Setting up Embedded Rust for Microbit (v1.1)

Sep 2025 ENR325

The most pain for doing embedded, is not the coding, but setting up your coding stuff. Embedded is NOT hard. A vending machine is an embedded system. Who's afraid of a vending machine?

This manual is built based on <micro::bit v2 Embedded Discovery Book>. Thanks to Embedded Working Group at the Rust community.

1) Tooling:

1.1) Install Rust:

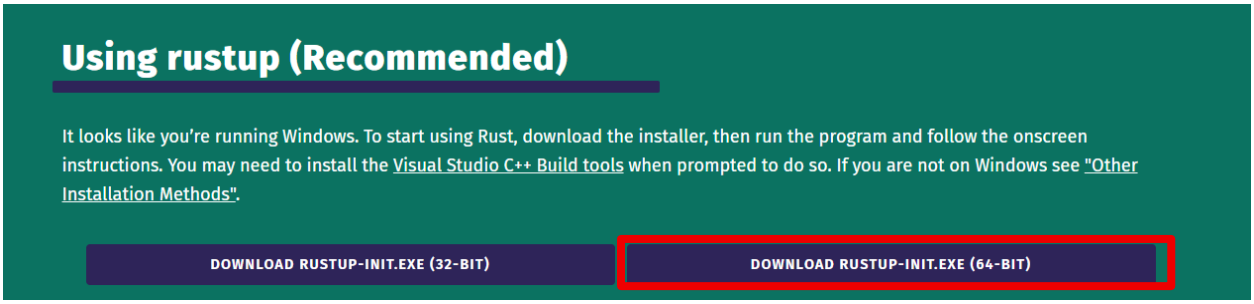
https://www.rust-lang.org/tools/install

*For Linux or Mac, it's just a cmd line.

For installation on windows, go to:

https://www.rust-lang.org/tools/install

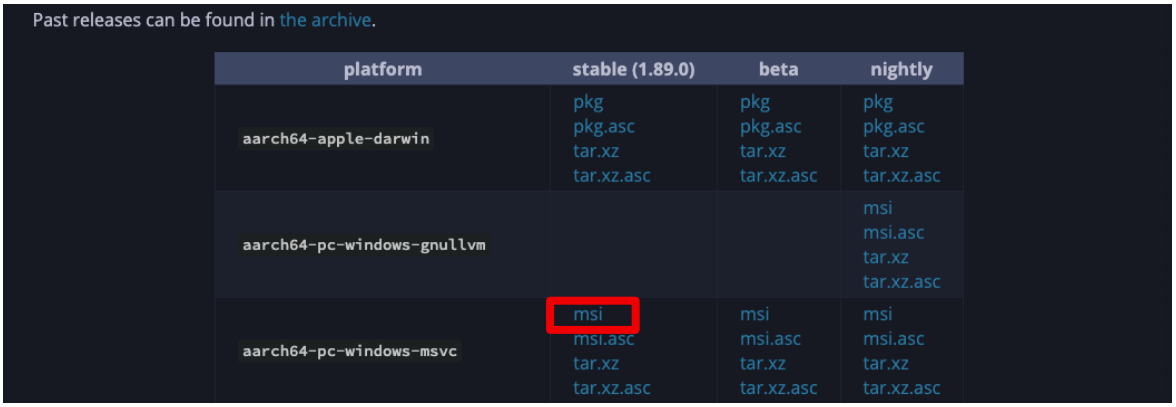
Download the exe file. Most likely your PC is 64-BIT.



Or you can go to:

https://forge.rust-lang.org/infra/other-installation-methods.html

Look for the mirror (msi) file:



Past releases can be found in the archive.

platform	stable (1.89.0)	beta	nightly
aarch64-apple-darwin	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc	pkg pkg.asc tar.xz tar.xz.asc
aarch64-pc-windows-gnullvm			msi msi.asc tar.xz tar.xz.asc
aarch64-pc-windows-msvc	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc	msi msi.asc tar.xz tar.xz.asc

