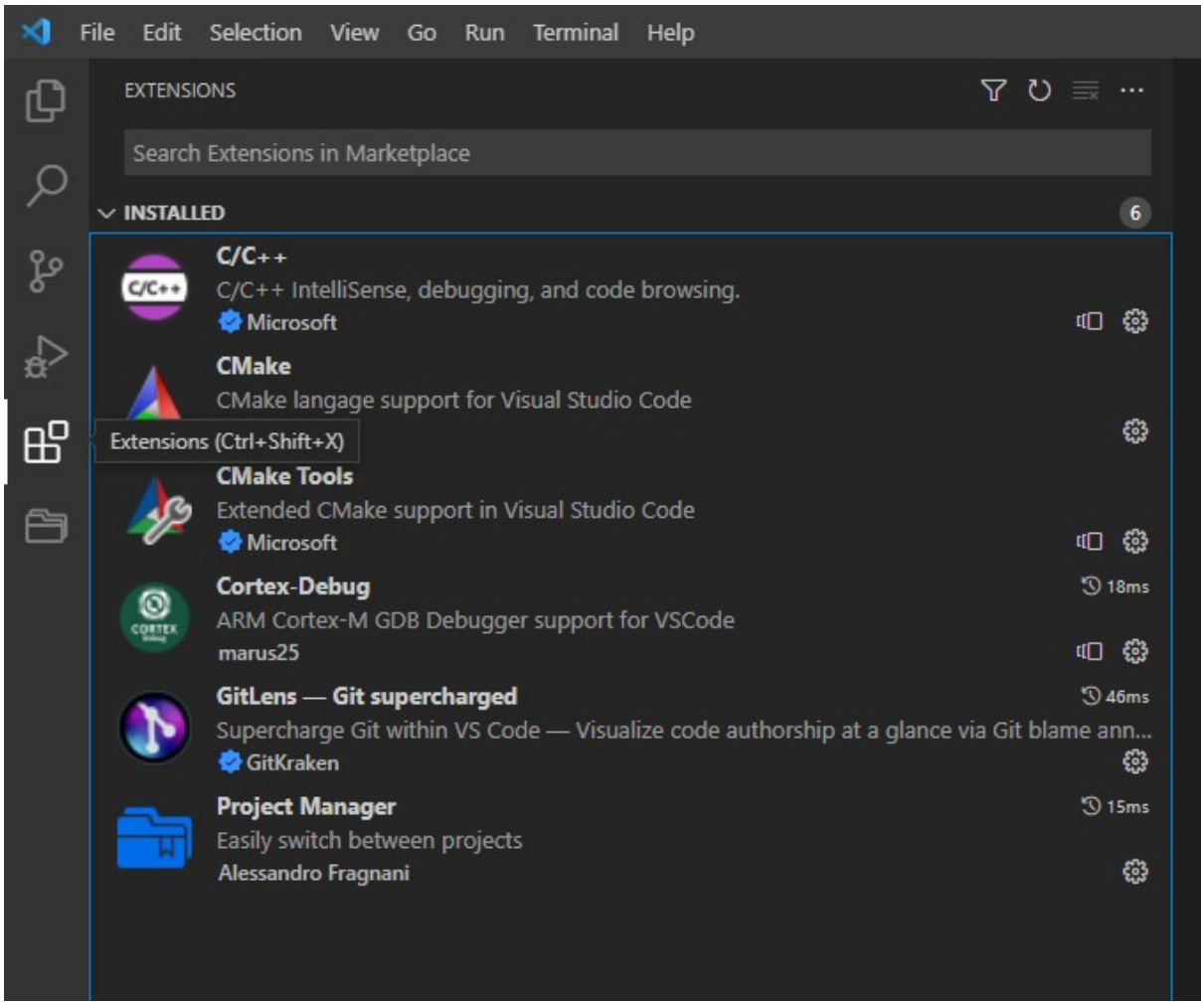
1. Installing extensions and tools

1.1 Install VS Code

Download installer from https://code.visualstudio.com/download
Run installer and follow install steps. I recommend to use next settings while install:



1.1.1 Install extensions for Visual Studio Code



It is required to install:

- C/C++
- CMake
- CMake Tools
- Cortex-Debug

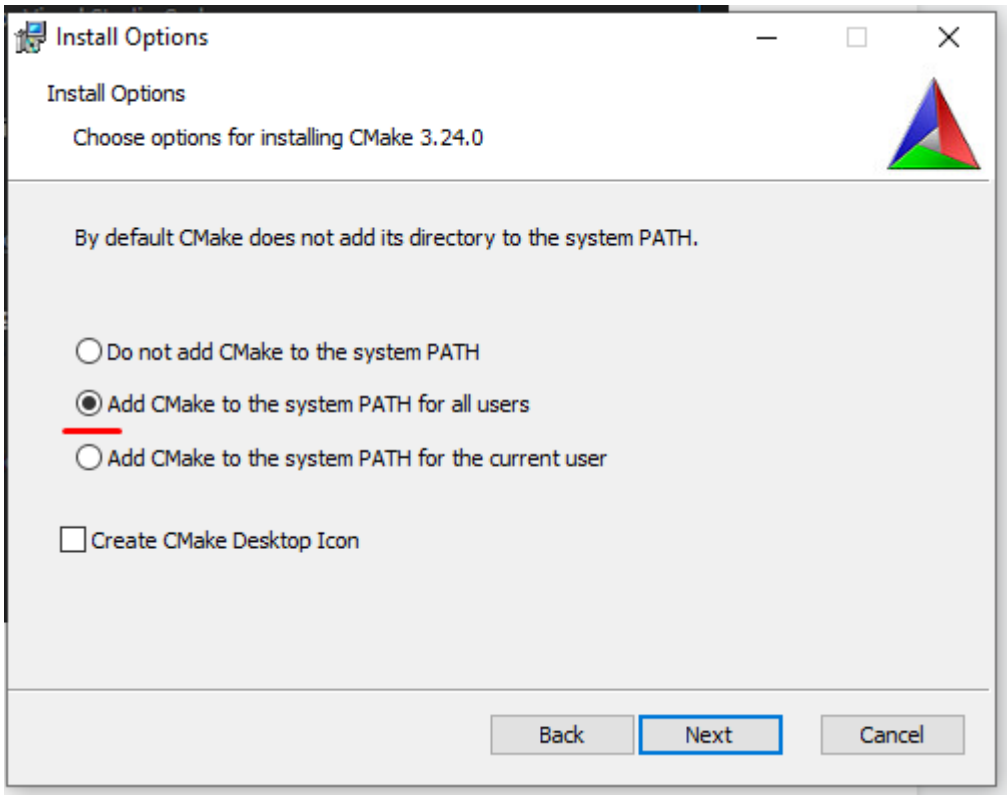
Also it is nice to have (but it is optional):

- GitLens
- Project Manager

You can also install any additional extensions that you want.

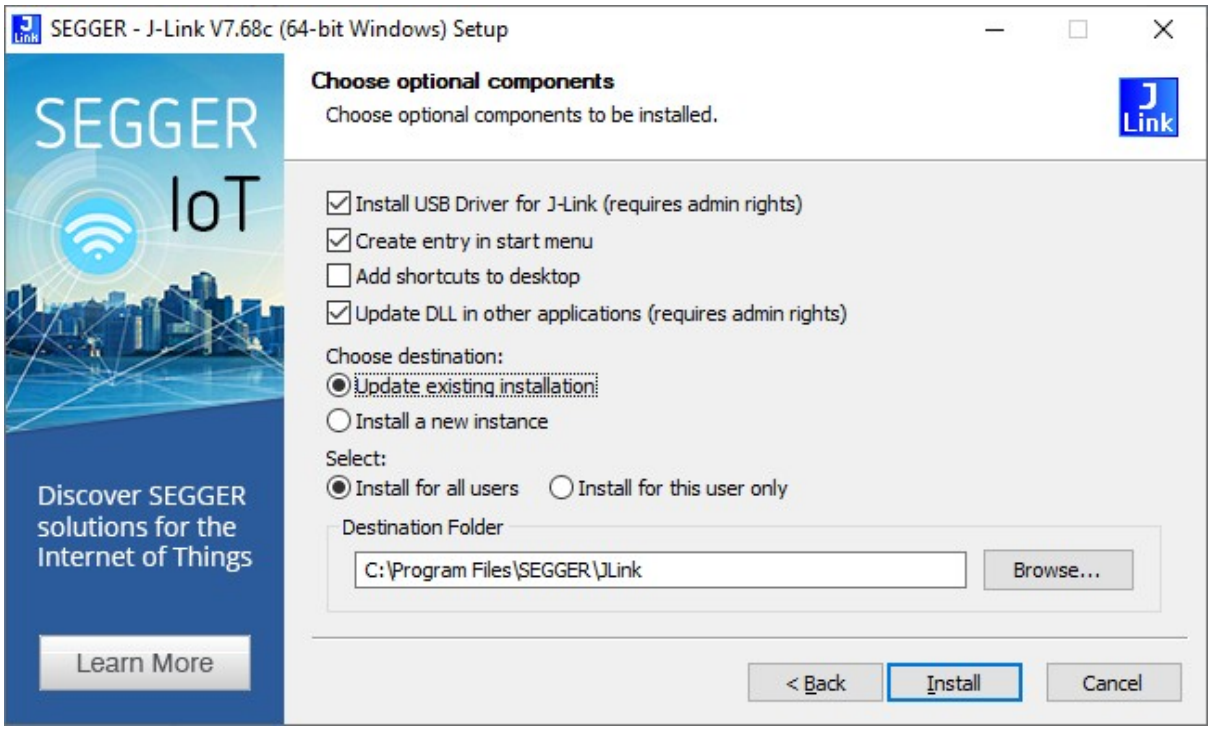
1.2 Install CMake

Download installer from https://cmake.org/download/
While installing you should select “Add to PATH”, or you can do it manually later in [1.9 Edit environment variables](#).



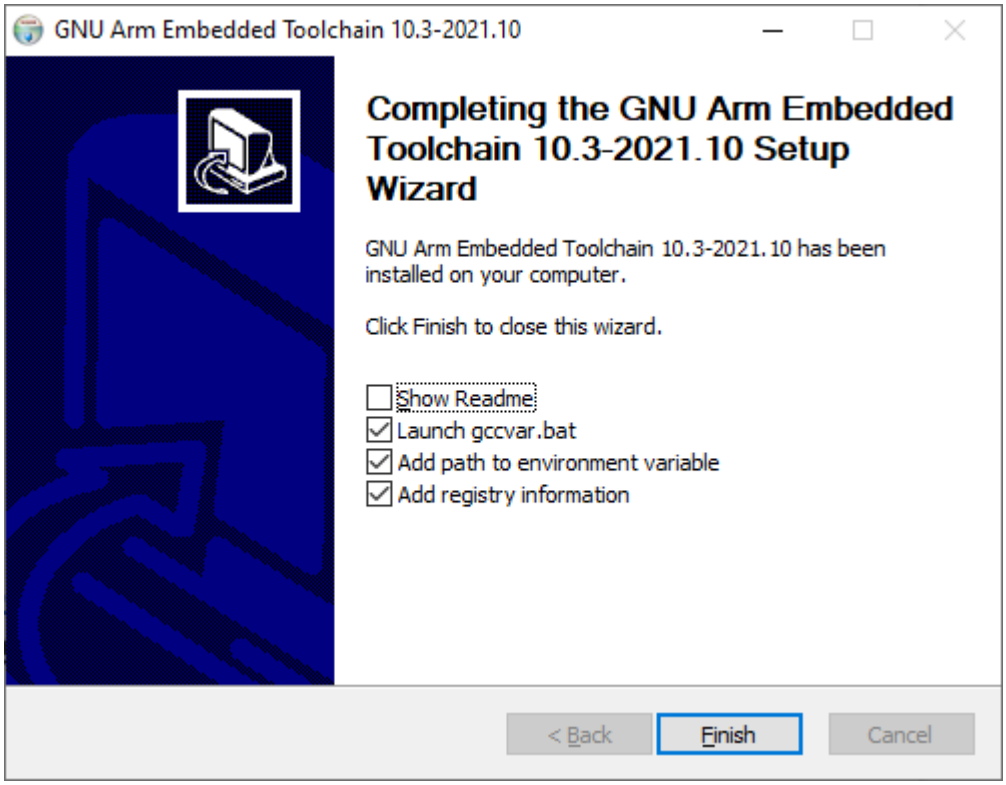
1.3 Install JLink

Download JLink from https://www.segger.com/downloads/jlink/
While installing you should select that settings



