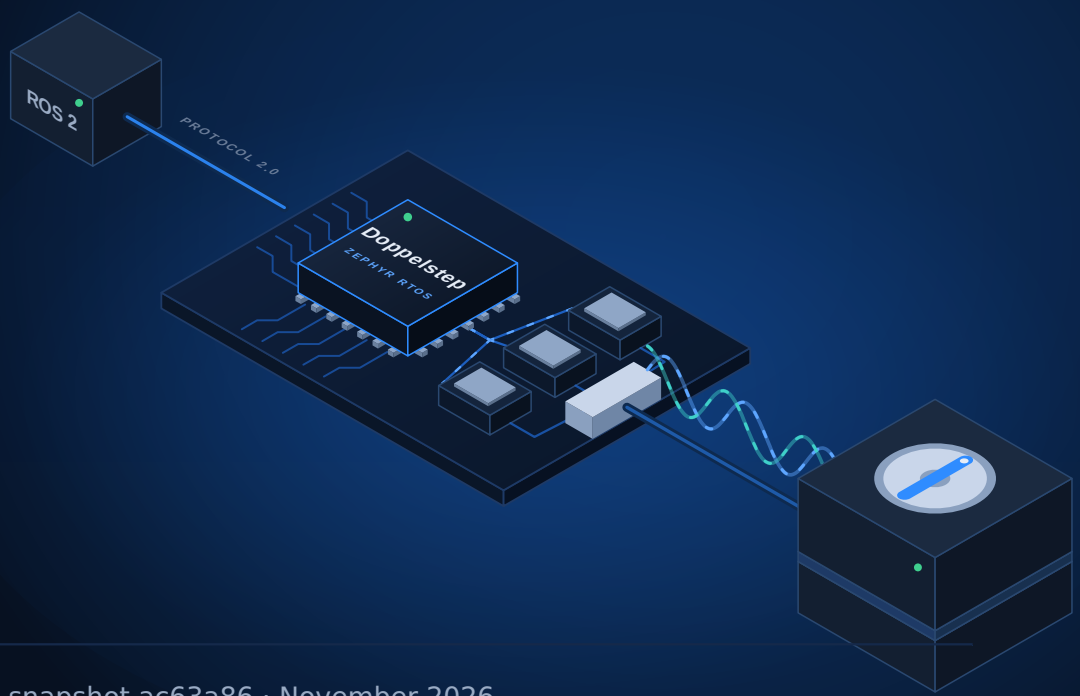
SystemFromScratch

OPEN-SOURCE ZEPHYR RTOS FIRMWARE

Doppelstep

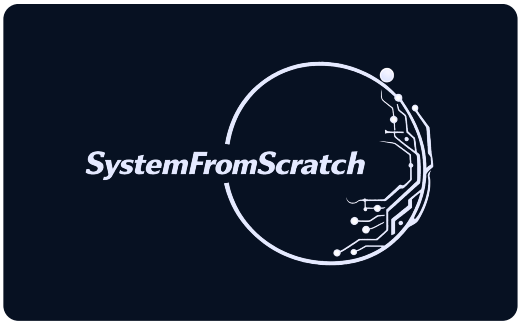
Your stepper motor, seen as a DYNAMIXEL smart servo

USER MANUAL



Version development snapshot ac63a86 · November 2026
Anton Vynohradov
systemfromscratch.github.io

Imprint



Doppelstep User Manual
Firmware version development snapshot ac63a86
Edition: November 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.github.io
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
ORCID: orcid.org/0009-0005-6965-0911
E-mail: avynohradov@systemfromscratch.com

Online
Documentation: https://systemfromscratch.github.io/sfs-doppelstep/
Source code: https://github.com/SystemFromScratch/sfs-doppelstep

Copyright © 2026 Anton Vynohradov.

The Doppelstep firmware and this manual are licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in Appendix A. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SystemFromScratch® and the SystemFromScratch® logo are registered trademarks. The Apache License 2.0 grants no rights to use them; they may not be used to endorse or promote products derived from Doppelstep without prior written permission.

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. STM32 is a registered trademark of STMicroelectronics International N.V. Arm and Cortex are trademarks of Arm Limited. Hardware and protocol names are used only to describe compatibility; Doppelstep is not affiliated with or endorsed by their owners. All other trademarks are the property of their respective owners.

This manual is generated from the same sources as the online documentation. The information in it was checked against the firmware version stated above; later versions may differ.

Contents

1 Introduction 6
- 1.1 The idea in plain words 6
- 1.2 The chain, from the robot controller to the motor 6
- 1.3 Who it is for 7
- 1.4 What it does, and what it does not do 8
- 1.5 Key properties 8
- 1.6 Status and known limits 9
- 1.7 Terms used in this documentation 9

2 Technology stack 11
- 2.1 At a glance 11
- 2.2 Firmware platform 11
- 2.3 Motor control 12
- 2.4 Servo protocol 13
- 2.5 Simulation 14
- 2.6 Tests and continuous integration 14
- 2.7 Build and development environment 14
- 2.8 Host tools 15
- 2.9 Documentation 15
- 2.10 Engineering practices 15

3 Getting started 17
- 3.1 Get the source 17
- 3.2 1. On your PC, no hardware 17
- 3.3 2. On the board, simulated motor 18
- 3.4 3. On the board, real motor 18
- 3.5 What a build produces 19
- 3.6 Next steps 19

4 Hardware and bring-up 20
- 4.1 The board 20
- 4.2 Motor wiring 20
- 4.3 Current and voltage scaling 22
- 4.4 The DYNAMIXEL bus 22
- 4.5 Bring-up checklist 22

5 DYNAMIXEL emulation 24
- 5.1 Register mapping 24
- 5.2 Units 25
- 5.3 Startup position 25
- 5.4 Factory values and the EEPROM 25
- 5.5 Connecting a host program 25
- 5.6 OpenMANIPULATOR-X 26
- 5.7 Differences from a real XM430-W350 26

6 Configuration 27
- 6.1 Where each setting lives 27
- 6.2 The motor node 27

6.3 The power stage node ... 28
6.4 The servo node ... 28
6.5 Variants of the build ... 29

7 Architecture ... 30
7.1 Layout ... 30
7.2 Four rules ... 31
7.3 Drivers ... 31
7.4 Contexts ... 32
7.5 Queues ... 32
7.6 A command, from the bus to the motor ... 32
7.7 Startup ... 33
7.8 Configuration ... 34

8 Simulated motor ... 35
8.1 How the pieces fit ... 35
8.2 The model ... 36
8.3 Parameters ... 36
8.4 Two ways to run it ... 37
8.5 Workflows ... 38
8.6 Checks ... 39
8.7 Limits ... 39

9 Firmware update ... 40
9.1 Flash layout ... 40
9.2 Build and flash ... 40
9.3 Serial recovery ... 40
9.4 Upload with the GUI ... 41
9.5 Upload with smpmgr ... 41
9.6 Signature ... 42
9.7 Configuration ... 42
9.8 Troubleshooting ... 42

10 Porting to a new MCU family ... 43
10.1 What a port needs ... 43
10.2 Backend checklist ... 43
10.3 What the code above the backend does not assume ... 44
10.4 Example: TI MSPM0G3519 ... 44

11 Development environment ... 46
11.1 Requirements ... 46
11.2 Repository layout ... 46
11.3 Two platforms, one set of tasks ... 47
11.4 Tasks ... 48
11.5 Dev Container ... 49
11.6 Flashing and debugging ... 50
11.7 CI and the environment image ... 51
11.8 Releasing ... 51
11.9 Codespaces ... 52
11.10 Python virtual environment ... 52
11.11 Documentation and PDF manual ... 52

12 Tests ... 54

12.1 ztest suites 54
12.2 SIL and integration tests 54
12.3 Running them 55
12.4 CI 57

13 Hardware checklist after the driver refactor 58
13.1 A: firmware without a motor 58
13.2 B: simulated motor on the board 59
13.3 C: with a motor, after the dead time is set 59
13.4 D: firmware update over USART2 59

14 GUI 61
14.1 Starting the GUI 61
14.2 Connecting 62
14.3 Motion and live plots 62
14.4 Control Table 63
14.5 Test scenarios 64
14.6 Firmware update 67
14.7 Structure and tests 68

Author & Company 69
Author 69
SystemFromScratch® 69
Use it, build on it 70
Contact 70
Support and feedback 71
Trademarks and acknowledgements 71

How Doppelstep got its name 72
The project before the name 72
What the name had to do 72
Candidates 72
The check 73
The meaning 73
How to write it 73

Appendix A License 74
A.1 Components under their own licenses 74
A.2 Trademarks 74
A.3 Full license text 75

CHAPTER 1

Introduction

Doppelstep is open-source firmware that makes an ordinary stepper motor behave like a smart servo. It runs on a small motor-control board from STMicroelectronics, the **B-G431B-ESC1**, drives a two-phase stepper motor with precise current control, and answers on its serial bus exactly like a **DYNAMIXEL XM430-W350** servo. Software written for that servo, such as the ROS 2 driver of the OpenMANIPULATOR-X robot arm, works with it unchanged.

This page explains the idea in plain words, who the project is for, what it does and does not do, and the terms used in the rest of the documentation.

1.1 The idea in plain words

A **smart servo** is a motor with its own electronics. A robot controller sends it short messages over a cable ("go to 90 degrees at this speed", "how much current do you use?"), and the servo does the rest. A **stepper motor** is simpler and cheaper and holds its position firmly, but it understands no messages: it needs electronics that feed its two coils with carefully shaped currents, thousands of times per second.

Doppelstep is that electronics' firmware. It does two jobs at once:

Job	How
Speak like the servo	It receives and answers DYNAMIXEL Protocol 2.0 messages and keeps the same table of settings and readings (the *Control Table*) that a real XM430-W350 keeps
Drive the motor	It turns each command into a smooth motion profile and controls the two coil currents 20 000 times per second, in steps 16 times finer than the motor's own 200 steps per revolution

The controller on the other end of the cable cannot tell the difference. That is where the name comes from: Doppelstep is the *double* (German *Doppelgänger*) of a servo (Naming (p. 72)).

1.2 The chain, from the robot controller to the motor

Stage	What happens	Where in the documentation
Host PC	ROS 2, DynamixelSDK or DYNAMIXEL Wizard sends Protocol 2.0 packets at 1 Mbps	DYNAMIXEL emulation
Serial bus	USART2 of the board (for now through the USB port of the on-board debugger)	Hardware and bring-up
Servo emulator	Checks every packet and every register access as the real servo does	Architecture
Application	Converts servo units into motor commands and back, in separate threads for the bus and for control	Architecture

Stage	What happens	Where in the documentation
Motor control	Stepper-C plans the motion at 1 kHz and controls the coil currents at 20 kHz	Technology stack
Power stage	A timer switches three half-bridges; the ADCs measure the currents and the supply voltage	Hardware and bring-up
Motor	A NEMA 17 stepper, wired with one shared leg for both coils	Hardware and bring-up

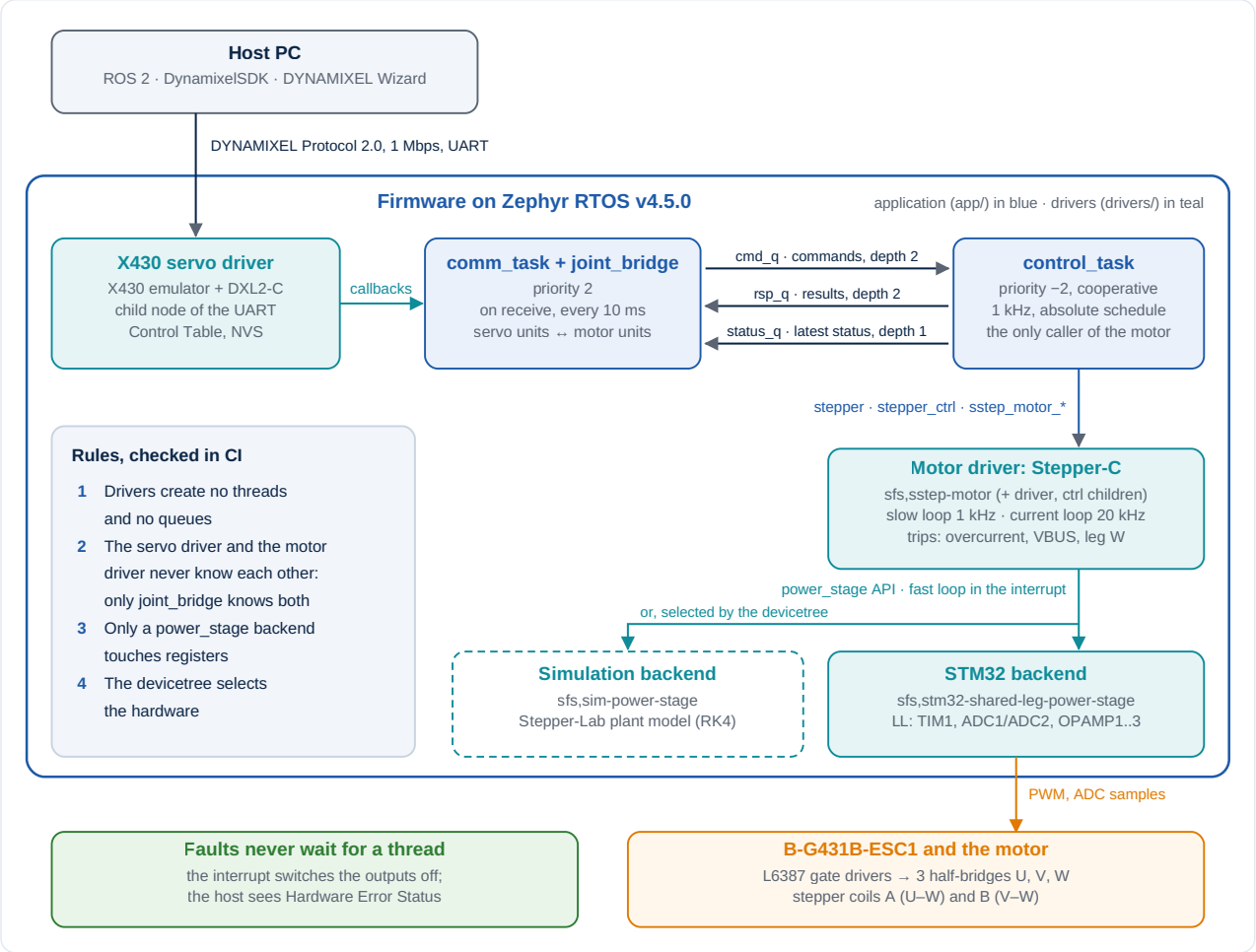


Figure 1: The firmware between the host and the motor

1.3 Who it is for

You are…	Doppelstep gives you
A robotics developer using ROS 2	A low-cost joint that your existing DYNAMIXEL software already knows how to drive
An embedded engineer	A complete, documented Zephyr RTOS firmware to study or build on: drivers, devicetree, threads, bootloader, tests
A team evaluating a supplier	A real product, not a demo: measured figures, tests on every change, honest limits

You are…	Doppelstep gives you
A student or maker	A project you can run on your PC in minutes, without any hardware, and then on an off-the-shelf evaluation board

1.4 What it does, and what it does not do

Doppelstep…	Why
Does emulate one XM430-W350 on the bus, in position, extended position and velocity mode	These are the modes the OpenMANIPULATOR-X and most ROS 2 setups use
Does control the coil currents in closed loop, with microstepping	Smooth, quiet motion and a defined torque, instead of switching full voltage onto the coils
Does protect the hardware inside the 20 kHz interrupt	A fault must switch the outputs off within one control period, whatever the threads are doing
Does run unchanged on a PC with a simulated motor	Protocol, motion and protection can be tested on every change, without risking hardware
Does update its own firmware over the servo bus	Only the bus cable is needed in the field; a signature check rejects foreign images
Does not measure the rotor position	The motor has no encoder. Reported positions are the commanded ones, and a lost step is not detected yet (Status)
Does not support current-based and PWM modes of the servo (Operating Modes 0, 5, 16)	Without a position sensor they would not behave like the real servo; the firmware refuses them as the servo refuses invalid values
Does not share a bus with real DYNAMIXEL servos yet	The bus runs through the debugger's USB port for now; a TTL half-duplex bus is planned

1.5 Key properties

Measured on this version with the CI build (`scripts/ci.sh`), unless noted.

Property	Value
Microcontroller	STM32G431CB: Arm Cortex-M4F, 170 MHz, 128 KiB flash, 32 KiB RAM
RTOS	Zephyr RTOS v4.5.0, with sysbuild and MCUboot
Application size	77 356 B flash of the 87 712 B slot, 19 032 B RAM
Bootloader size	MCUboot, 37 148 B flash of its 38 KiB partition
Control rates	Current loop 20 kHz (PWM rate), motion loop 1 kHz, bus status every 10 ms
Motor resolution	3200 microsteps per revolution (200 steps × 16)
Bus	DYNAMIXEL Protocol 2.0, 1 Mbps (factory value), XM430-W350, model number 1020

Property	Value
Automated tests	55 ztest cases in 4 suites, 28 SIL and integration tests, 8 motion tests on the fixed-point backend: 91 per CI run
Toolchain	Zephyr SDK 1.0.1 in a pinned Docker image
License	Apache License 2.0

1.6 Status and known limits

Doppelstep is under active development. These items are open, and the documentation says so wherever they matter:

- **Dead time.** `dead-time-ns` in `app/boards/b_g431b_esc1.overlay` is 0 until the value of the board's motor-control reference project is entered. With 0 the firmware runs but never switches the power stage on (bring-up step 1 (p. 22)).
- **Lost steps are not reported.** A load above the holding torque makes the motor lose synchronism silently. A test documents this as an expected failure (Tests (p. 56)).
- **The driver refactor is not yet checked on the board.** The checks to run are listed in the hardware checklist (p. 58).
- **Rotor vibration at rest in the model:** one ADC step in the current loop excites the mechanical resonance (up to about 3 rpm, 0.2 microstep). Not yet measured on a real motor.
- **Fast-loop execution time** on the target is not measured yet; the log of the simulated motor on the board shows the longest fast-loop callback.
- **Temperature:** the NTC sensor is not converted yet. Present Temperature keeps its table value and there is no overheating trip.
- **Bus:** a single-wire TTL bus instead of the debugger's serial port, and a shared bus with real DYNAMIXEL servos, are planned.

1.7 Terms used in this documentation

Term	Meaning
Stepper motor	A motor that turns in fixed steps (here 200 per revolution) when its two coils are energised in sequence
Microstepping	Shaping the two coil currents as sine and cosine, so the rotor stops between the full steps (here 16 microsteps per step)
Closed-loop current control	Measuring the coil currents and correcting the voltage every PWM period, so the current follows its set value
Smart servo	A motor with built-in control electronics that takes position or velocity commands over a bus
DYNAMIXEL, Protocol 2.0	A family of smart servos by ROBOTIS and the packet protocol they speak
Control Table	The numbered registers of a DYNAMIXEL servo: settings (EEPROM area, kept after power off) and live values (RAM area)
Zephyr RTOS	An open-source real-time operating system for microcontrollers, hosted by the Linux Foundation

Term	Meaning
Devicetree, Kconfig	Zephyr's two configuration languages: the devicetree describes the hardware, Kconfig selects the software features
Power stage, half-bridge	The transistors that switch the supply voltage onto the motor coils; a half-bridge is one pair of them
Fast loop, slow loop	The current controller in the PWM interrupt (20 kHz) and the motion planner in a thread (1 kHz)
MCUboot	An open-source secure bootloader: it starts only firmware with a valid signature
SIL (software in the loop)	Testing the real firmware against a simulated motor instead of the real one
native_sim	A Zephyr target that builds the firmware as a normal Linux program
ROS 2, ros2_control	The Robot Operating System and its framework for hardware drivers of robot joints

CHAPTER 2

Technology stack

Doppelstep is built from established open-source components and from the libraries of the System-FromScratch® portfolio. This page lists every one of them: what it does in the project, which version is used, and why it was chosen. All versions are pinned. The source of truth is `west.yml` for the firmware, the `Dockerfile` for the toolchain, and the requirement files for the Python tools.