When you double click the exe file, you will open a cmd window. Choose standard installation (press enter or 1).

```
1) Proceed with standard installation (default - just press enter)
2) Customize installation
3) Cancel installation
>1
```

```
info: profile set to 'default'
info: default host triple is x86_64-pc-windows-msvc
warn: Updating existing toolchain, profile choice will be ignored
info: syncing channel updates for 'stable-x86_64-pc-windows-msvc'
info: latest update on 2025-08-07, rust version 1.89.0 (29483883e 2025-08-04)
info: downloading component 'cargo'
info: downloading component 'clippy'
info: downloading component 'rust-docs'
info: downloading component 'rust-std'
info: downloading component 'rustc'
 75.9 MiB /  75.9 MiB (100 %)  11.4 MiB/s in  6s
info: downloading component 'rustfmt'
info: removing previous version of component 'cargo'
info: removing previous version of component 'clippy'
info: removing previous version of component 'rust-docs'
info: removing previous version of component 'rust-std'
info: removing previous version of component 'rustc'
info: removing previous version of component 'rustfmt'
info: installing component 'cargo'
info: installing component 'clippy'
info: installing component 'rust-docs'
 20.2 MiB /  20.2 MiB (100 %)   1.5 MiB/s in 40s
```

This is your PC's CPU "type", member that!

When installing Rust, if prompt, allow it to install the Visual Studio C++ Build tools too. MS Windows needs these to run Rust.

1.2) Install code editor (IDE)

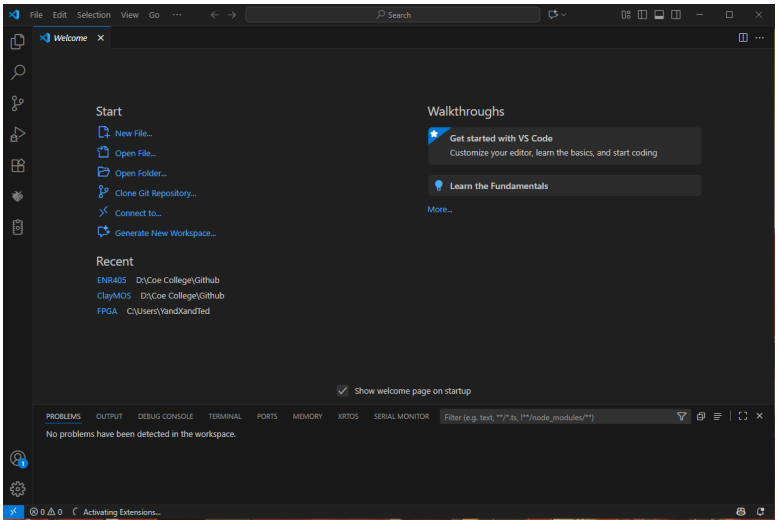
VS code works, so let's just use that.
Notes: there are many IDE available in the wild. VS code is not the fastest, but it's one of the popular ones to get rolling.

To install VS code on windows, go to:
https://code.visualstudio.com/
or
https://code.visualstudio.com/download

Download the installer for your OS. You will be required to create an account, or link your account to Github. If you haven't done so, now it's the time.

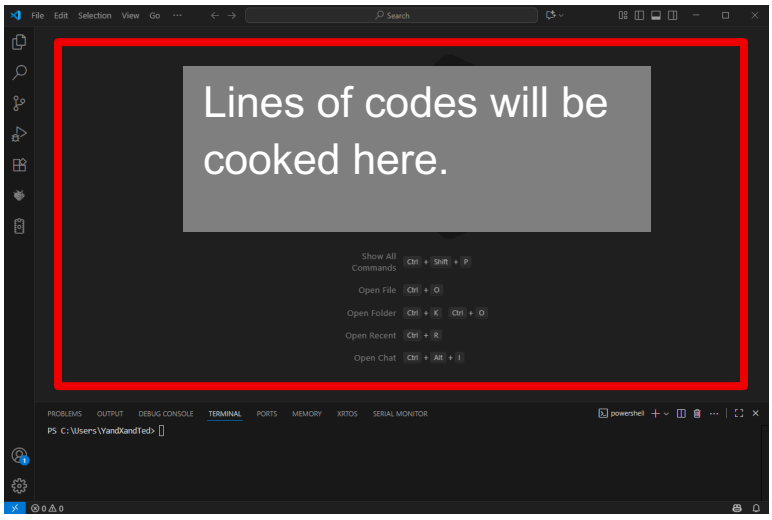
1.3) Get familiar with Visual Studio Code (VS Code)