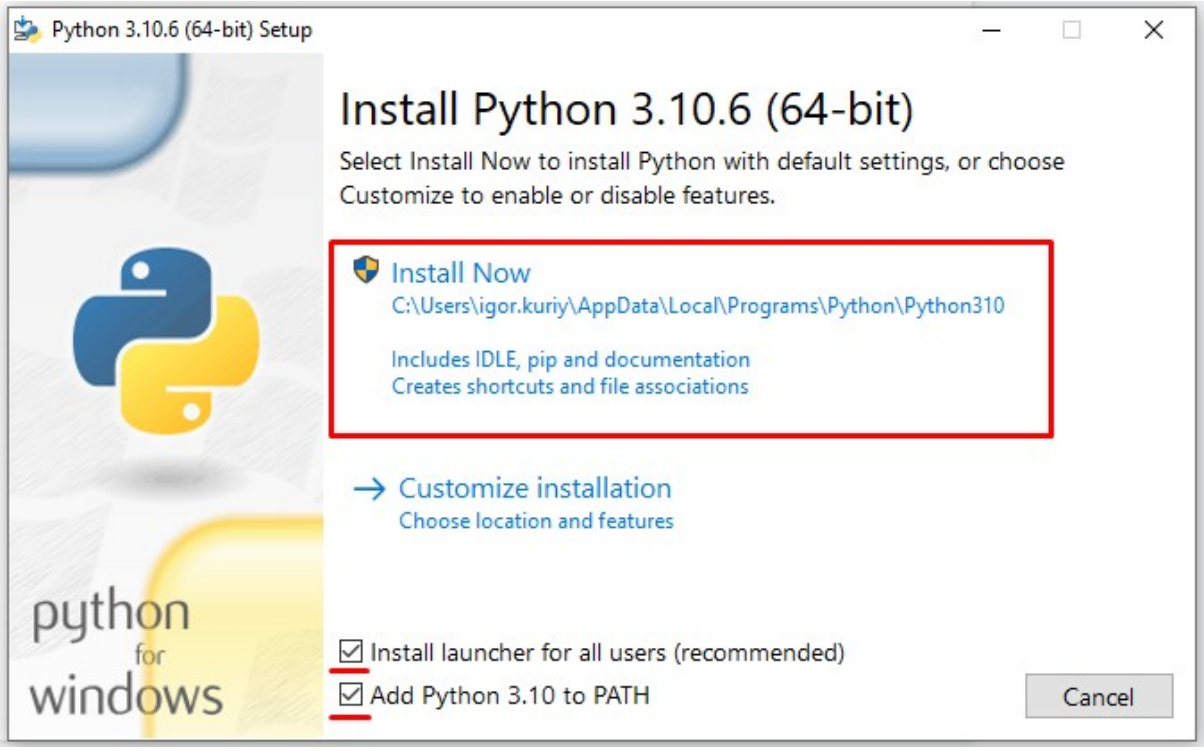
1.4 Install compiler

Download toolchain from https://developer.arm.com/downloads/-/gnu-rm
I highly recommend not changing the default install path. Also you should select next settings after install



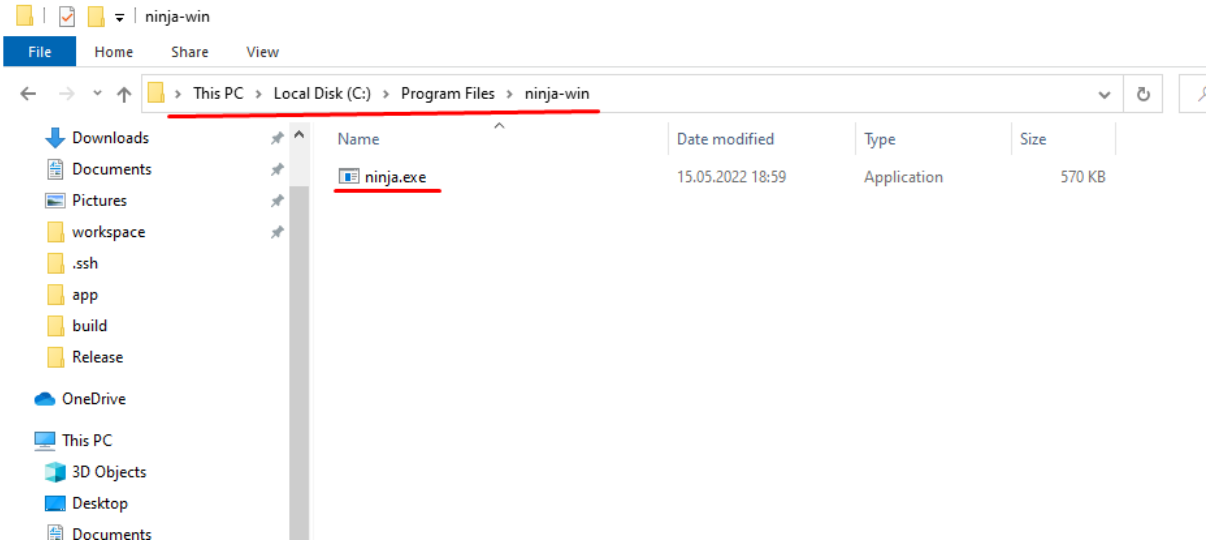
1.5 Install python

Download python from https://www.python.org/downloads/



1.6 Install Ninja

Download ninja-win.zip from https://github.com/ninja-build/ninja/releases
Then you should unzip it to Program Files folder



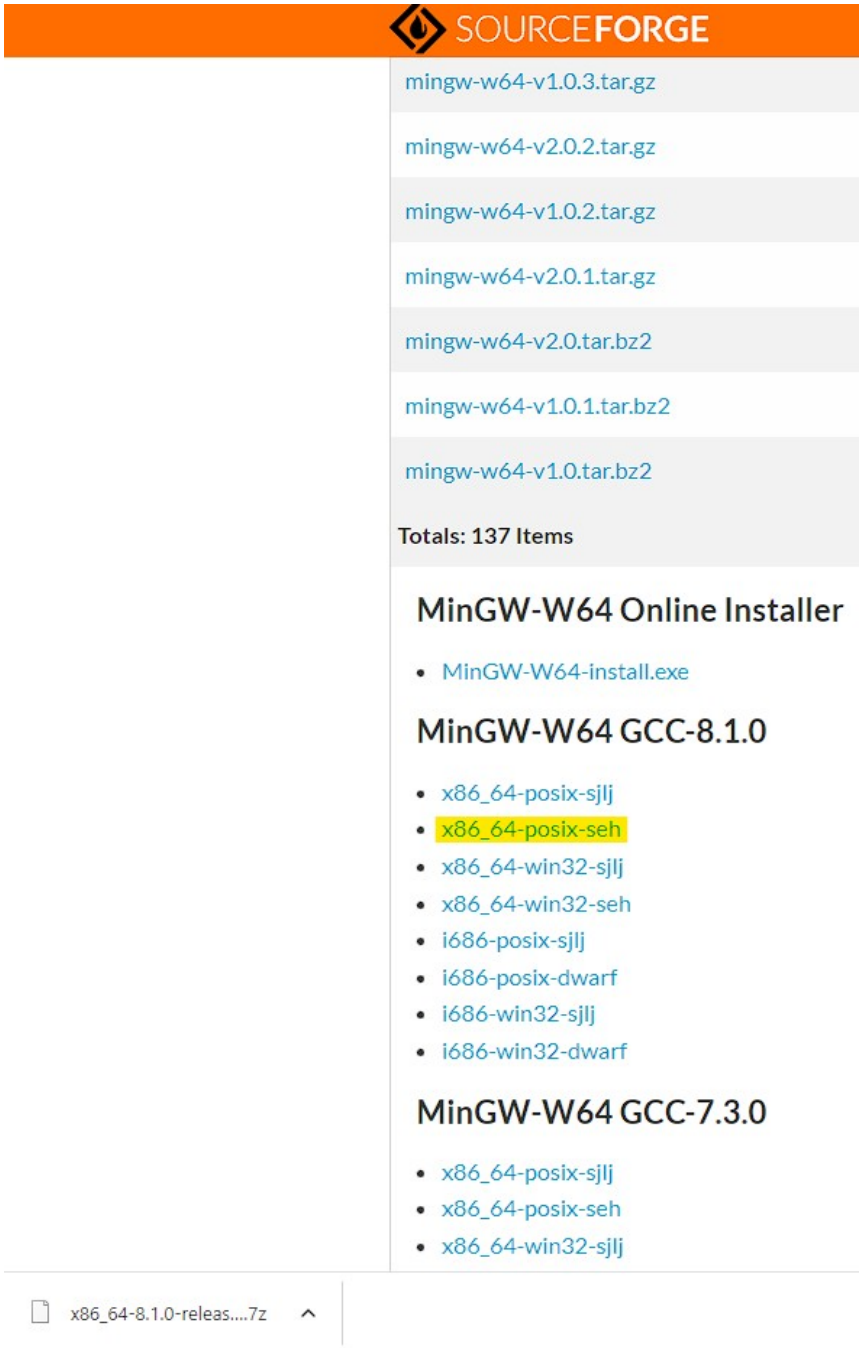
Add Ninja to the “Path” variable later in 1.9 Edit environment variables section.

1.7 Install MinGW

Open
https://sourceforge.net/projects/mingw-w64/files/mingw-w64/mingw-w64-release/

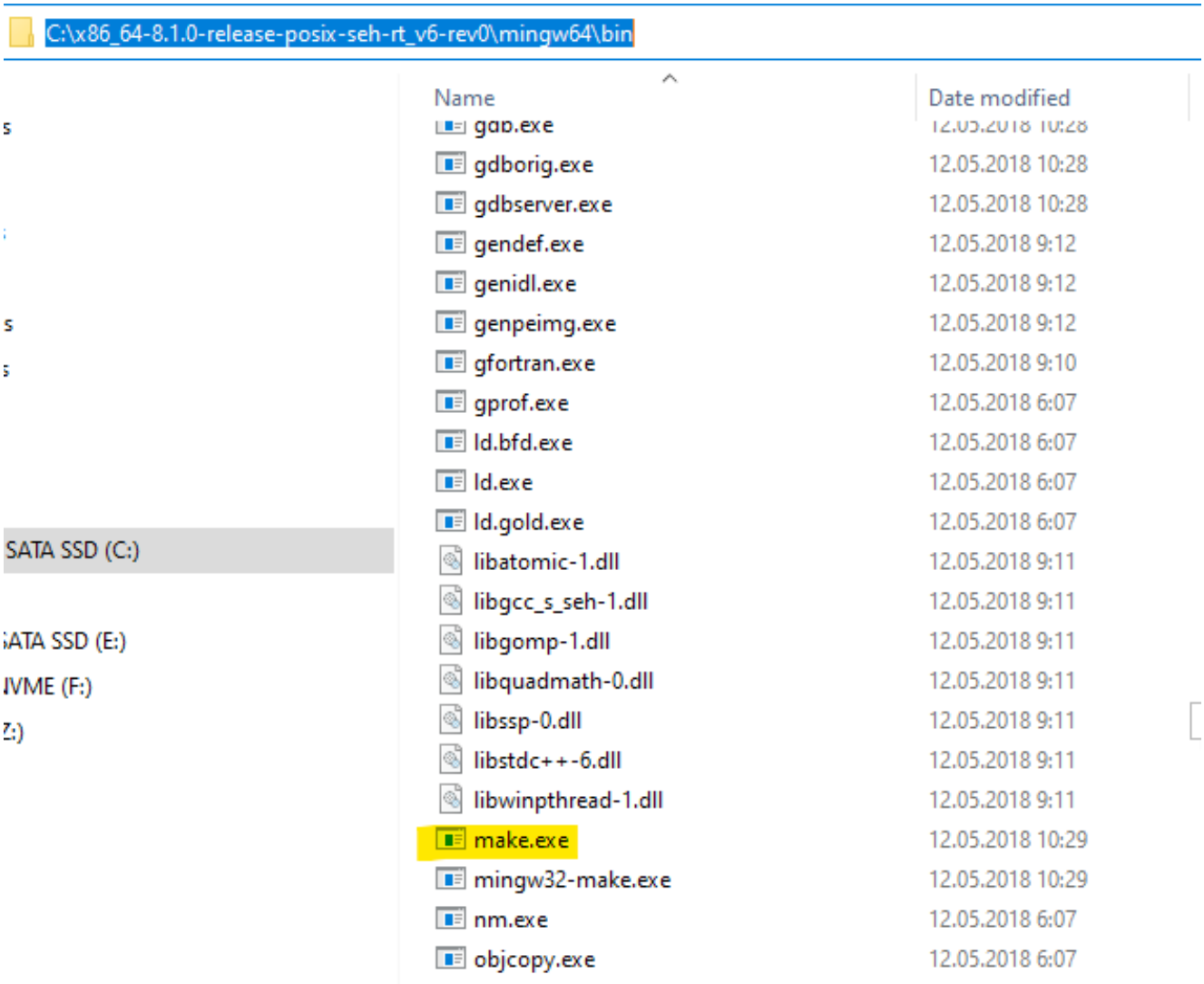
Scroll down and download x86_64-posix-seh archive, you will need 7-zip to unzip it.

Online Installer also can be used, but sometimes it fails to download files and throws errors, so the archive download is preferred.



After archive downloaded you need to unzip it to a destination folder.
The next step is to add the path of the “bin” folder to the “Path” environmental variable, see [1.9 Edit environment variables](#).
For example, if we unzip archive in the root of the disk C:\
the path of the “bin” folder will be C:\x86_64-8.1.0-release-posix-seh-rt_v6-rev0\mingw64\bin

By default the is no make.exe program in the “bin” folder.
It is called mingw32-make.exe
Feel free to copy mingw32-make.exe and rename the copy to make.exe, should look like this. After this step makefile generation should be working.



