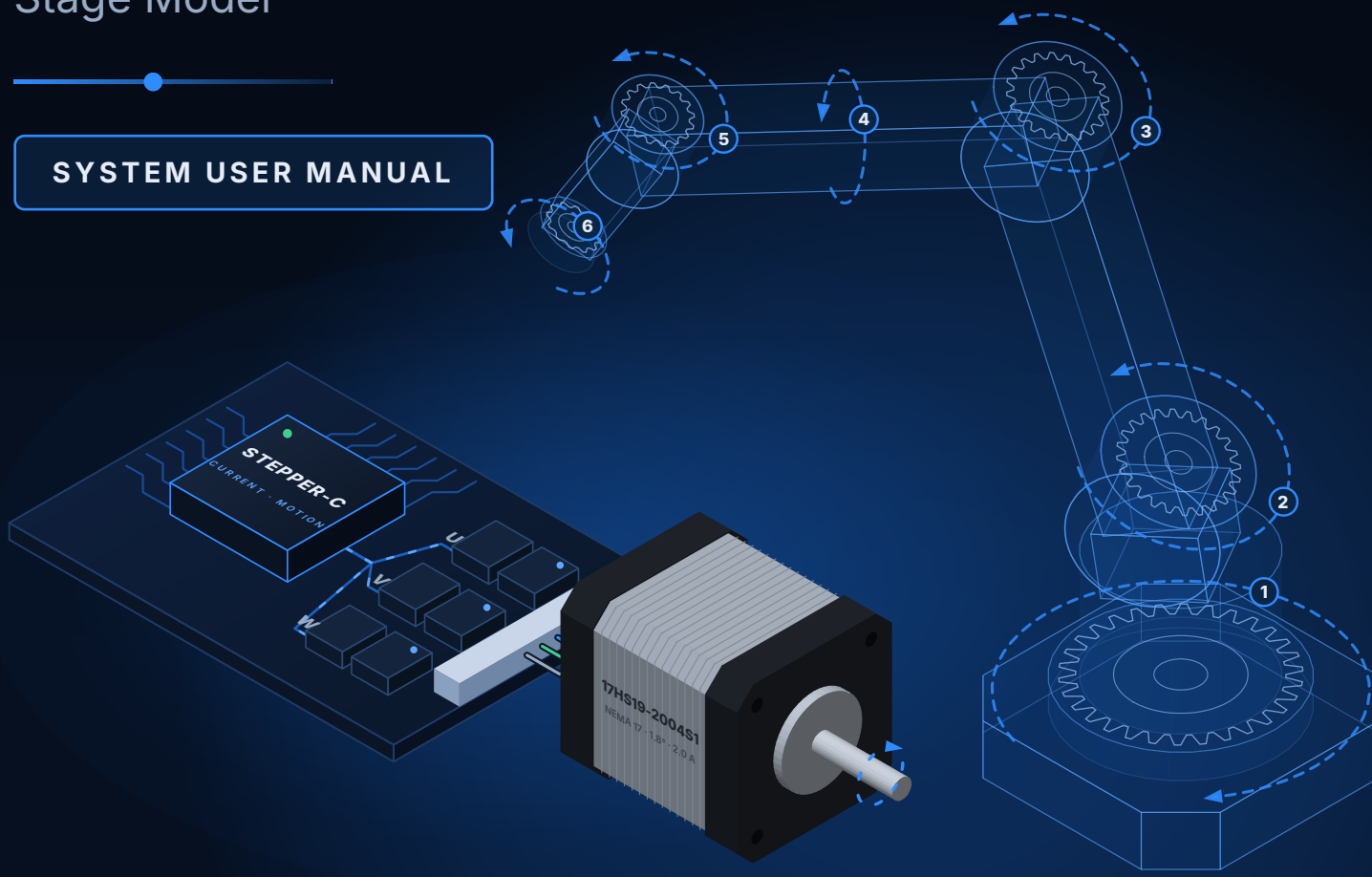
SystemFromScratch

OPEN-SOURCE MOTOR CONTROL SYSTEM

Stepper-C

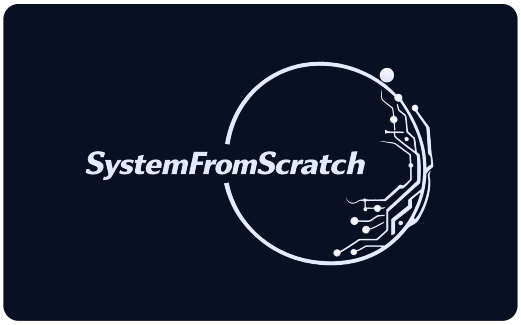
Stepper Motor Control Library, Simulation and Power Stage Model

SYSTEM USER MANUAL



Version 0.1.0 · October 2026
Anton Vynohradov
systemfromscratch.com

Imprint



Stepper-C System User Manual
Version 0.1.0
Edition: October 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
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-articles.github.io/Stepper-C-Motor-Control-Library/
Source code: https://github.com/AntonVynohradov/Stepper-C

Copyright © 2026 Anton Vynohradov.

The Stepper-C software (library, simulation and power stage models) 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 D. 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 Stepper-C without prior written permission.

STM32, STM32G4 and B-G431B-ESC1 are trademarks of STMicroelectronics. Scilab and Xcos are trademarks of their respective owners. They are used only to identify the products this manual refers to; Stepper-C 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 software versions stated in the Introduction; later versions may differ. The models are engineering tools: verify every limit on the real hardware before relying on it.

Contents

1 Introduction 6
- 1.1 What the system is 6
- 1.2 What it does and does not do 6
- 1.3 Who it is for 7
- 1.4 Key properties 7
- 1.5 Versions covered 8
- 1.6 Conventions 8
- 1.7 Terminology 9

2 System Architecture 10
- 2.1 The three repositories 10
- 2.2 Signal chain 11
- 2.3 How the parts are verified against each other 12
- 2.4 Development workflow 12
- 2.5 Hardware reference 13

3 Getting Started 14
- 3.1 Requirements 14
- 3.2 Library: build and test 14
- 3.3 Simulation: run the tests 15
- 3.4 Software in the loop: see the motor turn 15
- 3.5 Power stage: run ngspice 16
- 3.6 This manual 16

4 Motor Model 17
- 4.1 State equations 17
- 4.2 Sign convention 18
- 4.3 Parameters of the 17HS19-2004S1 18
- 4.4 Characteristic values 19
- 4.5 Numerical solution 19
- 4.6 Limits of the model 20

5 Control Algorithms 21
- 5.1 Two loops and the hand-off 21
- 5.2 Numbers: per unit, fixed point and float 21
- 5.3 Current references 22
- 5.4 Current controller 22
- 5.5 Voltage limit and leg mapping 24
- 5.6 Motion generator 24
- 5.7 States and faults 25