When you first open VS code, it might look like this:

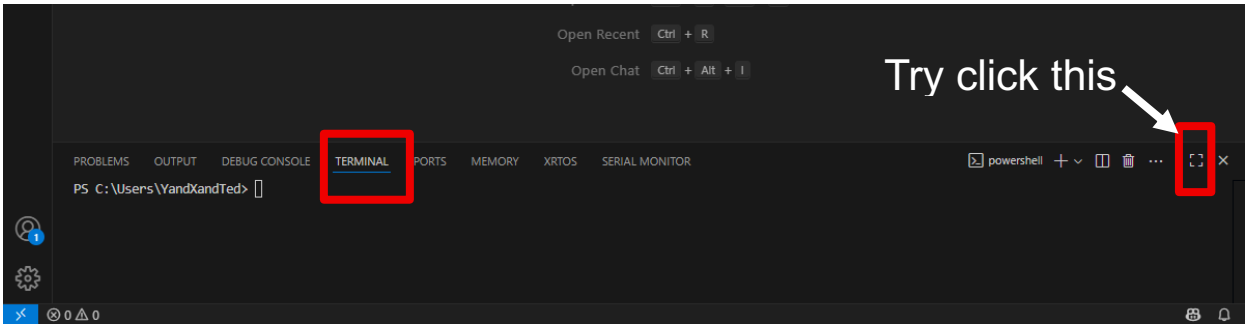


You will mostly work within two windows/panels, for now.

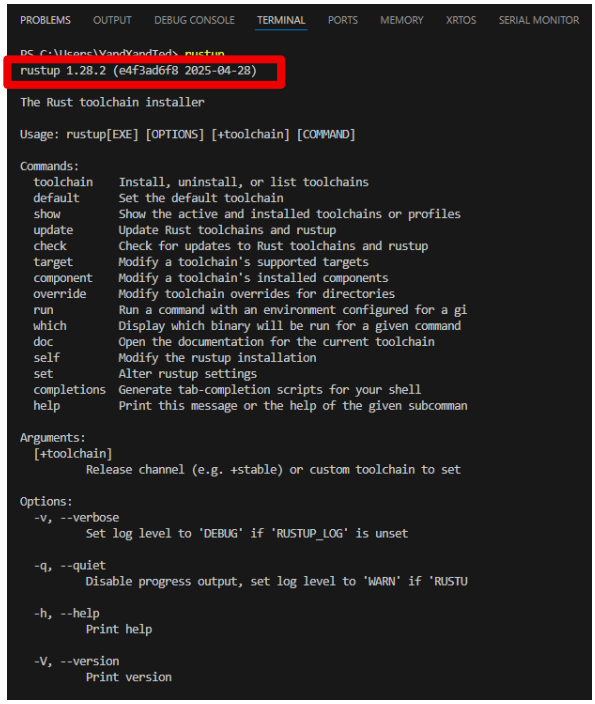
i) The work bench (where we do the coding stuff):



ii) The terminal (defaults on the bottom, where we do *hacker* stuff):



Let's type "rustup" (not including the quotation mark) to check the version.



Notes: Rust is maintained and updated often via the rustup tool every 6 weeks. Be sure to update often!
To do that, type: "rustup update" (not including the quotation mark).

For Rust embedded, we need more tools, type (or copy paste the command line, not including the quotation mark): "rustup component add llvm-tools"

```
PS C:\Users\YandXandTed> rustup component add llvm-tools
info: downloading component 'llvm-tools'
info: installing component 'llvm-tools'
 48.1 MiB /  48.1 MiB (100 %)  13.5 MiB/s in  3s
```

Again, type (or copy paste the command line, not including the quotation mark):

