# Software needed

Installing the whole package of available utilities: IDE, compiler etc is a relatively complex process. There are many installation methods/sources. The sequence of steps below is one of the variants that allow a complete installation.

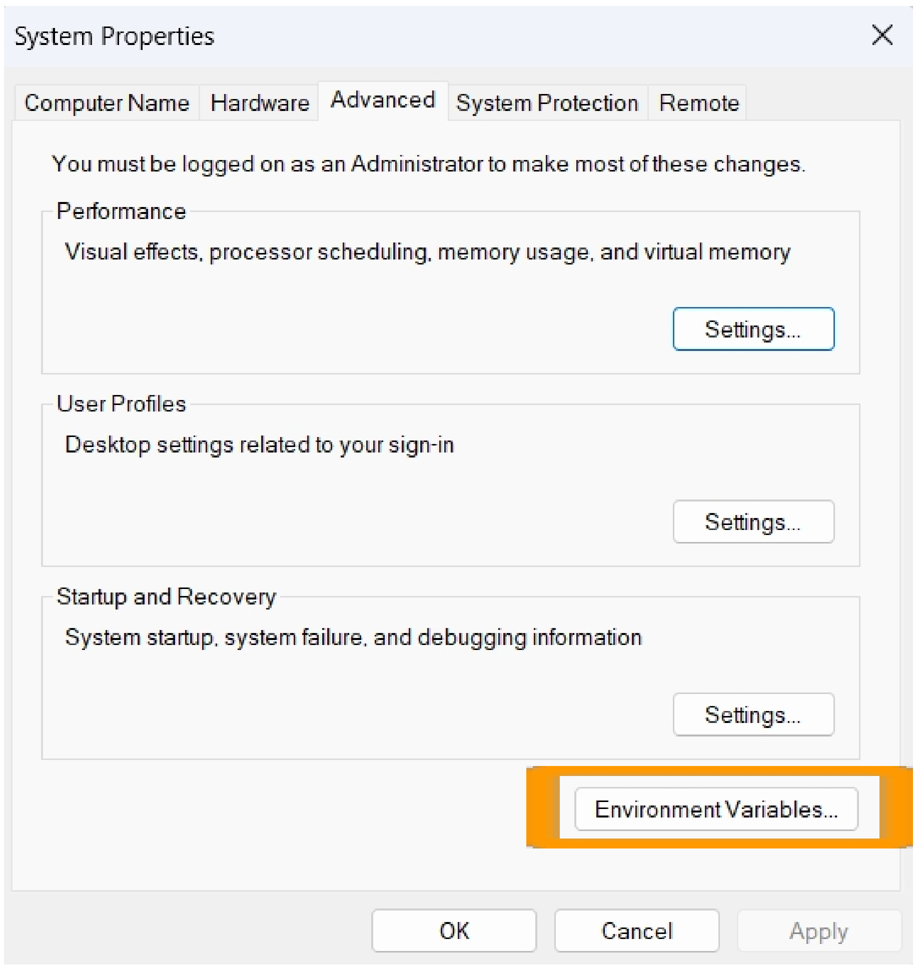
## 1. Install STM32 utilities

**1.1 Install STM32 CubeMX CubeMX** - automatic initialization code generator
Install from here: https://www.st.com/en/development-tools/stm32cubemx.html
It may be necessary to create an ST account for download. This package will be needed to make changes to the software, not to compile and load the initial test software already provided.

**1.2 STM32 Cube CLT installation -** this package contains, among other things, the main GNU C/C++ compiler (called "Gcc toolchain", gcc-arm-eabi variant for ARM "Embedded Application Binary Interface" processors) and other components.
Install from here: https://www.st.com/en/development-tools/stm32cubeclt.html

**1.3 Install Make for Windows**: is the utility that allows working with makefiles generated by STM32CubeMX.
This package is generic, not just for STM32, so install it directly from source (GNU):
https://gnuwin32.sourceforge.net/packages/make.htm
from here download the "Binaries" and "Dependencies" packages.
Unpack both archives in **the same** folder of your choice, created in advance. For example, C:\\ST\make-3.81 (or whatever specific version was downloaded). Note that some of the archive folders are common, but the files are not the same.