1.8 Check from command prompt

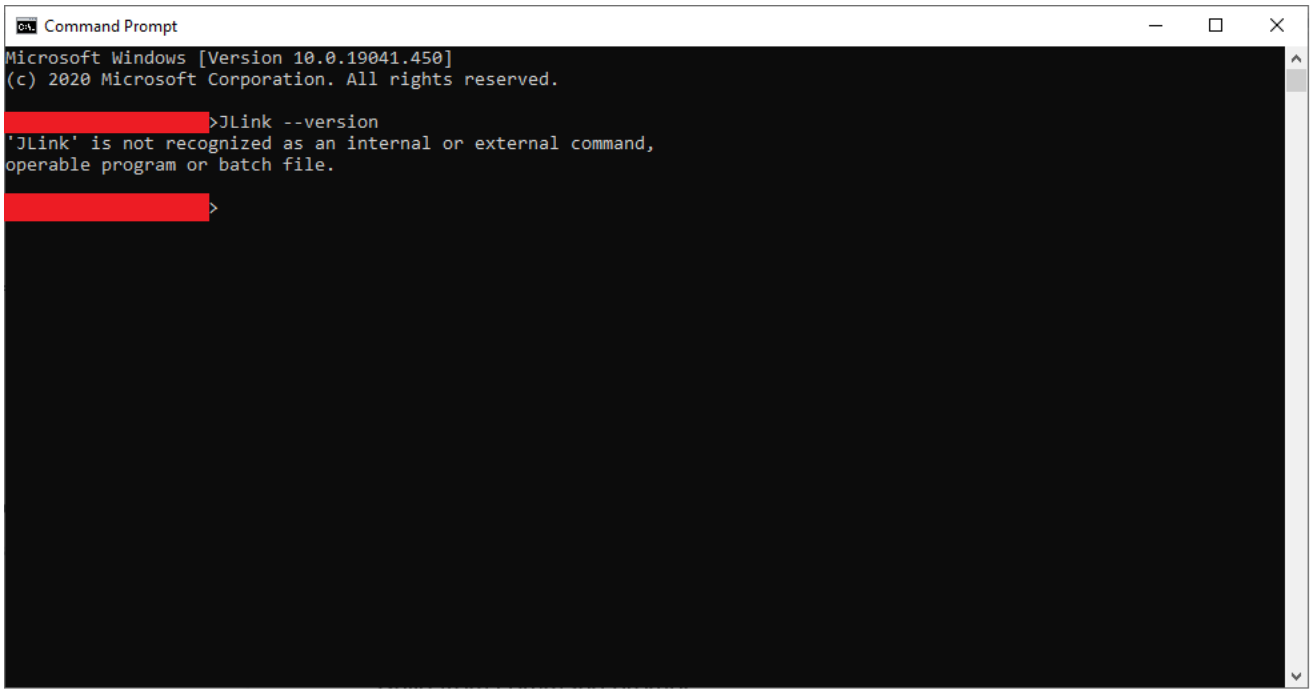
You can use next commands to check that program is visible in command prompt:

cmake --version
JLink --version
arm-none-eabi-gcc --version
python --version
ninja --version

If everything well you will get next output:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

[redacted]>cmake --version
cmake version 3.24.0-rc5

CMake suite maintained and supported by Kitware (kitware.com/cmake).

[redacted]>JLink --version
SEGGER J-Link Commander V7.68c (Compiled Jul 28 2022 15:47:10)
DLL version V7.68c, compiled Jul 28 2022 15:45:26

Unknown command line option --version.

C:\Users\igor.kuriy>arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824 (release)
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