“cargo install cargo-binutils --vers '^0.3'”

```
PS C:\Users\YandXandTed> cargo install cargo-binutils --vers '^0.3'
    Updating crates.io index
  Downloaded cargo-binutils v0.3.6
  Downloaded 1 crate (25.5KiB) in 0.60s
  Installing cargo-binutils v0.3.6
    Updating crates.io index
     Locking 58 packages to latest compatible versions
      Adding cargo_metadata v0.14.2 (available: v0.22.0)
      Adding clap v2.34.0 (available: v4.5.47)
      Adding rustc-cfg v0.4.0 (available: v0.5.0)
      Adding toml v0.5.11 (available: v0.9.6)
  Downloaded atty v0.2.14
  Downloaded rustc_version v0.4.1
  Downloaded windows-targets v0.52.6
  Downloaded textwrap v0.11.0
  Downloaded unicode-ident v1.0.19
  Downloaded memchr v2.7.5
  Downloaded backtrace v0.3.75
  Downloaded aho-corasick v1.1.3
  Downloaded regex v1.11.2
  Downloaded serde_json v1.0.145
  Downloaded clap v2.34.0
  Downloaded toml v0.5.11
```

```
   Compiling failure v0.1.8
   Compiling strsim v0.8.0
   Compiling bitflags v1.3.2
   Compiling vec_map v0.8.2
   Compiling clap v2.34.0
   Compiling cargo_metadata v0.14.2
   Compiling rustc-cfg v0.4.0
   Compiling regex v1.11.2
   Compiling toml v0.5.11
   Compiling rustc_version v0.4.1
   Compiling cargo-binutils v0.3.6
    Building [==================>       ] 61/81: cargo-binutils
```

Now check and remove the older versions of stuff type:

cargo uninstall cargo-embed

cargo uninstall probe-run

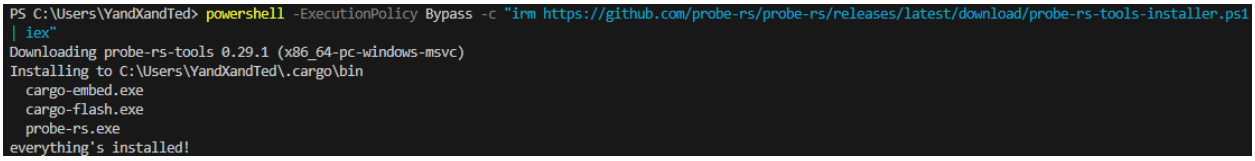
cargo uninstall probe-rs

cargo uninstall probe-rs-cl

If not then all good.

```
PS C:\Users\YandXandTed> cargo uninstall cargo-embed
error: package ID specification `cargo-embed` did not match any packages
PS C:\Users\YandXandTed> cargo uninstall probe-run
error: package ID specification `probe-run` did not match any packages
PS C:\Users\YandXandTed> cargo uninstall probe-rs
error: package ID specification `probe-rs` did not match any packages
PS C:\Users\YandXandTed> cargo uninstall probe-rs-cl
error: package ID specification `probe-rs-cl` did not match any packages
```

Now install probe-rs by copy-paste the whole red ling below:
powershell -ExecutionPolicy Bypass -c "irm https://github.com/probe-rs/probe-rs/releases/latest/download/probe-rs-tools-installer.ps1 | iex"

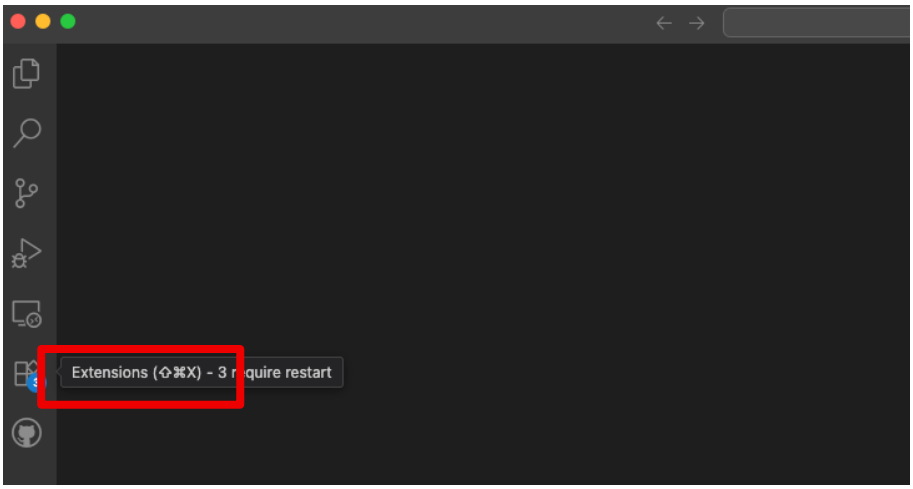


For more info regarding installing Rust on windows, please check: https://learn.microsoft.com/en-us/windows/dev-environment/rust/setup