**1.3.1 Set *PATH* under Windows to include Make:**
search for "System Properties" in Windows:



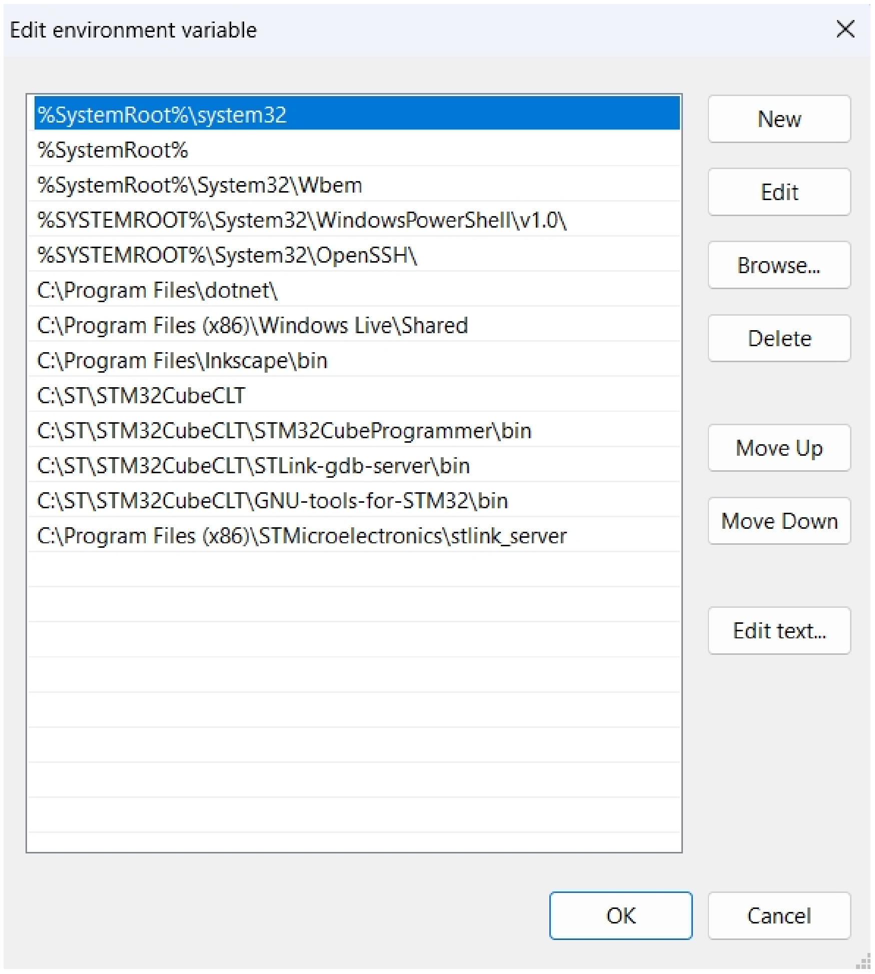
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

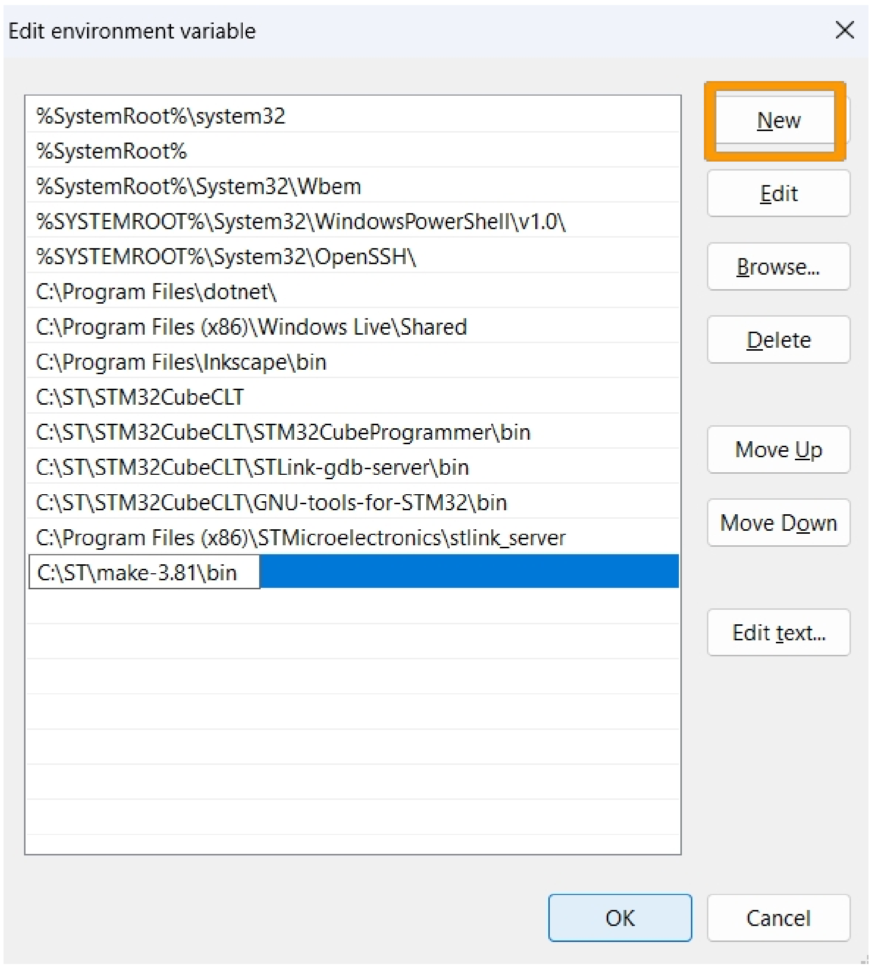
Startup and Recovery
System startup, system failure, and debugging information