[redacted]>python --version
Python 3.10.6

[redacted]>_
```

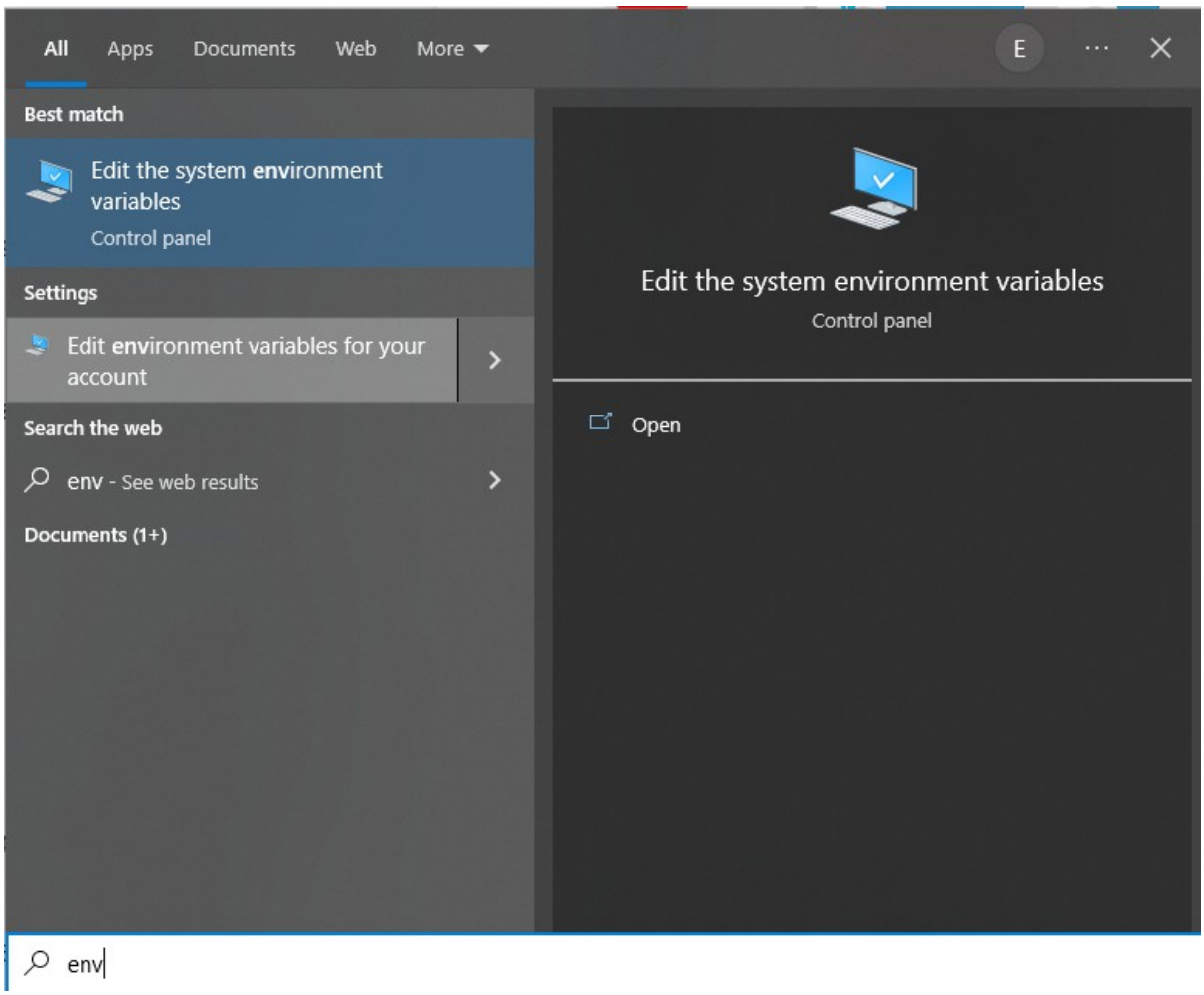
If some program is not visible from command prompt, you will get the next result:



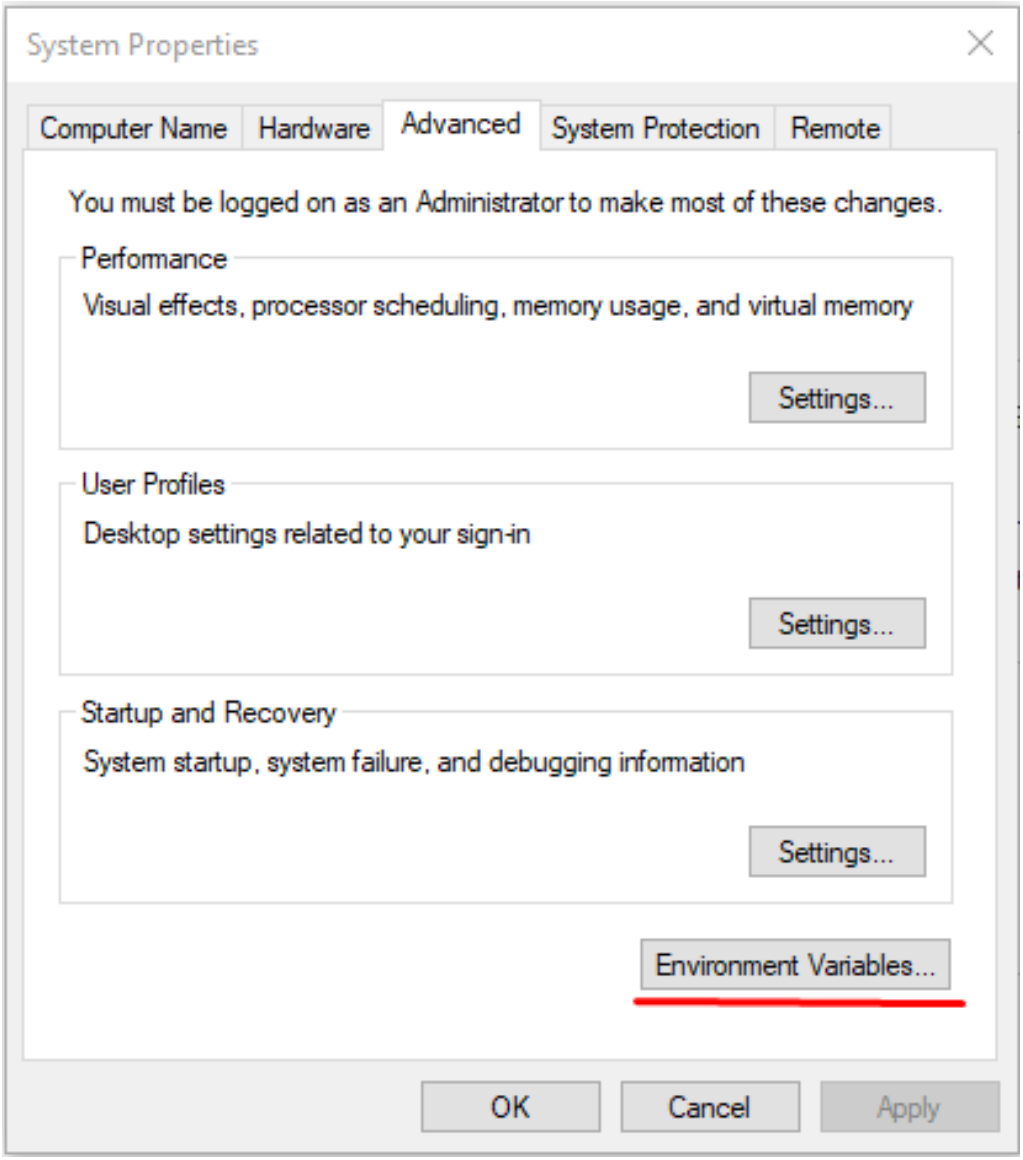
and you need to add the program path to Environment Variables (see [1.9 Edit environment variables](#)).

1.9 Edit environment variables

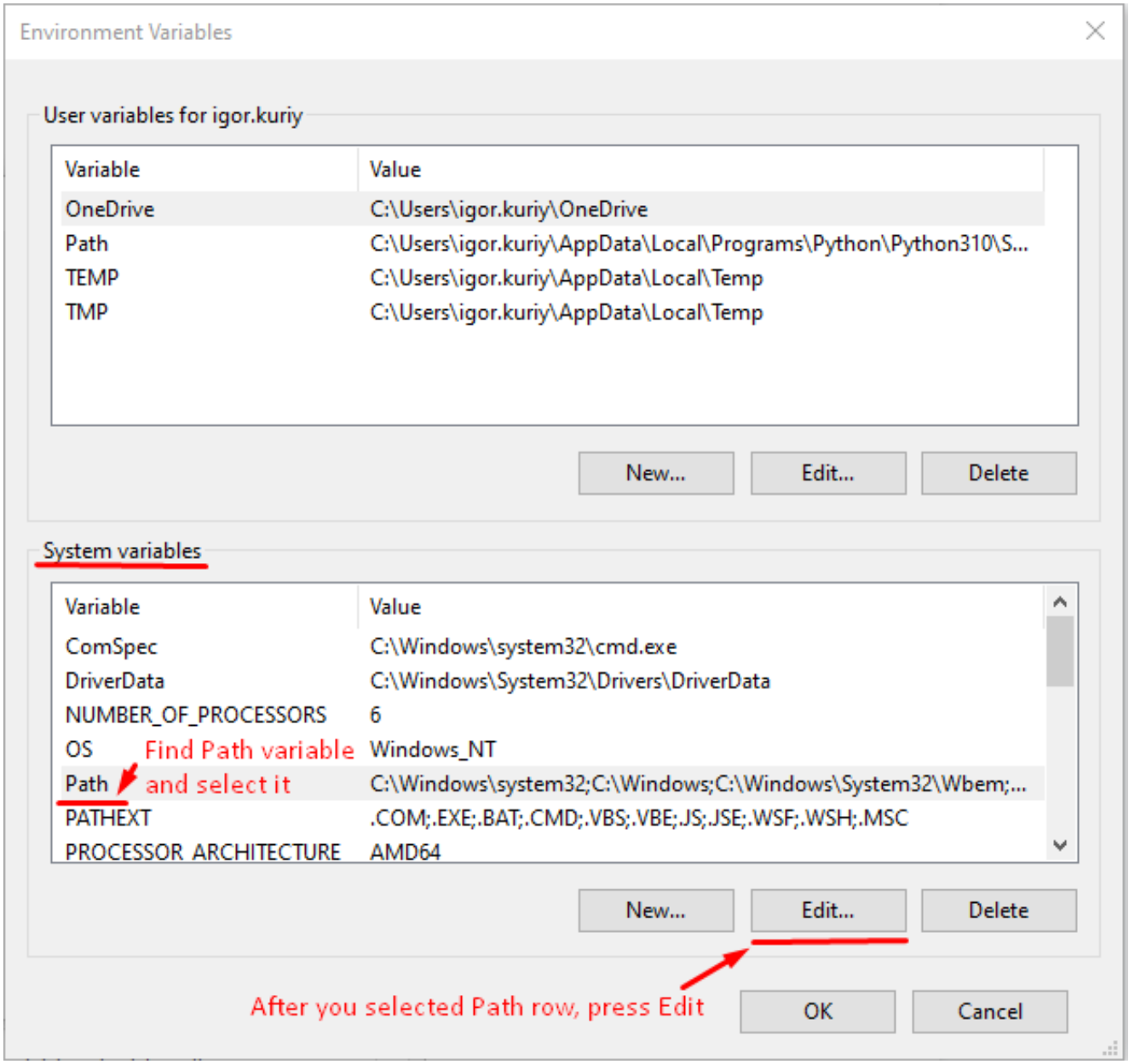
Press Win key and type “env”, Edit the system environment variables should appear, click it.



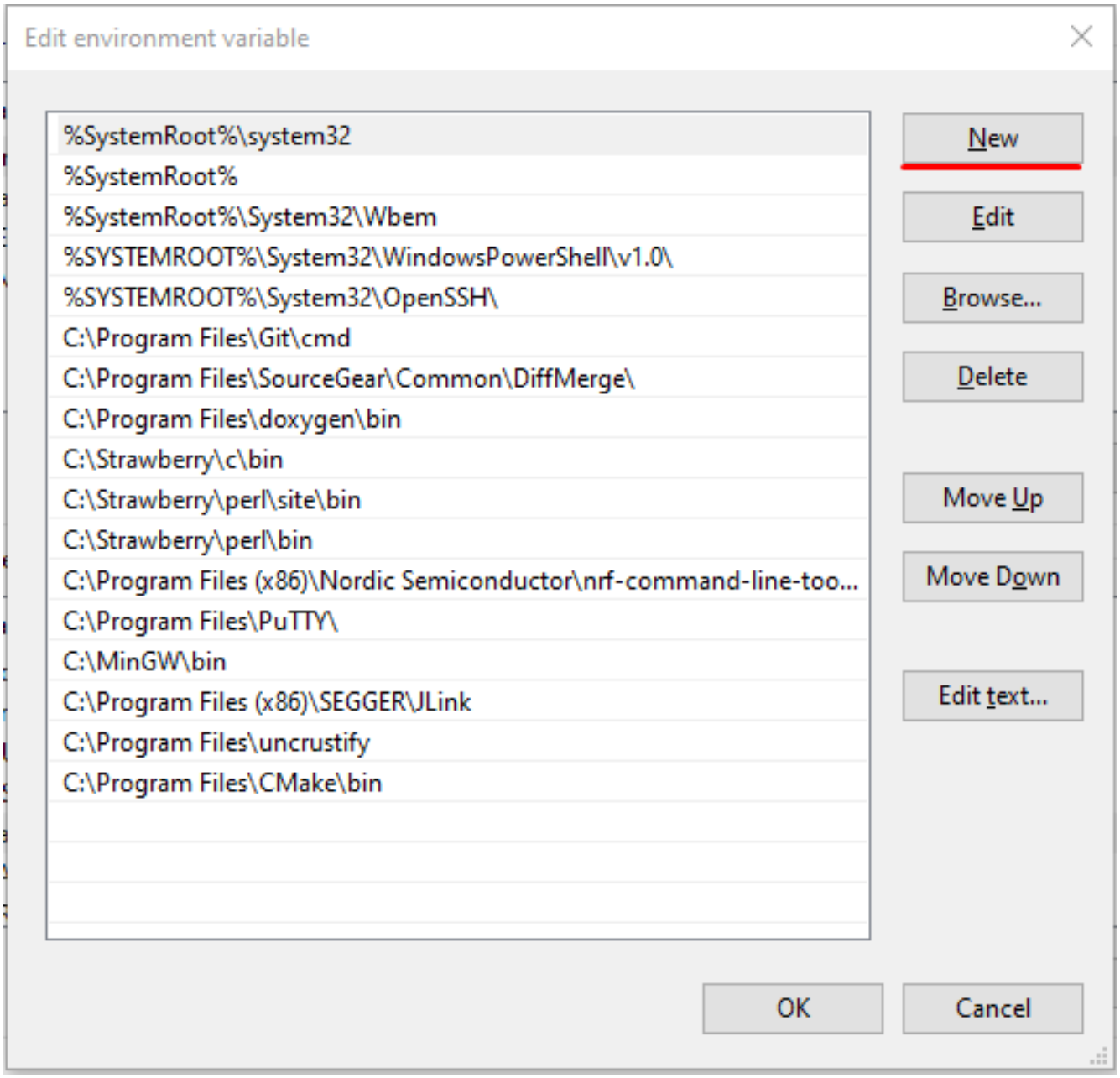
Then press “Environment variables” button


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

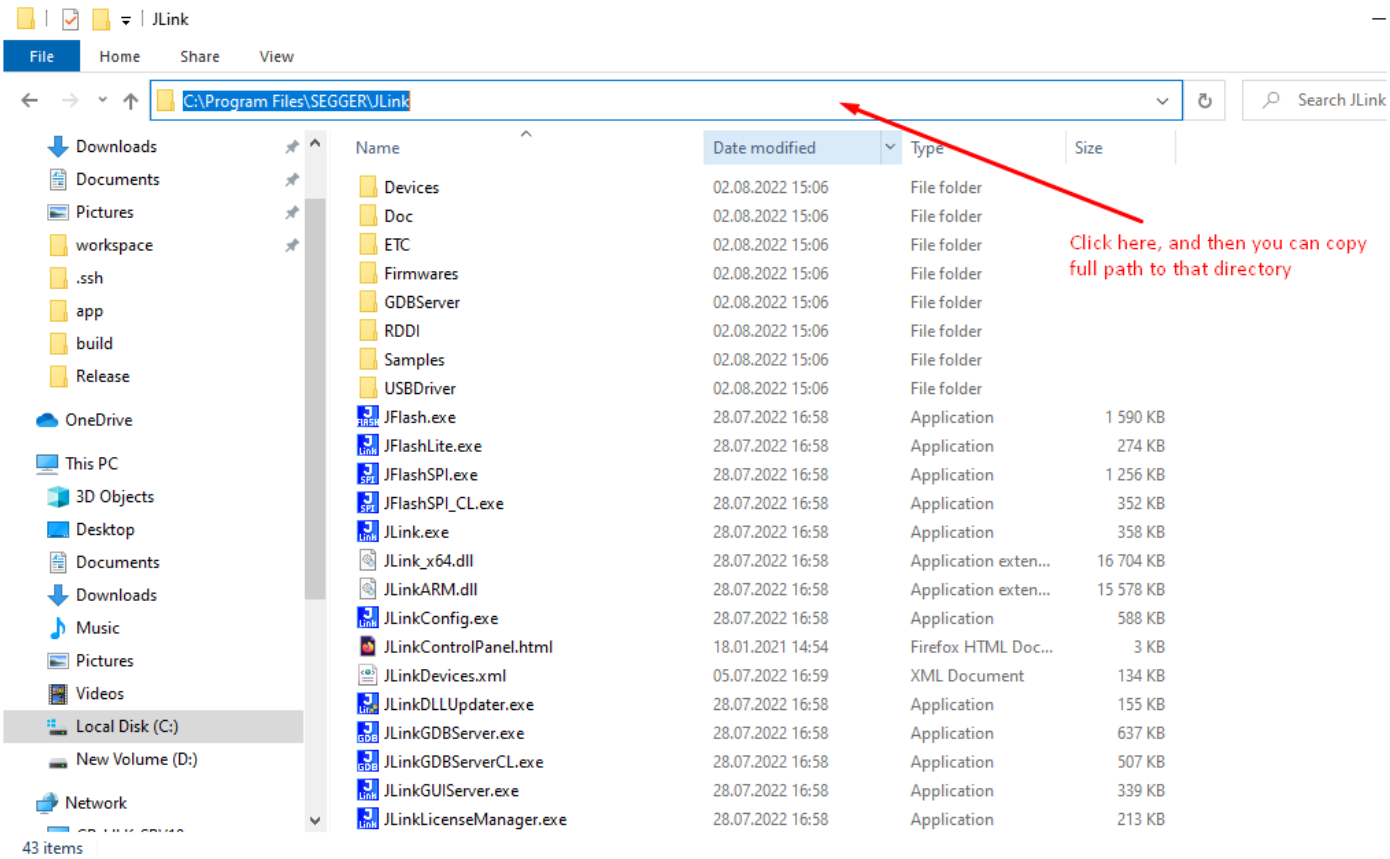
Then you need to find “Path” variable in System variables, then select it, and press Edit

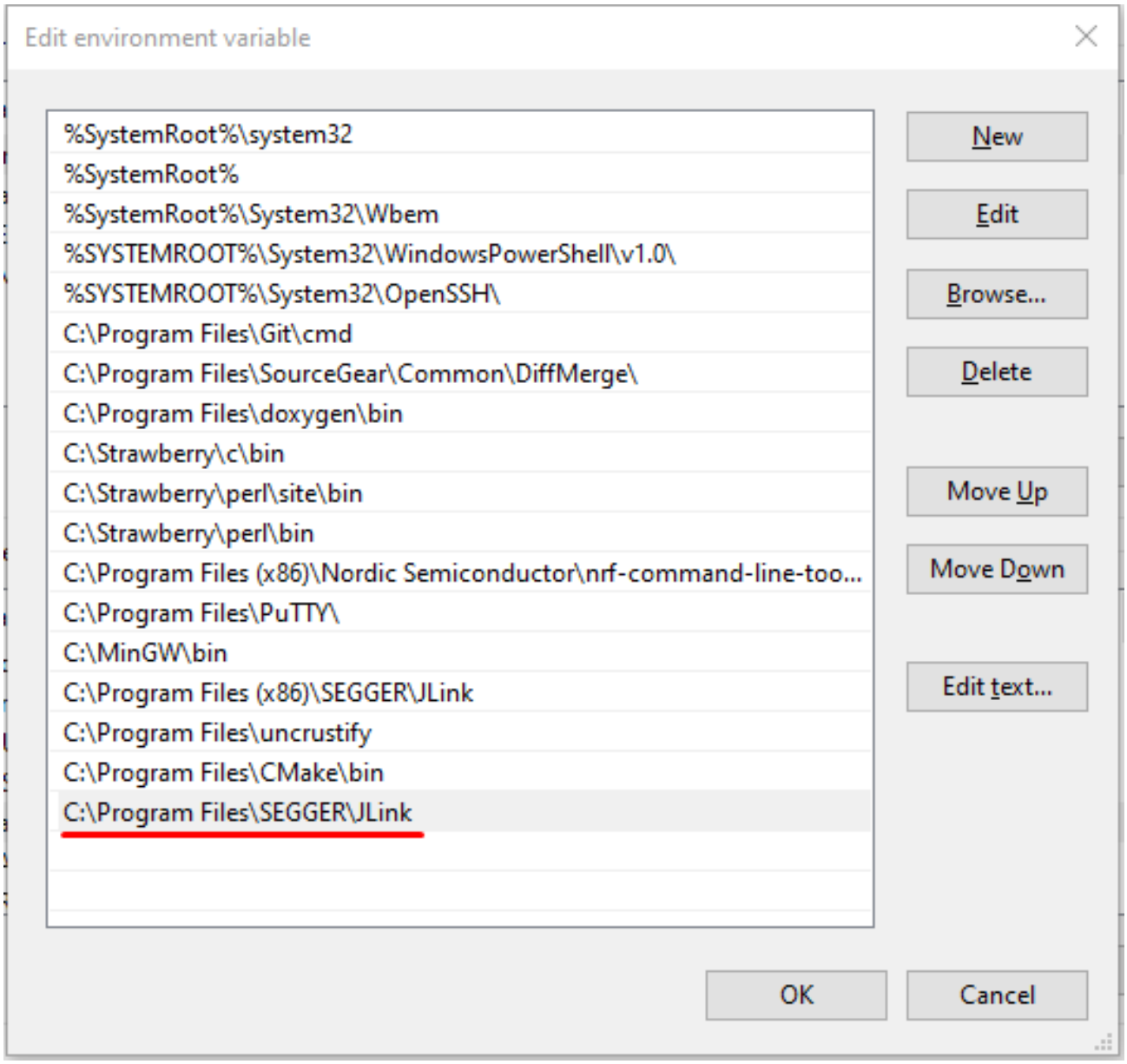


Now you need to press “New” button



And then you need to write a path to the missing program, for example JLink. You can just copy paste that path from the explorer window if you have opened it before.



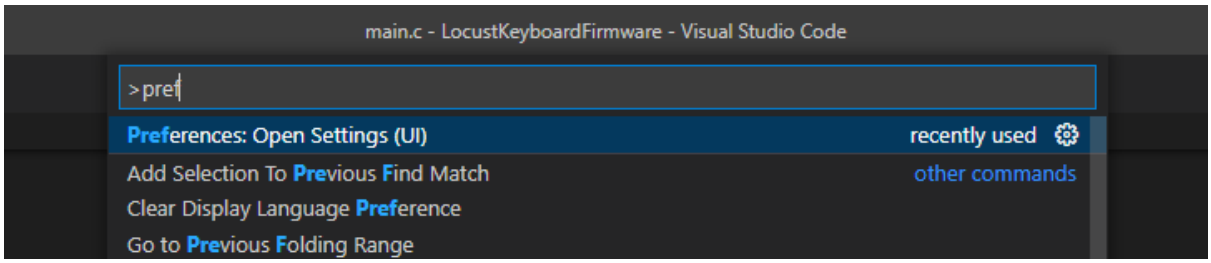


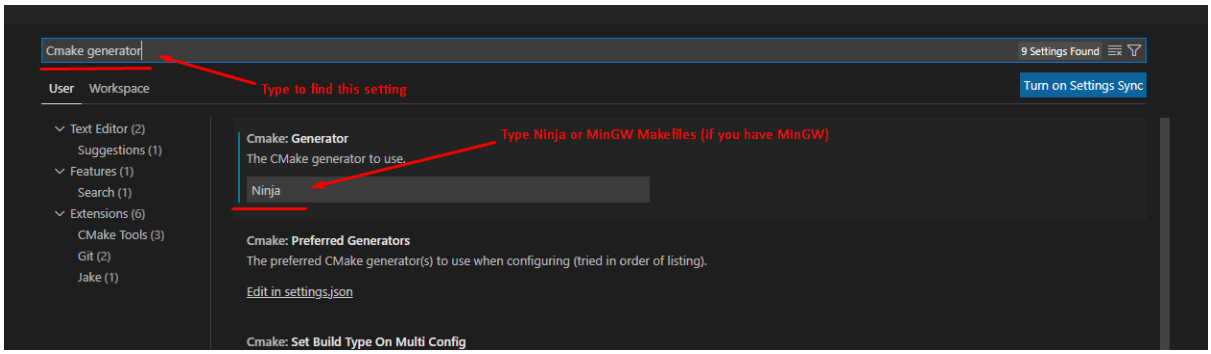
After you add new Environment Variables, it is better to restart your PC.

2. Setup CMake extension

2.1 Setup generator

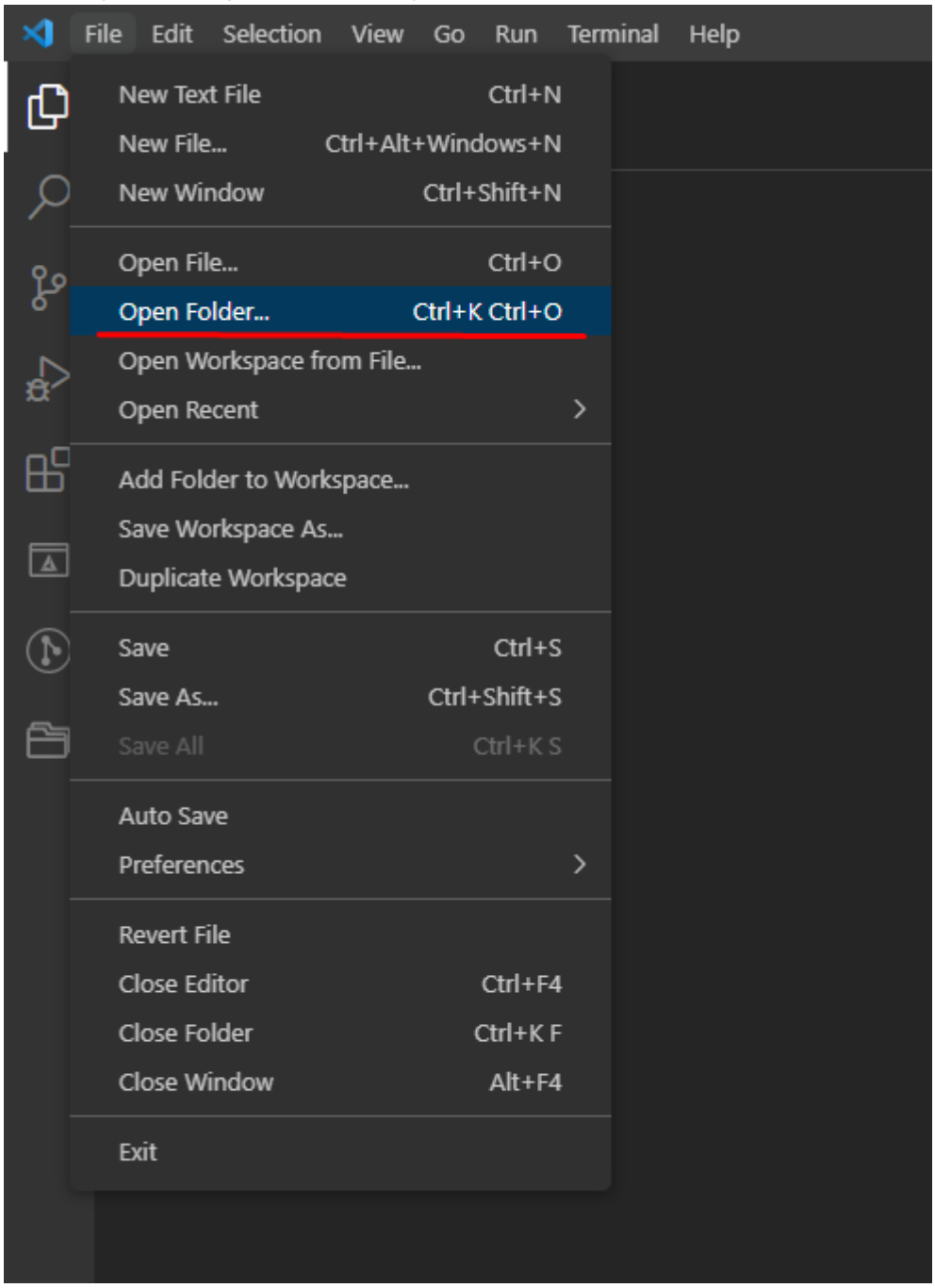
CMake doesn’t build project by it self, CMake generates builders for other tools like Makefile, Ninja, etc.
You can use MinGW Makefiles if you already installed MinGW, or you can use Ninja. Both variants are fine, but you need to set it in VS Code preferences. To do this press F1 and open preferences:



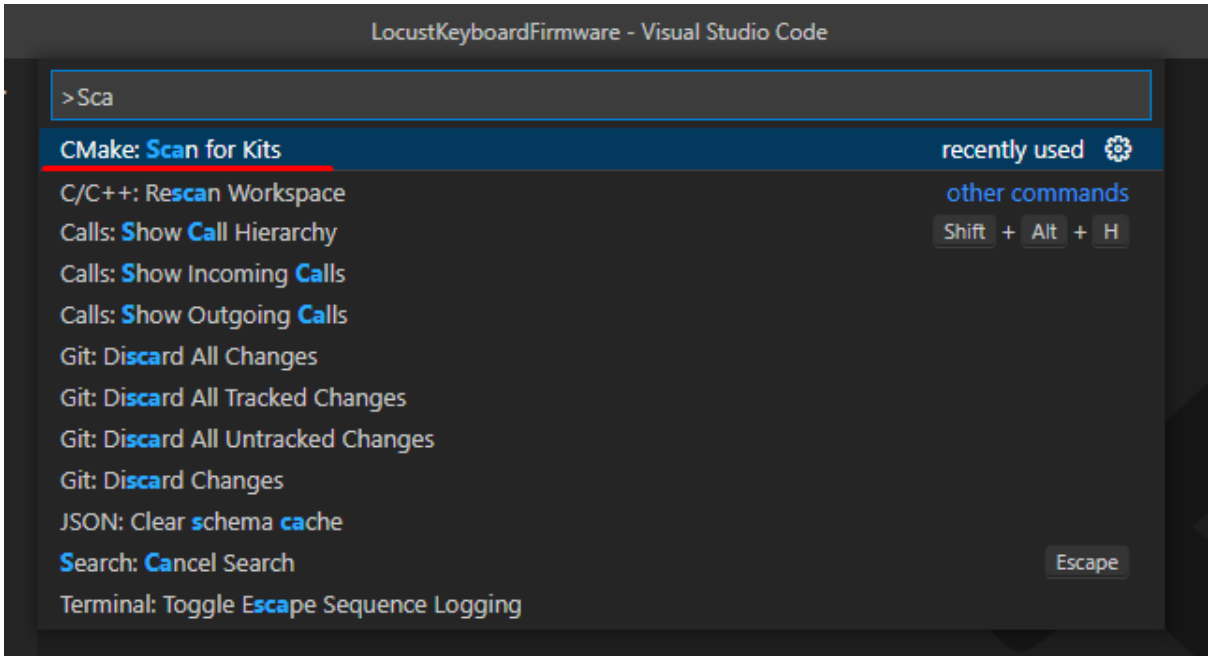


2.2 Setup CMake kits

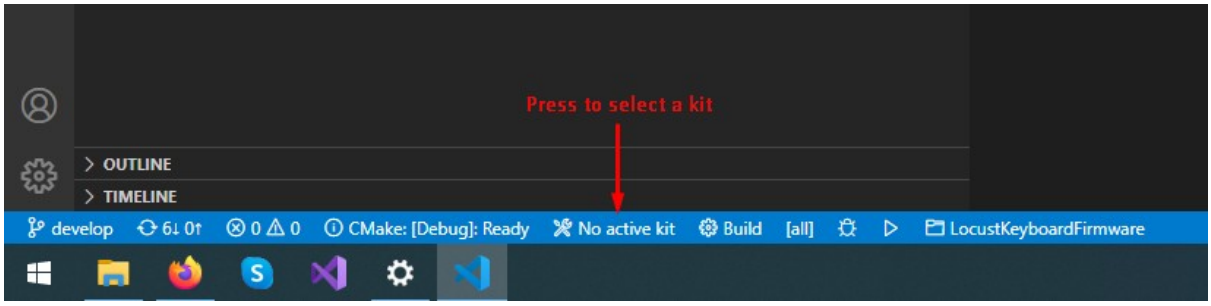