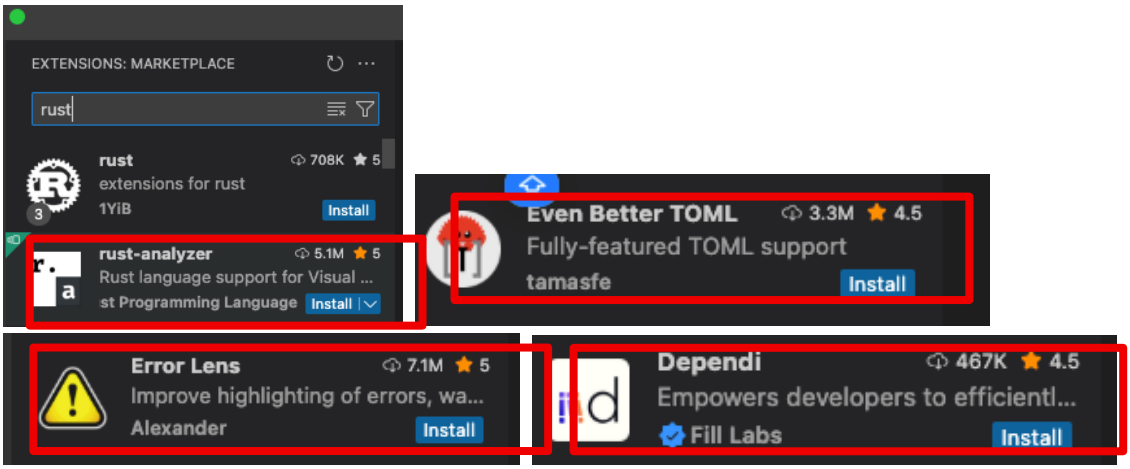
1.4) Install IDE for Arm chips: https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads

1.5) Install PUTTY: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

1.6) Install Rust related extension in the VS code.



Search and install
rust-analyzer
Even Better TOML
Error Lens
Dependi

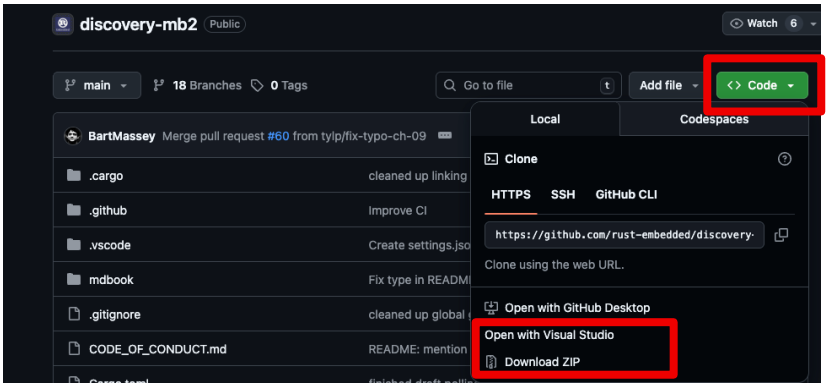


1.7) Hardware

Everyone already got a Microbit for digital class, all good. Microbit is one of the development boards which also serves as a STEM educational “toy”. For more info: https://microbit.org/code/

2) Install the github package

Someone already did the hard work and put everything we needed in a folder here: https://github.com/rust-embedded/discovery-mb2/
Either copy the code:



Or clone the whole link via github:

Now we are ready to do:

3) Embedded Rust on Microbit

3.1) First connect your microbit to your PC with a USB cable. At least one yellow LED light near the cable connection should be on:

3.2) Type “probe-rs list” in terminal

If you see this, your probe-rs could see microbit:

```
ML-PH-XL:discovery-mb2 xili$ probe-rs list
The following debug probes were found:
[0]: BBC micro:bit CMSIS-DAP -- 0d28:0204:990360200052820ab6ba0791a39aeab000000006e052820 (CMSIS-DAP)
```

Nice!