6 Stepper-C Library 27
- 6.1 Repository layout 27
- 6.2 Adding the library to a project 27
- 6.3 Build-time configuration 28
- 6.4 Configuring a driver 28
- 6.5 Connecting the loops to the hardware 30
- 6.6 Commanding motion 31

6.7 Memory, timing and reentrancy ... 32
6.8 Quality ... 32
6.9 Porting checklist ... 32

7 API Reference ... 34
7.1 Headers ... 34
7.2 Status codes ... 34
7.3 Enumerations and flags ... 34
7.4 Callbacks ... 35
7.5 Configuration structures ... 35
7.6 Functions ... 37
7.7 Calling contexts ... 40

8 Simulation ... 41
8.1 Repository layout ... 41
8.2 Parameters ... 41
8.3 Open-loop motor simulation ... 42
8.4 Closed-loop prototype ... 42
8.5 Motion generator twin ... 43
8.6 Power stage models ... 44
8.7 Python twins ... 45
8.8 Running the tests ... 46
8.9 Adapting the model to another motor or board ... 46

9 Software in the Loop ... 47
9.1 Start here ... 47
9.2 The diagram ... 47
9.3 Scenario: the diagram context ... 48
9.4 Results in the workspace ... 49
9.5 Build and rebuild ... 49
9.6 Agreement with the prototype ... 49
9.7 Troubleshooting ... 50

10 Power Stage Model ... 51
10.1 The leg circuit ... 51
10.2 Device models ... 52
10.3 Board values and assumptions ... 52
10.4 What happens in the dead time ... 53
10.5 Dead time voltage error ... 54
10.6 Shunt sampling ... 54
10.7 Export to the simulation ... 55
10.8 Tests ... 55

11 Test Catalogue ... 56
11.1 Naming ... 56
11.2 Motor model ... 56
11.3 Power stage and current loop ... 56
11.4 Motion ... 57
11.5 Resonance and friction ... 57
11.6 Software in the loop ... 58
11.7 Library unit tests (GoogleTest) ... 58
11.8 ngspice ... 58
11.9 Running everything ... 59

12 Results and Design Guidance 60
12.1 Current loop 60
12.2 Voltage limit and speed 60
12.3 Rotor resonance 61
12.4 Robustness 62
12.5 Power stage 62
12.6 Open points 62

13 Hardware Bring-up 64
13.1 Connection 64
13.2 Measurements before the first run 64
13.3 Firmware configuration (STM32G431) 65
13.4 First power-up sequence 65
13.5 What to compare with the models 65

Author & Company 67
Author 67
SystemFromScratch® 68
Contact 68
Support and feedback 69
Trademarks and acknowledgements 69

Appendix A Parameter Reference 70
A.1 Stepper-C configuration 70
A.2 Simulation parameters 71
A.3 ngspice board values 72
A.4 Derived values 72

Appendix B Glossary 73

Appendix C Changelog 75
C.1 [Unreleased] 75

Appendix D License 76
D.1 What the license allows 76
D.2 What it requires 76
D.3 What it does not give 76
D.4 Trademarks 77
D.5 Full license text 77

CHAPTER 1

Introduction

This manual describes the **Stepper-C system**: a control library for two-phase hybrid stepper motors written in C11, the Scilab/Xcos model it is developed and verified against, and the ngspice model of the power stage it will drive first. It is written for embedded engineers who want to use the library, check its behaviour in simulation, or extend the models. Every number in it was measured with the models and tests of the versions listed below.

1.1 What the system is

Stepper-C replaces the closed "step/direction" driver chip with software: the microcontroller measures the two phase currents, runs a current controller per phase and writes the PWM of the power stage itself. The library computes everything between a motion command (*move to position 3200*) and the PWM compare values; the board code only reads the ADC and writes the timer.

Three repositories make up the system:

Repository	Local folder	Content	Language / tool
Stepper-C (github.com/AntonVynohradov/Stepper-C)	`sfs-stepper-ctrl`	**Stepper-C**, the control library	C11, CMake, GoogleTest
sfs-stepper-sim (github.com/AntonVynohradov/sfs-stepper-sim)	`sfs-stepper-sim`	Motor model, prototype of the control algorithms, software in the loop	Scilab 2027, Xcos, Python twins
sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice)	`sfs-stepper-power-spice`	Transistor-level model of one leg of the B-G431B-ESC1 power stage	ngspice, Python

The library repository is published as **Stepper-C**; it is cloned into the folder `sfs-stepper-ctrl`, the name the simulation looks for.

The library is the product; the two models exist to find its limits before hardware does. The chain is closed in both directions: the C code runs unchanged inside Xcos, and the parameters the ngspice model extracts (effective dead time, leg resistance, diode drop) are written into the simulation.

1.2 What it does and does not do

Stepper-C does	Why
Closed-loop current control of both phases (PI with back-calculation anti-windup, back-EMF feed-forward)	Microstepping torque follows the current, not the voltage; at speed only a current loop keeps the amplitude
Microstepping 1 … 256 per full step from a 32-bit electrical angle	Smooth motion, low noise; resolution independent of the step rate
Trapezoidal motion profile with exact integer braking	No overshoot, exact stop on the target, no velocity step when a move is replaced

Stepper-C does	Why
Two power stage topologies: two H-bridges, or three legs with a shared leg	The shared-leg layout lets three-phase BLDC boards (B-G431B-ESC1) drive a stepper
Bus voltage compensation and voltage vector limiting with the direction kept	Torque direction is preserved when the bridge runs out of voltage
Overcurrent, over- and undervoltage trips; external faults	Protects the board in the fast loop, within one PWM period
Fixed point (Q15/Q31) or float arithmetic, chosen at build time	Runs on MCUs with and without FPU

Stepper-C does not	Why / what to use instead
Generate step/direction pulses for an external driver chip	It *is* the driver; use the chip's own interface for that case
Read the clock, sleep or allocate memory	Time comes from the configured loop rates; the instance is allocated by the application
Configure timers, ADC or GPIO	Board-specific; done by the application through the callbacks
Close a position loop on an encoder (yet)	Open-loop microstepping; the load angle is visible in simulation (planned extension)
Compensate the dead time (yet)	The configuration field exists; the ngspice results define the value (planned)

1.3 Who it is for

- **Firmware engineers** integrating stepper control on an STM32, a Cortex-M0 or any MCU with two ADC channels and a PWM timer: read Stepper-C Library (p. 27) and the API Reference (p. 34).
- **Control and system engineers** who want to know where the drive loses steps, resonates or saturates: read Motor Model (p. 17), Control Algorithms (p. 21) and Results (p. 60).
- **Hardware engineers** choosing dead time, gate resistors and shunt sampling: read Power Stage Model (p. 51).
- **Students** who want to open one file, press Start and see a stepper motor being controlled by real embedded C code: read Software in the Loop (p. 47).