Open your project directory in VS Code



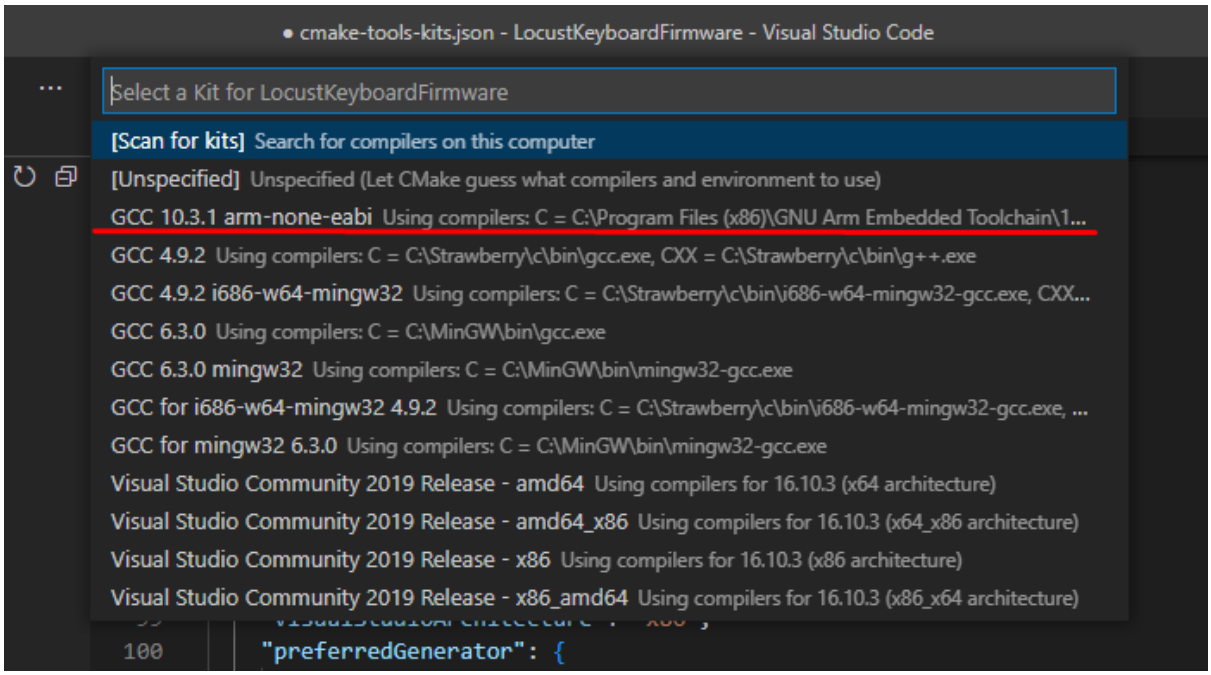
Then you need to press F1 and do “CMake: Scan for Kits”



Then press on kit button

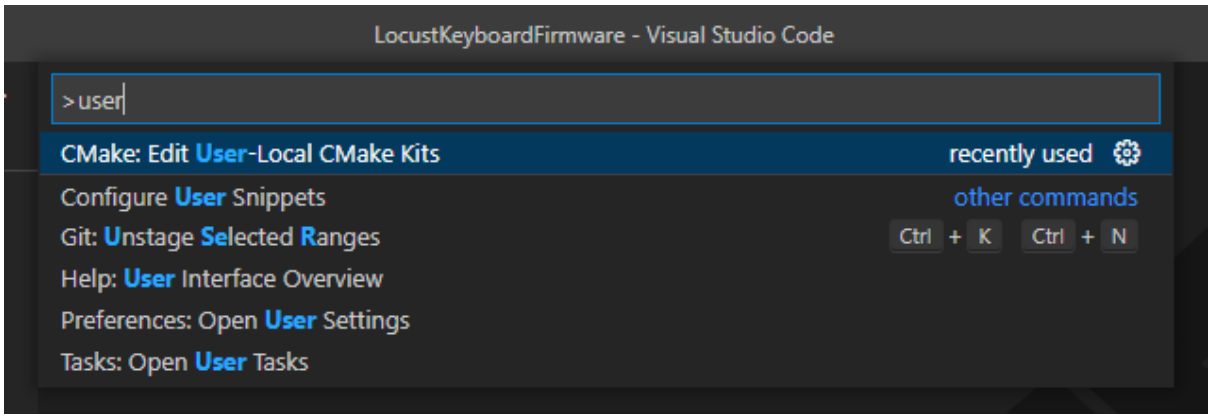


And select arm-none-eabi kit

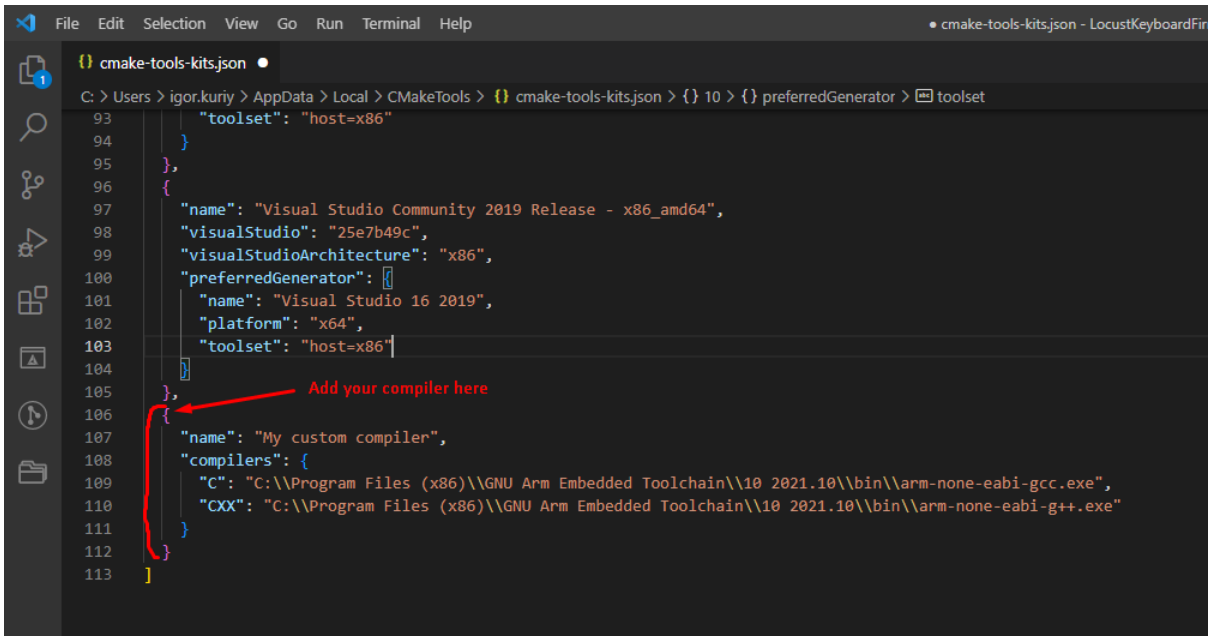


In some cases when you cannot find your ARM GCC kit, you need to restart VS Code.

There are some cases when CMake extension cannot find your kit, in those cases you can add them manually. To do this you need press F1 and run “CMake: Edit User-Local CMake Kits” command



That command will open the cmake-tool-kits.json file, where you can add your custom compiler. You need to add a name with a path to your compiler.

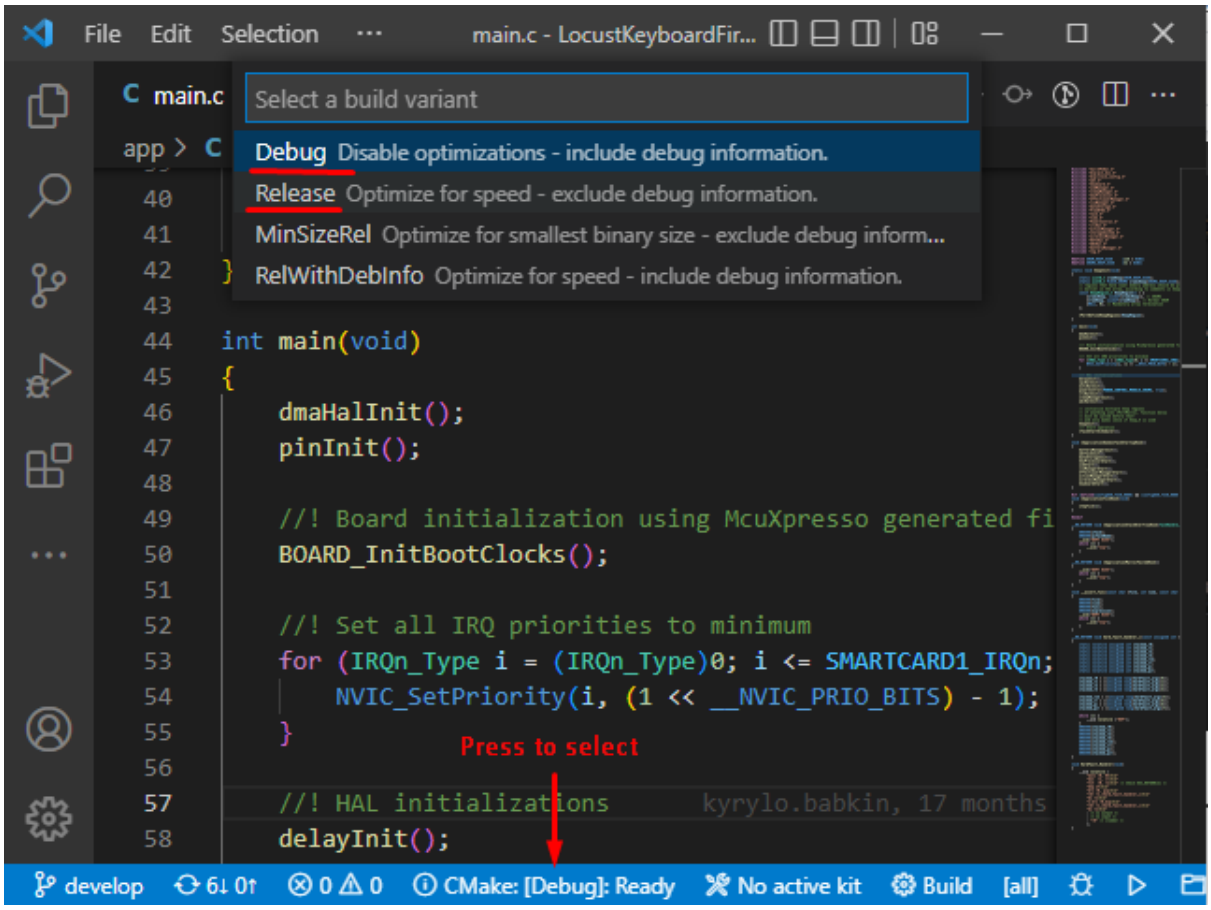


After that you save this file, you can select your custom kit to build your project.

3. Using CMake (based on NoName project)

3.1 Build from VS Code

You can select build variants. We have implemented 2 variants: Debug and Release. You can select build variant in GUI



For build your project you can use next commands:
CMake: Delete Cache and Reconfigure - this command should be used if you need to generate MakeFile/Ninja/etc structure
CMake: Configure - this command will create Makefile/Ninja structure
CMake: Clean Rebuild - this command will clean all binaries, compile all files again
CMake: Build - this command will build only modified files

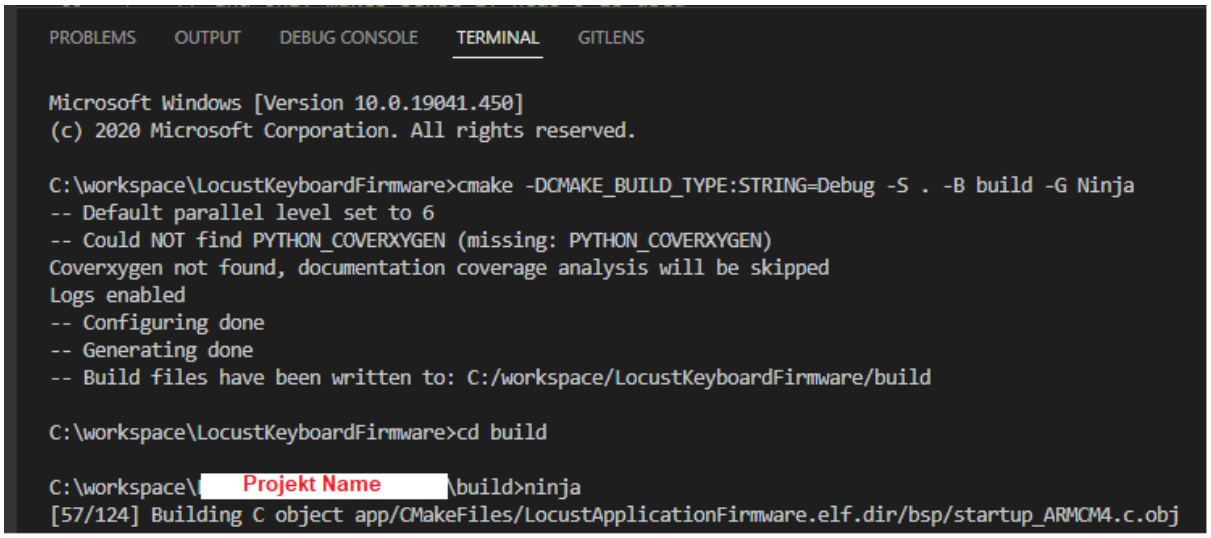
3.2 Build from command prompt

For building from command prompt you need to be sure that all tools are accessible in the console (see section 1.7).

3.2.1 Building with Ninja

Use command cmake to generate build structure and ninja to build:

```
>cmake -DCMAKE_BUILD_TYPE:STRING=Debug -S . -B build -G Ninja
>cd build
>ninja
```



3.2.2 Building with Makefile

Use command cmake to generate build structure and mingw makefiles to build:

```
>cmake -DCMAKE_BUILD_TYPE:STRING=Debug -S . -B build -G
“MinGW Makefiles”
>cd build
>mingw32-make
```

```
Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\workspace\LocustKeyboardFirmware>cmake -DCMAKE_BUILD_TYPE:STRING=Debug -S . -B build -G "MinGW Makefiles"
-- Default parallel level set to 6
-- Could NOT find PYTHON_COVERXYGEN (missing: PYTHON_COVERXYGEN)
Coverxygen not found, documentation coverage analysis will be skipped
Logs enabled
-- Configuring done
-- Generating done
-- Build files have been written to: C:/workspace/LocustKeyboardFirmware/build

C:\workspace\Project name>cd build

C:\workspace\Project name\build>mingw32-make
Consolidate compiler generated dependencies of target LittleFS
[  2%] Built target LittleFS
Consolidate compiler generated dependencies of target sdk
[ 38%] Built target sdk
Consolidate compiler generated dependencies of target FreeRTOS
[ 46%] Built target FreeRTOS
Consolidate compiler generated dependencies of target LocustApplicationFirmware.elf
[100%] Built target LocustApplicationFirmware.elf

C:\workspace\Project name\build>
```

3.3 Adding new source files

To add your new source files open CMakeLists.txt in the app directory and add it to the long .c files list

CMakeLists.txt - LocustKeyboardFirmware - Visual Studio Code

main.c | CMakeLists.txt .\ | CMakeLists.txt app

app > CMakeLists.txt

```
            -Wextra
        )

    target_link_options(${TARGET}
        PRIVATE
            "-Wl,--cref"
            "-Wl,-Map=${CMAKE_CURRENT_BINARY_DIR}/${MAP}"
        )

    target_sources(${TARGET}
        PUBLIC
        dataStorage.c
        debugPin.c
        fwStorage.c
        keyboard.c
        keyboardPhysicalLayout.c
        keyProcessor.c
        keyscan.c
        keyToHidCodeMap.c
        main.c
        PowerControl.c
        sleepManager.c
        time.c
        usbManager.c
        wdt.c
        hidUsb.c
        hid.c
        matrixIndexToKeyId.c
        effectLayer/effectLayer.c
        effectLayer/effectLayerManager.c
        effectLayer/effectParameters.c
        bakedInLighting/bakedInLighting.c
        lightingsSettings.c
        protocolIndex/keyIdToProtocolKeyIndex.c
        protocolIndex/protocolKeyIndexToProtocolLedIndex.c
        protocolIndex/protocolLedIndexToLedId.c
        util/colors.c
        util/general.c
        led/led.c
```

You can add your .c files here

3.4 Adding new header files

To add a new .h file you should be sure that the directory with your .h file is present in target_include_directories list.

```
40      FireworkLightingSystem
41      ActionPlaybackSystem
42      ActionRecorderSystem
43      ${LOG_LIBRARY}
44  )
45
46  target_include_directories(${TARGET}
47      PUBLIC
48      .
49      effectLayer
50      bakedInLighting
51      protocolIndex
52      led
53      ledDriver
54      log
55      myNewDirectory
56      util
57      profile
58      actionManager
59      journal
60      demoMode
61      swd
62      statusIndication
63      cco
64  )
```

If your .h file is not in app directory or not in that list, you should add directory for you header here

You should write here only directory name NOT file name

3.5 Changing project name and output binary name

To change project/binary name you can just change it in the first string of CMakeLists.txt file.