3.3) Now we need to specify cross-compiling[1]
Try to get to the following folder:

[1] Your PC runs a powerful CPU, but Microbit runs a much worse one (Nordic nRF52833, an Arm Cortex-M4 32 bit processor with FPU). So, we have to let Rust knows the spec for Microbit. All info can be dug out through the chip designer’s datasheet, Arm and Rust website. No worries.

discovery-mb2/mdbook/src/03-setup

Type:

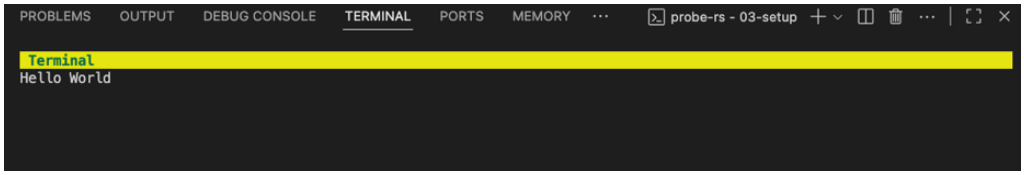
```
rustup target add thumbv7em-none-eabihf
```

Notes: You only need to do it once.

Type:

```
cargo embed --target thumbv7em-none-eabihf
```[2]

If everything goes on well, you will see:

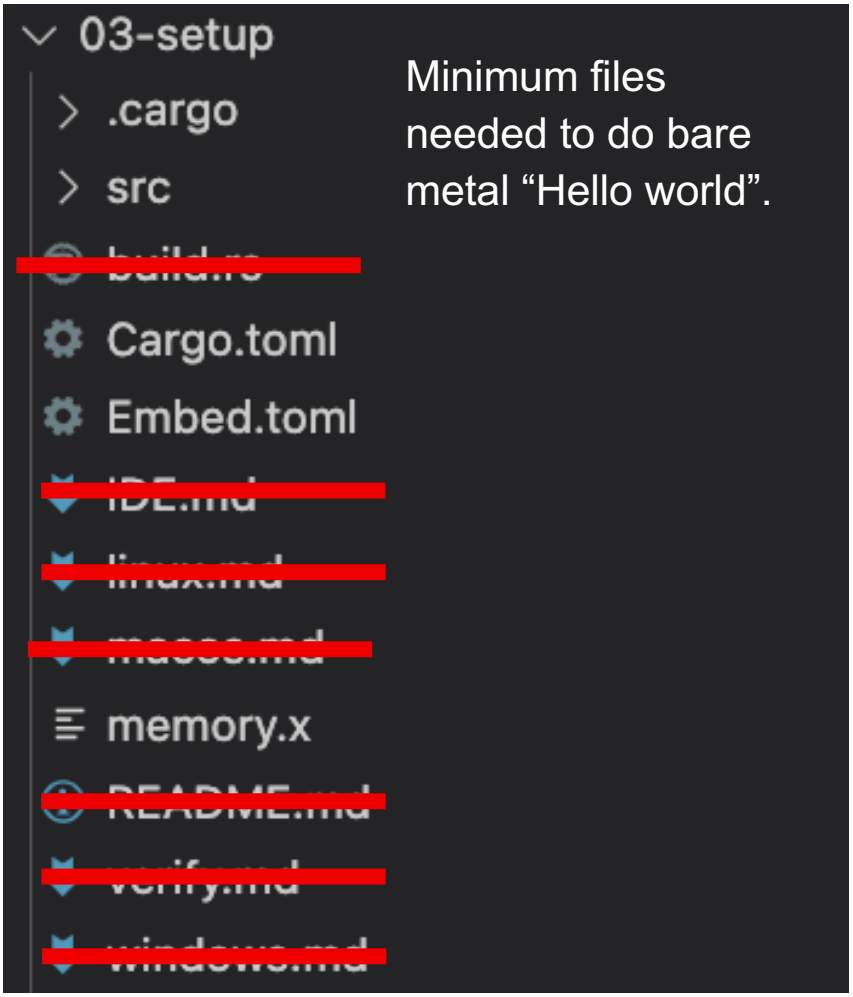


CONGRADULATIONS! You just did your first embedded coding on Microbit. (Microbit says “Hello World”, not your PC.)