1.4 Key properties

Property	Value
Language	C11 (library), Scilab 2027 (model), Python 3.9+ (twins, tools), ngspice 42+
Dependencies of the library	None (`<stdint.h>`, `<stdbool.h>`, `<stddef.h>` only; no `<string.h>`)
Memory	One `struct sstep` per motor, allocated by the application; no heap, no global state
Code size	8.2 kB Cortex-M4F fixed point, 9.0 kB Cortex-M4F float, 8.5 kB Cortex-M0+ (clang `-Os`, both topologies, sine table included)
Execution model	`sstep_fast_loop()` in the PWM interrupt (20 kHz), `sstep_slow_loop()` at 1 kHz

Property	Value
Unit tests	76 (fixed point), 75 (float); 100 % line and branch coverage of `src/`; clean under ASan/UBSan
Model tests	T00–T12, B01, X01–X04 (prototype), S01–S04 (software in the loop), SP01–SP03 (ngspice)
Coding rules	MISRA C:2012 followed in the code; the Cppcheck MISRA target is configured, a full report is pending
License	Apache License 2.0

1.5 Versions covered

Component	Version	State
Stepper-C (github.com/AntonVynohradov/Stepper-C) (`sfs-stepper-ctrl`)	0.1.0 (unreleased)	control loops complete, tested on models
Simulation (sfs-stepper-sim (github.com/AntonVynohradov/sfs-stepper-sim))	0.1.0 (unreleased)	prototype, switching stage, SIL, resonance and robustness tests
Power stage (sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice))	0.1.0 (unreleased)	ESC1 leg model, export to the simulation
Hardware port (B-G431B-ESC1)	—	next step (P3), see Hardware Bring-up

> **Models are not hardware**
>
> The results in this manual come from models with datasheet parameters and several stated assumptions (detent torque, dry friction, gate resistors, dead time). They show *where* to look and *what* to expect. Check every limit on the real motor and board before relying on it, and start with a current-limited supply.

1.6 Conventions

Item	Convention
Phase currents	`i_A = I cos φ`, `i_B = I sin φ`; the rotor settles at `p θ = φ`
Electrical angle	32-bit unsigned, one turn = 2^{32}, wraps naturally
Positions	microsteps (API), 1/65536 microstep inside the motion generator
Load angle	`φ − p θ` in electrical degrees; positive = rotor behind the field; loss of synchronism above 90° … 180°
Per unit	current base = ADC full scale current, voltage base = nominal bus voltage
Configuration	integer engineering units: mΩ, µH, mA, mV, Hz, ns
Code	function names with `()`, values with units, hex as `0x0010`
Tests	`T` prototype tests, `X` cross-checks with the Python twins, `B` bridge, `S` software in the loop, `SP` ngspice

1.7 Terminology

Full step
: 1.8° for a 200-step motor: one quarter of an electrical period.

Microstep
: A fraction 1/M of a full step; M = 16 gives 3200 positions per revolution.

Fast loop
: The code that runs every PWM period: current measurement, PI controllers, voltage limit, PWM write.

Slow loop
: The code that runs at 1 kHz: motion profile, current amplitude, events.

Shared leg
: A power stage with three half-bridges where phase A is connected between legs U and W and phase B between V and W; W is common to both phases.

Software in the loop (SIL)
: The real library code, compiled for the PC, running inside the Xcos simulation against the motor model.

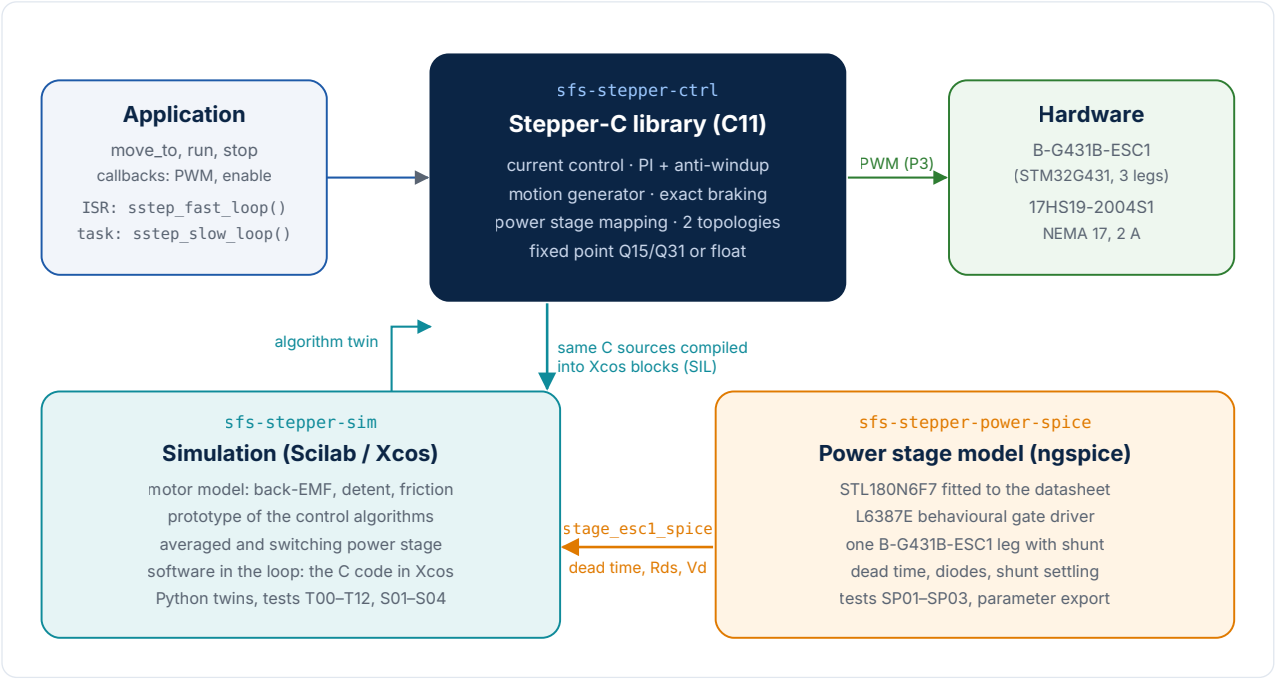
Twin
: A second implementation of the same algorithm in another language (Scilab ↔ Python, Scilab ↔ C) used to cross-check results sample by sample.

CHAPTER 2

System Architecture

This chapter shows how the library, the simulation and the power stage model fit together, which data flows between them, and the development workflow that keeps them consistent. The individual modules are described in their own chapters.