```
1  set (TARGET "LocustApplicationFirmware.elf")
2  string(REPLACE ".elf" ".map" MAP ${TARGET})
3  string(REPLACE ".elf" ".bin" BINARY ${TARGET})
4  string (TOUPPER ${CMAKE_BUILD_TYPE} BUILD_TYPE_DEFENITION)
5
6  add_executable(${TARGET})
7  add_subdirectory(bsp)
8  add_subdirectory(usb)
9  add_subdirectory(config)
```

If project uses different naming mechanism, that uses target list, it is changed here:

```
58  add_subdirectory (lib)
59  foreach (_TARGET Locust LocustWhite)
60      string (TOUPPER ${_TARGET} TARGET_DEFINITION)
61      set (DEFINES ${TARGET_DEFINITION} ${BUILD_TYPE_DEFENITION})
62      set (TARGET "${_TARGET}ApplicationFirmware.elf")
63      add_subdirectory (${APPLICATION_SOURCE_DIR} ${_TARGET})
64  endforeach()
65
```

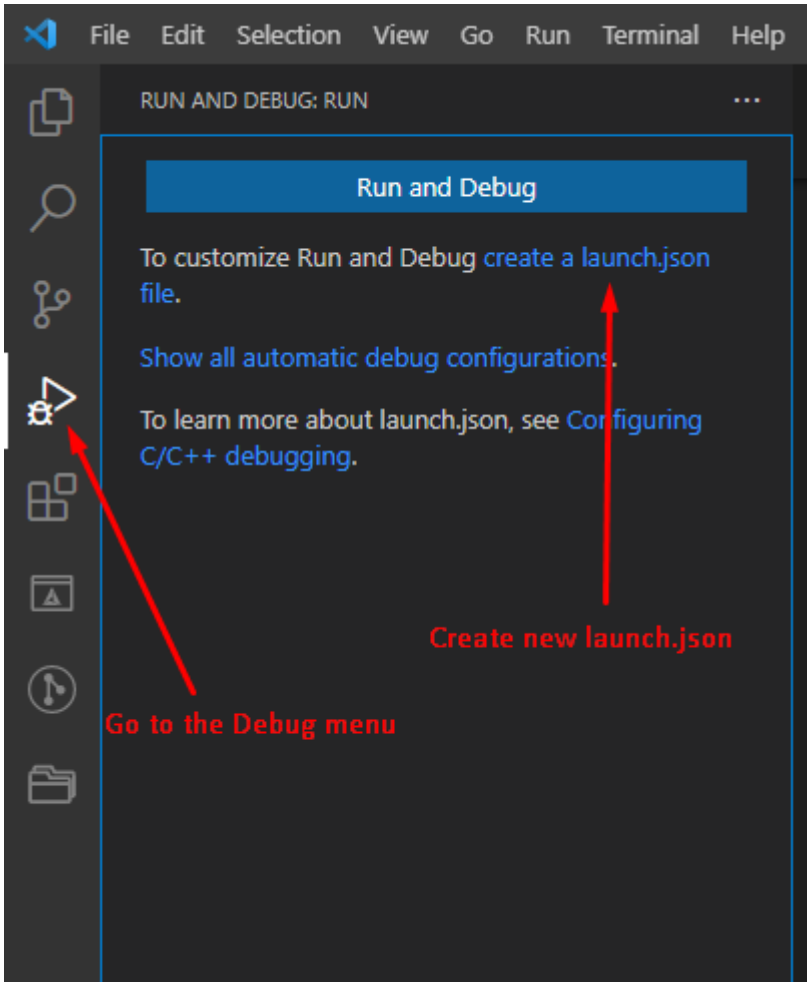
Add, remove or change first part of the output file name here

The rest can be changed here

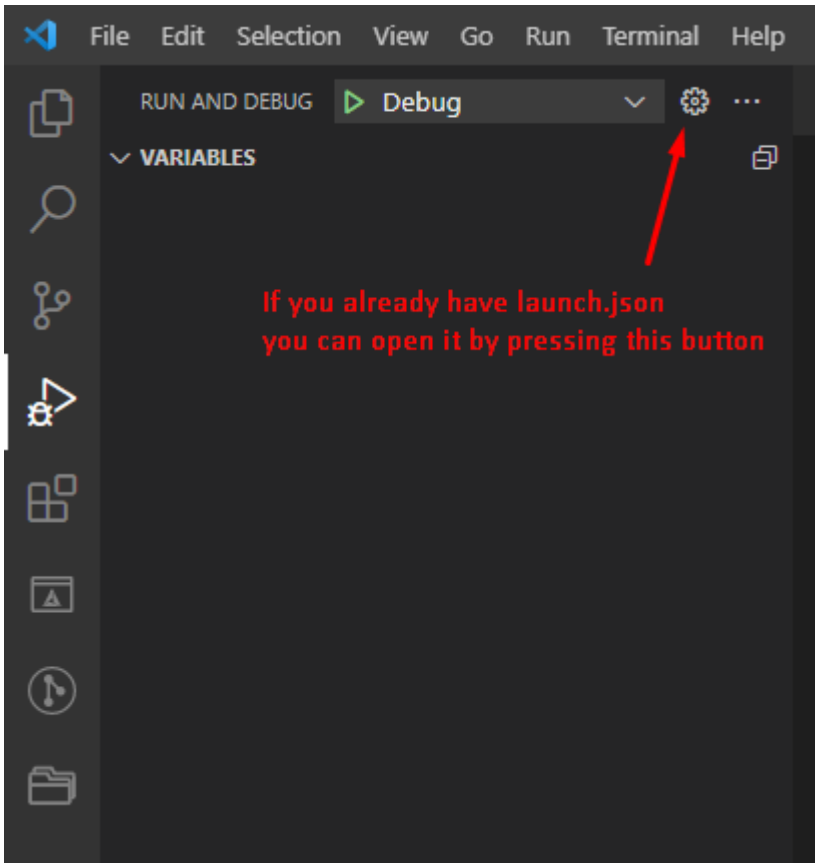
4. Setup Cortex Debug

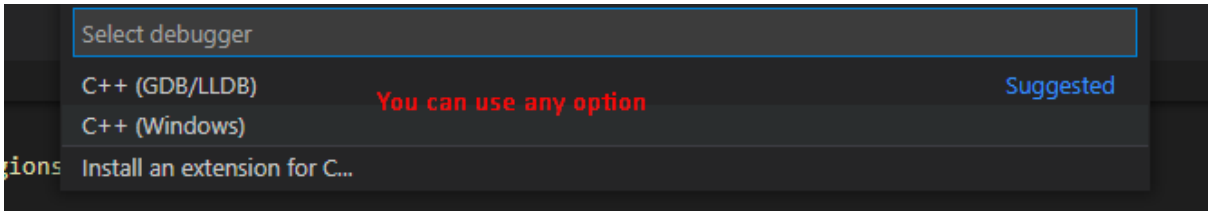
4.1 Setup launch.json

If you have no launch.json file you need to create it.



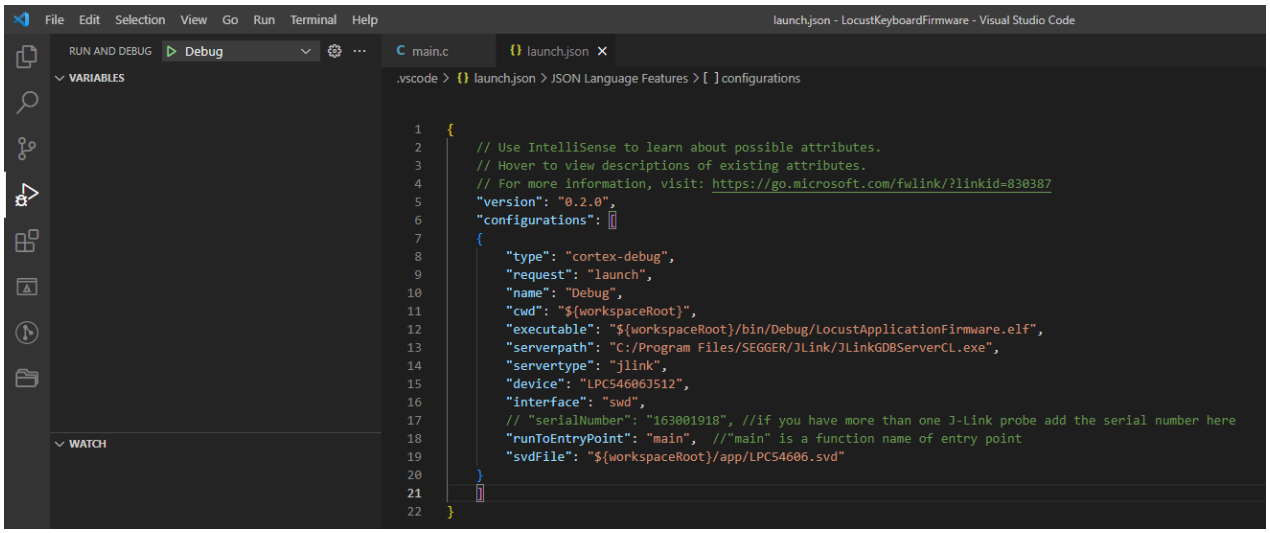
If you already have a launch.json file, you need to edit it.


File
Edit
Selection
View
Go
Run
Terminal
Help
RUN AND DEBUG
Debug
VARIABLES
If you already have launch.json
you can open it by pressing this button

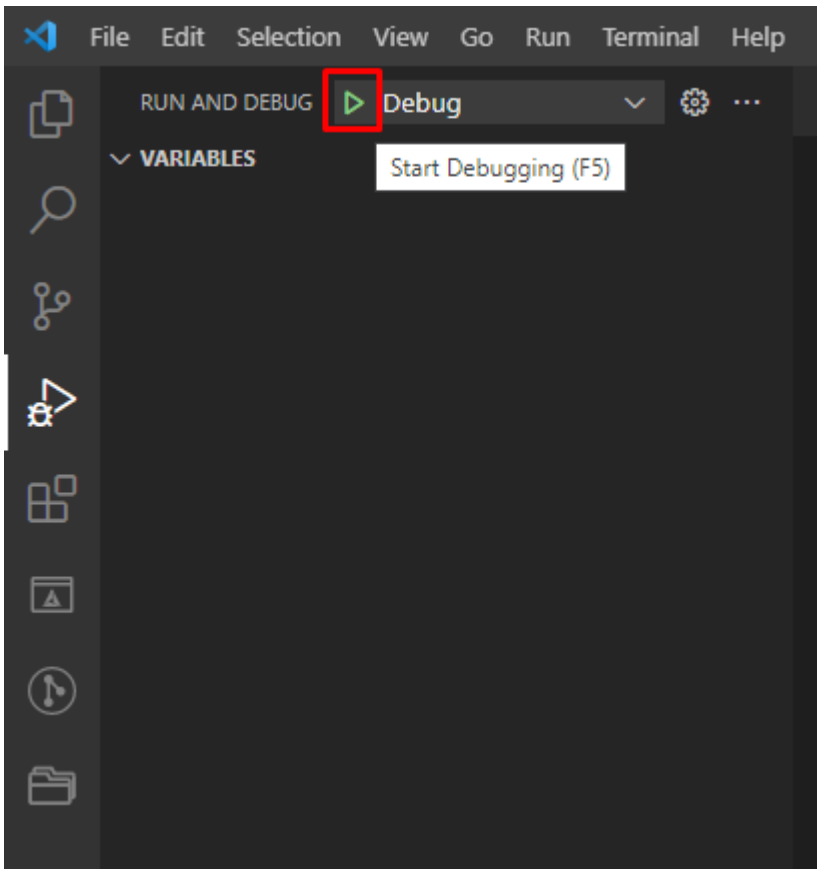

Select debugger
C++ (GDB/LLDB)
Suggested
C++ (Windows)
You can use any option
Install an extension for C...

When you open launch.json, please, copy paste text below:

```
{
    // 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": [
    {
        "type": "cortex-debug",
        "request": "launch",
        "name": "Debug",
        "cwd": "${workspaceRoot}",
        "executable":
"${workspaceRoot}/bin/Debug/LocustApplicationFirmware.elf",
        "serverpath": "C:/Program
Files/SEGGER/JLink/JLinkGDBServerCL.exe",
        "servertype": "jlink",
        "device": "LPC54606J512",
        "interface": "swd",
        // "serialNumber": "163001918", //if you have more than one J-
Link probe add the serial number here
        "runToEntryPoint": "main",  //"main" is a function name of
entry point
        "svdFile": "${workspaceRoot}/app/LPC54606.svd",
        "postLaunchCommands": [
      "-interpreter-exec console \"mem 0x00000000 0x0007FFFF rw\"",
      "-interpreter-exec console \"mem 0x20000000 0x20027FFF rw\"",
      "-interpreter-exec console \"mem 0x04000000 0x04007FFF rw\"",
      "-interpreter-exec console \"mem 0x40100000 0x40101FFC rw\"",
      "-interpreter-exec console \"mem 0x40085000 0x40100000 rw\"",
      "-interpreter-exec console \"mem 0x40082000 0x40090000 rw\"",
"-interpreter-exec console \"set mem inaccessible-by-default on\"",
"-interpreter-exec console \"set remote hardware-breakpoint-limit 5\"",
"-interpreter-exec console \"set remote hardware-watchpoint-limit 3\"",
"-interpreter-exec console \"tpiu config internal itm.fifo uart off
8000000\""
    ]
   }
}
```



After you save launch.json you can run Debug by pressing F5 or by press Debug button



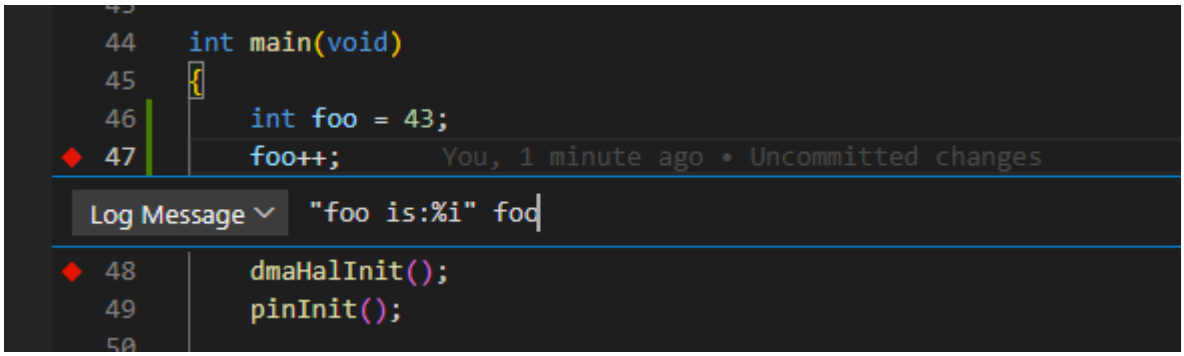
5. Using Cortex Debug

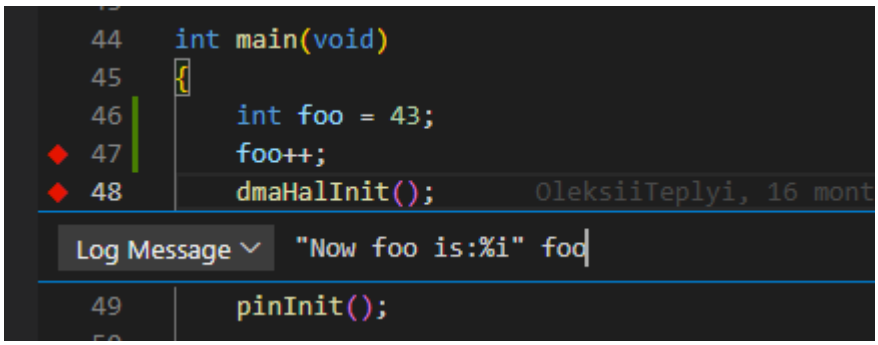
5.1 Basic stepping over code, watch and variables, call stack

You can find a lot of useful information about debugging in VS Code on that link: https://code.visualstudio.com/docs/editor/debugging

5.2 Logpoints

Logpoints may help you to use logging without any firmware modifications. It works similar to breakpoints, but it shows you messages instead of just stop code. It works something like that:
Breakpoint happen -> Write message -> Run code
So it is expected that your code may take more time if Logpoints happen often.
Log points in Cortex Debug work in a bit different format than described in VS Code docs. Logpoints in Cortex Debug is more like printf function, but you need to use ‘ ‘ (space) instead of ‘,’ (comma). For example:
“Hello World! My number is:%u, My var is:%i“ 42 myVariable
But you need to be sure that at the moment when break happens, that variable is reached.





```
44  int main(void)
45  {
46      int foo = 43;
47      foo++;
48      dmaHalInit();
```

Log Message "Now foo is:%i" foo

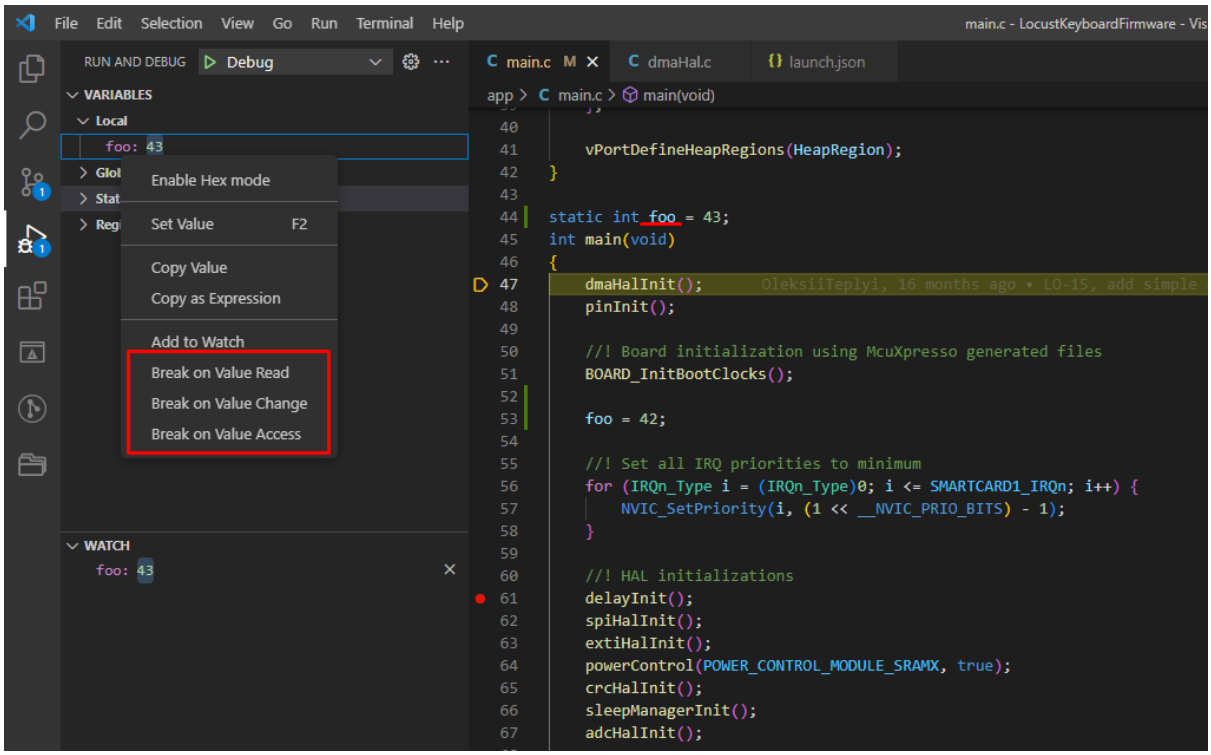
```
49      pinInit();
50
51      //! Board initialization using McuXpresso generated files
52      BOARD_InitBootClocks();        kyrylo.babkin, 17 months ago • DMF-509 initial commit
53
54      //! Set all IRQ priorities to minimum
55      for (IRQn_Type i = (IRQn_Type)0; i <= SMARTCARD1_IRQn; i++) {
56          NVIC_SetPriority(i, (1 << __NVIC_PRIO_BITS) - 1);
57      }
58
59      //! HAL initializations
60      delayInit();
61      spiHalInit();
62      extiHalInit();
63      powerControl(POWER_CONTROL_MODULE_SRAMX, true);
64      crcHalInit();
65      sleepManagerInit();
66      adcHalInit();
67
68      // initialize multiple heap regions
69      // if anything uses vPortMalloc, function below
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

```
Program
 received signal SIGTRAP, Trace/breakpoint trap.
POWER_EnterSleep () at C:/mywork/mcu-sdk-1/mcu-sdk-2.0/devices/LPC54608/fsl_power_lib/fsl_power_lib.c:263
263     C:/mywork/mcu-sdk-1/mcu-sdk-2.0/devices/LPC54608/fsl_power_lib/fsl_power_lib.c: No such file or directory.
Resetting target

Temporary breakpoint 4, main () at C:\workspace\LocustKeyboardFirmware\app\main.c:46
46          int foo = 43;
foo is:43
Now foo is:44

Breakpoint 9, main () at C:\workspace\LocustKeyboardFirmware\app\main.c:52
52          BOARD_InitBootClocks();
```

5.3 Memory breakpoints

Memory breakpoints may help you to find a place of variable modification. To use it, find a variable in the Debug menu and right click on it, then you can add a breakpoint there.


File Edit Selection View Go Run Terminal Help
main.c - LocustKeyboardFirmware - Vis
RUN AND DEBUG
Debug
main.c M
dmaHal.c
launch.json
app > main.c > main(void)
VARIABLES
Local
foo: 43
Enable Hex mode
Set Value F2
Copy Value
Copy as Expression
Add to Watch
Break on Value Read
Break on Value Change
Break on Value Access
WATCH
foo: 43
vPortDefineHeapRegions(HeapRegion);
}
static int foo = 43;
int main(void)
{
dmaHalInit();
pinInit();
//! Board initialization using McuXpresso generated files
BOARD_InitBootClocks();
foo = 42;
//! Set all IRQ priorities to minimum
for (IRQn_Type i = (IRQn_Type)0; i <= SMARTCARD1_IRQn; i++) {
NVIC_SetPriority(i, (1 << __NVIC_PRIO_BITS) - 1);
}
//! HAL initializations
delayInit();
spiHalInit();
extiHalInit();
powerControl(POWER_CONTROL_MODULE_SRAMX, true);
crcHalInit();
sleepManagerInit();
adcHalInit();

File Edit Selection View Go Run Terminal Help — main.c - LocustKeyboardFirmware - Visua

RUN AND DEBUG ▷ Debug

VARIABLES
- Local
- Global
- Static
  - foo: 43
- Registers

WATCH
- foo: 43

CALL STACK — Paused on entry
- main@0x0000c316 C:\workspace\LocustKeyboar...

BREAKPOINTS
- foo Write
- main.c app 76

CORTEX PERIPHERALS

CORTEX REGISTERS

main.c M | dmaHal.c | launch.json

app > main.c > main(void)

```c
}

static int foo = 43;
int main(void)
{
    dmaHalInit();        OleksiiTeplyi, 16 months ago • LO-15, add simple dm
    pinInit();

    //! Board initialization using McuXpresso generated files
    BOARD_InitBootClocks();

    //! Set all IRQ priorities to minimum
    for (IRQn_Type i = (IRQn_Type)0; i <= SMARTCARD1_IRQn; i++) {
        NVIC_SetPriority(i, (1 << __NVIC_PRIO_BITS) - 1);
    }

    //! HAL initializations
    delayInit();
    spiHalInit();
    extiHalInit();
    powerControl(POWER_CONTROL_MODULE_SRAMX, true);
    crcHalInit();
    sleepManagerInit();
    adcHalInit();

    // initialize multiple heap regions
    // if anything uses vPortMalloc, function below
    // must be called before that!
    // and only makes sense if heap_5 is used
    heapInit();
    //! Start operation

    foo = 42;
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | GITLENS

```
[main] Building folder: LocustKeyboardFirmware
[build] Starting build
[proc] Executing command: "C:\Program Files\CMake\bin\cmake.EXE" --build c:/works
[build] [  2%] Built target LittleFS
[build] [ 38%] Built target sdk
[build] [ 46%] Built target FreeRTOS
[build] Consolidate compiler generated dependencies of target LocustApplicationFi
[build] [ 47%] Building C object app/CMakeFiles/LocustApplicationFirmware.elf.dir
[build] [ 47%] Linking C executable LocustApplicationFirmware.elf
[build] Memory region         Used Size  Region Size  %age Used
[build]           FLASH:      225976 B       512 KB     43.10%
[build]            RAMX:         32 KB        32 KB    100.00%
[build]             RAM:      149136 B       160 KB     91.03%
```

develop* | 6↓ 0↑ | ⊗ 0 ⚠ 0 | Debug (LocustKeyboardFirmware) | CMake: [Debug]: Ready | No active kit | Build [all] | LocustKeyboardFirmware

File Edit Selection View Go Run Terminal Help — main.c - LocustKeyboardFirmware - Visual Studio Code

RUN AND DEBUG ▷ Debug

VARIABLES
- Local
- Global
- Static
  - foo: 42
- Registers

WATCH
- foo: 42

CALL STACK — Paused on data breakpoint
- vTaskStartScheduler@0x00022a7c C:\worksp...
- main@0x0000c376 C:\workspace\LocustKeyboa...

BREAKPOINTS
- foo Write
- main.c app 76

CORTEX PERIPHERALS

CORTEX REGISTERS

main.c M | tasks.c | dmaHal.c | launch.json

app > main.c > main(void)

```c
    spiHalInit();
    extiHalInit();
    powerControl(POWER_CONTROL_MODULE_SRAMX, true);
    crcHalInit();
    sleepManagerInit();
    adcHalInit();

    // initialize multiple heap regions
    // if anything uses vPortMalloc, function below
    // must be called before that!
    // and only makes sense if heap_5 is used
    heapInit();
    //! Start operation

    foo = 42;

    vTaskStartScheduler();        kyrylo.babkin, 17 months ago • DMF-509 initial commit
}

void vApplicationDaemonTaskStartupHook()
{
    batteryManagerInit();
    wdtTaskInit();
    dataStorageInit();
    keyProcessorStart();
    usbStart();
    ccoManagerStart();
    effectLayerManagerStart();
    actionManagerStart();
    wirelessManagerStart();
    keyboardStart();
}
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | GITLENS

```
Debug-13: Elapsed time for Disassembly Request: 14298 ms

Hardware watchpoint 3: foo

Old value =
43
New value =
42
vTaskStartScheduler () at C:\workspace\LocustKeyboardFirmware\lib\FreeRTOS\tasks.c:2001
2001                    xReturn = xTaskCreate(  prvIdleTask,
Debug-28: Enqueuing {"command":"disassemble","arguments":{"memoryReference":"0x00022a7c","o
a":28}
```

develop* | 6↓ 0↑ | ⊗ 0 ⚠ 0 | Debug (LocustKeyboardFirmware) | CMake: [Debug]: Ready | No active kit | Build [all] | LocustKeyboardFirmware