2.1 At a glance

Layer	Technology
Real-time operating system	**Zephyr RTOS v4.5.0**: drivers, devicetree, Kconfig, threads, sysbuild
Bootloader	**MCUboot** with an RSA-2048 signature check and serial recovery
Motor control	**Stepper-C v1.2.0**: closed-loop microstepping current control
Servo protocol	**DYNAMIXEL X430 Control Table Emulator** on **DXL2-C v1.0.0** (Protocol 2.0)
Simulation	**Stepper-Lab** plant model, power stage values from **ngspice** (sfs-stepper-power-spice), Zephyr **native_sim**
Hardware access	STM32Cube **LL drivers** (hal_stm32), **CMSIS 6**, **SEGGER RTT**
Tests	Zephyr **ztest** and **Twister**, **pytest**, the official **DynamixelSDK**
Build and environment	**west**, **Zephyr SDK 1.0.1**, **CMake**, **Docker**, Dev Container, **GitHub Codespaces**
CI	**GitHub Actions** with an environment image in the **GitHub Container Registry**, ccache
Host tools	**Python**: Tkinter GUI, **matplotlib**, **pyserial**, **smpmgr** / smpclient
Debugging	**OpenOCD** (ST-LINK), **SEGGER J-Link**, VS Code **Cortex-Debug**
Documentation	**MkDocs Material**, a PDF manual rendered with **WeasyPrint**

2.2 Firmware platform

Zephyr RTOS

Item	Detail
Version	v4.5.0 (`west.yml`)
Role	Kernel, threads and message queues, UART and flash drivers, NVS storage, logging, the devicetree and Kconfig configuration, the build system (sysbuild)

Item	Detail
Why	A vendor-neutral RTOS with a long-term support model and a driver model that separates hardware from application. The same application builds for the STM32 board and for the PC (`native_sim`), which is what makes the simulated tests possible

Doppelstep is a proper **Zephyr module**, laid out like the official `example-application`: its own drivers, devicetree bindings with the `sfs,` vendor prefix, and an out-of-tree board definition for the B-G431B-ESC1, which Zephyr 4.5 does not include. The motor is exposed through Zephyr's standard `stepper` and `stepper_ctrl` driver classes, so other Zephyr code can drive it without knowing Stepper-C (Architecture (p. 31)). Only the modules the firmware needs are imported (`name-allowlist` in `west.yml`), which keeps the workspace small.

MCUboot

Item	Detail
Version	The revision pinned by Zephyr v4.5.0, with mbedtls, tf-psa-crypto and zcbor
Role	Secure bootloader: checks the RSA-2048 signature of the application at every start; serial recovery accepts a new image over the DYNAMIXEL bus (SMP protocol)
Why	The standard bootloader of the Zephyr ecosystem, built together with the application by sysbuild. Field updates need only the bus cable, and foreign images do not start

Details: Firmware update (p. 40).

STM32Cube LL, CMSIS 6, SEGGER RTT

Component	Role	Why
hal_stm32 (STM32Cube LL drivers)	Timer TIM1, ADC1/ADC2 injected conversions, OPAMP1..3 in the power stage backend	Zephyr's PWM and ADC APIs have no dead time, complementary outputs or timer-triggered injected conversions, which a 20 kHz current loop needs. LL code stays in one file
CMSIS 6	Core access for the Cortex-M4F: zero-latency interrupts, the cycle counter	Arm's standard core interface
SEGGER RTT	Console and log through the debug probe	USART2 carries the servo bus, so the log needs another channel; RTT costs almost no CPU time

2.3 Motor control

Stepper-C

Item	Detail
Version	v1.2.0 (`west.yml`)
Source	github.com/AntonVynohradov/Stepper-C, SystemFromScratch®

Item	Detail
Role	Closed-loop microstepping current control of a two-phase stepper motor: the current controller (fast loop, 20 kHz), the motion profiles (slow loop, 1 kHz), current and voltage protection
Why	A hardware-agnostic library in portable C, with no heap and no RTOS, that reaches the hardware only through callbacks. Doppelstep wraps it in a Zephyr driver instead of re-implementing control in the application

Build options used here (`modules/stepper-c/Kconfig`):

Option	Doppelstep uses
Numeric backend	Single-precision float on the STM32G431 (hardware FPU). The fixed-point backend (Q15/Q31, integer only) is tested in CI for cores without an FPU
Power stage topology	Three half-bridges with a shared leg: the B-G431B-ESC1 has three legs, not four
Microstepping	16 microsteps per full step, 3200 per revolution

2.4 Servo protocol

DYNAMIXEL X430 Control Table Emulator

Item	Detail
Version	a pinned commit of `main` until its first release (`west.yml`)
Source	github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator, SystemFromScratch®
Role	The complete Control Table of an XM430-W350 (model 1020): access rules, ranges and limits, the EEPROM lock, Indirect Addresses, Sync/Bulk and Fast Sync/Bulk Read, Bus Watchdog, Reboot and Factory Reset
Why	The servo's behaviour is far more than reading and writing registers. A dedicated, separately tested emulator keeps that behaviour exact, and keeps DYNAMIXEL details out of the motor code

DXL2-C

Item	Detail
Version	v1.0.0 (`west.yml`), the tag the emulator expects
Source	github.com/AntonVynohradov/DXL2-C, SystemFromScratch®
Role	DYNAMIXEL Protocol 2.0 packet codec and link layer: framing, byte stuffing, CRC
Why	A released protocol library of the portfolio. The emulator normally downloads it with CMake FetchContent; the module glue points `FETCHCONTENT_SOURCE_DIR_DXL2` at the west project instead, so a build needs no network access, and warns when the `dxl2-c` revision in `west.yml` differs from the tag the emulator expects

2.5 Simulation

Component	Version	Role	Why
Stepper-Lab	pinned commit until its first release	The motor and power stage model in C (`sil/c/sstep_plant.c`) and its parameters (`params/*.json`)	It is the same C code as the Xcos block of the Scilab prototype, so the firmware is tested against the model the controller was designed with
sfs-stepper-power-spice	pinned commit until its first release	ngspice model of one leg of the B-G431B-ESC1; writes the stage parameters of Stepper-Lab	The leg voltage error (dead time, body diodes, resistance) comes from a circuit simulation, not from a guess
Zephyr native_sim	part of Zephyr v4.5.0	Builds the whole firmware as a Linux program	The complete chain, protocol to motor model, runs on any PC and in CI

How the three repositories work together: Simulated motor (p. 35).

2.6 Tests and continuous integration

Component	Version	Role
ztest and **Twister**	Zephyr v4.5.0	55 test cases in 4 suites: unit conversions, queues, the power stage API, the motor driver on a fake power stage
pytest (+ pytest-timeout)	9.0.2 (2.4.0)	28 SIL and integration tests against the `native_sim` firmware over Protocol 2.0, plus 8 motion tests on the fixed-point backend
DynamixelSDK	4.1.0	The official ROBOTIS library under ROS 2 `dynamixel_hardware_interface`: the integration tests run the ros2_control loop with it
Layer check	`scripts/check-layers.sh`	Enforces the architecture rules: no register access outside the backends, no floating point above them, no DYNAMIXEL code in the motor driver
GitHub Actions	`.github/workflows/ci.yml`	Runs `scripts/ci.sh` on every push to `main` and every pull request: layer check, three pristine builds, all tests
GitHub Container Registry	`ghcr.io/systemfromscratch/sfs-doppelstep-env`	The environment image with the toolchain and the west workspace, tagged by the hash of the files it is made from
ccache	in the image	Keeps CI and local rebuilds fast

Details: Tests and CI (p. 54).

2.7 Build and development environment

Component	Version	Role
Docker image	base `python:3.12.15-slim-bookworm`, digest pinned	The whole toolchain in one image, identical on Windows, Linux, CI and Codespaces

Component	Version	Role
Zephyr SDK	1.0.1 (minimal, `arm-zephyr-eabi`, host tools with OpenOCD)	GCC toolchain for Arm and the host tools
west	1.5.0	Workspace manager and build front end of Zephyr
CMake	3.30.5	Build system under west
VS Code tasks and **Dev Container**	`.vscode/`, `.devcontainer/`	The same tasks on Windows and Linux; a container with the debug probe passed through USB or a debug server on the host
GitHub Codespaces	`.devcontainer/codespaces/`	Builds and the simulated tests in the browser, nothing to install
OpenOCD, **SEGGER J-Link**, **Cortex-Debug**	SDK / host installation	Flashing, debugging with Zephyr thread awareness, RTT log

Details: Development environment (p. 46).

2.8 Host tools

Component	Version	Role
Python with **Tkinter**	3.11 or newer	The bench GUI: motion, live plots, Control Table, test scenarios, firmware update
matplotlib	3.11.2	Live plots of position, velocity, current and voltage
pyserial	3.5	The serial link to the bus
smpmgr / smpclient	0.19.1	Firmware upload through MCUboot serial recovery, from the command line or the GUI

Details: GUI (p. 61).

2.9 Documentation

Component	Role
MkDocs with **Material for MkDocs**	This website, in the SystemFromScratch® style, built in strict mode (a broken link fails the build)
WeasyPrint (with Pango)	The printable A4 PDF manual from the same sources (`tools/docs/build_manual.py`): cover, imprint, contents with page numbers, numbered chapters and figures, the license as an appendix, bookmarks

2.10 Engineering practices

The technologies matter less than how they are used. These rules hold across the project:

Practice	In Doppelstep
Everything pinned	Zephyr, libraries, SDK, Python packages and the base image have fixed versions or digests; the CI image is tagged by the hash of its inputs
Layers enforced by a tool	The architecture rules are checked in CI, not only written down
Builds without warnings	The CI builds are kept free of warnings; the one known exception, a warning of the X430 emulator under host gcc, is reported upstream
Hardware isolated	One backend file per MCU family touches registers; the rest is portable C on Zephyr APIs
Tests before hardware	Protocol, motion and protection are tested on the PC against a physical model on every change
Measured numbers only	Sizes, rates and test counts in this documentation come from the build and the tests of this version; what is not measured yet is marked as open
Honest limits	Known gaps are documented and, where possible, held by a test that will flag the fix

CHAPTER 3

Getting started

There are three ways to meet Doppelstep, each one step closer to the real motor. Start with the first: it needs no hardware, and everything you see there behaves the same way on the board.

Path	You need	You get
1. On your PC	Docker and Git, or a GitHub account (Codespaces)	The firmware as a Linux program with a simulated motor; the tests; a bus you can talk to
2. On the board, simulated motor	B-G431B-ESC1 and a USB cable	The real timing (20 kHz interrupt, 1 Mbps UART) with the motor model; the power stage stays off
3. On the board, real motor	Also a two-phase stepper motor and a DC supply (firmware limits 10–28 V, nominal 24 V)	The real thing, after the bring-up checklist

3.1 Get the source

```
git clone https://github.com/SystemFromScratch/sfs-doppelstep.git doppelstep
```

The folder may have any name on your PC; inside the container it is always `/workdir/doppelstep`. Zephyr and the libraries live in a Docker volume next to it (`doppelstep-west`), so they survive container rebuilds and keep file access fast on Windows.

3.2 1. On your PC, no hardware

With Docker, any operating system

```
cd doppelstep
docker compose build                                    # the toolchain image, once
docker compose run --rm zephyr bash scripts/west-init.sh    # Zephyr and the libraries,
once
docker compose run --rm zephyr bash scripts/run-tests.sh    # build and run the SIL and
integration tests
```

The last command builds the `native_sim` firmware with the simulated motor and runs 28 tests against it over DYNAMIXEL Protocol 2.0, in about 30 s after the build. The report is in `build-tests/junit.xml`, the firmware output of every test in `build-tests/pytest/`.

To drive the simulated servo yourself:

```
docker compose run --rm zephyr west build -b native_sim/native/64 -s app -d build-native
docker compose run --rm -p 5555:5555 zephyr python3 tools/sim/run_native.py run --tcp 5555
```

The second command starts the firmware in real time and serves its bus on TCP port 5555 of your PC. In another terminal, on the PC itself:

```
python tools/sim/dxl_smoke_test.py --tcp localhost:5555
# ping, torque on, move, read back
```

The smoke test needs only the Python standard library. For ROS 2 on Linux or WSL, `python3 tools/sim/run_native.py connect --host localhost --port 5555` creates the serial port `/tmp/ttyDXL`; give that port to `dynamixel_hardware_interface` (DYNAMIXEL emulation (p. 25)).

With VS Code

Open the folder in VS Code and press `Ctrl+Shift+B`: the **Workspace Menu** lists every action, on Windows and on Linux. The tasks **Zephyr: Run tests (native_sim)** and **Zephyr: Run (native_sim)** do the steps above. With the Dev Containers extension, **Dev Containers: Reopen in Container** gives you a shell with `west` ready (Development environment (p. 46)).

In the browser, with GitHub Codespaces

Open Doppelstep in GitHub Codespaces (codespaces.new/SystemFromScratch/sfs-doppelstep?quick-start=1&devcontainer_path=.devcontainer%2Fcodespaces%2Fdevcontainer.json): the link selects the configuration **Doppelstep (Codespaces: build and native_sim tests)**. By hand: **Code → Codespaces → … → New with options**, then that configuration. It starts from the published environment image, so nothing is downloaded or built at the start. Then run `bash scripts/run-tests.sh` in its terminal.

3.3 2. On the board, simulated motor

You need the **B-G431B-ESC1** and a USB cable to its ST-LINK. No motor and no supply are needed: the motor model replaces the real samples, and the firmware never switches the power stage on.

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build -- -
DEXTRA_DTC_OVERLAY_FILE=sim.overlay
west flash -d build -r openocd
```

On Windows, run `west` in the container (`docker compose run --rm zephyr …`), or use the VS Code tasks **Zephyr: Pristine Build (simulated motor)** and **Zephyr: Flash (OpenOCD, ST-LINK)**, which use the OpenOCD installed on Windows for flashing.

The board now answers on the ST-LINK Virtual COM Port at 1 Mbps as an XM430-W350 with ID 1. Try it with DYNAMIXEL Wizard, the GUI (p. 61), or on Linux with the smoke test (`python3 tools/sim/dxl_smoke_test.py /dev/ttyACM0`). The log, with the rotor position of the model, comes over RTT: `west rtt -d build -r openocd`.

3.4 3. On the board, real motor

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build
west flash -d build -r openocd
```

> **Follow the bring-up checklist first**
>
> The firmware refuses to switch the power stage on until the dead time of the gate drivers is configured (`dead-time-ns`, 0 by default). Wire the motor as described in Hardware and bring-up (p. 20), then follow its checklist step by step: polarity, current loop, motion, protection.

3.5 What a build produces

File	Use
`build/mcuboot/zephyr/zephyr.hex`	MCUboot, the bootloader
`build/app/zephyr/zephyr.signed.hex`	The signed application, for a debug probe
`build/app/zephyr/zephyr.signed.bin`	The signed application, for an update over the bus (Firmware update)
`build/app/zephyr/zephyr.elf`	Symbols for the debugger
`build-native/zephyr/zephyr.exe`	The `native_sim` firmware (Linux program)

`west flash` loads MCUboot and the signed application together. A pristine build (`-p always`) is needed after a change to `prj.conf`, Kconfig, the devicetree, `west.yml` or the toolchain.

3.6 Next steps

Next	Read
Debug with breakpoints, read the log	Development environment
Connect ROS 2 or the OpenMANIPULATOR-X	DYNAMIXEL emulation
Change the motor, the ID, the limits	Configuration
Understand the code before changing it	Architecture

CHAPTER 4

Hardware and bring-up

Doppelstep runs on the STMicroelectronics **B-G431B-ESC1** Discovery kit, a small electronic speed controller (ESC) board for drone motors, with a detachable debugger. This page describes the board, how the stepper motor is wired to it, how the firmware scales the measurements, and the steps to bring a new setup up safely.

4.1 The board

Part	Detail
Microcontroller	STM32G431CBU6: Arm Cortex-M4F, 170 MHz, 128 KiB flash, 32 KiB SRAM
Power stage	L6387 gate drivers, STL180N6F7 MOSFETs, three half-bridges (legs U, V, W)
Current sensing	Three 3 mΩ low-side shunts, amplified by the internal OPAMPs of the MCU
Debugger	ST-LINK/V2-1 on a detachable part of the board, with a Virtual COM Port
Pinout	UM2516 Rev 4, Table 4 (`boards/st/b_g431b_esc1/doc/index.rst`)

The board definition is part of this repository (`boards/st/b_g431b_esc1/`), because Zephyr 4.5 does not include it: 170 MHz from the 8 MHz crystal, the pinout of the user manual, and the flash runners OpenOCD (default), J-Link, STM32CubeProgrammer and pyOCD.

4.2 Motor wiring

The board has three half-bridges, while a two-phase stepper has two coils. Doppelstep uses the **shared-leg topology** of Stepper-C: each coil sits between its own leg and the common leg W.

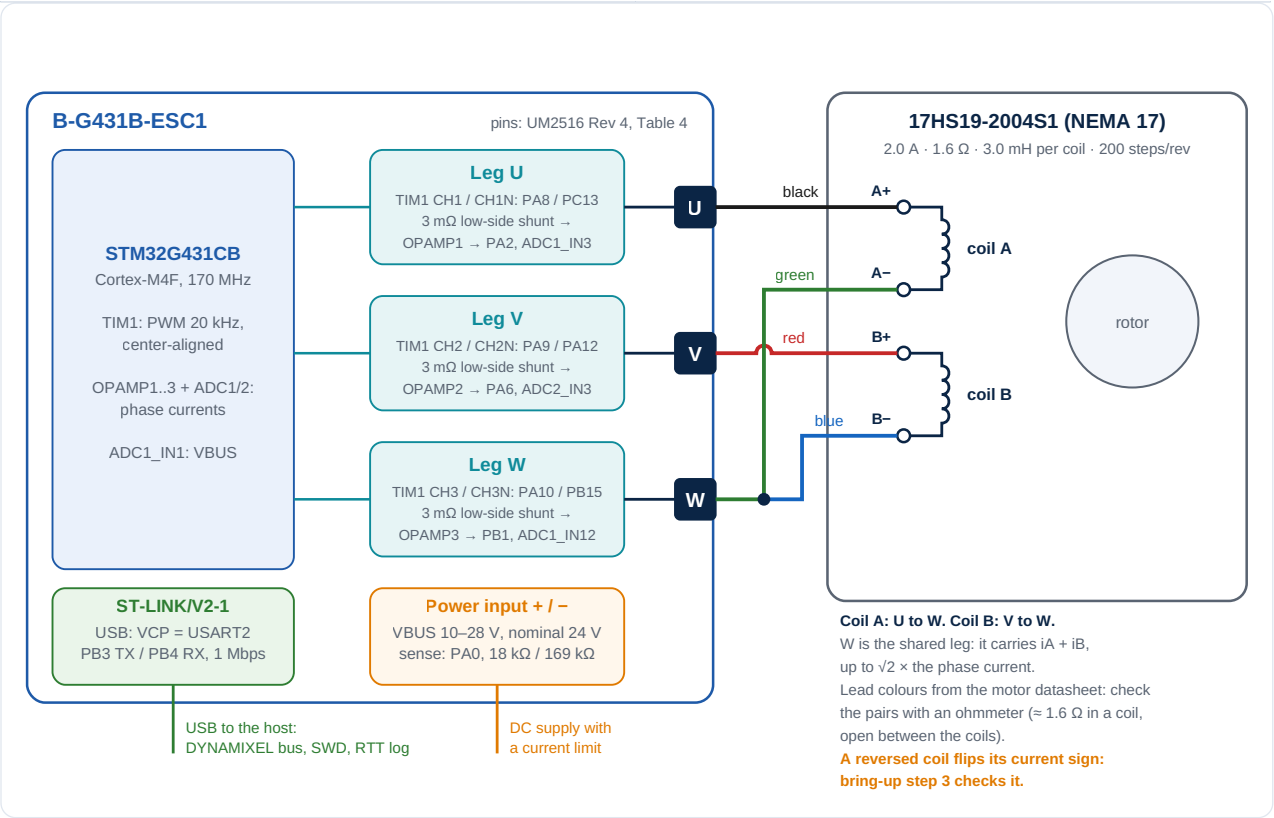


Figure 2: Wiring of the 17HS19-2004S1 to the B-G431B-ESC1: coil A between U and W, coil B between V and W, with the MCU pins of every leg

Motor lead (17HS19-2004S1)	Coil end	ESC1 output	Leg pins (TIM1)	Current sense
Black	A+	U	CH1 / CH1N: PA8 / PC13	OPAMP1 → PA2 (ADC1_IN3)
Green	A−	W	CH3 / CH3N: PA10 / PB15	OPAMP3 → PB1 (ADC1_IN12)
Red	B+	V	CH2 / CH2N: PA9 / PA12	OPAMP2 → PA6 (ADC2_IN3)
Blue	B−	W	CH3 / CH3N: PA10 / PB15	OPAMP3 → PB1 (ADC1_IN12)

The lead colours are those of the motor datasheet. Other motors, and some batches, use other colours: identify the two coils with an ohmmeter before connecting (about 1.6 Ω within a coil, open between the coils). Swapping the two ends of one coil reverses the sign of its current; bring-up step 3 checks it.