Select "Environment Variables" then "System variables" (note, not "user") and select Path→ Edit



```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\dotnet\
C:\Program Files (x86)\Windows Live\Shared
C:\Program Files\Inkscape\bin
C:\ST\STM32CubeCLT
C:\ST\STM32CubeCLT\STM32CubeProgrammer\bin
C:\ST\STM32CubeCLT\STLink-gdb-server\bin
C:\ST\STM32CubeCLT\GNU-tools-for-STM32\bin
C:\Program Files (x86)\STMicroelectronics\stlink_server
```

Press the "New" button to add a new line in Path; manually add the "bin" folder from the folder where the 2 Make archives were unpacked:



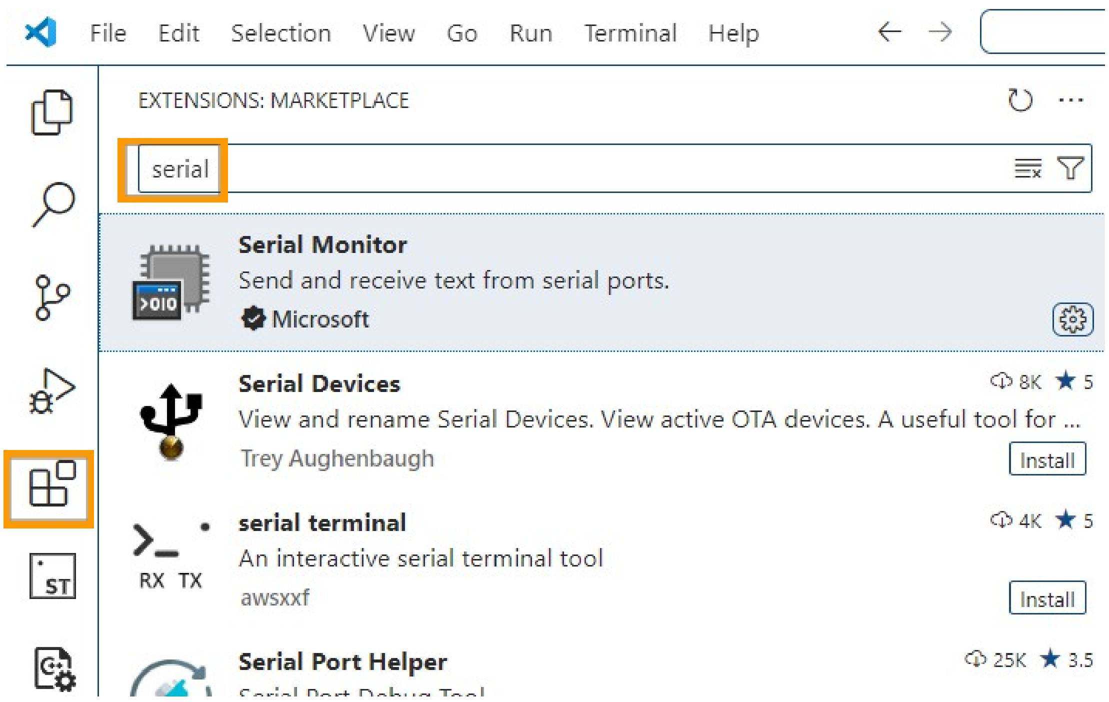
Edit environment variable

```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\dotnet\
C:\Program Files (x86)\Windows Live\Shared
C:\Program Files\Inkscape\bin
C:\ST\STM32CubeCLT
C:\ST\STM32CubeCLT\STM32CubeProgrammer\bin
C:\ST\STM32CubeCLT\STLink-gdb-server\bin
C:\ST\STM32CubeCLT\GNU-tools-for-STM32\bin
C:\Program Files (x86)\STMicroelectronics\stlink_server
C:\ST\make-3.81\bin
```