2.1 The three repositories



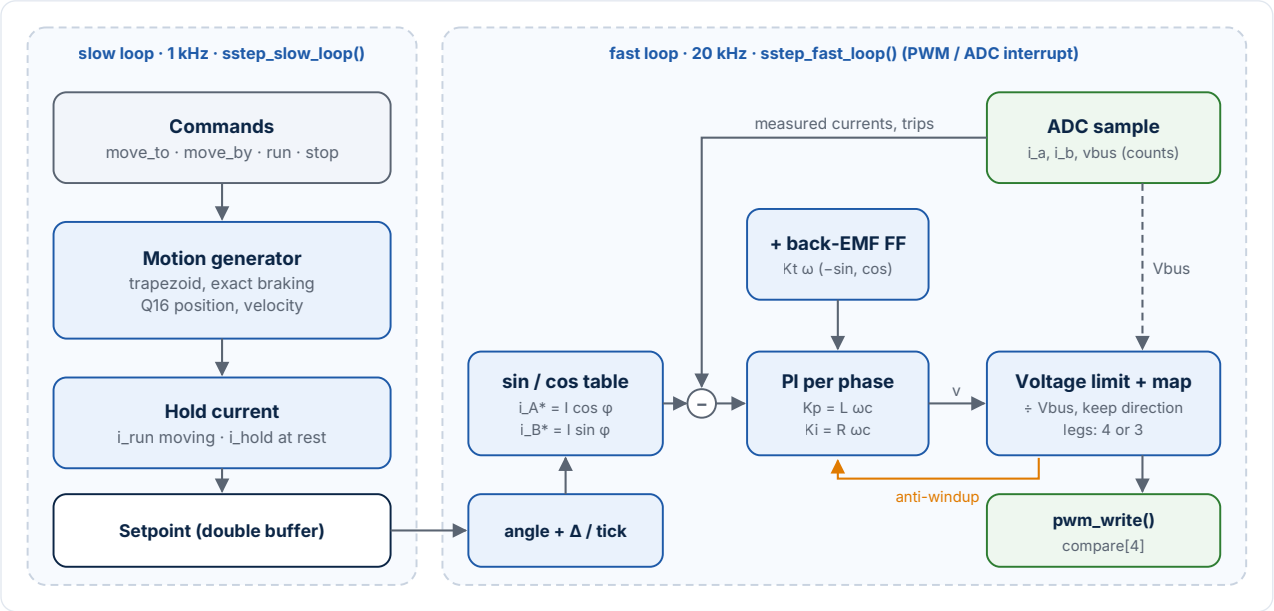
Part	Role in the system	Gives to the others	Takes from the others
Stepper-C (GitHub (github.com/AntonVynohradov/Stepper-C), folder `sfs-stepper-ctrl`)	The product: control loops for the MCU	C sources compiled into the Xcos blocks	Algorithms first proven in the prototype; dead time value for compensation
Simulation (sfs-stepper-sim (github.com/AntonVynohradov/sfs-stepper-sim))	Motor model, prototype of every algorithm, SIL test bench	Reference results, limits (speed, resonance, robustness)	The library sources (SIL); the switching stage parameters from ngspice
Power stage model (sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice))	Transistor-level view of one ESC1 leg	`params/stage_esc1_spice.sci` / `.json`: effective dead time, leg resistance, diode drop	Board values (gate resistors, shunt, dead time)
Hardware (B-G431B-ESC1 + 17HS19-2004S1)	The target of the first port (P3)	Measured R, L, Kt, dead time, gate waveforms	The library and its configuration

The repositories are independent Git repositories that sit next to each other:

```
sfs-stepper/
├── sfs-stepper-ctrl/          Stepper-C library
├── sfs-stepper-sim/           Scilab/Xcos model, SIL, Python twins
├── sfs-stepper-power-spice/   ngspice model of the power stage
└── sfs-stepper-article/       this manual (website + PDF), repository Stepper-C-Motor-
Control-Library
```

The simulation finds the library through the environment variable `SSTEP_CTRL_DIR`, a Git submodule `ext/sfs-stepper-ctrl`, or the sibling folder `../sfs-stepper-ctrl` (in that order). The ngspice runner writes its export into `../sfs-stepper-sim/params/` when that folder exists.

2.2 Signal chain



From a command to a turning shaft:

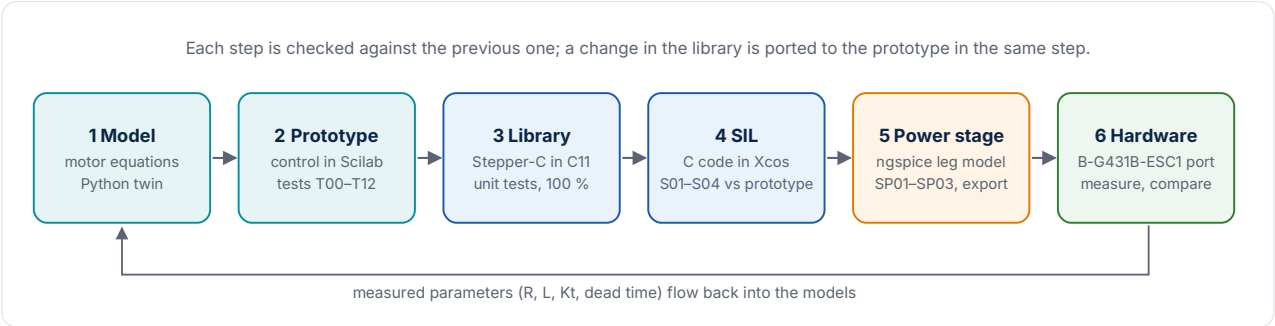
1. The application calls `sstep_move_to()` (or `sstep_run()`, `sstep_stop()`).
2. Every slow loop period (1 ms) the **motion generator** advances the position with the acceleration and velocity limits, chooses the current amplitude (run or hold) and publishes a **setpoint**: the electrical angle at the start of the period, its increment per fast period, the current amplitude and the back-EMF feed-forward.
3. Every fast loop period (50 µs at 20 kHz) `sstep_fast_loop()` converts the ADC counts to currents, checks the trip levels, computes the current references `I cos φ`, `I sin φ`, runs one **PI controller per phase**, adds the feed-forward, limits and maps the voltage vector to the legs of the topology and calls `pwm_write()`.
4. The **power stage** applies the leg voltages; the **motor** turns the phase currents into torque; the ADC samples the currents at the centre of the next PWM period.

The same chain exists three times — in C (the library), in Scilab (the prototype) and in Python (the twin) — and the three are compared sample by sample.

