NeoECU Development Environment

Hector van der Aa

hector@h3cx.dev

March 21, 2026

Table of contents

Background

This document describes the setup and usage of the development and build environment for the NeoECU project.

NeoECU uses an AMP (Asymmetrical Multiprocessing) architecture. Both cores of the STM32H747 are used for different tasks and run different software stacks.

Core responsibilities:

- **M7 core** — Real-time, safety-critical engine control (e.g. spark timing, ignition)
- **M4 core** — Telemetry, logging, and user interface

Because of this separation:

- M7 code must prioritise reliability and determinism
- M4 code can tolerate higher latency and complexity

Selected tools:

- **M7 core**
  - STM32CubeMX — MCU initialisation code generation
  - FreeRTOS — Lightweight task scheduling
  - STM32 HAL — Hardware abstraction layer
- **M4 core**
  - Zephyr RTOS — Microkernel with full I/O stack

Firmware flashing is performed over JTAG using a SEGGER J-Link.

> **Note**
> This environment is designed for Linux. Windows and macOS may work, but are not officially supported.

Setup

Prerequisites

uv (Python Tooling)

Install `uv`:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Restart your terminal, then verify installation:

```
which uv && uv --version
```

Expected output (example):

```
/home/user/.local/bin/uv
uv 0.10.x
```

Install Python 3.14:

```
uv python install 3.14
```

Documentation: https://docs.astral.sh/uv/getting-started/installation/

Build Tools

Install CMake and Ninja:

```
# Arch
sudo pacman -S cmake ninja

# Debian
sudo apt update
sudo apt install cmake ninja-build

# Fedora
sudo dnf install cmake ninja-build
```

Verify installation:

```
which cmake && cmake --version
which ninja && ninja --version
```

Build Dependencies

Install ARM toolchain:

```
# Arch
sudo pacman -S \
  arm-none-eabi-gcc \
  arm-none-eabi-newlib \
  arm-none-eabi-binutils\

# Debian
sudo apt update
sudo apt install \
  gcc-arm-none-eabi \
  libnewlib-arm-none-eabi \
  binutils-arm-none-eabi\

# Fedora
sudo dnf install \
  arm-none-eabi-gcc-cs \
  arm-none-eabi-newlib \
  arm-none-eabi-binutils\
```

Zephyr

Create workspace:

```
mkdir ~/zephyrproject
cd ~/zephyrproject
```

Create virtual environment:

```
uv venv --python 3.14
source .venv/bin/activate
```

Install west:

```
uv pip install west
```

Initialise workspace:

```
west init ~/zephyrproject
cd ~/zephyrproject
west update
```

Export environment:

```
west zephyr-export
```

Install dependencies:

```
uv pip install -r zephyr/scripts/requirements.txt
```

If errors occur, install missing packages and re-run until successful.

Install SDK:

```
cd ~/zephyrproject/zephyr
west sdk install
```

Add the ZEPHYR_BASE variable to your .bashrc:

```
export ZEPHYR_BASE=$HOME/zephyrproject/zephyr
```

STM32CubeMX

Download STM32CubeMX from
https://st.com/en/development-tools/stm32cubemx.html

In your downloads, unzip the package:

```
unzip stm32cubemx-lin-*.zip -d STM32CubeMxSetup
cd STM32CubeMxSetup
```

Run the setup binary:

```
./SetupSTM32CubeMX-*
```

Add the install directory to your path, ie for bash add the following to `.bashrc`:

```
export PATH=$PATH:$HOME/STM32CubeMX
```

It can take a few minutes for the application to start for the first time.

Then open the 'Install or remove embedded software pacakges' window:

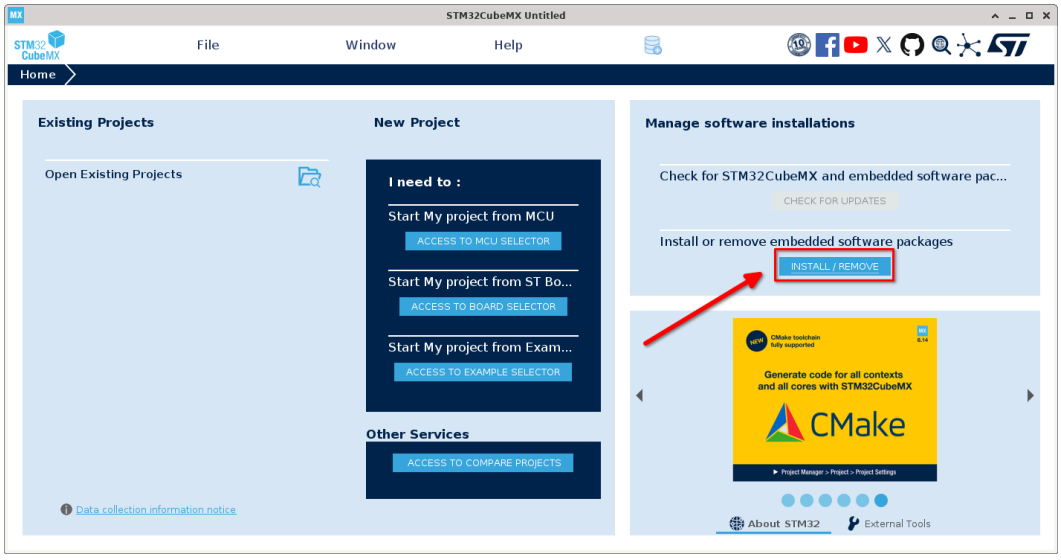

Figure 1: STM32CubeMX Home Screen

In the 'STM32Cube MCU Packages' tab, find the 'STM32H7' entry and install the latest version:

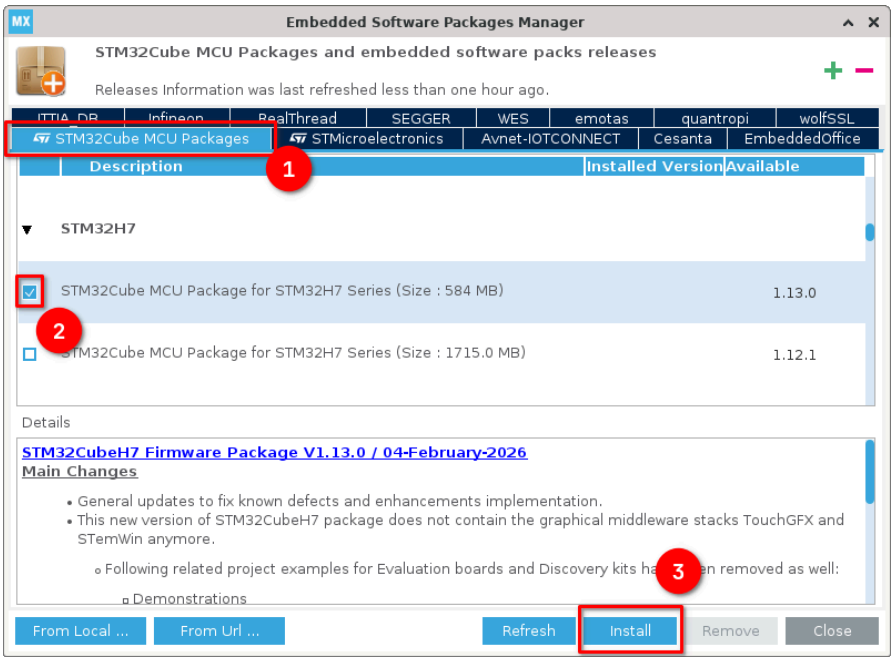

Figure 2: STM32CubeMX Install MCU Package

JLink Tools

Download the '*J-Link Software and Documentation Package*' from https://www.segger.com/products/debug-probes/j-link

From your downloads, install the package:

```
# Arch
sudo mkdir /opt/SEGGER/JLink
sudo tar -xvzf JLink_Linux_*_x86_64.tgz -C target --strip-components=1

# Debian
sudo apt install ./JLink_Linux_*_x86_64.deb

# Fedora
sudo dnf install ./JLink_Linux_*_x86_64.rpm
```

If JLink tooling was installed with the tar bundle, the following directory needs to be added to your path:

```
export PATH=$PATH:/opt/SEGGER/JLink
```

Repository Setup

Cloning

Clone the repository:

```
git clone gitea@git.h3cx.dev:Exergie/BuildDemoNeoECU.git
cd BuildDemoNeoECU
```

Virtual Environment

Create a project-specific virtual environment:

```
uv venv --python 3.14
```

Activate it:

```
source .venv/bin/activate
```

This step must be repeated for each new terminal session.

NeoECU Tooling Setup

Install the neoecu cli tooling:

```
uv pip install "git+https://git.h3cx.dev/h3cx/NeoECU-Tooling.git"
```

Help can be found by running:

```
neoecu
```

Start by checking dependencies:

```
neoecu envcheck
```

Then init the tooling system:

```
neoecu init
```

You can now test build the demo project:

```
neoecu build --debug --clean
```

If this succeeds then your build environement is set up correctly!

NeoECU Tooling Overview

The NeoECU tooling harmonizes the STM32 and Zephyr environements into one set of easy to use commands. It currently supports the following commands:

```
neoecu envcheck
neoecu init
neoecu build
neoecu clean
```

envcheck

The environement checker is a simple tool that checks that all the required tools and dependencies are installed and accessible from your path.

init

The init command installs all the required python dependencies for west and initializes the cmake build environment for the STM32 side of the project.

build

The build command builds both sides of the project, STM32 and Zephyr. It defaults to release mode when run without arguments.

Multiple arguments can be passed as follows:

```
--release
# or
--debug
```

```
--clean
```

For example to build clean in debug mode:

```
neoecu build --debug --clean
```

clean

The clean command cleans up all build directories in the project