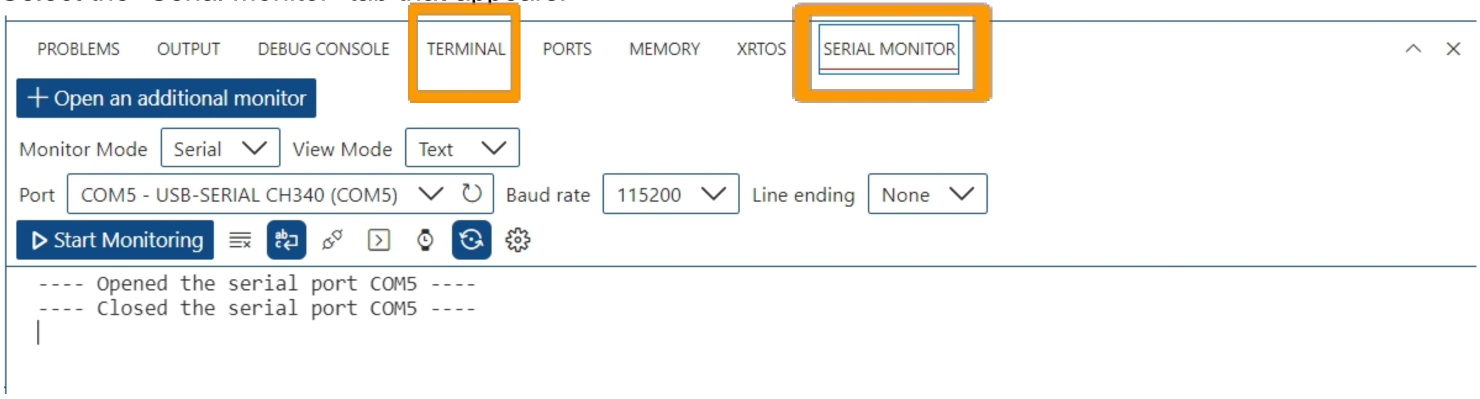
2. **Install Visual Studio Code (VS Code).**

   This is the IDE for writing code and calling the compiler and other tools. Install from here: https://code.visualstudio.com/download

**2.1 Install VS Code extension**: serial terminal
Purpose: To install and test a serial terminal (*serial monitor*). Start VS Code installed above. Press button ⊞ (Extensions manager), type "Serial" in the text box, several variants will appear. Select "Serial Monitor" (Microsoft) and press the *Install* button. After installation, the installed extensions no longer show the *Install* button but the "cogwheel" for settings.


File Edit Selection View Go Run Terminal Help
EXTENSIONS: MARKETPLACE
serial
Serial Monitor
Send and receive text from serial ports.
Microsoft
Serial Devices
8K 5
View and rename Serial Devices. View active OTA devices. A useful tool for ...
Trey Aughenbaugh
Install
serial terminal
4K 5
An interactive serial terminal tool
awsxxf
Install
Serial Port Helper
25K 3.5

Note that in VS Code, there is not only *Serial Monitor*, but also a command line called "Terminal". If the window with the 2 tabs "Terminal" and "Serial Monitor" doesn't appear as below, on the right of the screen, bottom half, the command CTRL-J (Toggle Panel) makes it appear/disappear.
Select the "Serial Monitor" tab that appears:


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS MEMORY XRTOS SERIAL MONITOR
+ Open an additional monitor
Start Monitoring

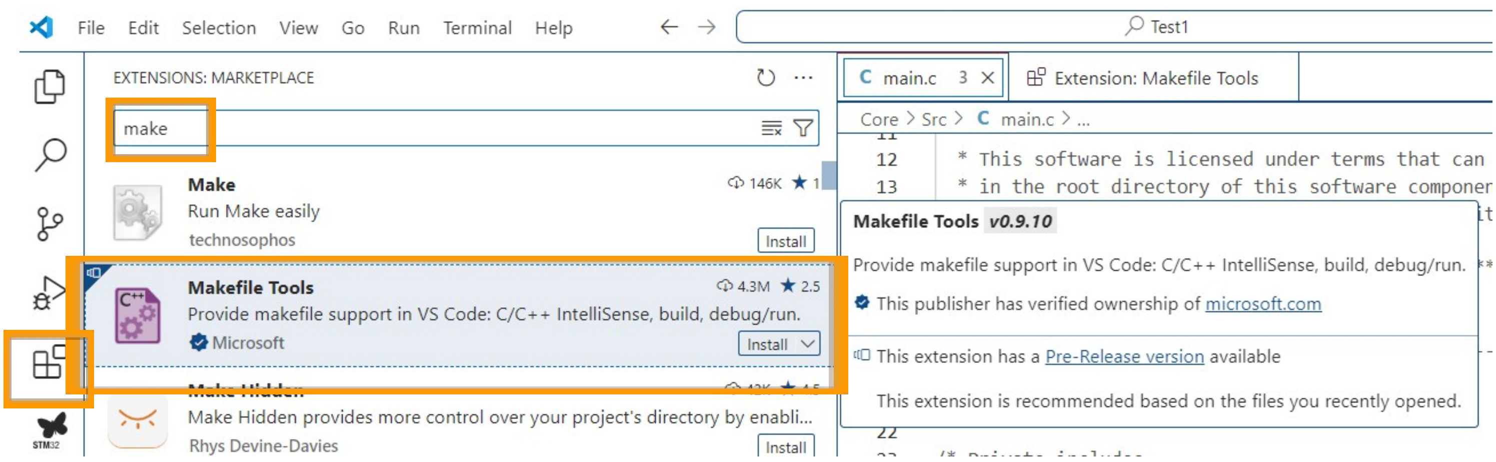
Monitor Mode: Serial | View Mode: Text
Port: COM5 - USB-SERIAL CH340 (COM5) | Baud rate: 115200 | Line ending: None

```
---- Opened the serial port COM5 ----
---- Closed the serial port COM5 ----
```