[2] `thumbv7em-none-eabihf` is what rust used to talk to the Microbit chip. It is provided by the Arm instruction, and it is a smaller instruction set for embedded. The last two letters, “hf” means hardware floating point acceleration. AKA the chip can do fractional computations faster.

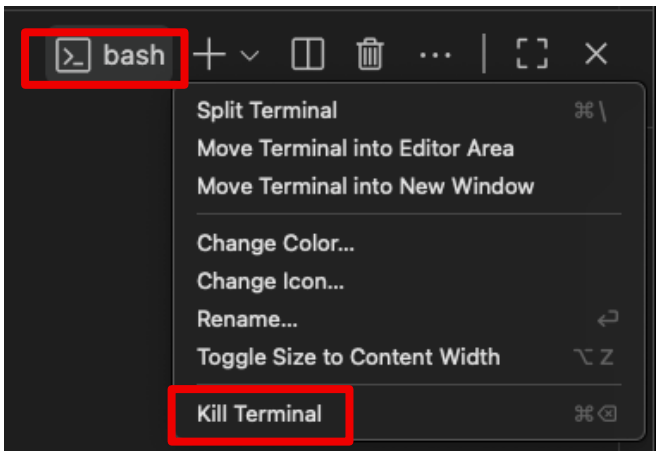
```rust
#![no_main]
#![no_std]

use cortex_m::asm::wfi;
use panic_rtt_target as _;
use nrf52833_pac as _;
use rtt_target::{rprintln, rtt_init_print};

use cortex_m_rt::entry;

▶ Run | ⚙ Debug
#[entry]
fn main() -> ! {
    rtt_init_print!();
    rprintln!("Hello World");
    loop {
        wfi();
    }
}

```

This is the code you flashed onto your Microbit.

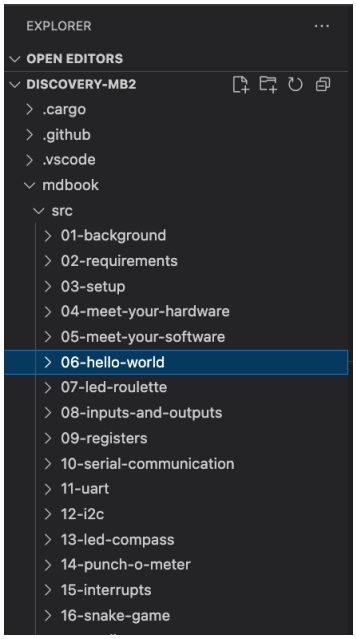


Minimum files needed to do bare metal “Hello world”.

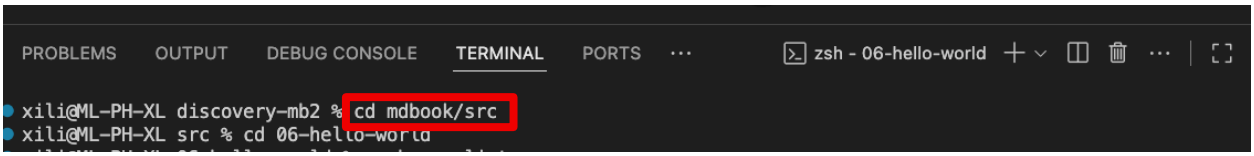
To stop, Kill Terminal.