W carries the sum of both coil currents, iA + iB, up to √2 times the phase current. The firmware measures its shunt too and switches the outputs off when |iW| exceeds the trip current (`aux-trip` of the motor node).

The supply goes to the power input pads of the board (+ and −). The firmware accepts 10–28 V (`vbus-min-mv`, `vbus-max-mv`) and is tuned for 24 V; use a laboratory supply with a current limit for the first runs. The USB cable to the ST-LINK carries the DYNAMIXEL bus, the debugger and the RTT log.

The tested motor is a **17HS19-2004S1** (NEMA 17, 2.0 A rated, 1.6 Ω and 3.0 mH per coil, 200 steps per revolution). Another motor needs its values in the motor node (Configuration (p. 27)).

4.3 Current and voltage scaling

The scaling comes from the power stage node of the devicetree (`app/boards/b_g431b_esc1.overlay`):

Quantity	Value
Shunt	3 mΩ
Amplifier gain	−64/7 ≈ −9.14 (OPAMP PGA 16 with VINM0 as the bias input, as in ST MCSDK, NuttX and SimpleFOC)
Current full scale	±60.2 A with a 3.3 V reference, about 29 mA per ADC count
VBUS divider	18 kΩ / (18 kΩ + 169 kΩ)
VBUS full scale	34.3 V

4.4 The DYNAMIXEL bus

The bus is **USART2** (PB3 TX, PB4 RX), for now through the ST-LINK Virtual COM Port of the debugger, at 1 Mbps. A DYNAMIXEL X430-T has no direction pin either: for a TTL half-duplex bus later, USART2 can run in single-wire mode (`single-wire;` on `&usart2`, TX pin only). Because USART2 carries the bus, the console and the log use SEGGER RTT through the debug probe.

4.5 Bring-up checklist

Follow these steps in order on a new board or after a change to the power stage. They follow the integration guide of Stepper-C.

1. **Dead time (required).** Enter the value of the B-G431B-ESC1 MCSDK (Motor Control Workbench) project in `dead-time-ns` of `app/boards/b_g431b_esc1.overlay`. While it is 0 the firmware runs, but never enables the gate drivers: Torque Enable answers Result Fail and the log says "enable refused". Check the gate driver input polarity (TIM1 CHx/CHxN active high) against the same project.
2. **Outputs off.** The RTT log shows the current offsets (about 2048 counts) and the ADC readings. Compare Present Input Voltage with a multimeter.
3. **Polarity.** Set `voltage-drive;` and a low `run-current-milliamp` in the motor node. After Torque Enable the measured phase A current must be positive. If not, set `current-sense-invert;` in the power stage node.
4. **Current loop.** Back to current drive with a low run current: check that the current settles at the run current.
5. **Motion.** Small Goal Position steps (DYNAMIXEL Wizard or the GUI (p. 61)), then the OpenMANIPULATOR-X profile.
6. **Protection.** Lower `trip-current-milliamp` and `vbus-max-mv`, then check the trips and the Hardware Error Status.

After a change to the drivers or the threads, also run the hardware checklist (p. 58): it covers what the PC tests cannot see.

CHAPTER 5

DYNAMIXEL emulation

To a host program, Doppelstep is a **DYNAMIXEL XM430-W350** (model number 1020) with one ID, speaking Protocol 2.0. The X430 emulator validates every access exactly as the servo does: ranges and limits, the EEPROM lock, Indirect Addresses, Fast Sync/Bulk Read, the Bus Watchdog. The firmware then maps the registers onto the motor (`app/src/joint_bridge.c`). This page lists that mapping, the units, and what differs from a real XM430.

5.1 Register mapping

Register	Mapping
Torque Enable (64)	`stepper_enable()` / `stepper_disable()`; refused (Result Fail) while the dead time is not set or with a fault
Operating Mode (11)	3 position, 4 extended position, 1 velocity; 0, 5, 16 are refused (Data Range Error)
Goal Position (116)	`stepper_ctrl_move_to()`; limits from Profile Velocity/Acceleration (velocity- or time-based, Drive Mode bit 2), Velocity Limit and the `max-velocity` / `max-acceleration` properties of the motor node
Goal Velocity (104)	`stepper_ctrl_run()` or `stepper_ctrl_stop()` (velocity mode)
Drive Mode (10) bit 0, Homing Offset (20)	Direction and offset of the position mapping
Current Limit (38)	Run current = min(limit × 2.69 mA, `run-current-milliamp`)
Present Position (132), Position Trajectory (140)	Commanded position (no encoder)
Present Velocity (128), Velocity Trajectory (136)	Commanded velocity
Present Current (126)	Phase current amplitude $\sqrt{(iA^2 + iB^2)}$
Present Input Voltage (144)	Measured VBUS (0.1 V)
Moving (122), Moving Status (123)	From the status of the motor driver
Hardware Error Status (70)	Over/undervoltage → Input Voltage; overcurrent, W trip, HAL → Electrical Shock
Bus Watchdog	`stepper_ctrl_stop()`
REBOOT / FACTORY_RESET	Outputs off, faults cleared; on the board (MCUboot) REBOOT then resets the MCU: silent for 500 ms plus startup, Present Position restarts (Firmware update)
CLEAR (multi-turn)	Position redefined within one turn
Baud Rate (8)	USART2 reconfigured after the status packet

5.2 Units

The conversions are in `app/src/joint.c`:

Quantity	DYNAMIXEL unit	Motor side
Position	4096 pulses per joint revolution	3200 microsteps per motor revolution (200 × 16)
Velocity	0.229 rev/min	microsteps per second
Accelera-tion	214.577 rev/min²	microsteps per second²
Current	2.69 mA	mA
Gear	`gear-motor-revs / gear-joint-revs` of the servo node (1/1 by default)	

5.3 Startup position

The motor has no encoder. At power-on the rotor position is reported as Present Position **2048** (`start-position` of the servo node, 0 rad for `dynamixel_hardware_interface`): power the joint up in that position. With the torque off the rotor can be moved by hand, and the reported position is then no longer the real one.

5.4 Factory values and the EEPROM

Value	Doppelstep	A real XM430
ID	`factory-id` of the servo node, 1	1
Baud rate	`factory-baud-code` 3 = 1 Mbps, the OpenMANIPULATOR-X bus speed	57600 bps

The factory values apply while no EEPROM block is stored and after FACTORY_RESET. ID, Baud Rate and the other EEPROM registers written over the bus are stored in flash (NVS, the `storage` partition of the servo node) and survive a power cycle and a firmware update. Flash writes are refused while the power stage is enabled: the MCU has a single flash bank, and an erase stalls the CPU.

5.5 Connecting a host program

Host program	Port
On the board	ST-LINK Virtual COM Port (`COM5` on Windows, `/dev/ttyACM0` on Linux), 1 Mbps
`native_sim` on the same Linux machine or Dev Container	`build-native/ttyDXL`
`native_sim` in Docker, host program on WSL or another Linux machine	`python3 tools/sim/run_native.py connect --host localhost --port 5555` → `/tmp/ttyDXL`
`native_sim` in Docker, smoke test on Win-dows	`python tools\sim\dxl_smoke_test.py --tcp localhost:5555`

For ROS 2, set the port of the `dynamixel_hardware_interface` description to `/tmp/ttyDXL` (or `build-native/ttyDXL`); the baud rate is accepted and ignored on `native_sim`. The pseudoterminal driver polls for the first byte of a packet every 10 ms, so an answer can take up to about 10 ms (a real bus: well under 1 ms): keep the host's packet timeout above that.

5.6 OpenMANIPULATOR-X

Its `ros2_control` description expects five XM430-W350 on one bus (IDs 11–15, 1 Mbps, Operating Mode 3, Drive Mode 4 = time-based profile). One Doppelstep answers for **one** ID, so over the Virtual COM Port the host sees one joint: set `factory-id` to the joint's ID (for example 11). Fast Sync/Bulk Read answers come only from the IDs of one emulator bus.

The integration tests run this loop against the firmware with the official DynamixelSDK: Sync Read of current, velocity and position, Sync Write of the goal, at 100 Hz (Tests (p. 54)).

5.7 Differences from a real XM430-W350

Topic	Doppelstep
Position and velocity readings	Commanded values: there is no encoder, and lost steps are not detected yet
Operating Modes 0, 5, 16	Refused with Data Range Error
Factory baud rate	1 Mbps instead of 57600 bps
Reboot on the board	Resets the MCU: 500 ms of silence plus startup, Present Position restarts at 2048
Present Temperature	Keeps its table value: the NTC sensor is not converted yet, no overheating trip
Several IDs on one bus	One ID per board

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. Doppelstep is an independent project and is not affiliated with or endorsed by ROBOTIS.

CHAPTER 6

Configuration

Doppelstep is configured the Zephyr way. The **devicetree** describes the hardware: the power stage, the motor and its limits, the servo on its UART. **Kconfig** selects the compiled features, the threads and the buffers. Both are read at build time, so a change needs a rebuild (a pristine one for the devicetree and Kconfig). Settings that a host may change at run time, such as the ID or the baud rate, are servo registers (DYNAMIXEL emulation (p. 24)).

6.1 Where each setting lives

What	Where
Motor (R, L, rated current, back-EMF, steps), stage limits, control (microsteps, bandwidth, currents, velocity, acceleration)	Motor node, `app/boards/<board>.overlay`
PWM frequency, dead time, ADC channels, shunt, amplifier gain, VBUS divider	Power stage node, `app/boards/b_g431b_esc1.overlay`
DYNAMIXEL ID, factory baud, start position, gear, NVS partition	Servo node, a child of the UART
Threads, queues, offset samples, command timeout	`CONFIG_SFS_*`, `app/Kconfig`
Stepper-C build options (backend, topology, sine table, `-O2`)	`CONFIG_SSTEP_*`, `modules/stepper-c/Kconfig`
X430 model, devices per bus, protocol features	`CONFIG_X430_*` (Kconfig of the emulator)
Bus buffer, poll period, persistent storage, Reboot behaviour	`CONFIG_SFS_DXL_*`, `drivers/x430/Kconfig`
Simulated motor (model parameters, scenario, log)	`CONFIG_SFS_SIM_*`, `app/sim.overlay` (Simulated motor)
Board-specific Zephyr options (FPU, RTT, OPAMP; `native_sim` tick and logs)	`app/boards/<board>.conf`
Bootloader, flash layout, serial recovery	`app/sysbuild.conf`, `app/sysbuild/`, partitions in the board devicetree (Firmware update)

Open the Kconfig menu of a build with `west build -d build -t menuconfig`.

6.2 The motor node

The values for the 17HS19-2004S1 motor and the B-G431B-ESC1, from `app/boards/b_g431b_esc1.overlay`:

Property	Value	Meaning
`phase-resistance-milliohm`	1600	Coil resistance (datasheet)

Property	Value	Meaning
`phase-inductance-microhenry`	3000	Coil inductance (datasheet)
`rated-current-milliamp`	2000	Rated phase current (datasheet)
`back-emf-uv-s-per-rad`	208600	Back-EMF constant
`full-steps-per-rev`	200	Full steps per revolution
`vbus-nominal-mv`, `vbus-min-mv`, `vbus-max-mv`	24000, 10000, 28000	Supply voltage: nominal and trip limits
`trip-current-milliamp`	3500	Overcurrent trip
`aux-trip`	set	Also trip on the shared leg W
`slow-loop-hz`	1000	Motion loop rate; the PWM rate must be an integer multiple
`microsteps`	16	Microsteps per full step
`current-bandwidth-hz`	1000	Bandwidth of the current controller
`run-current-milliamp`, `hold-current-milliamp`	1500, 500	Current while moving and at rest
`hold-delay-ms`	500	Time at rest before the hold current applies
`max-velocity`, `max-acceleration`	16000, 64000	Upper limits of every motion profile

6.3 The power stage node

Property	Value	Meaning
`pwm-frequency`	20000	PWM and fast-loop rate, Hz
`dead-time-ns`	0	Dead time of the half-bridges. **Must be set before the outputs can be enabled** (bring-up)
`vbus-divider`	18000, 169000	VBUS divider resistors, Ω
`current-sense-invert`	not set	Inverts the current measurement, if bring-up step 3 requires it

Shunt, amplifier gain and the ADC channels are in the same node; their values and the resulting full scale are in Hardware and bring-up (p. 22).

6.4 The servo node

Property	Default	Meaning
`factory-id`	1	DYNAMIXEL ID while no EEPROM is stored and after Factory Reset
`factory-baud-code`	3 (1 Mbps)	Baud Rate register value, same conditions
`start-position`	2048	Present Position at power-on (no encoder)

Property	Default	Meaning
`gear-motor-revs`, `gear-joint-revs`	1, 1	Gear ratio between motor and joint
`storage`	`&storage_partition`	Flash partition of the EEPROM area (NVS)

6.5 Variants of the build

Variant	Command
Board, real motor	`west build -p always --sysbuild -b b_g431b_esc1 -s app -d build`
Board, simulated motor	add `-- -DEXTRA_DTC_OVERLAY_FILE=sim.overlay`
PC (`native_sim`)	`west build -b native_sim/native/64 -s app -d build-native`
Fixed-point Stepper-C backend	add `-- -DCONFIG_SSTEP_MATH_FIXED=y`

`sim.overlay` stays in the CMake cache of a build directory, so a following incremental build keeps the simulated motor until the next pristine build.

CHAPTER 7

Architecture

Doppelstep is a Zephyr module with one application. Hardware and libraries sit behind drivers. The application owns the threads and the queues and connects the drivers. This page explains the layers, their rules and the path of a command, so you can read the code or add to it. For a new MCU, continue with Porting to a new MCU family (p. 43).

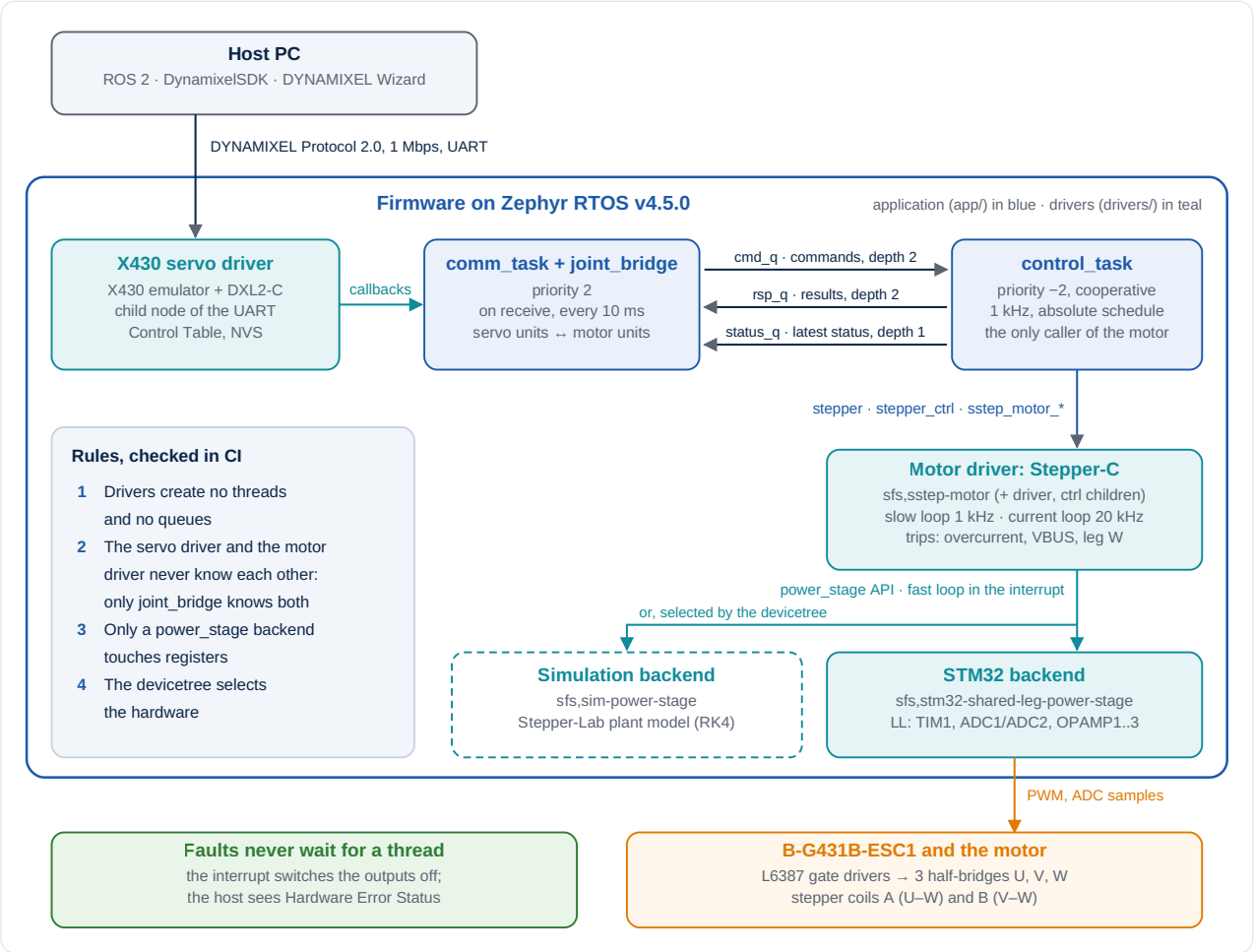


Figure 3: Layers of the firmware, from the host to the motor

7.1 Layout

```
doppelstep/
├── zephyr/module.yml     the repository is a Zephyr module
├── CMakeLists.txt, Kconfig   build modules/ and drivers/
├── app/                  application: tasks, queues, register mapping
├── drivers/              power_stage/, motor/, x430/
├── include/doppelstep/drivers/   public driver APIs
├── dts/bindings/         devicetree bindings and the sfs vendor prefix
├── boards/               out-of-tree board
├── modules/              build glue of the libraries
└── tests/                ztest suites and pytest SIL and integration tests
```

The layout follows the Zephyr `example-application`. West finds the module because the manifest repository is a west project, so the application and every test application get the drivers, the board and the bindings without extra CMake.

7.2 Four rules

1. **Drivers create no threads and no queues.** A driver offers a function such as `sstep_motor_process()` or `x430_servo_process()`, and the application calls it from a loop it owns. The only exception is the fast loop, which stays in the interrupt inside the driver.
2. **The X430 driver does not know the motor, and the motor driver does not know DYNAMIXEL.** Only `app/src/joint_bridge.c` knows both, so X430 units never reach the motor driver.
3. **Only a `power_stage` backend touches registers.** LL calls, the cycle counter, OPAMPs and direct interrupts appear nowhere else.
4. **The devicetree selects the hardware.** The `power-stage` phandle of the motor selects the backend, the servo is a child node of its UART, and a phandle selects the flash partition.

`scripts/check-layers.sh` enforces rules 2 and 3 in CI, and also rejects floating point above the backends and any output switching in the simulation backend.

7.3 Drivers

Driver	Compatible	API
Power stage, STM32	`sfs,stm32-shared-leg-power-stage`	Class `power_stage` (`power_stage.h`)
Power stage, simulation	`sfs,sim-power-stage`	Class `power_stage`, plus `power_stage_sim.h`
Motor (Stepper-C)	`sfs,sstep-motor` with the children `sfs,sstep-stepper-driver` and `sfs,sstep-stepper-ctrl`	Zephyr classes `stepper` and `stepper_ctrl`, plus `sstep_motor.h`
Servo (X430 emulator)	`sfs,x430-servo`, a child of the UART	`x430_servo.h`

Why a custom class for the power stage? Zephyr's PWM API has no dead time, center alignment or complementary outputs, and its ADC API has no timer-triggered injected conversions. A 20 kHz current loop needs all of them, so MCU-specific code is unavoidable. The `power_stage` class keeps that code in one file per MCU family, the way Zephyr does for its own driver classes.