Check the available ports with nothing connected to the PC. Connect a TTL-USB adapter. Check the available serial ports again, noting what has newly appeared (example on the figure: COM5, the figure may vary). If it does not appear, there is a problem with the Windows driver, search the Internet for "CH340 Windows driver" and install it manually.

**2.2 You can also install the following VSCode extensions** by going to the extensions button and typing the first few letters, e.g. "make" in the search box:

1. Makefile Tools (Microsoft)

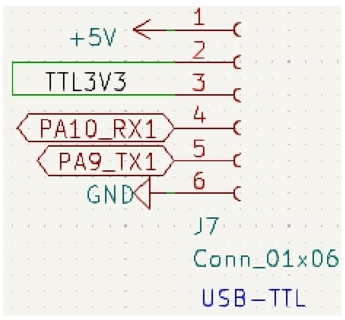

File Edit Selection View Go Run Terminal Help
Test1
EXTENSIONS: MARKETPLACE
make
Make
146K 1
Run Make easily
technosophos
Install
Makefile Tools
4.3M 2.5
Provide makefile support in VS Code: C/C++ IntelliSense, build, debug/run.
Microsoft
Install
Make Hidden
Make Hidden provides more control over your project's directory by enabli...
Rhys Devine-Davies
Install
main.c 3
Extension: Makefile Tools
Core > Src > main.c > ...
Makefile Tools v0.9.10
Provide makefile support in VS Code: C/C++ IntelliSense, build, debug/run.
This publisher has verified ownership of microsoft.com
This extension has a Pre-Release version available
This extension is recommended based on the files you recently opened.

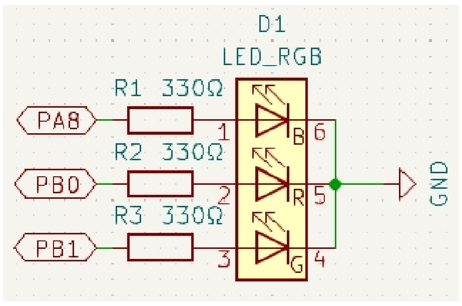
2. C/C++ Intellisense, debugging, and code browsing (Microsoft) - this will install a few more tools
3. STM32-for-VSCode (Bureau Moeilijke Moeilijke Dingen)
4. CSK Terminal - *optional but recommended* - another serial terminal that allows continuous transmission of characters without pressing ENTER after each character; useful to visualize serial characters on the oscilloscope without using SINGLE SWEEP mode

## 3. <u>Loading test software into the processor</u>

The STM32F103 processor contains a *bootloader* in the ROM that allows loading using the internal serial port UART1 (pins PA9, PA10). These pins are available at connector J7 to which a USB-TTL adapter set to 3.3V will be connected. A *bootloader* is a super-minimal operating system variant that only allows user software to be loaded into the processor. It can only communicate with a PC program (sometimes called a *PC-loader*) specially written according to a protocol compatible between the 2.

***Important note:*** in the figure below, pins 2,3 of the J7 are shorted to each other, which is equivalent to setting the jumper to position "3V3" when using the TTL-USB adapter separately. This is because all signals from the STM32 chip are at 3.3V and can be destroyed if it receives 5V on a pin! The 5V voltage is only used for the input of a stabilizer which produces 3.3V for all components on the board.