2.3 How the parts are verified against each other

Comparison	Test	Agreement
Scilab motor model ↔ Python model (scipy, Radau)	X01	$4 \cdot 10^{-8}$ A
Scilab closed loop ↔ Python closed loop	X02, X03	$5 \cdot 10^{-13}$ A
Scilab motion generator ↔ C motion generator	`tools/ref_motion.py` against `src/sstep_motion.c`	bit-exact over 3 052 885 periods
C library in Xcos ↔ Scilab prototype	S01–S04	S02, S03: currents within 1.05 mA (ADC step 1.95 mA), commanded angle identical; S01, S04: re-run pending
Scilab switching stage ↔ Python switching stage ↔ C plant block	X04, host check	$4 \cdot 10^{-16}$ (C ↔ Python)
ngspice device model ↔ STL180N6F7 datasheet	SP01	Rds(on), Ciss, Coss, Crss, Qg within 10 %
C unit tests with a motor model ↔ Scilab prototype	`gtest_sstep_loop.cpp`	current step 1.11 % overshoot (prototype 1.12 %)

2.4 Development workflow



The work follows the order of the figure, and each step is checked against the previous one before the next starts:

1. **Model.** The motor equations are written once (`stepper_motor_f`) and checked against analytic results (T00–T02) and an independent Python solver (X01).
2. **Prototype.** Each control algorithm is first written in Scilab and tested on the model (T03–T12).
3. **Library.** The algorithm is implemented in C11 for both numeric backends, with unit tests and 100 % coverage.
4. **Software in the loop.** The C code runs in Xcos against the same model; S01–S04 must match the prototype.
5. **Power stage.** ngspice extracts the switching parameters of the real board; the simulation uses them.
6. **Hardware.** The library is ported to the B-G431B-ESC1; measured motor and board values flow back into the models.

Rule: the prototype follows the library

The Scilab prototype always implements the *current* algorithm of the library. A change in Stepper-C is ported to the prototype and to its Python twin in the same step, so that the SIL comparison compares identical algorithms.

2.5 Hardware reference

Item	Part	Key data
Motor	StepperOnline 17H-S19-2004S1 (NEMA 17)	2 phases, 200 steps, 1.60 Ω, 3.0 mH, 2.0 A, 0.59 N·m, 82 g·cm²
Board	STMicroelectronics B-G431B-ESC1	STM32G431CB (Cortex-M4F, 170 MHz), 3 half-bridges, L6387E drivers, STL180N6F7 MOSFETs (60 V), three low-side shunts
Connection	Shared leg	phase A between U and W, phase B between V and W
Supply	Laboratory supply	24 V nominal (the board accepts 3S … 6S LiPo voltages)

With three legs the largest sinusoidal phase voltage is `Vbus/√2` (17.0 V at 24 V) instead of `Vbus` with two H-bridges; the consequences are measured in Results (p. 60).

CHAPTER 3

Getting Started

This chapter installs the tools, gets the three repositories and runs each part once: the library tests, the simulation tests, the software-in-the-loop diagram and the ngspice model. Each section can be done on its own; the order below is the one that needs the fewest tools first.

3.1 Requirements

Tool	Version	Needed for	Notes
Git	any	all	
CMake	3.21+	library	presets need 3.21
C compiler	GCC 11+, Clang 14+ or MSVC 2022	library, SIL	the SIL build uses the compiler Scilab finds
Python	3.9+	twins, ngspice runner, docs	`numpy`, `scipy`, `matplotlib`
Scilab	2027.0.0 (2025 and 2026 work)	simulation, SIL	includes Xcos
Visual Studio 2022 or Build Tools	"Desktop development with C++"	SIL on Windows	Scilab compiles the Xcos blocks with it
ngspice	42+ (tested 45.2)	power stage model	console executable
GoogleTest, gcov	from the presets	library unit tests	fetched by CMake

Get the repositories into one folder:

```
mkdir sfs-stepper && cd sfs-stepper
git clone https://github.com/AntonVynohradov/Stepper-C.git sfs-stepper-ctrl
git clone https://github.com/AntonVynohradov/sfs-stepper-sim.git
git clone https://github.com/AntonVynohradov/sfs-stepper-power-spice.git
```

The library repository Stepper-C (github.com/AntonVynohradov/Stepper-C) is cloned into the folder `sfs-stepper-ctrl`: the simulation looks for the library under that name (or in `SSTEP_CTRL_DIR`).

3.2 Library: build and test

The library uses CMake presets. The preset names carry the configuration in brackets:

Preset	What it builds
`[Debug]Stepper Motor Control`	static library, debug
`[Release]Stepper Motor Control`	static library, optimised
`[UnitTests]Stepper Motor Control`	GoogleTest suite, fixed point backend, coverage

Preset	What it builds
`[UnitTestsFloat]Stepper Motor Control`	GoogleTest suite, float backend
`[IntegrationTests]Stepper Motor Control`	pytest integration tests (placeholder)
`[Examples]Stepper Motor Control`	host examples (placeholder)

```
cd sfs-stepper-ctrl
cmake --preset "[UnitTests]Stepper Motor Control"
cmake --build "build/[UnitTests]Stepper Motor Control"
ctest --test-dir "build/[UnitTests]Stepper Motor Control" --output-on-failure
```

Expected result: 76 tests pass (75 with the float preset). VS Code users find the same steps as tasks (*Terminal → Run Task*).

3.3 Simulation: run the tests

1. Open Scilab and change to the `sfs-stepper-sim` folder.
2. Run the test runner:

```
exec("tests/run_all.sce", -1);
```

or from a terminal (exit status 1 on a failure, for CI):

```
scilab-cli -f tests/run_all.sce
```

Each test prints one line `PASS name: measured values` or `FAIL …`, and the runner ends with the number of failures. The full run takes a few minutes; T10 (switching stage) and T11 (resonance sweep) take the longest.

The Python twins run without Scilab:

```
cd sfs-stepper-sim
python -m pip install -r requirements-dev.txt
python tools/ref_model.py        # T00–T02, motor model
python tools/ref_ctrl.py         # B01, T03, T04, current loop
python tools/ref_motion.py       # T05–T07, T09, motion generator
python tools/ref_switching.py    # T10, switching stage
python tools/ref_ext.py          # T08, T11, T12
```

3.4 Software in the loop: see the motor turn

1. Make sure `sfs-stepper-ctrl` is next to `sfs-stepper-sim` (or set `SSTEP_CTRL_DIR`).
2. Open `start_sil.sce` in Scilab and press **F5**. The first time, Scilab compiles the library and the Xcos blocks (a few seconds); later it only links them, and it rebuilds automatically when a source file changed.
3. Xcos opens `xcos/stepper_sil.zcos`. Press **Start** (▶).

The scope shows the phase currents with their references, the speed, the load angle and the commanded and real rotor positions. The scenario is edited in **Simulation → Set Context**; see Software in the Loop (p. 47) for every variable.

> **After an update of the blocks**
>
> When the block interface changed (new outputs, parameters), set `sil_regenerate = %t` in `start_sil.sce` once, or delete `xcos/stepper_sil.zcos`; the diagram is generated again from `xcos/sil_diagram.sci`. Restart Scilab when the library was rebuilt, because Windows keeps the old DLL loaded.