Why three nodes for the motor? Zephyr splits stepper devices into a hardware driver (`stepper`: enable, disable, microsteps) and a motion controller (`stepper_ctrl`: moves, ramp, position). Integrated chips such as `adi,tmc50xx` use a parent node with one child for each class, and Stepper-C plays the same role. What the classes lack (slow loop, status, currents, faults) is the extension on the parent node.

7.4 Contexts

Context	Priority	Rate	Work	Never
Interrupt of the backend	Highest (on the STM32: zero-latency ADC1_2 interrupt)	PWM rate, 20 kHz	Currents and VBUS, shared-leg trip, fast loop `sstep_fast_loop()`, immediate shutdown on a fault	Kernel calls
`control_task`	−2, cooperative	1 kHz, absolute schedule	Slow loop `sstep_motor_process()`, commands, publication of the status	Waiting on an outgoing queue
`comm_task`	2	On receive, and every 10 ms	`x430_servo_process()`, `joint_bridge`, live registers	Calling the motor driver
`main`	0	—	Startup, then the STATUS LED	Sending commands

Every call into the motor driver comes from `control_task`, so the driver needs no lock for its status. With `CONFIG_ASSERT` the driver checks this.

7.5 Queues

`app/src/app_msg.h` defines three queues. Messages travel by value.

Queue	Direction	Depth	Use
`cmd_q`	comm → control	2	Commands, with a sequence number
`rsp_q`	control → comm	2	Results, 0 or `-errno`
`status_q`	control → readers	1	Latest status only

`control_task` never waits on an outgoing queue, so communication cannot stall the control loop. `app_call()` waits at most `CONFIG_SFS_CMD_TIMEOUT_MS` and discards a late answer by its sequence number, so `comm_task` cannot hang either. A command that timed out may still execute later, and the log records the timeout.

7.6 A command, from the bus to the motor

Goal Position:

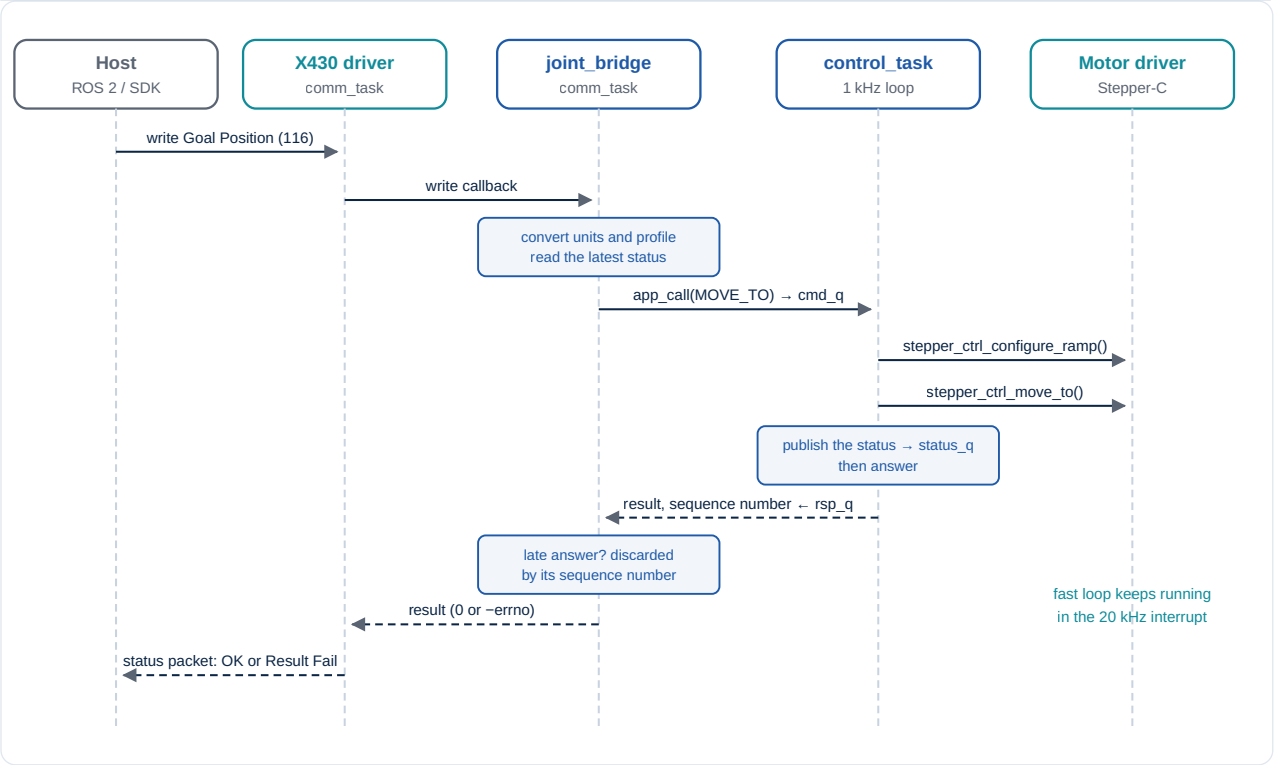


Figure 4: A Goal Position write, from the bus to the motor and back

The host gets the result in the status packet, so `joint_bridge` waits for it. Because the status is published before the answer, the sender always sees a status that reflects its command.

Situation	Reaction	Host sees
The driver rejects a command	Result in `rsp_q`	Result Fail
Unsupported Operating Mode	`joint_bridge`, motor not involved	Data Range Error
Overcurrent, bus voltage, stall	Outputs off in the interrupt, flag in the status	Hardware Error Status, Torque Enable 0
`cmd_q` full, or no answer in time	`-EBUSY` or `-ETIMEDOUT`, log entry on a timeout	Result Fail

Motor protection depends on no task, because the interrupt handles electrical faults.

7.7 Startup

On the board MCUboot runs first: it checks the signature of the application and waits 500 ms for a firmware update over USART2 (Firmware update (p. 40)). Then the application starts:

1. Device initialisation at `POST_KERNEL`: power stage, motor driver, servo. Outputs stay off.
2. `main` checks the devices and measures the current sensor offsets.
3. `main` starts `control_task`, which publishes the first status, and connects the servo through `joint_bridge`.
4. `main` starts `comm_task` and continues with the LED.

The fast-loop interrupt already runs after step 1, while the motor driver is ready and the sensor offsets are not yet known. That is safe because the outputs are off, the current trips of Stepper-C only act when the outputs are on, and `aux_trip` stays false until calibration has succeeded.

Until step 3 finishes, nothing can send an enable command, so the motor cannot start with uncalibrated sensors. On an error in steps 1 or 2 the tasks do not start. In every error case the outputs stay off and the LED blinks fast.

7.8 Configuration

What	Where
Motor, limits, tuning, joint, power stage, PWM, scaling	Devicetree properties (`app/boards/*.overlay`)
Compiled features, threads, buffers, timeouts	Kconfig
Board-specific Zephyr options	`app/boards/<board>.conf`
Simulated motor on the board	`-DEXTRA_DTC_OVERLAY_FILE=sim.overlay`
Bootloader, flash layout, serial recovery	`app/sysbuild.conf` , `app/sysbuild/` , partitions in the board devicetree

CHAPTER 8

Simulated motor

Doppelstep can run without a motor: a model of the power stage and the stepper motor takes the place of the hardware, and everything above it (ROS 2, DYNAMIXEL Protocol 2.0, the X430 emulator, Stepper-C) runs unchanged. The model is not a copy made for the firmware. It is the plant model of Stepper-Lab (github.com/AntonVynohradov/Stepper-Lab), the same C code as its Xcos block. Its parameters come from the same files the Scilab model uses, and the power stage values come from the ngspice model of sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice).

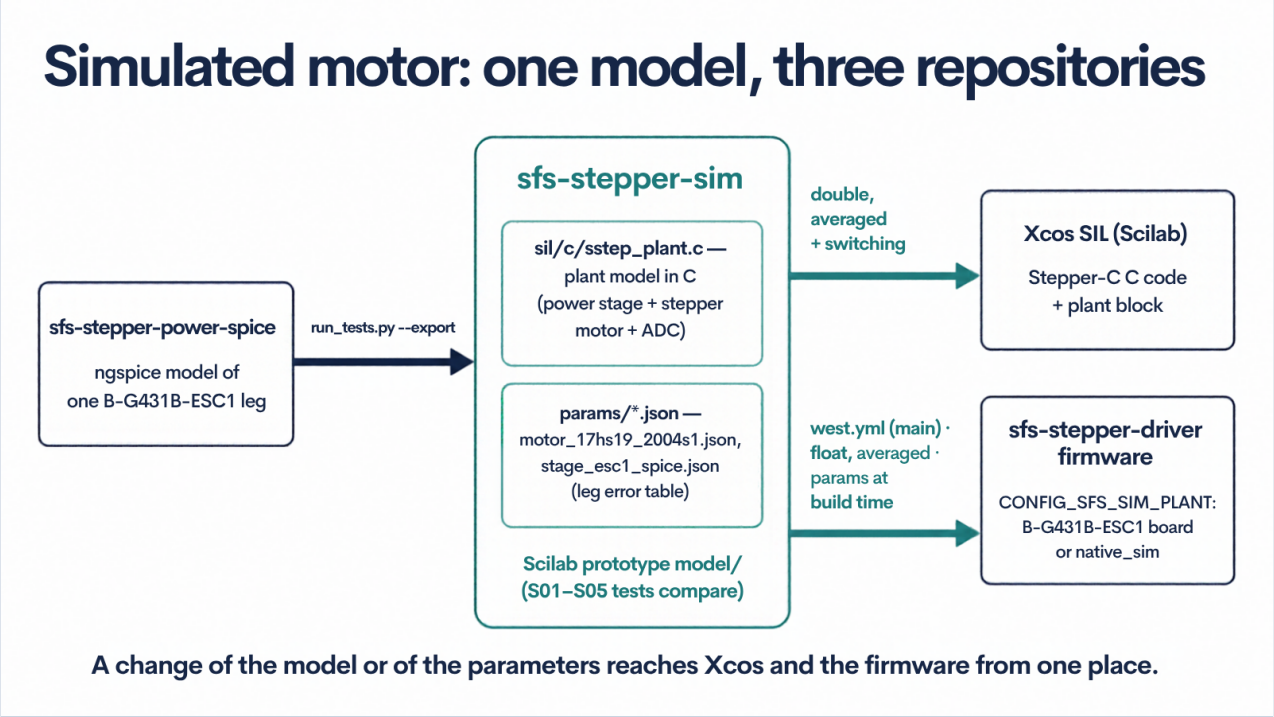


Figure 5: One model, three repositories

8.1 How the pieces fit

Repository	Role	In the firmware build
sfs-stepper-power-spice	ngspice model of one ESC1 leg. Writes the stage parameters of Stepper-Lab.	Not built. West project, next to Stepper-Lab.
Stepper-Lab	Scilab prototype, Xcos SIL, the plant model in C (`sil/c/sstep_plant.c`), parameters (`params/*.json`).	`sstep_plant.c` compiled as a Zephyr library (`float`, averaged stage). The JSON files are read at configure time.
Doppelstep	This firmware. The backend `sfs,sim-power-stage` (`drivers/power_stage/`) adapts the model to Zephyr: ADC scaling, publication. `app/src/sim_log.c` writes the log line.	Driver and application

Both simulation repositories are west projects, pinned to the tested commit of their `main` until their first release (`west.yml`). The build glue is `modules/stepper-lab/`.

What is shared and what stays offline:

- **Shared:** the model code (`sstep_plant.c`) and its parameters (`params/*.json`). A change made there reaches the Xcos block and the firmware.
- **Offline:** the ngspice simulations, the switching power stage (switching instants, dead time, body diodes) and the Xcos diagram. They are too slow for a 50 µs control period. Their results enter the firmware model as parameters, most of all the **leg voltage error table**.

8.2 The model

State `x = [iA, iB, ω, θe]` with `θe = p·θ` and `p = Ns / 4`:

```
L diA/dt = vA − R iA + Kt ω sin θe
L diB/dt = vB − R iB − Kt ω cos θe
J dω/dt  = Kt (−iA sin θe + iB cos θe) − Td sin 4θe − TL − B ω − Tc tanh(ω / ωc)
dθe/dt   = p ω
```

- **Power stage:** shared leg of the B-G431B-ESC1, so `vA = vU − vW` and `vB = vV − vW`. Each leg gives `duty · Vbus` plus, by default, the **leg voltage error** measured with ngspice as a function of the leg current. The error covers the dead time, the body diodes and the leg resistance. In Xcos (test S05) this brings the voltage the current loop commands to within 19 mV of the switching stage, against 118 mV with ideal legs.
- **Timing:** one PWM period of computation delay, as on the MCU. In every period: `sim_plant_sample()` (ADC counts of the state) → Stepper-C fast loop → `sim_plant_step()` (RK4 over the period with the voltages written in the previous period).
- **Outputs off:** the windings are open. The currents are zero and the rotor coasts.
- **Precision:** single precision, with `θe` kept within one turn plus a turn count, so long runs do not lose accuracy.
- **Load:** `CONFIG_SFS_SIM_LOAD_UNM`, or `power_stage_sim_set_load()`. It is a constant active torque, like gravity on a joint, so it pulls the rotor back when the motor loses synchronism or the torque is off.

8.3 Parameters

Source file (Stepper-Lab)	Written by	Values
`params/motor_17hs19_2004s1.json`	`python tools/export_params.py`, from `params/motor_17hs19_2004s1.sci`	R, L, Kt, Ns, J, B, Td, Tc, ωc
`params/stage_esc1_spice.json`	sfs-stepper-power-spice `python tools/run_tests.py --export`	leg resistance, diode drop, effective dead time, leg error table and its operating point (Vbus, fs)

At configure time the glue checks `"format": 1` and generates `build*/zephyr/include/generated/sstep_plant_params.h`. A change of a JSON file re-runs CMake on the next build. The file names are `CONFIG_SSTEP_PLANT_MOTOR_PARAMS` and `CONFIG_SSTEP_PLANT_STAGE_PARAMS`.

The backend is selected by the devicetree: a `sfs,sim-power-stage` node and the `power-stage` phandle of the motor. The firmware settings on top of the files:

Option	De-fault	Meaning
`CONFIG_SFS_SIM_R_MOHM`, `_L_UH`, `_KT_UNM_PER_A`, `_INERTIA_G_MM2`, `_VISCOUS_UNM_S`, `_DETENT_UNM`, `_COULOMB_UNM`	−1	Override one value of the motor file (−1 keeps the file value). Useful to test the controller against a motor that differs from the values it is tuned for.
`CONFIG_SFS_SIM_LEG_ERROR`	y	Use the ngspice leg error table. A warning is logged when Vbus or fs differ by more than 5 % from the operating point of the table.
`CONFIG_SFS_SIM_VBUS_MV`	24000	Supply voltage of the model
`CONFIG_SFS_SIM_LOAD_UNM`, `_LOCKED_ROTOR`	0, n	Scenario
`CONFIG_SFS_SIM_SUBSTEPS`, `_LOG_PERIOD_MS`	1, 1000	RK4 steps per period, period of the log line

The controller keeps its own values (properties of the motor node, the datasheet). The startup log names the model file and its main values.

8.4 Two ways to run it

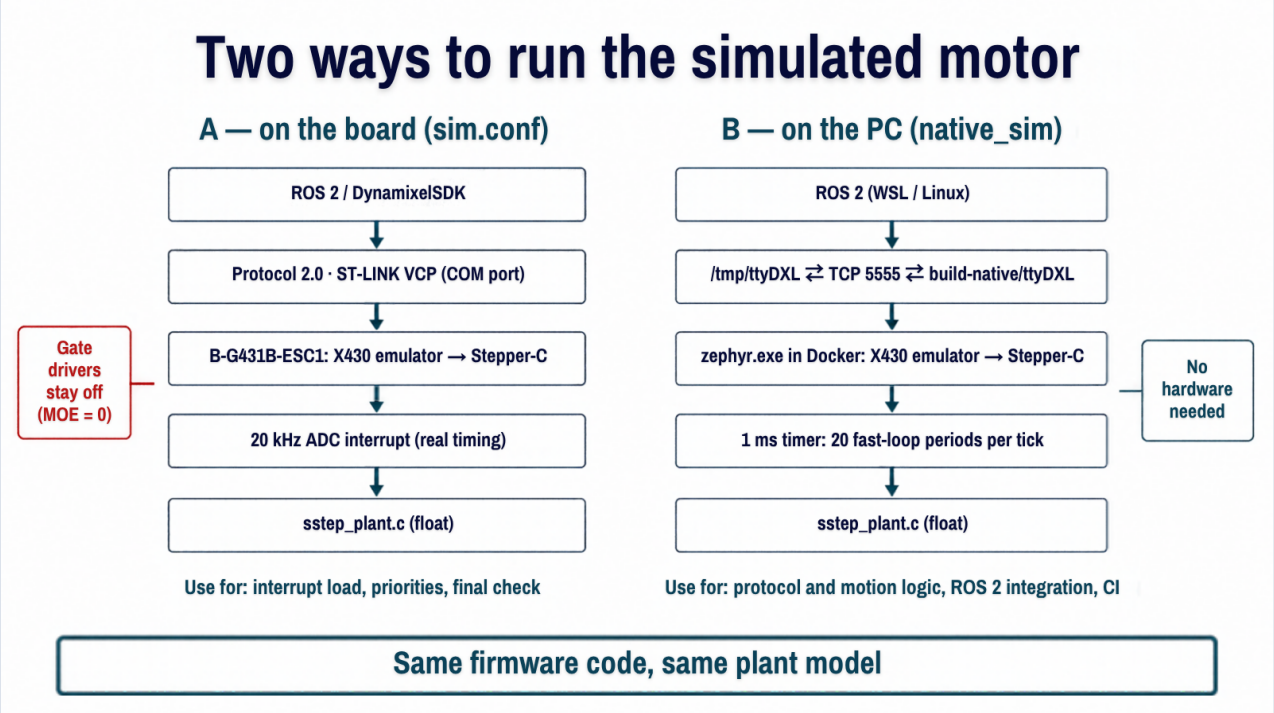


Figure 6: Two ways to run the simulated motor

Aspect	A: on the board	B: on the PC (`native_sim`)
Hardware	B-G431B-ESC1, no motor needed	none
Timing	real: 20 kHz interrupt, UART 1 Mbps	simulated, kept to real time

Aspect	A: on the board	B: on the PC (`native_sim`)
Bus	ST-LINK VCP (COM port)	pseudoterminal, or TCP 5555 from Docker
Debugging	J-Link / OpenOCD, RTT	gdb on `zephyr.exe`, logs on stdout
Use it for	interrupt load, priorities, final check	protocol and motion logic, ROS 2 integration, CI

A: on the board

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build -- -
DEXTRA_DTC_OVERLAY_FILE=sim.overlay
```

Or use the VS Code task **Zephyr: Pristine Build (simulated motor)**. TIM1 and the ADCs keep the real timing, and the samples come from the model. The simulation backend never switches the outputs of the real stage, so the main output enable (MOE) stays off and the board runs with or without a motor and bus voltage. The RTT log shows the commanded and rotor position, the lag in electrical degrees, the currents, and the longest fast-loop callback in cycles and µs.

B: on the PC

```
west build -b native_sim/native/64 -s app -d build-native   # task "Zephyr: Build
(native_sim)"
python3 tools/sim/run_native.py run [--tcp 5555]
# task "Zephyr: Run (native_sim)"
python3 tools/sim/dxl_smoke_test.py build-native/ttyDXL                # ping, torque,
move, without ROS 2
```

`run_native.py` links the bus pseudoterminal to `build-native/ttyDXL`. In Docker (the task on the Windows host) it also serves the bus on TCP 5555. On the ROS 2 machine, `python3 tools/sim/run_native.py connect --host localhost --port 5555` creates `/tmp/ttyDXL`. From Windows, `python tools\sim\dxl_smoke_test.py --tcp localhost:5555` runs the smoke test. The pseudoterminal driver polls for the first byte of a packet every 10 ms, so keep the host's packet timeout above that.

8.5 Workflows

Change a motor parameter. Edit `params/motor_*.sci` in Stepper-Lab, then run `python tools/export_params.py` and its tests, and push. In the Doppelstep workspace run `west update` and rebuild. CMake regenerates `sstep_plant_params.h`.

Change the power stage model (ngspice). In sfs-stepper-power-spice run `python tools/run_tests.py --export --sim <path to Stepper-Lab>`. It updates `export/stage_esc1.json` and `params/stage_esc1_spice.*` of Stepper-Lab. In the west workspace the two repositories are neighbours under `modules/lib`. `--export-only` rewrites those files from the last export without ngspice. Commit both repositories, then `west update` and rebuild.

Change the model itself. Edit `sil/c/sstep_plant.c` and the Scilab prototype (`model/`) together. Run `tests/run_all.sce` in Stepper-Lab (S01–S04 compare Xcos with the prototype sample by sample, S05 checks the leg error table). Then push, run `west update` here and rebuild.

Try local changes before pushing. Point the build at a local checkout:

```
west build ... -- -DSTEPPER_LAB_SOURCE_DIR=/path/to/Stepper-Lab
```

With Docker, mount the checkout first, for example `docker compose run --rm -v C:/path/to/Stepper-Lab:/sim:ro zephyr ...` with `-DSTEPPER_LAB_SOURCE_DIR=/sim`. The value stays in the CMake cache of that build directory until a pristine build.

8.6 Checks

The firmware tests on `native_sim` (SIL with the model, integration with DynamixelSDK) are described in SIL and integration tests (p. 54).

Where	Check
sfs-stepper-power-spice	`tools/run_tests.py`: SP01–SP03 against the datasheet; `--export` writes the contract file
Stepper-Lab	`tests/run_all.sce`: prototype tests and S01–S05 (Xcos SIL with `sstep_plant.c`); `tools/export_params.py --check`
Doppelstep	pristine builds of `b_g431b_esc1` (with and without `sim.overlay`) and `native_sim/native/64` without warnings; `tools/sim/dxl_smoke_test.py` against `run_native.py`

8.7 Limits

- Present Position and Present Velocity are the commanded values, as with the real motor, which has no encoder. The rotor position of the model appears only in the log.
- The rotor of the model is never exactly still. With the 12-bit ADC the current loop toggles by one count (29.4 mA, about 570 mV of phase voltage), which excites the weakly damped mechanical resonance, about 125 Hz at the 0.5 A hold current. It reaches about 3 rpm (0.2 microstep), depending on the rotor angle. Tests therefore check a stopped motor with a threshold, not with 0 rpm. A real motor has friction and ADC noise: measure it (p. 58).
- The leg error table is valid at its operating point (24 V, 20 kHz, timer dead time 500 ns, assumed board values). Re-run ngspice when these change.
- On `native_sim`, code takes no simulated time. Interrupt load and priorities are validated only on the board (variant A).
- `west.yml` pins both simulation repositories to a commit of `main`. Move them to tags at their first release, so that the model version has a name and not only a hash.

CHAPTER 9

Firmware update

The board updates its firmware over USART2, the UART of the DYNAMIXEL bus. The bootloader is MCUboot, built by Zephyr sysbuild, and the update uses its serial recovery: an SMP (MCUmgr) server in the bootloader that writes the new image into the application slot. The application itself contains no update code.

9.1 Flash layout

STM32G431CB: 128 KiB of flash, 2 KiB pages.

Partition	Address	Size	Content
`boot_partition`	`0x00000`	38 KiB	MCUboot
`slot0_partition`	`0x09800`	86 KiB	Application, signed
`storage_partition`	`0x1f000`	4 KiB	NVS: EEPROM area of the emulated DYNAMIXEL

`boot_partition` is the measured size of MCUboot rounded up to a page. An update never touches `storage_partition`, so the EEPROM values (ID, Baud Rate, limits) survive it.

9.2 Build and flash

Every build for the board is a sysbuild build:

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build
west flash -d build -r openocd     # MCUboot and the signed application
```

File	Content
`build/mcuboot/zephyr/zephyr.hex`	MCUboot
`build/app/zephyr/zephyr.signed.bin`	Signed application: the file for an update
`build/app/zephyr/zephyr.signed.hex`	Signed application for a debug probe
`build/app/zephyr/zephyr.elf`	Symbols of the application for the debugger

A build directory made before MCUboot was added needs one pristine build. The VS Code tasks and debug configurations already load both images.

9.3 Serial recovery

MCUboot listens on USART2 at **1 Mbps**, 8N1. The rate is fixed: the bootloader does not read the Baud Rate register of the application.

Situation	MCUboot
Any reset (power-on, NRST, ST-LINK, DYNAMIXEL Reboot)	Waits 500 ms for an SMP request. With a request it stays in serial recovery, without one it starts the application
Slot 0 holds no valid image (first flash, interrupted upload)	Stays in serial recovery

The host must get an SMP request to MCUboot inside the 500 ms window. The window opens only after the reset and the start of MCUboot, so a request sent right after the status packet of the Reboot can arrive before MCUboot listens and be lost. A host program therefore sends the Reboot, then repeats a short SMP request (for example `os echo`) every few tens of milliseconds until MCUboot answers, and then uploads. A separate smpmgr process started after the Reboot may start too late: its startup can take longer than 500 ms.

DYNAMIXEL Reboot on the board

With MCUboot, the Reboot instruction (0x08) resets the MCU after its status packet (`CONFIG_SFS_DXL_REBOOT_RESETS_MCU`, default on with MCUboot). This differs from an XM430:

- Present Position restarts. The open-loop position lives in RAM and there is no encoder.
- The device does not answer for 500 ms plus the startup time.
- The motor outputs are off before the reset, as with every Reboot.

On `native_sim` Reboot stays emulated, so the SIL tests keep the XM430 behaviour.

9.4 Upload with the GUI

The **Firmware update** tab of the GUI (p. 61) runs the whole sequence in one process: Reboot, SMP echo repeated until MCUboot answers, image list, upload, image list again, reset, and a DYNAMIXEL ping of the new application. It passes only when slot 0 holds an image whose hash equals the SHA-256 TLV of the chosen `zephyr.signed.bin`. MCUboot lists only images that pass its signature check, so a listed image with that hash is the file, verified.

9.5 Upload with smpmgr

smpmgr (github.com/intercreate/smpmgr) is in `requirements.txt`. The line options match the SMP buffers of MCUboot (`CONFIG_BOOT_MAX_LINE_INPUT_LEN=128`, `CONFIG_BOOT_LINE_BUFS=8`).

```
smpmgr --port COM5 --baudrate 1000000 --line-length 128 --line-buffers 8 \
    image upload build/app/zephyr/zephyr.signed.bin
smpmgr --port COM5 --baudrate 1000000 --line-length 128 --line-buffers 8 \
    os reset
```

Use `image upload`, not `upgrade`: with one slot there is no test image to mark, and serial recovery writes slot 0 directly. `COM5` is the ST-LINK Virtual COM Port (Linux: `/dev/ttyACM0`).

9.6 Signature

MCUboot checks an RSA-2048 signature. The build signs with the development key of MCUboot, which is public: any image signed with it boots. For a product, generate a key with `imgtool keygen`, keep it outside the repository and set `SB_CONFIG_BOOT_SIGNATURE_KEY_FILE` in `app/sysbuild.conf`. Boards that already run MCUboot with the development key reject images signed with the new key, so they need MCUboot flashed again through the debug probe.

9.7 Configuration

What	Where
MCUboot on, one slot, signature	`app/sysbuild.conf`
Serial recovery, wait window, RAM	`app/sysbuild/mcuboot.conf`
USART2 and its rate for MCUboot	`app/sysbuild/mcuboot.overlay`
Partitions	`boards/st/b_g431b_esc1/b_g431b_esc1.dts`
Reboot resets the MCU	`CONFIG_SFS_DXL_REBOOT_RESETS_MCU` (`drivers/x430/Kconfig`)

9.8 Troubleshooting

Symptom	Cause	Remedy
smpmgr times out, the application answers DYNAMIXEL	No request reached MCUboot inside the 500 ms window	Repeat a short SMP request after the Reboot until MCUboot answers, then upload
smpmgr gets no answer at all	The Baud Rate register is not 1 Mbps: the application answers at that rate, MCUboot listens at 1 Mbps	Use `--baudrate 1000000` for smpmgr whatever the register says
The application does not start after loading `zephyr.elf` with a debugger	The ELF has no MCUboot header and signature: MCUboot stays in serial recovery	Load `zephyr.signed.hex` (the VS Code configurations do) or upload with smpmgr
The application does not start after an interrupted upload	Slot 0 holds no valid image: MCUboot stays in serial recovery	Upload again

CHAPTER 10

Porting to a new MCU family

To move Doppelstep to another MCU family, you write one `power_stage` backend and describe the board. The motor driver, the X430 driver and the application do not change. Within a family that already has a backend, only the devicetree and the board configuration change.

10.1 What a port needs

Item	Where
A backend of the `power_stage` class, with its binding	`drivers/power_stage/`, `dts/bindings/sfs/`
The board, unless Zephyr already has it	`boards/`
The nodes of the stage, the motor and the servo	`app/boards/<board>.overlay`
Zephyr options of the board: FPU or not, console, flash writes	`app/boards/<board>.conf`

Copy `drivers/power_stage/power_stage_stm32.c` as the starting point, and use the fake backend in `tests/drivers/common/` as the shortest example of the API.

10.2 Backend checklist

- ☐ **The fast callback makes no kernel calls.** It runs at the PWM rate in your interrupt. Choose the interrupt type in the backend: zero-latency where the core has it, otherwise the highest normal priority.
- ☐ **Sampling instant.** With low-side shunts, sample when the low-side switches conduct: at the center of a center-aligned period, triggered by the timer in hardware.
- ☐ **Offset calibration** runs with the outputs off and blocks only the caller.
- ☐ **Polarity.** `adc_i_span` follows the gain sign; `current-sense-invert` corrects the result of bring-up step 3.
- ☐ **`outputs_allowed`** is false until the dead time is configured, and `output_enable` then returns `-EPERM`.
- ☐ **Outputs start off.** Drive the idle levels before the pins connect to the timer.
- ☐ **`aux_trip`** covers a leg the sample does not show and stays false until calibration has succeeded: with an unknown offset a false trip would latch a fault at every start. Without such a measurement, leave `set_aux_trip` out and remove `aux-trip` from the motor node, because the motor driver refuses to start when the node asks for a check the stage cannot do.
- ☐ **`get_stats`** is optional: without a cycle counter, leave `fast_cycles_max` at 0.
- ☐ `pwm-frequency` is an integer multiple of `slow-loop-hz`.

- ☐ Add the board build to `scripts/ci.sh` and extend the allowed file list of `scripts/check-layers.sh` with the new backend.

10.3 What the code above the backend does not assume

Feature	Rule
Zero-latency interrupts	The contract is "no kernel calls in the fast callback"
FPU	Without an FPU, Kconfig selects the Stepper-C fixed-point backend. CI runs the motion tests on it
Cycle counter	`fast_cycles_max` is optional
On-chip amplifiers	They belong to the backend and the board
Simulated motor on the board	The plant model uses floating point, so it needs an FPU or a PC

10.4 Example: TI MSPM0G3519

This is the planned next family. The figures below come from the datasheet page and from Zephyr v4.5.0, not from a built port.

Feature	STM32G431	MSPM0G3519	Consequence
Core	Cortex-M4F, 170 MHz	Cortex-M0+, 80 MHz	Half the cycles per PWM period
Zero-latency interrupts	Yes	No: Zephyr needs BASEPRI	A normal high-priority interrupt, subject to the kernel's interrupt locks
FPU	Yes	No	Fixed-point backend
Cycle counter (DWT)	Yes	No	No `fast_cycles_max`, or another timer
Motor timers	TIM1	Two advanced timers with dead band and complementary outputs	New backend
ADC	ADC1, ADC2, injected	Two 12-bit ADCs, simultaneous sampling	New backend
On-chip amplifiers	OPAMP1..3	Not listed by TI for this part	External amplifiers on the board

Zephyr v4.5.0 already has `mspm0g3519.dtsi` and the board `lp_mspm0g3519`.

Time budget

The main risk is time. At 20 kHz and 80 MHz, one PWM period has 4000 cycles and one millisecond has 80000. Stepper-C (commit `3ef0f7e`, just before v1.2.0) was compiled for Cortex-M0+ with GCC 15.2, fixed-point backend, shared-leg topology, and run in a Cortex-M0 emulator with an M0+ cycle model:

Build	Image	Fast loop, max	CPU without stall detection	CPU with stall detection
`-Os`	12.6 KB	2305 cycles	64 %	102 %
`-O2`	16.5 KB	1516 cycles	44 %	68 %

CPU figures are the worst case: sustained voltage limiting and a status read every millisecond.

> **These are estimates**
>
> The model has no flash wait states, no Zephyr interrupt entry and no ADC register reads, so real load is higher. Measure on the part before you rely on these numbers.

What to do with them:

- Set `CONFIG_SSTEP_OPTIMIZE_SPEED=y`, which builds Stepper-C with `-O2`. With 512 KB of flash, 4 KB is a small price.
- With stall detection on, `sstep_get_status()` costs about 9000 cycles. Raise `CONFIG_SFS_STATUS_PERIOD_MS`: 10 ms saves about 10 percentage points.
- If the budget is still short, lower the fast-loop rate, which Stepper-C accepts as an integer fraction of the PWM rate, or move the fast path to SRAM with Zephyr's code relocation.

The fast loop itself needs no 64-bit arithmetic: the fixed-point backend uses 32-bit multiplications there. 64-bit helpers appear in the slow loop.

CHAPTER 11

Development environment

Doppelstep keeps the whole toolchain in a Docker image, so a build gives the same result on Windows, on Linux, in CI and in GitHub Codespaces. This page describes the environment: the repository, the VS Code tasks, the Dev Container and the debug probes, CI and its image, and the documentation tooling.

11.1 Requirements

Platform	Install
Windows	Docker Desktop (WSL 2 backend), Git, VS Code with the Dev Containers extension. For debugging on the host: OpenOCD 0.12+ (for example xPack) with the ST-LINK driver, and/or the SEGGER J-Link Software
Linux	Docker and Git, or a native Zephyr installation with `west`; VS Code optional
Browser only	A GitHub account: Codespaces

11.2 Repository layout

```
doppelstep/                              west manifest repository and Zephyr module
├── west.yml                             Zephyr v4.5.0, Stepper-C v1.2.0, X430 emulator
(commit), DXL2-C v1.0.0,
│                                        Stepper-Lab and sfs-stepper-power-spice (commits)
├── zephyr/module.yml                    the repository is a Zephyr module: board,
bindings, drivers
├── CMakeLists.txt, Kconfig              build modules/ and drivers/
├── app/                                 Zephyr application (west build -s app)
│   ├── Kconfig, Kconfig.app_msg         threads, queues, startup
│   ├── prj.conf                         options common to every board
│   ├── sim.overlay                      simulated motor on the board
(EXTRA_DTC_OVERLAY_FILE)
│   ├── sim-test.conf                    log period of the model for the tests
│   ├── boards/b_g431b_esc1.overlay      power stage, motor, servo: the hardware of this
board
│   ├── boards/b_g431b_esc1.conf         FPU, zero-latency IRQ, OPAMP, RTT
│   ├── boards/native_sim_native_64.*    simulated stage, servo on uart1 (PTY), 1 ms tick,
stdout logs
│   ├── sysbuild.conf, sysbuild/         MCUboot: one slot, signature, serial recovery
│   ├── VERSION                          firmware version (log, MCUboot image header)
│   └── src/
│       ├── main.c                       startup, STATUS LED
│       ├── control_task.c               slow loop, commands, publication of the status
│       ├── comm_task.c                  DYNAMIXEL bus
│       ├── joint_bridge.[ch]            X430 registers <-> motor commands
│       ├── joint.[ch]                   microsteps <-> DYNAMIXEL units, profiles
│       ├── app_msg.[ch], app_tasks.h    queues and task ids
│       └── sim_log.c                    periodic log line of the simulated motor
├── drivers/
│   ├── power_stage/                     STM32 and simulation backends, model adapter
│   ├── motor/                           Stepper-C behind the classes stepper and
stepper_ctrl
│   └── x430/                            X430 emulator on a UART, NVS
├── include/doppelstep/drivers/          public driver APIs
├── dts/bindings/                        sfs,* bindings, vendor prefix sfs
```

```
├── boards/st/b_g431b_esc1/            out-of-tree board (not in Zephyr 4.5)
│   ├── b_g431b_esc1.dts               pinout from UM2516 Table 4, 170 MHz from 8 MHz
HSE, partitions
│   ├── board.cmake                    runners: openocd (default), jlink,
stm32cubeprogrammer, pyocd
│   └── support/openocd.cfg            ST-LINK/V2-1, STM32G4
├── modules/                           build glue of the libraries
│   ├── stepper-c/                     Kconfig (CONFIG_SSTEP_*), CMakeLists.txt
│   ├── dynamixel-x430/                Kconfig (sources Kconfig.x430), CMakeLists.txt
│   └── stepper-lab/                   plant model sstep_plant.c, params/*.json ->
sstep_plant_params.h
├── tests/
│   ├── app/, drivers/                 ztest suites, run by Twister
│   └── sil/, integration/             pytest against the native_sim firmware
├── tools/
│   ├── gui/                           host GUI: test bench and firmware update
│   ├── sim/                           native_sim launcher (PTY link, TCP bridge),
Protocol 2.0 smoke test
│   └── docs/                          PDF manual builder and styles, MkDocs hooks,
diagrams
├── docs/, overrides/, mkdocs.yml      this documentation (MkDocs Material)
├── scripts/
│   ├── west-init.sh                   west init -l + west update (in the container)
│   ├── run-tests.sh / ci.sh           tests on native_sim / the whole of CI
│   ├── check-layers.sh                dependency rules between the layers
│   ├── junit_summary.py               test table for the GitHub job summary
│   ├── package_release.py             release checks and release files
│   └── host/                          GDB servers on the host (OpenOCD, J-Link; Windows
and Linux)
├── .github/workflows/                 ci.yml (image, builds, tests), docs.yml (website,
PDF),
│                                      release.yml (GitHub Release on a vX.Y.Z tag)
├── CHANGELOG.md                       changes of every release
├── .devcontainer/                     Dev Container configurations (host debug server,
USB probe, Codespaces)
├── .vscode/                           tasks for Windows and Linux, Cortex-Debug launch
configurations
├── .agents/skills/doppelstep/         engineering rules for agents
├── docker/jlink/                      optional J-Link Software archive for the image
├── Dockerfile, compose.yaml           Zephyr toolchain image
├── requirements.txt                   host Python tools: GUI, firmware update,
documentation
└── LICENSE (Apache-2.0)
```

The west workspace lives in the container under `/workdir` (the named Docker volume `doppelstep-west`): `doppelstep/` is a bind mount of this repository, next to it are `zephyr/` and `modules/` (the HAL, CMSIS, SEGGER, MCUboot and its crypto modules, and the libraries of `west.yml` under `modules/lib/`). Zephyr and the modules stay in the Linux volume (fast file access, kept across container rebuilds); only this repository and the build directories live in the host folder.

11.3 Two platforms, one set of tasks

The VS Code tasks in `.vscode/tasks.json` work on Windows and on Linux. Each task has a Linux command (Dev Container or Linux host) and a `windows` override; both call a menu script with the same action name:

Where VS Code runs	Script	Zephyr (`west`)	Debug probe	GUI and documentation
Dev Container (Linux)	`.vscode/workspace-menu.sh`	in the container	USB pass-through (mode A) or host GDB server (mode B)	`.venv` / `.venv-linux`

Where VS Code runs	Script	Zephyr (`west`)	Debug probe	GUI and documentation
Linux host	`.vscode/workspace-menu.sh`	native `west`, or the Docker image when `west` is missing	native tools, or the `zephyr-usb` compose service	`.venv`
Windows host	`.vscode/host-menu.ps1`	Docker image (`docker compose run --rm zephyr`)	OpenOCD / J-Link installed on Windows (`scripts/host/`)	`.venv`

On the Windows host, Zephyr runs in the same image and the same `doppelstep-west` volume as the Dev Container, so both share the west workspace, ccache and `build/`.