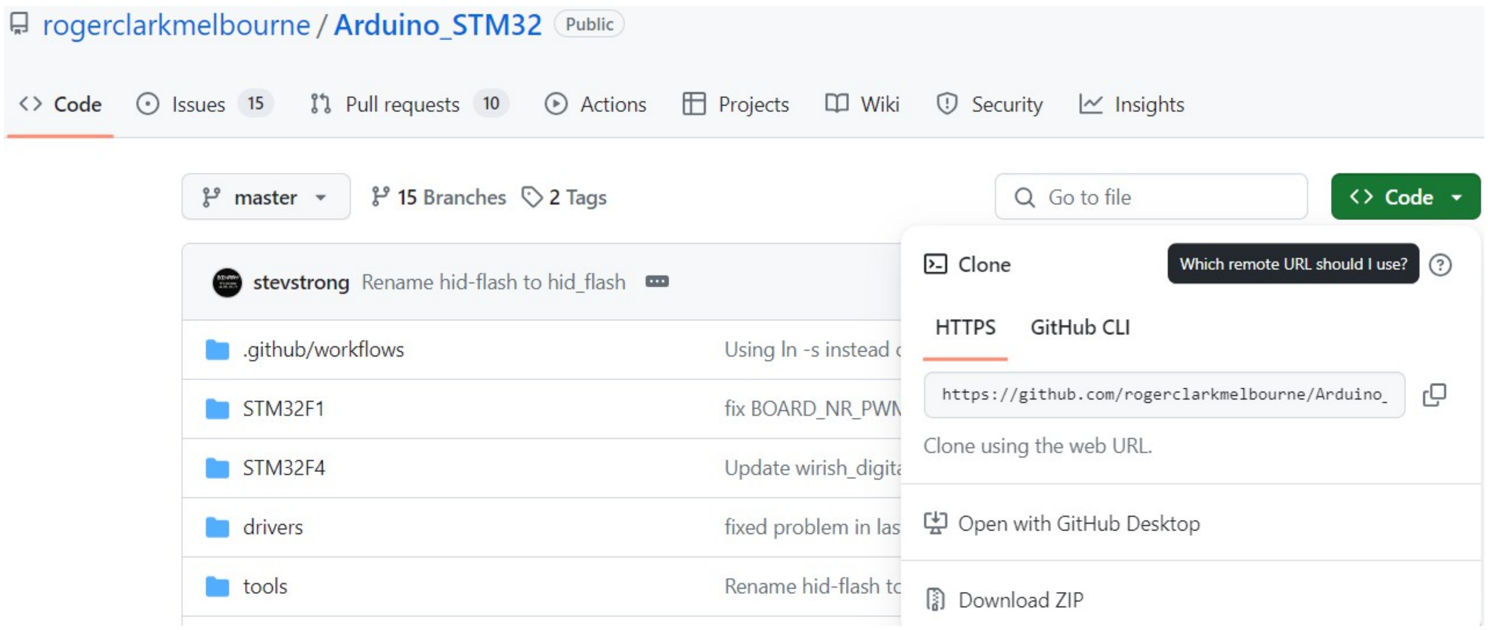




There is also a tricolor LED on the board consisting of 3 R,G,B LEDs connected to PB0, PB1, PA8. These 3 LEDs can be used to signal program operation. When ROM *bootloader* is active, the LEDs are off. When the user program is active, the LEDs will blink as defined in the program. This way we know who is running: *the bootloader* or the user program.

### 3.1 Installing serial bootloader program (STM32Flash) on PC

Download from here: https://github.com/rogerclarkmelbourne/Arduino_STM32/tree/master
(note that these tools are not Arduino specific, but can be used in the Arduino environment, hence the name).
select the <> Code button on the page and from there "download ZIP"


rogerclarkmelbourne / Arduino_STM32 Public

Code | Issues 15 | Pull requests 10 | Actions | Projects | Wiki | Security | Insights

master | 15 Branches | 2 Tags | Go to file | Code

stevstrong Rename hid-flash to hid_flash

| | |
|---|---|
| .github/workflows | Using ln -s instead ( |
| STM32F1 | fix BOARD_NR_PWN |
| STM32F4 | Update wirish_digit |
| drivers | fixed problem in las |
| tools | Rename hid-flash to |

Clone | Which remote URL should I use?

HTTPS | GitHub CLI

```
https://github.com/rogerclarkmelbourne/Arduino_
```

Clone using the web URL.