3.5 Power stage: run ngspice

1. Install ngspice. On Windows, download the 64-bit zip from https://ngspice.sourceforge.io/download.html, unpack it and point `NGSPICE` to `Spice64\bin\ngspice_con.exe` (or install `mingw-w64-ucrt-x86_64-ngspice` in MSYS2). On Linux, `apt install ngspice`.
2. Run the tests and export the parameters to the simulation:

```
cd sfs-stepper-power-spice
python -m pip install -r requirements-dev.txt
python tools/run_tests.py --export     # about 40 s
python tools/plot_leg.py               # docs/leg_edges.png
```

The runner prints SP01–SP03 and writes `export/stage_esc1.json` and, when `../sfs-stepper-sim` exists, `params/stage_esc1_spice.sci` and `.json` there.

3.6 This manual

The manual is built from Markdown with MkDocs Material (website) and WeasyPrint (PDF):

```
git clone https://github.com/SystemFromScratch-Articles/Stepper-C-Motor-Control-
Library.git sfs-stepper-article
cd sfs-stepper-article
python -m pip install -r requirements-docs.txt
mkdocs serve                           # http://127.0.0.1:8000
python tools/docs/build_manual.py      # pdf/Stepper-C_System_Manual_v0.1.0.pdf (not
committed)
python tools/docs/diagrams.py          # regenerate the diagrams
python tools/docs/figures.py           # regenerate the data figures (needs the sim
twins)
```

WeasyPrint needs the Pango library: on Windows install `mingw-w64-x86_64-pango` in MSYS2 (the build script finds `C:\msys64\mingw64\bin` by itself), on Linux `apt install libpango-1.0-0 libpangoft2-1.0-0`.

CHAPTER 4

Motor Model

The motor model is a two-phase hybrid stepper with sinusoidal back-EMF, a detent torque term, viscous and dry friction and a rigid load. It is written once (`model/stepper_motor_f.sci`) and used by the prototype, the Xcos plant block (in C) and the Python twins. This chapter gives the equations, the sign conventions, the parameters of the 17HS19-2004S1 and the limits of the model.

4.1 State equations

The state is $x = [\, i_A,\ i_B,\ \omega,\ \theta \,]$: the two phase currents (A), the mechanical speed (rad/s) and the mechanical rotor angle (rad). With $p = N_s/4$ electrical periods per mechanical turn (50 for a 200-step motor):

Electrical part

$$\begin{aligned} L\frac{di_A}{dt} &= v_A - R\,i_A - e_A, \qquad & e_A &= -K_t\,\omega\sin(p\theta) \\ L\frac{di_B}{dt} &= v_B - R\,i_B - e_B, \qquad & e_B &= K_t\,\omega\cos(p\theta) \end{aligned}$$

Mechanical part

$$\begin{aligned} J\frac{d\omega}{dt} &= T_e - T_L - T_f \\ \frac{d\theta}{dt} &= \omega \end{aligned}$$

Torques

$$\begin{aligned} T_e &= K_t\,(-i_A\sin(p\theta) + i_B\cos(p\theta)) - T_d\sin(4p\theta) \\ T_f &= B\,\omega + T_c\tanh\left(\frac{\omega}{\omega_c}\right) \end{aligned}$$

Symbol	Meaning	Unit
v_A, v_B	phase voltages applied by the power stage	V
R, L	phase resistance and inductance (no mutual inductance between phases)	Ω, H
K_t	torque constant = back-EMF constant	N·m/A = V·s/rad
T_d	detent torque amplitude (period: one full step)	N·m
T_L	load torque	N·m
B	viscous friction	N·m·s/rad

Symbol	Meaning	Unit
T_c, ω_c	dry (Coulomb) friction and its smoothing speed	N·m, rad/s
J	rotor + load inertia	kg·m²

The back-EMF and the torque come from the same flux linkage, so the electrical power $e_A i_A + e_B i_B$ equals the mechanical power $T_e\,\omega$ (without the detent term, which does no net work over a step). Test **T00** checks this balance on 1000 random states (largest error 1.1·10⁻¹³ W).

4.2 Sign convention

With the current vector

$$i_A = I\cos\varphi, \qquad i_B = I\sin\varphi$$

the torque is $T_e = K_t\, I \sin(\varphi - p\theta)$: it is zero and restoring at $p\theta = \varphi$. The rotor therefore **settles at** $p\theta = \varphi$, increasing φ turns it forward, and phase A leads phase B by 90°. The electrical angle between the field and the rotor, the **load angle** (electrical degrees),

$$\delta = \varphi - p\theta$$

is the most useful quantity in every test: with a load T_L the rotor lags by $\delta = \arcsin\left(T_L/(K_t I)\right)$, the largest static torque is $K_t I$ at $\delta = 90\,^\circ$, and beyond $\delta \approx 180\,^\circ$ the rotor falls into the next pole — the motor has lost steps.

> **Note**
>
> An early project note used $i_A = I\sin\varphi$, $i_B = I\cos\varphi$. With the torque equation above that turns the rotor backwards; the library, the prototype and this manual all use the cosine/sine form.

4.3 Parameters of the 17HS19-2004S1

Para-meter	Value	Source
R	1.60 Ω	datasheet (±10 %, 25 °C); older listings state 1.4 Ω — **measure**
L	3.0 mH	datasheet at 1 kHz (±20 %)
I_{rated}	2.0 A	datasheet
K_t	0.2086 N·m/A	derived: holding torque 0.59 N·m with both phases at 2 A, $K_t = 0.59/(\sqrt{2}\cdot 2)$ — **measure the back-EMF**
N_s	200 (1.8°)	datasheet
J	82 g·cm² = 8.2·10⁻⁶ kg·m²	datasheet, rotor only
B	1·10⁻⁵ N·m·s/rad	assumption
T_d	0 (tests: 0.02 N·m)	not in the datasheet; 0.02 N·m ≈ 3 % of the holding torque is an assumption

Para-meter	Value	Source
T_c	0 (tests: 0.01 N·m)	assumption
ω_c	0.05 rad/s	numerical smoothing of the dry friction

The parameters are a Scilab structure returned by `motor_17hs19_2004s1()` (`params/motor_17hs19_2004s1.sci`); a test or a scenario changes a field (`mot.Td = 0.02;`) without touching the equations.

> **Measuring K_t**
>
> Spin the unpowered shaft (a drill), measure the open-circuit phase voltage amplitude E and the speed ω: $K_t = E/\omega$. If the datasheet holding torque is meant for one phase only, `K_t` would be 0.295 N·m/A — 41 % more, which changes every speed limit in this manual.