3.4) Experience the day of a blessed embedded DEV:
By the work done so far, you have set up your IDE, flashed your first code into the Microbit. For embedded system, “Hello World” is often done by a “blinky”, i.e., make an LED flashing. That’s how you can do it:
Use the Terminal to navigate into the folder: 06-hello-world under the DISCOVERY-MB2 folder:

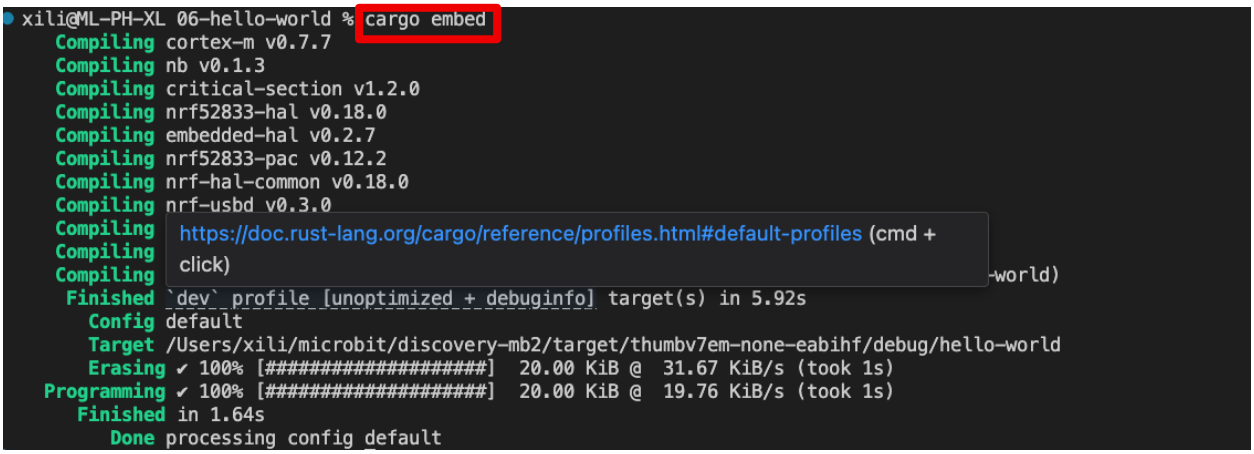


Here’s the quick cmd line:



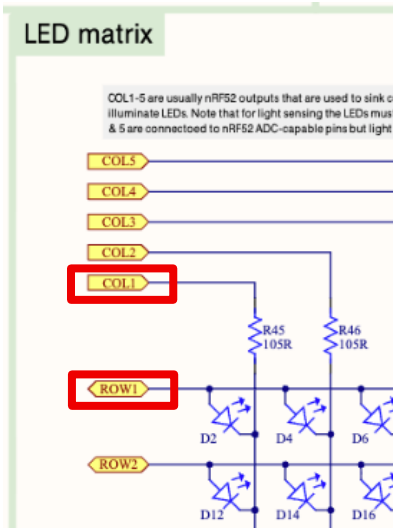
After you reconnect your microbit to your PC, you can use probe-rs to check if it's still there:

Now flash the new code into the MCU:



If everything goes on well, your first LED (0,0) should be blinky now.

What is going on?
First, take a look at the mcirobit-v2 hardware schematic:
https://github.com/microbit-foundation/microbit-v2-hardware/blob/main/V2.00/MicroBit_V2.0.0_S_schematic.PDF
For the LED matrix, if we put ROW 1 at high (3.3V), and COL1 at low (0 V), we will induce a current through D2 LED.
There are many levels of coding that function in Rust embedded. We will talk more about them in the next lab.



And that's all the codes are done, with a 500 ms timer to make it on and off.

mdbook > src > 06-hello-world > src > main.rs > ...

```rust
#![no_main]
#![no_std]

use cortex_m_rt::entry;
use embedded_hal::{delay::DelayNs, digital::OutputPin};
use microbit::hal::{gpio, timer};
use panic_halt as _;

▶ Run | ⚙ Debug
#[entry]
fn main() -> ! {
    let board: Board = microbit::Board::take().unwrap();

    let mut row1: P0_21<Output<PushPull>> = board.display_pins.row1.into_push_pull_output(gpio::Level::High);
    let _col1: P0_28<Output<PushPull>> = board.display_pins.col1.into_push_pull_output(gpio::Level::Low);

    let mut timer0: Timer<TIMER0> = timer::Timer::new(board.TIMER0);

    loop {
        timer0.delay_ms(500);
        row1.set_high().unwrap();
        timer0.delay_ms(500);
        row1.set_low().unwrap();
    }
}

```