Open with GitHub Desktop

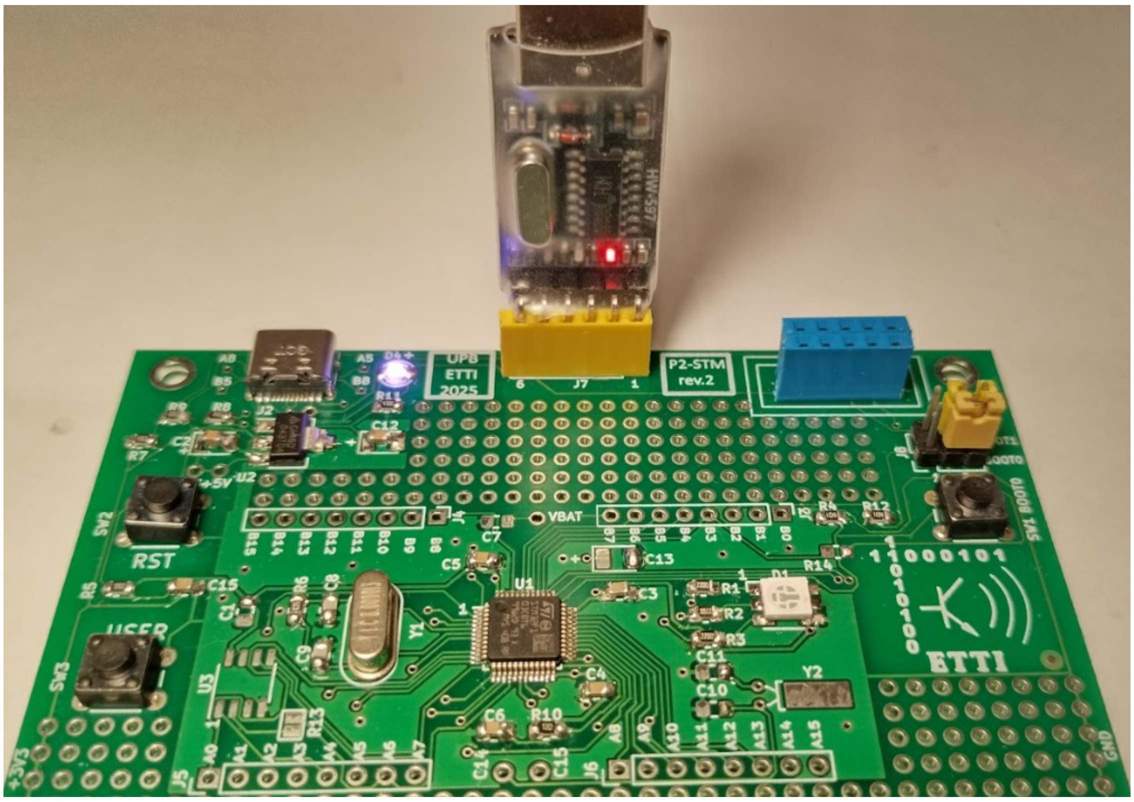
Download ZIP

Extract from the `tools` archive→ `win`→ `stm32flash.exe` into a working folder, e.g. `c:\project2` . This is the PC program that communicates with the bootloader on the microcontroller for loading a user program.

**3.2 Loading the test software using the serial port** (with USB-TTL serial adapter)

Download the ready compiled STM32F103 test software in binary format from here:
[http://ham.elcom.pub.ro/proiect2/STM32/Test1.bin](http://ham.elcom.pub.ro/proiect2/STM32/Test1.bin) . Copy the .bin file into the same folder as the executable stm32flash.exe, e.g. c:\project2

Connect the USB-TTL adapter to the board. ***Attention!*** Make sure that pin 1 of J7 on the board is connected to the "5V" pin of the adapter! *Do not connect the adapter backwards*, beware that the J7 does not have a key to allow single orientation.
Backwards connection may cause the board to fail, since the adapter also supplies power to the board.



From the Windows Device Manager under Ports: COM and LPT check for a USB-Serial CH340 port and its port number, e.g. COM5.

- Set jumpers BOOT0, BOOT1 to 0 (right).
- keep SW1 - BOOT0 button pressed down
- short press the RST (Reset) button and release. At this moment, the processor starts after reset and finds BOOT0=1 (because of the button) so the execution of the software will be done from the "system memory" where the bootloader is located
- release button SW1 - BOOT0
- on the PC, open a *Command Prompt*, switch to the folder that contains both the executable and the file .bin (for example: `cd c:\project2` ), then run: (change the COM port number as necessary)

```
stm32flash.exe -b 115200 -w test1.bin COM5
```

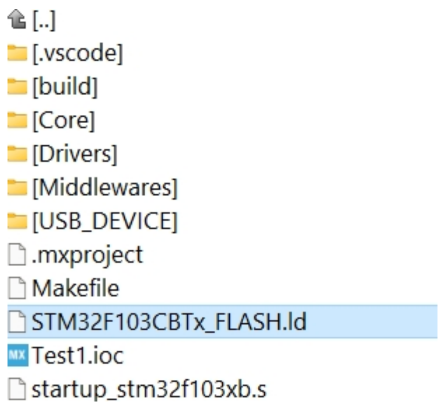
  wait for programming to finish with the message *Wrote address xxxxx (100%) Done*
- press briefly Reset; at this reset, the processor starts, it does not find BOOT0 pressed so the program execution is done from Flash, where the recently loaded user program is.

*Alternatively*, you can not use the BOOT0 button at all, but set the BOOT0 jumper to 1, press briefly and release Reset, then set the BOOT0 jumper to 0 and execute the program command. As you can see from the schematic, the SW1 - BOOT0 button is in parallel with the respective jumper - use one or the other according to preference.

After loading the test software watch for the RGB LED to blink in succession. This indicates successful programming and operation of the test software. Also, on the UART1 port (connector J7), any X character received at 9600bps will return X+1, pressing ? will read the version number and the button will change the blink rate.