11.4 Tasks

`Ctrl+Shift+B` opens **Workspace Menu**, which lists the actions of the current platform. The other tasks run one action directly (`Terminal → Run Task…`):

Task	Linux / Dev Container	Windows host
Zephyr: West Init/Update	`scripts/west-init.sh`	same, in Docker
Zephyr: Build	`west build --sysbuild -b b_g431b_esc1 -s app -d build`: MCUboot + signed application (Firmware update)	same, in Docker
Zephyr: Pristine Build	`west build -p always --sysbuild …`	same, in Docker
Zephyr: Pristine Build (simulated motor)	`west build -p always --sysbuild … -- -DEXTRA_DTC_OVERLAY_FILE=sim.overlay`	same, in Docker
Zephyr: Build (native_sim)	`west build -b native_sim/native/64 … -d build-native`	same, in Docker
Zephyr: Run tests (native_sim)	`scripts/run-tests.sh`: SIL and integration tests → `build-tests/junit.xml`	same, in Docker
Zephyr: Run (native_sim)	`tools/sim/run_native.py run` (bus at `build-native/ttyDXL`)	same, in Docker; bus also on TCP 5555 of the host
Zephyr: Clean	removes `build/`	removes `build/`
Zephyr: Flash (OpenOCD, ST-LINK)	`west flash -r openocd`: MCUboot and the signed application	`openocd … program build/mcuboot/zephyr/zephyr.hex`, then `build/app/zephyr/zephyr.signed.hex` (OpenOCD on Windows)
Python: Create .venv	`.venv` (or `.venv-linux`) with `requirements.txt`	`.venv` with `requirements.txt`
GUI: Run	`tools/gui/doppelstep_gui.py`	same
Docs: Serve	`mkdocs serve` on `http://127.0.0.1:8000`	same

Task	Linux / Dev Container	Windows host
Docs: Build PDF	`tools/docs/build_manual.py` → `site/Doppelstep.pdf`	same (installs Pango in the container when missing)

Actions only in the **Linux** menu: Menuconfig, Flash (J-Link), Debug server (OpenOCD / J-Link), RTT log (OpenOCD / J-Link), Ccache stats/clear.

Actions only in the **Windows** menu: GDB server on the host (OpenOCD, with or without RTT; J-Link) and the J-Link Remote Server — the servers that the Cortex-Debug configurations "… on host" connect to from the Dev Container.

11.5 Dev Container

`F1 → Dev Containers: Reopen in Container` and pick one of:

Container	Debug probe
Doppelstep (host debug server) — default	stays on Windows; GDB connects to `host.docker.internal` (mode B)
Doppelstep (USB probe in the container)	attached to WSL 2 with usbipd-win (mode A)

The first start runs `scripts/west-init.sh` (`west init -l`, `west update --narrow --fetch-opt=--tags`) into the `doppelstep-west` volume.

Both build the stage `tools` of the `Dockerfile`: the tools only, the workspace stays in the volume. A J-Link archive in `docker/jlink/` goes into this local image only.

Mode A: USB probe in the container (usbipd-win)

```
winget install usbipd                  # once
usbipd list                            # find the ST-LINK / J-Link BUSID
usbipd bind --busid <BUSID>            # once per device (administrator)
usbipd attach --wsl --busid <BUSID>    # after every replug, before the container starts
```

The container runs privileged with `/dev/bus/usb` mounted; `lsusb` must show the probe. For the J-Link runner put `JLink_Linux_V*_x86_64.tgz` into `docker/jlink/` before the image is built (`docker/jlink/README.md`).

Mode B: GDB server on the host

```
.\scripts\host\openocd-server.ps1          # ST-LINK, original firmware   -> :3333
.\scripts\host\jlink-gdb-server.ps1        # J-Link OB / J-Link           -> :2331
```

On a Linux or macOS host the same servers start with `bash scripts/host/openocd-server.sh` and `bash scripts/host/jlink-gdb-server.sh` (OpenOCD 0.12+ or the J-Link Software installed on the host; the port is set by `GDB_PORT` for OpenOCD and by `PORT` for J-Link).

Both listen on all interfaces: the container reaches the host as `host.docker.internal` (Windows Firewall may ask once).

Without VS Code

```
docker compose build
docker compose run --rm zephyr bash scripts/west-init.sh
docker compose run --rm zephyr                       # west build for the board
docker compose --profile usb run --rm zephyr-usb     # shell with the USB probe
```

11.6 Flashing and debugging

`west` (mode A, or the J-Link Remote Server)

```
west flash        -d build -r openocd     # ST-LINK/V2-1 (default runner)
west debug        -d build -r openocd
west debugserver  -d build -r openocd     # GDB server on :3333
west flash        -d build -r jlink       # J-Link OB / J-Link
west debug        -d build -r jlink
west debugserver  -d build -r jlink       # GDB server on :2331
west attach       -d build -r openocd|jlink
```

The openocd runner uses `boards/st/b_g431b_esc1/support/openocd.cfg` and the OpenOCD of the Zephyr SDK (Zephyr thread awareness). In mode B the J-Link runner in the container can still reach the probe through `scripts/host/jlink-remote-server.ps1` (`--dev-id <host IP> --dev-id-type ip`); this path is not yet verified with the J-Link OB firmware.

VS Code (Cortex-Debug, `.vscode/launch.json`)

Configuration	Server
Debug: OpenOCD in container (ST-LINK, USB pass-through)	OpenOCD of the SDK in the container, `rtos Zephyr`
Debug: J-Link in container (J-Link OB, USB pass-through)	`JLinkGDBServerCLExe` in the container
Debug: OpenOCD on host (host.docker.internal:3333)	`scripts/host/openocd-server.ps1`
Debug: J-Link GDB Server on host (host.docker.internal:2331)	`scripts/host/jlink-gdb-server.ps1`
Attach: ... on host (no download)	Attach to the running firmware

Every *Debug* configuration builds first, loads MCUboot and the signed application (`zephyr.elf` alone would not boot: MCUboot checks the signature), resets and stops at `main` of the application.

Logs (RTT)

USART2 carries the DYNAMIXEL bus, so the console and the log use SEGGER RTT:

```
west rtt -d build -r openocd      # or -r jlink
```

With a host GDB server: `openocd-server.ps1 -Rtt` (TCP 9090), or the RTT telnet port 19021 of the J-Link GDB Server.

11.7 CI and the environment image

The stage `ci` of the `Dockerfile` adds the west workspace of `west.yml` (Zephyr, the modules, MCUboot) in `/workdir`. GitHub Actions publishes it to the GitHub Container Registry as `ghcr.io/systemfromscratch/sfs-doppelstep-env:<hash>`, where `<hash>` is taken from the files the image is made from: `Dockerfile`, `.dockerignore`, `west.yml`, `scripts/west-init.sh`, `tests/requirements.txt`.

Event	Image with that hash in ghcr.io	Missing
Push to `main`	pulled; `:main` points at it	built, published as `:<hash>` and `:main`
Pull request, other branch	pulled	built in the job for that run, not published

The firmware job then runs `scripts/ci.sh` in the image, with the repository mounted at `/workdir/doppelstep` and `WEST_UPDATE=0`: the workspace is the one the image was built with. Every project in `west.yml` is pinned to a tag or a commit, so the workspace changes only with `west.yml`, and with it the image tag. The workflow writes the package with `GITHUB_TOKEN`. The package must be **public** (package settings on GitHub, **Change visibility**), so that anyone's codespace, and `docker pull` without a login, can use it. The image never contains the J-Link Software (not redistributable): `docker/jlink/` holds no archive in a checkout.

The workflow `.github/workflows/docs.yml` builds the website and the PDF manual on every pull request and publishes both on GitHub Pages from `main` (systemfromscratch.github.io/sfs-doppelstep, PDF at `Doppelstep.pdf` there). It needs one repository setting: **Settings → Pages → Source: GitHub Actions**.

11.8 Releasing

A release is a `vX.Y.Z` tag on `main`. The version follows Semantic Versioning (semver.org/) and lives in two places that must agree: the tag and `app/VERSION`, which Zephyr turns into `APP_VERSION_STRING` (logged at boot) and into the version of the signed image that MCUboot checks. The release workflow refuses a tag that does not match.

1. Set `app/VERSION` to the new version and move the entries of `## [Unreleased]` in `CHANGELOG.md` to `## [X.Y.Z] - YYYY-MM-DD`; update the compare links at its end.
2. Pin every project of `west.yml` to a tag or a full commit hash: a release must build the same code every time.
3. Re-measure the figures in the README and the documentation that changed (flash, RAM, test counts: the build log of CI) and update the project status.
4. Check locally, then in the container run what CI runs:

```
python3 scripts/package_release.py --version X.Y.Z --check-only
docker compose run --rm zephyr bash scripts/ci.sh
docker compose run --rm zephyr python3 scripts/package_release.py --version X.Y.Z
```

The last command writes the release files to `dist/`. **Actions → Release → Run workflow** does the same in GitHub Actions without publishing anything (artifact `release`). 5. Merge to `main`, then tag the merge commit and push the tag:

```
git tag -a vX.Y.Z -m "Doppelstep X.Y.Z"
git push origin vX.Y.Z
```

`.github/workflows/release.yml` then checks the tag, runs the whole CI workflow on the tagged commit, builds the PDF manual with the version on its cover and publishes the GitHub Release. Its notes are the CHANGELOG section of the version, and its files are the board images under release names, MCUboot and the application merged into one HEX for the first flashing, the manual and `SHA256SUMS`. A `0.x` version is published as a pre-release. The release images are signed with the public development key of MCUboot (Firmware update (p. 40)).

11.9 Codespaces

Code → Codespaces → … → New with options, configuration **Doppelstep (Codespaces: build and native_sim tests)** (`.devcontainer/codespaces/`). It starts from `ghcr.io/systemfromscratch/sfs-doppelstep-env:main` on 4 cores; nothing is downloaded or built at the start. `setup.sh` makes `/workspaces` the west topdir: links to the workspace of the image and a west configuration that points at the clone. When `west.yml` of the branch differs from the one of the image it says so; run `west update --narrow --fetch-opt=--tags` then.

In a codespace:

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build   # board images
bash scripts/run-tests.sh                                          # SIL and integration
tests
bash scripts/ci.sh                                                 # everything, as CI
```

Flashing, debugging and the GUI need the board and a display: they stay local. If a codespace cannot pull the image, check that the package is public (see above).

11.10 Python virtual environment

The GUI and the documentation use the packages in `requirements.txt`. The tasks create the environment on first use; **Python: Create .venv** creates or updates it explicitly.

- Windows: `.venv\Scripts\python.exe` (Python from python.org, with Tkinter).
- Linux: `.venv/bin/python`. In a folder bind-mounted from Windows where `.venv` already is a Windows environment, the Linux scripts use `.venv-linux` instead.

11.11 Documentation and PDF manual

The documentation is written in Markdown in `docs/` and built with MkDocs Material (`mkdocs.yml`):

```
mkdocs serve                       # live preview
mkdocs build                       # HTML site in site/
python tools/docs/build_manual.py  # PDF manual: site/Doppelstep.pdf
```

`tools/docs/build_manual.py` converts the pages of `nav` to one HTML document and renders it to A4 with WeasyPrint, in the layout of the SystemFromScratch® manuals (Stepper-C): a cover with the hero illustration, an imprint, the contents with page numbers, numbered chapters with numbered

sections and figures, the About pages unnumbered, and the license with its full text as Appendix A. Running header and footer, PDF bookmarks and metadata are added. Its settings are in `extra.manual` of `mkdocs.yml`, the print styles in `tools/docs/manual.css`. The version is the latest `vX.Y.Z` tag, otherwise `git describe`. WeasyPrint needs Pango: on Linux `apt install libpango-1.0-0 libpangoft2-1.0-0` (the task does it in the container), on Windows MSYS2 with `mingw-w64-x86_64-pango`, which the script finds in `C:\msys64\mingw64\bin`.

The website uses the SystemFromScratch® style: `overrides/home.html` renders the landing page from `extra.landing` in `mkdocs.yml` (hero, problem, what you get, measured figures, portfolio banner), with the quick start of `docs/index.md` below it; `extra.css` holds the company palette. The landing page (`docs/index.md` and `extra.landing`) is web only; its quick start is covered by Getting started (p. 17) in the PDF. The hero illustration and the diagrams in `docs/assets/images/` (`hero-3d.svg`, `architecture.svg`, `command-path.svg`, `test-setup.svg`, `motor-wiring.svg`) are generated by `python tools/docs/diagrams.py` from the structure of the code; change the script, never the SVG, and check the labels against the code after a change. The GUI screenshots in `docs/assets/images/gui/` show the GUI connected to the `native_sim` firmware (`socket://localhost:5555`); retake them when the GUI changes.

Every number on the landing page and in the Introduction (p. 8) is measured: the test counts from `build-ci/twister/twister.json` and the `junit.xml` files of `scripts/ci.sh`, the sizes from the memory report of the board build. Update them when they change.

New pages go into `docs/` and into `nav` of `mkdocs.yml`; the build is strict, so broken links fail it.

CHAPTER 12

Tests

Three levels run on the PC: ztest suites for the parts, and SIL and integration tests for the whole firmware. Checks on the board (p. 58) cover what a PC cannot see.

12.1 ztest suites

Suite	Checks
`tests/app/joint`	Unit conversions and motion profiles
`tests/app/msg`	Queues: timeout, discarded late answer, full queue, latest status
`tests/drivers/power_stage`	API contract of the class on the simulation backend, `-ENOSYS` of optional functions
`tests/drivers/sstep_motor`	Motor driver through the standard classes, on a fake power stage

```
west twister -T tests/app -T tests/drivers -p native_sim/native/64 --integration
```

`tests/drivers/common/` holds the fake power stage: the test fires the fast loop itself, so every step is deterministic.

12.2 SIL and integration tests

The whole firmware runs on the PC as a `native_sim` program, with the motor model of the simulated motor (p. 35) behind the power stage. Pytest drives it over DYNAMIXEL Protocol 2.0, the way a host program does. The tests run in the Docker image, both on a developer PC and in GitHub Actions, with the same command.

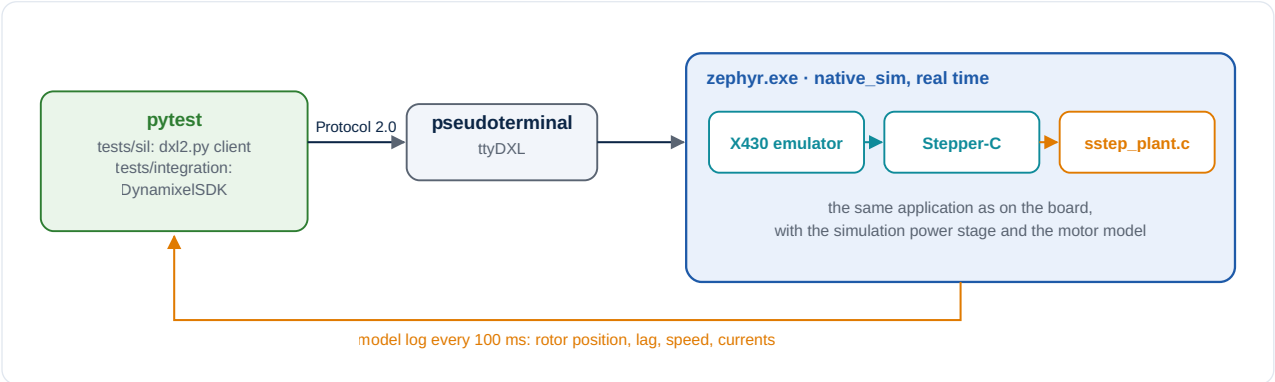


Figure 7: SIL and integration tests: pytest drives the native_sim firmware with the motor model

12.3 Running them

Where	Command
Windows host	VS Code task **Zephyr: Run tests (native_sim)**, or `docker compose run --rm zephyr bash scripts/run-tests.sh`
Dev Container / Linux	`bash scripts/run-tests.sh` (task of the same name)
Like CI (builds + tests)	`docker compose run --rm zephyr bash scripts/ci.sh`
GitHub	`.github/workflows/ci.yml` on every push to `main` and every pull request

`scripts/run-tests.sh` builds `build-tests/` (`native_sim/native/64` with `app/sim-test.conf`, which logs the model every 100 ms) and runs pytest. Arguments go to pytest:

```
bash scripts/run-tests.sh tests/sil                 # only the SIL tests
bash scripts/run-tests.sh -k protection -x          # a subset, stop at the first failure
TESTS_PRISTINE=1 bash scripts/run-tests.sh          # pristine build first
STEPPER_LAB_SOURCE_DIR=/sim bash scripts/run-tests.sh # unpublished Stepper-Lab checkout
TESTS_BUILD_DIR=build-tests-fixed TESTS_EXTRA_CMAKE_ARGS=-DCONFIG_SSTEP_MATH_FIXED=y \
    bash scripts/run-tests.sh tests/sil/test_motion.py # fixed-point backend
```

The results are written to the build directory:

- `build-tests/junit.xml`: the report.
- `build-tests/pytest/<test>/fw.log`: the complete firmware output of every test.

The full suite takes about 30 s.

Levels

Level	Directory	What runs	Checks
SIL	`tests/sil/`	the firmware with the plant model; client `tools/sim/dxl2.py` (raw Protocol 2.0, error byte, mal-formed packets)	Control Table behaviour, and the effect on the rotor of the model
Integration	`tests/integration/`	the official **DynamixelSDK** (`dynamixel-sdk`), the library under the ROS 2 `dynamixel_hardware_interface`	the ros2_control loop: Sync Read of current, velocity and position, Sync Write of the goal at 100 Hz
Hardware (manual)	Bring-up checklist	the board with a motor	dead time, polarity, current loop

The firmware and model checks of other repositories are separate. Stepper-Lab runs the Xcos SIL against the Scilab prototype (S01–S05), and sfs-stepper-power-spice runs the ngspice benches.

What the tests cover

File	Tests
`sil/test_boot.py`	clean startup, ping (model 1020), live registers at rest (24.0 V, position 2048), factory EEPROM
`sil/test_motion.py`	Goal Position moves the rotor of the model to the commanded position; run and hold current; Profile Velocity; time-based profile (Drive Mode 4, OpenMANIPULATOR-X); velocity mode; torque off releases the motor; refused Operating Mode; EEPROM locked while the torque is on
`sil/test_protection.py`	load angle against `T = Kt·I·sin(δ)` at the run and the hold current; overload (0.4 N·m), which trips with Hardware Error Status bit 4, the Alert bit and torque off, all cleared by Reboot; undervoltage (9 V), which sets bit 0; Torque Enable in a fault is refused; **lost synchronism, which is not reported** (expected failure, see below)
`sil/test_bus.py`	bad CRC and other IDs get no answer; Bus Watchdog stops the motor and refuses goals until cleared; Reboot; Factory Reset; Sync Read / Write; rapid goals during a move; Clear Multi-Turn during a move is refused; garbage on the bus
`sil/test_eeprom.py`	EEPROM values survive a power cycle (same `flash.bin`), RAM values do not
`integration/test_dynamixel_sdk.py`	SDK ping, read and write; 3 s of the ros2_control loop on a sine trajectory: no failed transaction, more than 50 Hz, tracking error within a few cycles, rotor synchronous

Expected failure: lost synchronism

`test_lost_synchronism_is_reported` is marked `xfail(strict=True)`. A load of 0.12 N·m is held at the run current (`Kt·1.5 A = 0.31 N·m`) but not at the hold current (`Kt·0.5 A = 0.10 N·m`), so the joint falls. The firmware still reports Torque Enable 1, no hardware error and the commanded position. A stepper motor without an encoder or stall detection cannot see this. The test documents the gap. When stall detection is added, the test starts to pass, and pytest reports the unexpected pass until the marker is removed.

Scenario of a test

Each test starts its own `zephyr.exe` in a fresh directory, so it gets a new emulated EEPROM (`flash.bin`). Options of the executable set the scenario without a rebuild:

Option	Effect
`-sim_load_unm=<µN·m>`	load torque (gravity-like), acting from the first Torque Enable
`-sim_vbus_mv=<mV>`	bus voltage of the model

Example:

```
def test_something(firmware_factory):
    fw = firmware_factory(["-sim_load_unm=80000"])
    bus = dxl2.Bus(fw.pty)
    bus.write(DXL_ID, dxl2.TORQUE_ENABLE, 1)
    s = fw.next_sample()           # rotor, lag (electrical degrees), rpm, currents
```

Fixtures (`tests/conftest.py`):

- `fw`: the firmware with the default scenario.
- `bus`: a `dxl2.Bus` on it.
- `firmware_factory(args)`: firmware with a scenario.

`fw.restart()` keeps the working directory, so the EEPROM survives. `support.wait_until()` polls with a timeout. `support.assert_stopped()` checks a stopped motor: the commanded position stands still and the rotor stays under 5 rpm, because the rotor of the model vibrates at rest (why (p. 39)). Times that matter come from the Realtime Tick register, because the simulated time of `native_sim` can run ahead of the host clock.

The firmware runs in real time, so the bus timing is the one a host program sees. The pseudoterminal driver polls for the first byte of a packet every 10 ms.

12.4 CI

`scripts/ci.sh` runs the same steps on a PC and in GitHub Actions:

1. `scripts/check-layers.sh`: the dependency rules of the architecture (p. 31).
2. `west update` (skipped in GitHub Actions, `WEST_UPDATE=0`: the image holds the workspace).
3. Pristine builds without warnings of `b_g431b_esc1`, `b_g431b_esc1` with `sim.overlay`, and the `native_sim` test firmware. The only known warning is the X430 emulator's `-Wmaybe-uninitialized` under host gcc, which is reported upstream.
4. The ztest suites, with Twister.
5. The SIL and integration tests.
6. The SIL motion tests on the Stepper-C fixed-point backend.

In GitHub Actions `scripts/ci.sh` runs in the environment image (tools and the west workspace, development environment (p. 51)); ccache is kept in the GitHub Actions cache.

It uploads the test reports, the firmware logs and the ESC1 images, and writes a test table to the job summary.

The pinned test packages are in `tests/requirements.txt` and in the `Dockerfile` (`ARG PYTEST_VERSION ...`). Keep both in step. For the local image, raise the revision in `compose.yaml` (`doppelstep:1.0.1-<n>`) when the Dockerfile changes; the CI image gets a new tag by itself.

CHAPTER 13

Hardware checklist after the driver refactor

CI runs on `native_sim`, so it cannot see the real interrupt, timer, ADC or UART. Run these checks on a B-G431B-ESC1 after a change to the drivers or the tasks, and write down the results. They need no motor unless stated, and the gate drivers stay off.

To compare timing, first measure the fast loop on the firmware before the change.

Value	Before	After
`fast loop max` [cycles], simulated motor		

13.1 A: firmware without a motor

Build and flash:

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build
west flash -d build -r openocd
```

Then read the log with `west rtt -d build -r openocd`.