4.4 Characteristic values

Quantity	Formula	17HS19-2004S1
Electrical time constant	$\tau = L/R$	1.875 ms
Steady current at 3.2 V	V/R	2.0 A (T01)
Rotor natural frequency at current I	$f_n = \frac{1}{2\pi}\sqrt{pK_tI/J}$	254 Hz at 2 A, no load
Back-EMF amplitude	$K_t\,\omega$	1.31 V per rev/s
Holding torque	$\sqrt{2}\,K_tI$ (both phases)	0.59 N·m at 2 A
Static torque of the current vector	K_tI	0.417 N·m at 2 A

The natural frequency is the stiffness of the "magnetic spring" pK_tI against the inertia; it decides the low-speed resonance (Results (p. 61)).

4.5 Numerical solution

Use	Solver
Open-loop studies (`stepper_simulate`)	Scilab `ode` (lsoda), rtol 10^{-8}, atol 10^{-10}
Closed loop, averaged power stage	fixed-step RK4, 4 steps per PWM period (12.5 µs)
Closed loop, switching power stage	RK4 between the switching instants, at least 16 steps per period
Python twin of the open-loop model	scipy `solve_ivp` (Radau), rtol 10^{-10}

The dry friction $T_c\tanh(\omega/\omega_c)$ is stiff near zero speed (time constant $J\omega_c/T_c$). RK4 stays stable while $h\,T_c/(J\omega_c) < 2.8$: with ω_c = 0.05 rad/s and the 12.5 µs step of the closed loop that is T_c up to about 0.09 N·m. Increase ω_c for larger friction values.

4.6 Limits of the model

The model is meant for designing and testing the controller. It does **not** include:

- magnetic saturation (inductance and `K_t` drop at high current);
- harmonics of the back-EMF and of the torque other than the detent term;
- mutual inductance between the phases;
- temperature (R rises about 0.4 %/K in copper);
- stiction (the dry friction is smoothed);
- elasticity of the coupling and the load (one rigid inertia).

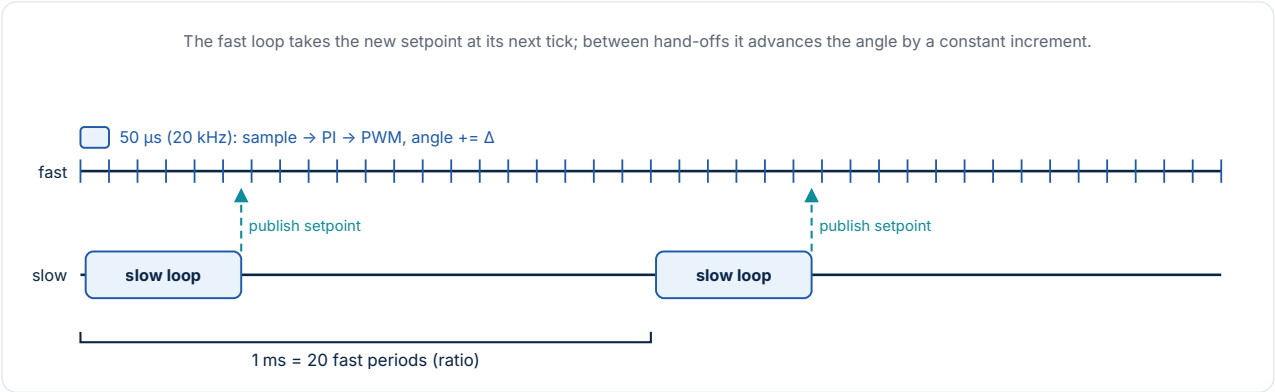
Effects that depend on these — the exact speed of the mid-band instability, the noise spectrum, the current at full saturation — must be checked on the hardware.

CHAPTER 5

Control Algorithms

This chapter explains every algorithm of Stepper-C: the timing of the two loops, the number formats, the current controller, the mapping of the voltage to the power stage legs, the motion generator and the fault handling. The same algorithms are implemented in the Scilab prototype (`model/`) and in the Python twins (`tools/`); where the prototype differs, it says so.

5.1 Two loops and the hand-off



Loop	Rate (default)	Context	Work
Fast	`fast_loop_hz` = 20 kHz (PWM frequency)	PWM / ADC interrupt	ADC conversion, trips, current references, PI, feed-forward, voltage limit, leg mapping, `pwm_write()`
Slow	`slow_loop_hz` = 1 kHz	main loop or timer task	motion generator, run/hold current, bus voltage scale, setpoint, events

`fast_loop_hz` must be an integer multiple of `slow_loop_hz`; the ratio (20 by default) is stored as `ratio`.

Hand-off. The slow loop writes a complete setpoint — electrical angle at the start of its period, angle increment per fast period, current amplitude, back-EMF feed-forward, bus voltage scale — into the buffer the fast loop is *not* using, then sets `sp_pending`. The fast loop swaps buffers at its next tick. On a single-core MCU this needs no lock: the fast loop never reads the buffer being written, and the flag is cleared before writing.

Angle interpolation. Between hand-offs the fast loop adds the increment `Δ` to the angle every tick, at most `ratio` times; if the slow loop is late, the angle stops instead of running ahead.

5.2 Numbers: per unit, fixed point and float

All signals inside the loops are **per unit**:

Quantity	Base	1.0 pu means
Current	`i_full_scale_ma` (ADC full scale)	the current at the end of the ADC range

Quantity	Base	1.0 pu means
Voltage	`vbus_nominal_mv`	the nominal bus voltage
Angle	2^{32}	one electrical turn

Type	Fixed point backend	Float backend
Signal `sstep_real_t`	Q15 in `int16_t` (−1 … 1)	`float`
Accumulator `sstep_acc_t`	Q15 in `int32_t` (±65536 pu)	`float`
PI integrator `sstep_integ_t`	Q27 in `int32_t` (keeps increments far below one LSB of Q15)	`float`
Gain `struct sstep_gain`	Q15 mantissa + shift exponent	`float`
Angle `sstep_angle_t`	`uint32_t`, wraps at one turn	same

Every operation saturates instead of wrapping, in both backends with identical ranges. `sstep_init()` derives all gains from the integer configuration without floating point (2π is taken as 710/113, error $8.5 \cdot 10^{-8}$). The sine and cosine come from a quarter-wave table (`SSTEP_CFG_SINE_TABLE_BITS` = 8: 257 entries) with linear interpolation; the measured error is 1.0 LSB ($3.05 \cdot 10^{-5}$) in fixed point and $4.7 \cdot 10^{-6}$ in float.

Microstep angle. One microstep is `2³⁰ / M` of an electrical turn (four full steps per turn). The electrical angle of a position is `pos << (14 − log₂ M)` with the position in 1/65536 microstep; positions therefore wrap exactly at one electrical turn.