## 4. Recompiling the test software

This step verifies that *the* toolchain is correctly installed and allows the creation of the Text1.bin binary from the source files.
Download the test software source archive - the test1.zip file - from http://ham.elcom.pub.ro/proiect2/STM32.
Unzip it into any folder, it will unpack the following directory structure into a folder named for exampleTest1:



```
[..]
[.vscode]
[build]
[Core]
[Drivers]
[Middlewares]
[USB_DEVICE]
.mxproject
Makefile
STM32F103CBTx_FLASH.ld
Test1.ioc
startup_stm32f103xb.s
```

The `Test1.ioc` file is the project for STM32CubeMX. The source code generated by this utility (to which I added the specific lines) is in **Core\Src** (the sources themselves) and **Core\Inc** (the #include files). The compilation and linking process will produce the executables `Test1.bin`, `Test1.elf`, `Test1.hex` in the **build**. All represent standard formats for several uC families. The .elf extension stands for "Executable and Linkable Format", the .hex extension is "Intel Hex" and .bin is a binary variant. All 3 formats are equivalent, but will

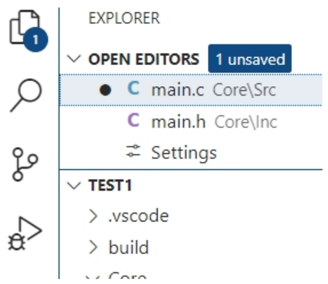
use one or the other depending on what the uploading software you are using supports (in our examples we use .bin). Watch the date and time of the file after you recompile to make sure it is the one produced at that time.
Compiling (and linking) is done using the **make** utility which reads and applies the instructions in the Makefile. To recompile, go to the folder containing the Makefile and, making sure you set the PATH for "Make" correctly in step 1.3, issue the command:

```
make all
```

If successful, messages of the following type will appear to confirm the executables produced:

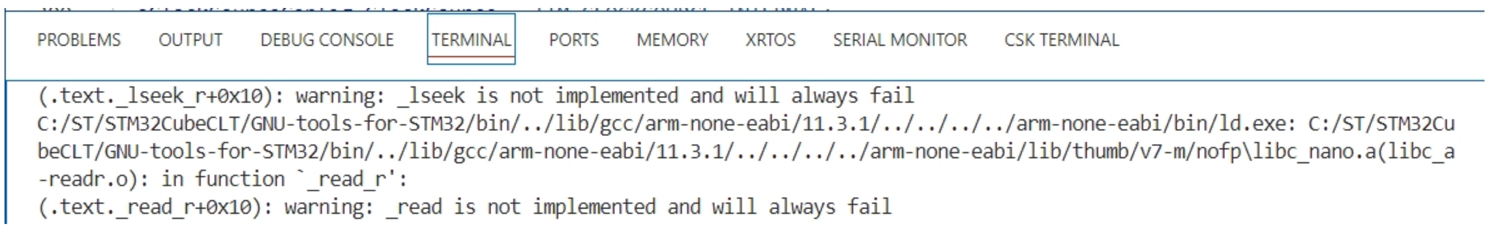
```
arm-none-eabi-size build/Test1.elf
   text     date      bss      dec      hex filename
  24660      480     6632    31772     7c1c build/Test1.elf
arm-none-eabi-objcopy -O ihex build/Test1.elf build/Test1.hex
arm-none-eabi-objcopy -O binary -S build/Test1.elf build/Test1.bin
```

By now you have recompiled the test code already provided. *Arm-None-EABI* refers to the GCC compiler variant used, for the ARM family of microcontrollers (GCC also exists for Intel X86, etc).
For code changes, open the code in the **Visual Studio Code** IDE installed in step 1. Select File→ Open Folder and select the folder where you unzipped the Test1 archive (where the Makefile is located). To view and edit the files use the tool in the left sidebar. For extensions select the .
If you make any changes in the code, this will be highlighted in file explorer:



**Warning.** Save all files in bulk using File → Save All (**CTRL K S**) For recompilation:

- Either run **make all** as above from the already opened folder in Command Prompt
- either use the Terminal included in VS Code: if it does not appear, press CTRL+J, then select Terminal (not to be confused with the *serial terminal* called *Serial Monitor*, they are separate tabs in the figure below). From the Terminal you can issue the same commands: **make all**, etc. Also there you see warnings and error messages.



```
(.text._lseek_r+0x10): warning: _lseek is not implemented and will always fail
C:/ST/STM32CubeCLT/GNU-tools-for-STM32/bin/../lib/gcc/arm-none-eabi/11.3.1/../../../../arm-none-eabi/bin/ld.exe: C:/ST/STM32Cu
beCLT/GNU-tools-for-STM32/bin/../lib/gcc/arm-none-eabi/11.3.1/../../../../arm-none-eabi/lib/thumb/v7-m/nofp\libc_nano.a(libc_a
-readr.o): in function `_read_r':
(.text._read_r+0x10): warning: _read is not implemented and will always fail
```

**The make** command will produce nothing if it detects that the source files have not changed since the last make.