#	Check	Expected
1	Startup log	No `<err>`. Offsets near 2048 counts, a warning that `dead-time-ns` is 0, the line `x430_servo: DYNAMIXEL`
2	STATUS LED	Slow blink: outputs off
2a	Hardware Error Status over 10 power cycles, the first with erased NVS	0 every time, LED slow blink. No false fault from the shared-leg check at startup
3	`tools/sim/dxl_smoke_test.py` on the ST-LINK COM port, 1 Mbps	Ping answers model 1020
4	DYNAMIXEL Wizard scan	Finds one XM430-W350 at ID 1, 1 Mbps
5	Torque Enable	Result Fail: outputs are refused while `dead-time-ns` is 0. No fault, LED unchanged
6	Present Input Voltage with a supply connected	Agrees with a multimeter
7	Write Baud Rate 2 (115200 bps), then reconnect at that speed	Answers at the new speed. `native_sim` cannot test this
8	Power cycle after step 7	Keeps the baud rate, so NVS works. Set it back to 3 afterwards

These checks do not need the outputs. A real trip is checked in part C.

13.2B: simulated motor on the board

```
west build -p always --sysbuild -b b_g431b_esc1 -s app -d build -- -
DEXTRA_DTC_OVERLAY_FILE=sim.overlay
west flash -d build -r openocd
```

#	Check	Expected
9	Startup log	`SIMULATED MOTOR`, no `<err>`
10	Torque Enable, Goal Position 3072 in the Wizard	Rotor of the model follows, lag well under 90 electrical degrees
11	STATUS LED during step 10	On
12	PWM pins with an oscilloscope, or the MOE bit of `TIM1_BDTR` in the debugger	Gate outputs stay at their idle level, MOE = 0
13	`fast loop max` in the log	Enter it in the table above. It must stay well under one PWM period, 50 µs. A rise over the old value should be small: a few tens of cycles for the extra API call
14	Spam Goal Position from the Wizard or a script for a minute	Every packet answered, no `<err>`, `fast loop max` stable

13.3C: with a motor, after the dead time is set

Follow the bring-up checklist (p. 22) first; its step 6 forces the trips, which confirms the fault path through the interrupt. Checks 15 and 16 concern a finding from the simulation.

#	Check	Expected
15	Phase current at rest with a current probe, at the 0.5 A hold current and at several rotor positions	Records the current ripple. In the model the current loop toggles by one ADC step (29.4 mA, about 570 mV of phase voltage)
16	Vibration and noise of the motor at rest, at the same positions	Records whether the motor audibly or measurably vibrates around 125 Hz at 0.5 A

If checks 15 and 16 show a real problem, consider oversampling in the ADC, a lower current loop bandwidth (`current-bandwidth-hz`) or a higher measurement gain. The model predicts the effect, but real friction and ADC noise may reduce it or remove it, so measure before changing the control.

13.4D: firmware update over USART2

MCUboot serial recovery (Firmware update (p. 40)). Flash with the debug probe first (task **Zephyr: Flash**), at 1 Mbps unless a check says otherwise.

#	Check	Expected
17	Startup log after the probe flash	The application starts, so MCUboot accepted the signature
18	VS Code configuration **Debug: OpenOCD on host**	Stops at `main` of the application

#	Check	Expected
19	Write a test value to Homing Offset (20), read it back	The value
20	Reboot, then an SMP request repeated until answered, then `image upload` and `os reset` of an image with another boot banner	Upload completes, the new banner in the RTT log
21	Baud Rate 1 (57600), Reboot	Complete status packet without error; the application answers again at 57600 after about 0.5 s plus startup
22	Upload interrupted halfway (unplug USB), power on	The application does not start; a second upload succeeds and the application starts
23	Read Homing Offset (20)	The value of check 19: the storage partition survived

CHAPTER 14

GUI

The Doppelstep GUI is a desktop program on the host that talks to the B-G431B-ESC1 over the DYNAMIXEL Protocol 2.0 bus (ST-LINK Virtual COM Port). It is the bench tool for judging the whole project on the board:

- **Motion and live plots**: torque, Goal Position, profiles, Goal Velocity, live plots of position, velocity, current and input voltage, CSV recording.
- **Control Table**: every register of the emulated XM430-W350, read in one block, written by double-click.
- **Test scenarios**: the register checks of the SIL tests (`tests/sil`), run on the board, with PASS / FAIL per check and an HTML report.
- **Firmware update**: MCUboot serial recovery over the same port, verified by the image hash (Firmware update (p. 40)).

The screenshots on this page show the GUI connected to the `native_sim` firmware with the simulated motor (port `socket://localhost:5555`, see Simulated motor (p. 38)); every value in them comes from the firmware. On the board, choose the ST-LINK Virtual COM Port instead.

14.1 Starting the GUI

Platform	VS Code task	Script	Virtual environment
Windows host	**GUI: Run**	`.vscode/host-menu.ps1`	`.venv`
Linux desktop	**GUI: Run**	`.vscode/workspace-menu.sh`	`.venv` (`.venv-linux` next to a Windows `.venv`)

The task creates the virtual environment with `requirements.txt` on first use. It needs Python 3.11 or newer with Tkinter: on Windows the python.org installer ("tcl/tk and IDLE"), on Debian/Ubuntu the packages `python3-tk` and `python3-venv`. The GUI needs a display, so it runs on the host and not inside the Dev Container.

By hand:

```
.\.venv\Scripts\python.exe tools\gui\doppelstep_gui.py --port COM5
```

Options: `--port` (preselected port), `--baud` (bps, default 1000000), `--id` (default 1).

14.2 Connecting

Choose the port (the ST-LINK port is preselected when found), the baud rate and the ID, then **Connect**. The GUI pings the ID and shows the model and firmware version; model 1020 is the XM430-W350 that Doppelstep emulates. While connected, the bar under the connection shows the live registers ten times per second: Torque, Hardware error (decoded), Moving, Position, Velocity, Current and Input voltage.

A changed ID or Baud Rate in the Control Table disconnects; connect again with the new value.

14.3 Motion and live plots

Control	Register
Torque ON / **STOP (torque off)**	Torque Enable (64)
Operating Mode: 3 position, 1 velocity (EEPROM: torque off first)	Operating Mode (11)
Goal Position: slider, value, −90° / 0° / +90° (2048 = 0°)	Goal Position (116)
Profile Velocity, Profile Acceleration	112, 108
Goal Velocity, ×0.229 rev/min	Goal Velocity (104)

The plots show Goal and Present Position [°], Present Velocity [rpm], Present Current [mA] and Present Input Voltage [V] over the last 10, 30 or 60 s, redrawn twice per second. The toolbar under them zooms and saves a PNG. **Record CSV** writes every sample (10 per second) with the columns `time_s, goal_position_deg, present_position_deg, present_velocity_rpm, present_current_ma, input_voltage_v, torque_enable, moving, hardware_error_status, realtime_tick_ms`.

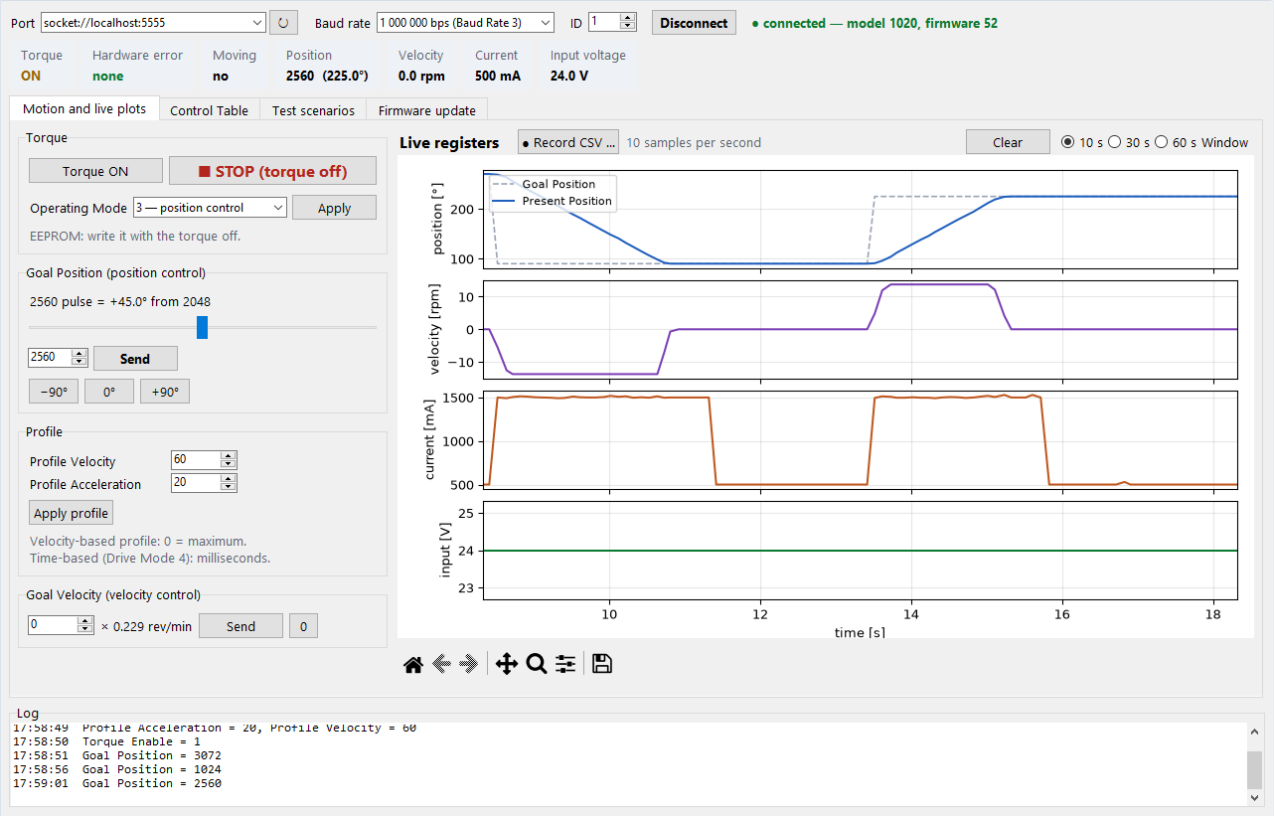


Figure 8: The Motion tab: Goal Position 1024, then 2560, with Profile Velocity 60 and Profile Acceleration 20; 1500 mA run current while moving, 500 mA hold current at rest

Torque Enable refused

While `dead-time-ns` is 0 in `app/boards/b_g431b_esc1.overlay` the firmware answers Torque Enable with Result Fail and never enables the gate drivers (hardware checklist (p. 58), check 5). The GUI says so when it happens.

No encoder

Present Position is the commanded open-loop position. The GUI cannot see lost steps: watch the shaft (a mark on it helps).

14.4 Control Table

Read all reads every register of the emulated XM430-W350 and shows its address, size, area, access, raw value and, where the register has a unit, the converted value. **Refresh every second** repeats the read. Double-click a writable register, or select it and press **Write selected …**, to write a new value. EEPROM registers (addresses below 64) are locked while the torque is on, as on the real servo.

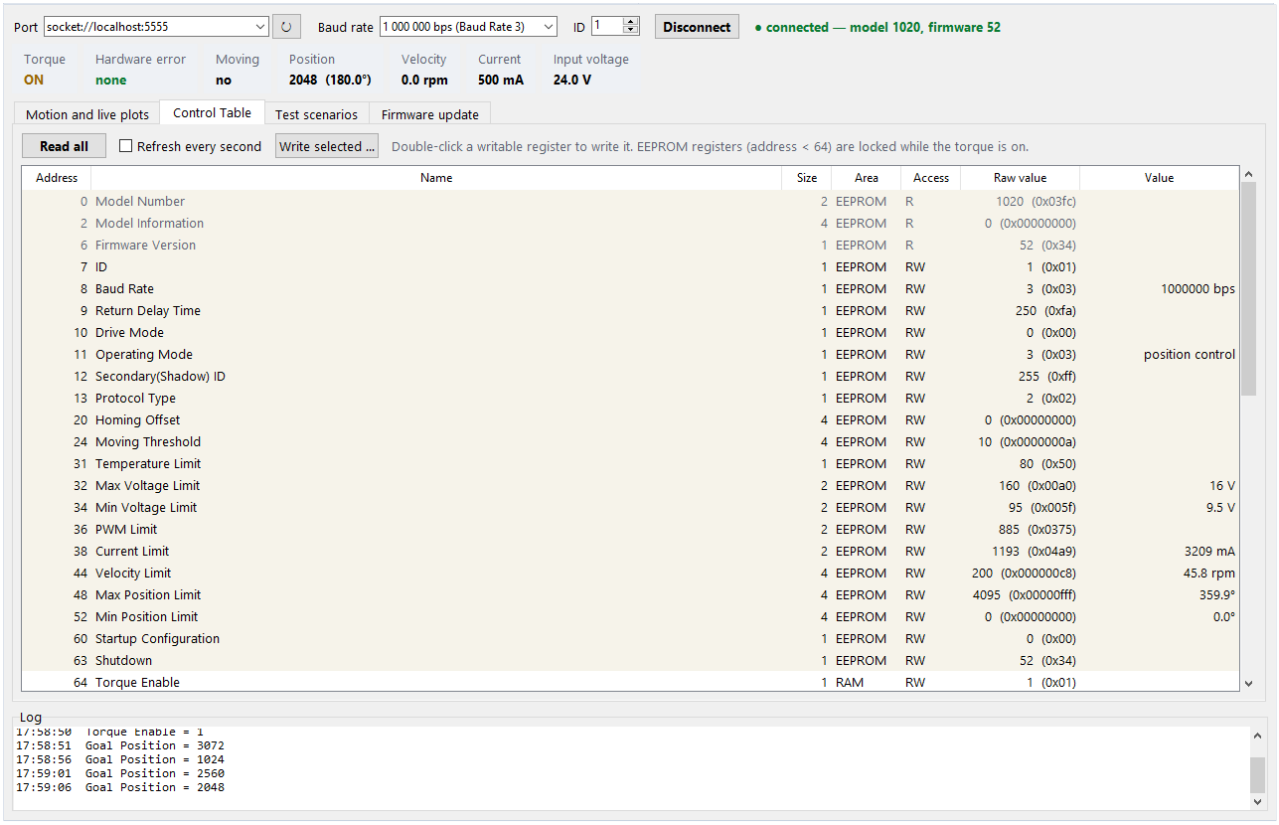


Figure 9: The Control Table tab after Read all: the EEPROM area with the factory values (ID 1, Baud Rate 3 = 1 Mbps, Operating Mode 3)

14.5 Test scenarios

Each scenario repeats one SIL test with the same writes, waits and limits, on the board instead of native_sim. A check that needs the log of the motor model (rotor position, lag, model currents) cannot run on the board and is left out; the scenario lists it under "Not checked on the board".

The suite expects the factory EEPROM: connect to **ID 1 at 1 000 000 bps**. Every scenario starts with a DYNAMIXEL Reboot (the MCU resets, like a fresh native_sim process for every SIL test) and ends with the torque off. Scenarios that change the EEPROM put the factory values back. Before the run the GUI asks for confirmation when the motor turns or the EEPROM changes.

Scenario	SIL test	Board
Ping answers as XM430-W350	`test_boot.py::test_ping_answers_as_xm430_w350`	
Live registers at rest	`test_boot.py::test_live_registers_at_rest`	Input voltage recorded, not checked (24.0 V is the model supply)
Factory EEPROM	`test_boot.py::test_factory_eeprom`	

Scenario	SIL test	Board
Bad CRC and other IDs get no answer	`test_bus.py::test_bad_crc_and_other_ids_get_no_answer`	
Bus Watchdog stops the motor	`test_bus.py::test_bus_watchdog_stops_the_motor`	model speed left out
Reboot disables the torque	`test_bus.py::test_reboot_disables_the_torque_and_keeps_the_position`	Present Position restarts at 2048: Reboot resets the MCU
Factory Reset restores the ID	`test_bus.py::test_factory_reset_restores_the_id`	resets **all** EEPROM registers
Sync Write and Sync Read	`test_bus.py::test_sync_write_and_sync_read`	
200 Goal Position writes during a move	`test_bus.py::test_rapid_goals_during_a_move_all_get_an_answer`	firmware log left out
Clear multi-turn during a move is rejected	`test_bus.py::test_clear_multi_turn_during_a_move_is_rejected`	
Garbage on the bus does not stop it	`test_bus.py::test_garbage_on_the_bus_does_not_stop_it`	firmware log left out
EEPROM survives a restart, RAM does not	`test_eeprom.py::test_eeprom_survives_a_restart_ram_does_not`	restart = Reboot (MCU reset)
Goal Position moves the joint	`test_motion.py::test_goal_position_moves_the_rotor`	model rotor left out
Run current, then hold current	`test_motion.py::test_run_then_hold_current`	1500 / 500 mA ± 40 from Present Current
Profile Velocity sets the speed	`test_motion.py::test_profile_velocity_sets_the_speed`	model speed left out
Time-based profile keeps the time	`test_motion.py::test_time_based_profile_keeps_the_time`	1.0 ± 0.15 s from Realtime Tick
Velocity control mode	`test_motion.py::test_velocity_mode`	model speed and lag left out

Scenario	SIL test	Board
Torque off releases the motor	`test_motion.py::test_torque_off_releases_the_motor`	model currents left out
Unsupported Operating Mode is refused	`test_motion.py::test_unsupported_operating_mode_is_refused`	
EEPROM is locked while the torque is on	`test_motion.py::test_eeprom_is_locked_while_torque_is_on`	

Not repeated: `test_boot.py::test_startup_log_has_no_error` (reads the firmware log, RTT on the board) and `test_protection.py` (the faults must be made on the bench).

Results: **PASS** every check met; **FAIL** a check not met or a status error (the first failed check ends the scenario, as an assert in pytest); **ERROR** the device stopped answering; **STOPPED** the operator pressed Stop. Select a scenario to see its checks, measured and expected values and notes. **Save HTML report** writes one self-contained file with the connection, every scenario and check, and the last firmware update of the session.

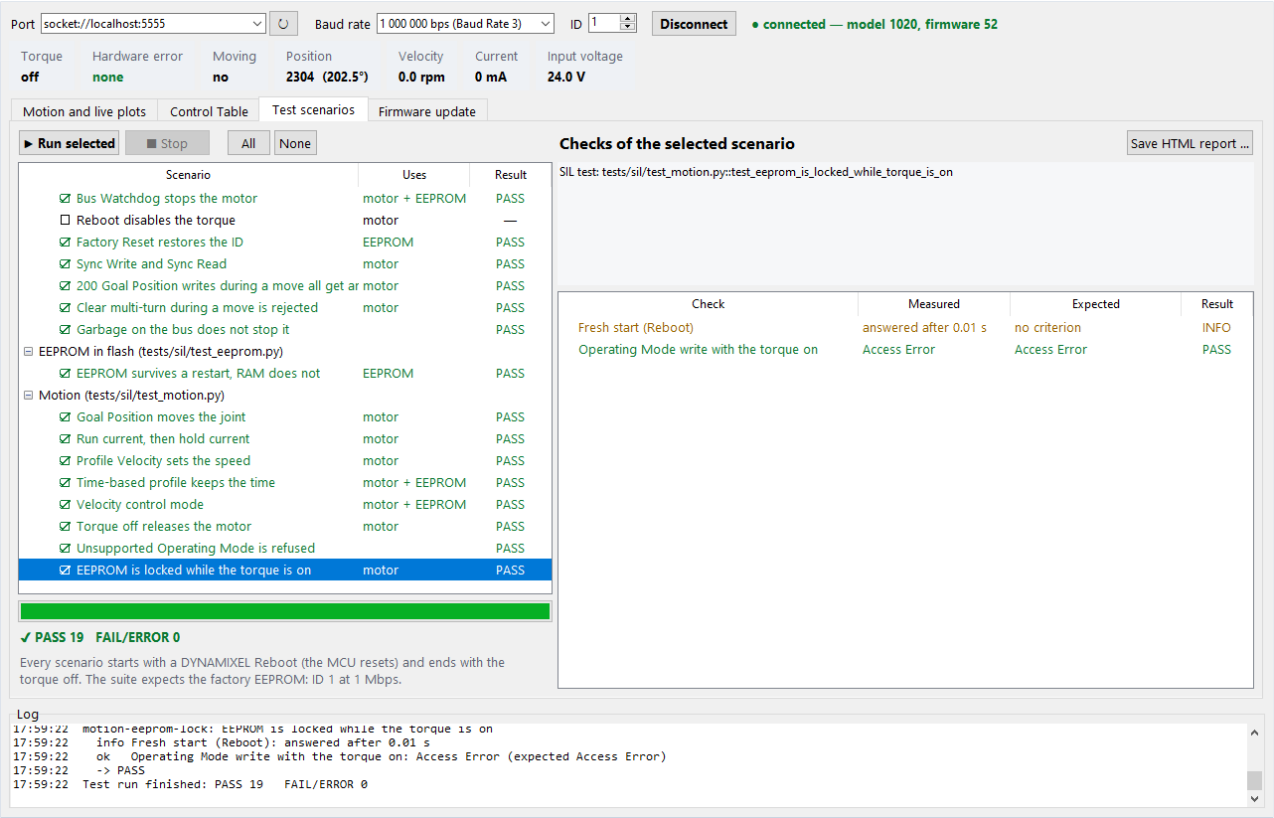


Figure 10: The Test scenarios tab after a run against the native_sim firmware: 19 scenarios PASS. "Reboot disables the torque" is deselected because it checks the MCU reset of the board, which native_sim emulates differently

14.6 Firmware update

1. Build the board (`west build --sysbuild …`). The tab preselects `build/app/zephyr/zephyr.signed.bin` and shows its version, size (slot 0 holds 86 KiB) and the SHA-256 TLV. Unsigned files (`zephyr.bin`, `zephyr.elf`) are rejected.
2. Connect as usual, or leave the GUI disconnected when MCUboot already waits in serial recovery (no valid image in slot 0, for example after an interrupted upload).
3. **Update firmware** and confirm.

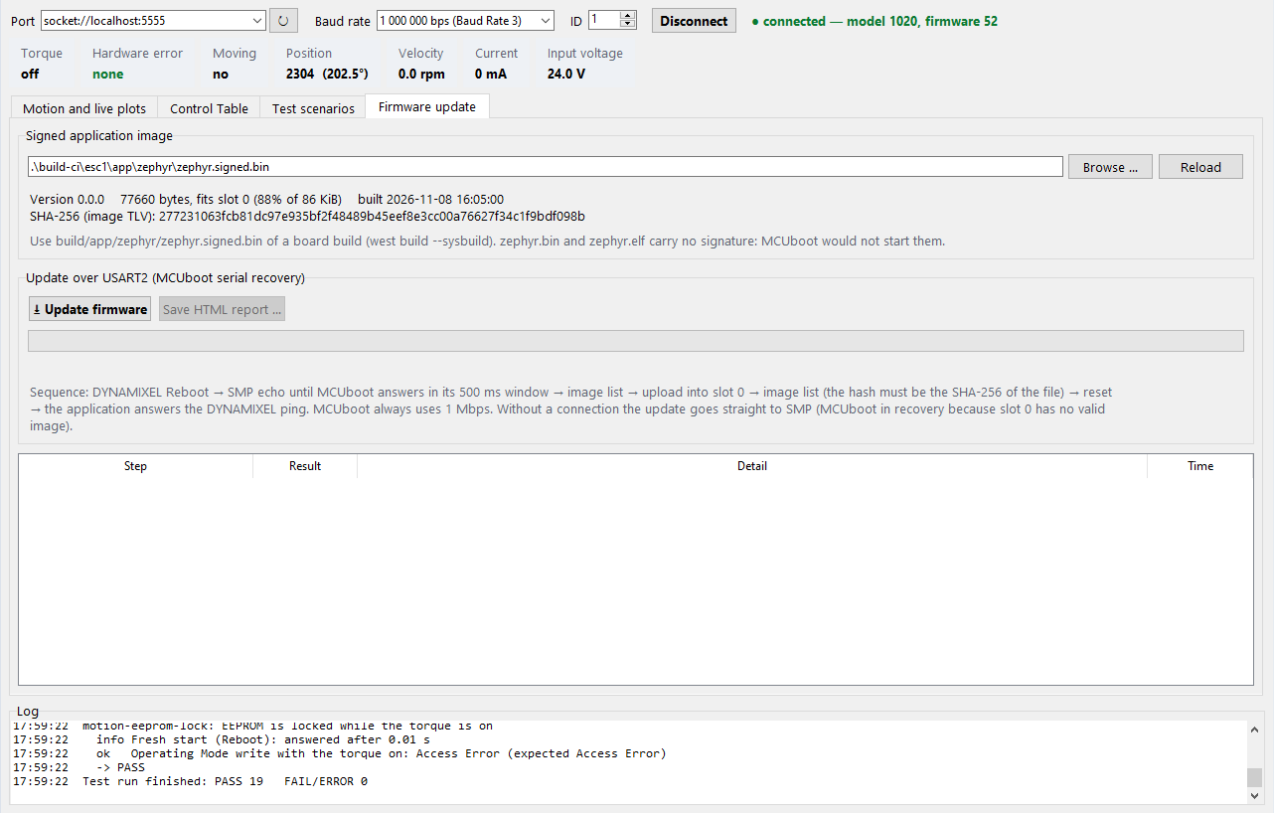


Figure 11: The Firmware update tab with a signed image loaded: version, size against slot 0, build time and the SHA-256 of the image TLV

Step	Passes when
Image file	signed MCUboot image with a SHA-256 TLV that fits slot 0
DYNAMIXEL Reboot	sent (skipped when not connected)
MCUboot serial recovery	an SMP echo is answered within 3 s, at 1 Mbps
Image list before	MCUboot answers (what slot 0 held)
Upload	the whole image is written to slot 0
Image check	slot 0 lists the version and the hash of the file
Reset	`os reset` sent
Application	the application answers the ping within 5 s as model 1020

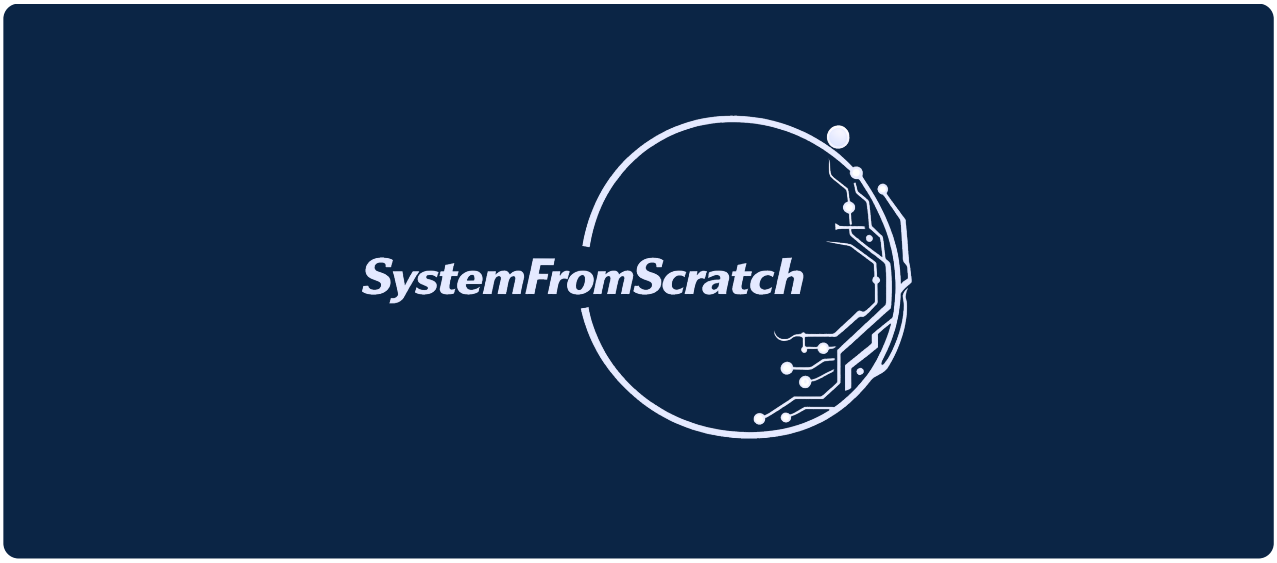
After a passed update the GUI is connected to the new application. **Save HTML report** writes the steps. If MCUboot does not answer: check that the port is the ST-LINK port, then retry; MCUboot always uses 1 Mbps, whatever the Baud Rate register says (troubleshooting (p. 42)).

14.7 Structure and tests

Module (`tools/gui/doppelstep_gui/`)	Role
`protocol.py`	Protocol 2.0 packets, CRC, stuffing, status parsing; also used by `tools/sim/dxl2.py`
`registers.py`	Control Table of the XM430-W350 (emulator model), units
`link.py`	serial port (pyserial); the only module that opens it
`device.py`	Protocol 2.0 master with a lock shared by the poller, the panels and the scenarios
`poller.py`	live registers, one Read of addresses 64–145 per sample, CSV recording
`scenarios.py`	the scenarios and their runner
`firmware.py`, `image.py`	update sequence (smpclient), MCUboot image and its hash
`report.py`	HTML report
`ui/`	Tkinter views, the only modules that import Tkinter and matplotlib

`tools/gui/tests/` (pytest, host-only, part of `scripts/ci.sh`) checks the protocol, the image parser, the scenarios against a small fake servo, the update sequence against a fake SMP session, and that every scenario names an existing SIL test.

Author & Company



Doppelstep is designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®**: firmware and embedded Linux for connected products, secure, MISRA-compliant and ready for the EU Cyber Resilience Act.

Author

Anton Vynohradov · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Embedded Systems Engineer · Founder of SystemFromScratch®

Zephyr, Embedded Linux, MISRA, CRA.

Anton founded SystemFromScratch® to help industrial IoT, automotive, robotics, medical and consumer electronics teams turn hardware into secure, maintainable products. He designs and writes the software: firmware, drivers, RTOS and embedded Linux, security and compliance. For hardware design he works with experienced partner engineers, so a customer has one point of contact from the first board on the bench to the product in the field. He is based in Poland and works remotely worldwide.

For Doppelstep, Anton wrote the firmware, its drivers and board support, the Stepper-C, DXL2-C and X430 emulator libraries it is built on, the simulation models, the tests, the development environment, the bench GUI and this documentation.

SystemFromScratch®

SystemFromScratch® is one partner for the whole software stack of an embedded product: firmware, operating systems, security and regulation, with specialist hardware engineers joining when a project needs them.

Service	What it covers
Embedded software development	Bare-metal, RTOS and Linux application firmware in C/C++, including OTA, built for performance, reliability and long-term maintainability
BSP & device drivers	Board support packages, bootloaders and low-level drivers for MCUs, SoCs, FPGAs and custom hardware, including Zephyr drivers and Linux kernel work
RTOS & Embedded Linux	Porting, configuration, integration and reproducible builds for Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot)
Hardware bring-up & integration	The firmware side of first boot, validation and production, including motor-control platforms; hardware design through trusted partners
Cybersecurity & CRA readiness	Secure boot, firmware integrity, threat modelling and a secure development lifecycle for the EU Cyber Resilience Act
Compliance & consulting	MISRA C/C++, static analysis, code and architecture reviews, compliance roadmaps

Doppelstep is one of the open-source projects of the SystemFromScratch® portfolio: public engineering work that can be inspected line by line. It shows several of these services in one real product: Zephyr drivers and an out-of-tree board, a signed bootloader with field update, motor control in a hard real-time interrupt, an industrial protocol, tests against a physical model, and a reproducible environment.

Use it, build on it

Doppelstep is licensed under the Apache License 2.0 (License (p. 74)). You may use it, change it and build your own products on it, commercially too, under the terms of that license. It is meant to be repeated: every step from an empty PC to a moving motor is documented, and every number in this documentation can be re-measured from the sources.

Contact

Channel	Address
Website	systemfromscratch.github.io
E-mail	avynohradov@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch
LinkedIn	linkedin.com/in/vynohradov
ORCID	0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)
Location	Poland, working remotely worldwide

Building a connected product, bringing up new hardware, or preparing for the Cyber Resilience Act? Get in touch through systemfromscratch.github.io or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue on GitHub (github.com/SystemFromScratch/sfs-doppelstep/issues). Please include the commit or version, the build command and board, the log (RTT or `native_sim` output), and the steps or the test that show the problem.
- **Contributions:** follow the architecture rules (p. 31); the layer check and the tests must pass (Tests and CI (p. 54)).
- **Commercial support, integration and custom development:** contact SystemFromScratch® through systemfromscratch.github.io.

Trademarks and acknowledgements

SystemFromScratch® is a registered trademark. The SystemFromScratch® name and logo are used in this documentation to identify the publisher of Doppelstep; the Apache License 2.0 grants no rights to use them, and they may not be used to endorse or promote products derived from Doppelstep without prior written permission.

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. STM32 is a registered trademark of STMicroelectronics International N.V. Arm and Cortex are trademarks of Arm Limited. Hardware and protocol names are used only to describe compatibility; Doppelstep is not affiliated with or endorsed by their owners.

Doppelstep builds on the work of the Zephyr Project, MCUboot, STMicroelectronics (STM32Cube), SEGGER (RTT) and ROBOTIS (DynamixelSDK, the published Control Table of the XM430-W350).

How Doppelstep got its name

> *Your stepper, seen as a smart servo.*

The project before the name

The project started as **sfs-stepper-driver**: the internal SystemFromScratch name for a Zephyr RTOS firmware on the STMicroelectronics B-G431B-ESC1. It drives a low-cost two-phase stepper motor with Stepper-C (github.com/AntonVynohradov/Stepper-C) and answers on the bus as a DYNAMIXEL XM430-W350, through the DYNAMIXEL X430 Control Table Emulator (github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator). The host — ROS 2 `dynamixel_hardware_interface`, DynamixelSDK, DYNAMIXEL Wizard — cannot tell the difference.

`sfs-stepper-driver` says what the code is. It does not say why anyone would want it, and it reads like one more stepper driver among hundreds. An open-source project that people should find, remember and recommend needed a public name.

What the name had to do

1. **Tell the story in one word.** What makes the project different is not "a stepper driver" but *a stepper that passes for a smart servo*: drop-in, no change on the host side.
2. **Stay clear of other people's trademarks.** No DYNAMIXEL, ROBOTIS or STM32 in the name; the name may hint at what it emulates, never borrow the brand.
3. **Fit the field.** Successful open motor-control projects have short, one-word names: ODrive, moteus, VESC, SimpleFOC.
4. **Be findable.** A word that is not already taken by a product, especially not in robotics, fitness or motion, so that searches lead to the project.
5. **Leave room to grow.** The technical prefix `sfs-` stays for repositories and identifiers of the SystemFromScratch family; the public name stands on its own.

Candidates

Name	Idea	Why not
Mimistep	*mimic* + step	"mimic" is worn out as a product word
Turncoat	the one who changes sides	fun for a talk, too flippant for production use
Servostep / StepServo	stepper as servo	descriptive, but very likely in use by stepper driver vendors ("easy servo" class products)
OpenJoint	open robot joint	generic, drowns in search results
Articula	articulation, joint	pleasant, but says nothing about the stepper
Kinestep	kinetics + step	the strongest neutral candidate — see below
Doppelstep	*Doppelgänger* + step	chosen

The check

On 1 November 2026 both finalists were searched on the web:

- **Kinestep:** no exact match, but two close names in a neighbouring field: *KinesteX*, an AI fitness trainer app, and *Kineticstep*, a crowdfunded walking exoskeleton. Together with "step" suggesting walking and fitness, searches for Kinestep would land on other products.
- **Doppelstep:** no exact match at all — no product, project or company. The nearest hits (a music act called *Doppelgang*, a GitHub tool named *DoppelSpotter*) are unrelated.

Doppelstep was clearly the cleaner and more distinctive of the two.

Not checked yet, to be done before the first public release: the GitHub organisation name, the `.dev` / `.io` / `.com` domains, and trademark registers (EUIPO / TMview for the EU, classes 7, 9 and 42; USPTO).

The meaning

A **Doppelgänger** is a look-alike double: someone indistinguishable from another person. **Doppelstep** is a stepper motor that is the double of a smart servo: it speaks the same protocol, exposes the same Control Table, follows the same profiles, and the controller on the other end simply sees the servo it expects.

The name also keeps a quiet nod to its parts — *Doppel-* for the double it plays, *-step* for the honest stepper motor underneath, driven by Stepper-C.

How to write it

Where	Form
Prose, documentation, marketing	**Doppelstep** (one word, capital D)
Identifiers: repository, west path, Docker image and volume, CMake project	`doppelstep`
SystemFromScratch technical prefix (Kconfig `SFS_*`, devicetree vendor `sfs,`)	unchanged
Former internal name	`sfs-stepper-driver`

Not "DoppelStep", "Doppel-Step" or "Doppelstepper".

Tagline: *Your stepper, seen as a smart servo.*

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. Doppelstep is an independent project and is not affiliated with or endorsed by ROBOTIS.

APPENDIX A

License

Doppelstep (the application, the drivers, the board definition, the module glue, the configuration, the tests, the host tools and this documentation) is licensed under the **Apache License 2.0**. The full text is in the file `LICENSE` of the repository and at apache.org/licenses/LICENSE-2.0.

In short, and without replacing the license text: you may use, copy, change and distribute Doppelstep, also in commercial and closed-source products, provided that you keep the copyright and license notices, state your changes, and include the license with what you distribute. The license includes a patent grant and comes without warranty.

Copyright © 2026 Anton Vynohradov.

A.1 Components under their own licenses

A build of Doppelstep includes or uses components that come under their own licenses. Check them before you distribute a product.

Component	Where it comes from
Zephyr RTOS, MCUboot, mbedtls, tf-psa-crypto, zcbor, CMSIS 6	West projects imported from Zephyr v4.5.0 (`west.yml`)
STM32Cube HAL/LL (hal_stm32)	West project imported from Zephyr
SEGGER RTT	West project imported from Zephyr (`segger`)
Stepper-C, DYNAMIXEL X430 Control Table Emulator, DXL2-C, Stepper-Lab, sfs-stepper-power-spice	West projects (`west.yml`)
Zephyr SDK and the Python packages of the image	`Dockerfile`
Python packages of the GUI and the documentation	`requirements.txt`

The SEGGER J-Link Software is not part of the repository or the image unless you add it yourself under SEGGER's license (`docker/jlink/README.md`).

A.2 Trademarks

The Apache License 2.0 grants no rights to trademarks. SystemFromScratch® is a registered trademark; its name and logo may not be used to endorse or promote products derived from Doppelstep without prior written permission. DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd.; STM32 is a registered trademark of STMicroelectronics International N.V. Doppelstep is not affiliated with or endorsed by them.

A.3 Full license text

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

Copyright 2026 Anton Vynohradov

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

```
   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
```

```
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!) The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```