5.3 Current references

From the angle φ and the amplitude I of the setpoint:

$$i_A^* = I\cos\varphi, \qquad i_B^* = I\sin\varphi$$

The amplitude is `i_run_ma` while the motor moves and `i_hold_ma` after `hold_delay_ms` of standstill (event `SSTEP_EVENT_HOLD_CURRENT`). `sstep_enable()` starts with the run current.

5.4 Current controller

One PI controller per phase, the same for both:

$$\begin{aligned} e_k &= i_k^* - i_k \\ v_k^* &= K_p\, e_k + \mathrm{integ}_k + v_{\mathrm{ff},\,k} \\ \mathrm{integ}_k &\leftarrow \mathrm{integ}_k + K_i T_s\, e_k + K_b \left(v_{k,\,\mathrm{applied}} - v_k^*\right) \end{aligned}$$

with $k \in \{A, B\}$, the integrator state integ_k and the back-EMF feed-forward $v_{\mathrm{ff},\,k}$; the K_b term acts only while the voltage is limited.

Gain	Formula	17HS19-2004S1, 1 kHz
K_p	$L\,\omega_c$	18.85 V/A

Gain	Formula	17HS19-2004S1, 1 kHz
K_i	$R\,\omega_c$	10 053 V/(A·s)
K_b (back-calculation)	$T_s R/L$ (tracking time L/R)	0.0267 per period

Pole-zero cancellation. The PI zero $K_i/K_p = R/L$ cancels the pole of the R-L phase, so the closed current loop is first order with the bandwidth $\omega_c = 2\pi \cdot$ `current_bw_hz` (1 kHz by default; `current_bw_hz` must not exceed `fast_loop_hz / 4`). The computation delay of one PWM period (the voltage computed now is applied in the next period) adds a small overshoot: the measured step 0 → 2 A has 1.12 % overshoot, 0.35 ms rise time (90 %) and 0.40 ms settling time (2 %) with both topologies (test T04).

Anti-windup. When the power stage cannot produce the requested voltage, the difference between the applied and the requested voltage pulls the integrator back with the gain K_b (back-calculation). The integrator does not wind up during acceleration into the voltage limit, and the current recovers without overshoot when the limit ends.

Back-EMF feed-forward. From the commanded speed (the position change of the slow period), the slow loop computes the back-EMF amplitude $K_t\,\omega$ and the fast loop adds

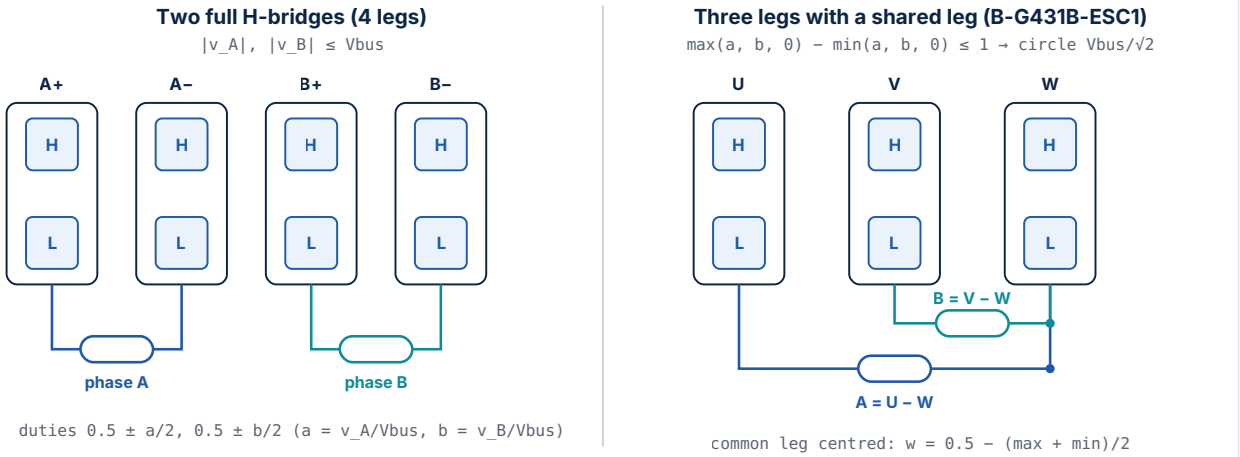
$$v_{\mathrm{ff},\,A} = -K_t\,\omega \sin\varphi, \qquad v_{\mathrm{ff},\,B} = K_t\,\omega \cos\varphi$$

to the controller outputs. The PI then only corrects the residual. With `ke_uv_s_per_rad = 0` the feed-forward is off.

Voltage drive. With `drive_mode = SSTEP_DRIVE_VOLTAGE` there is no current measurement: $v_k^* = R\,i_k^* + v_{\mathrm{ff},\,k}$. Torque then drops with speed (the inductance is not compensated); use it only for boards without current sensing.

> **Known limitation: phase lag at speed**
>
> The loop follows a rotating reference with a lag that grows with the electrical frequency (about 4° at 1 rev/s; at 4 rev/s the current vector error is 0.41 A rms). The amplitude stays close to the reference up to the voltage limit (1.97 A at 4 rev/s), but the lag shifts the torque angle. A delay compensation (angle advance by one period) is planned; an inductive feed-forward $L\,di/dt$ was tried in the prototype and made the result worse.

5.5 Voltage limit and leg mapping



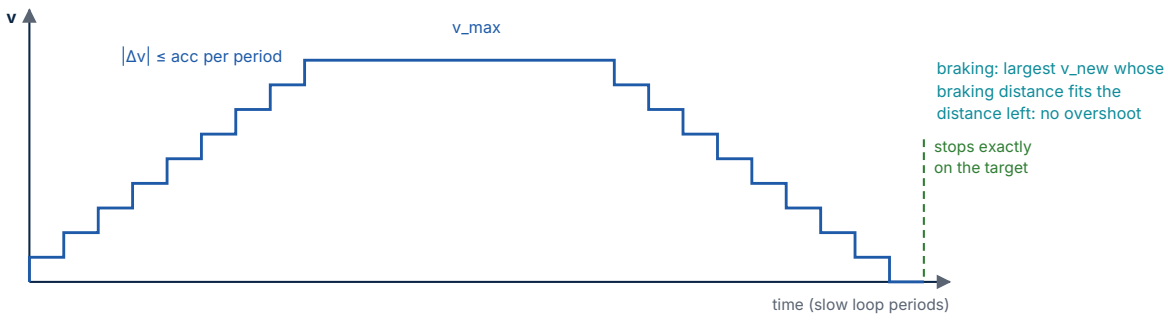
The requested phase voltages `(v_A, v_B)` are divided by the measured bus voltage (`a = v_A / Vbus`, `b = v_B / Vbus`) and mapped to leg duty cycles:

Topology	Feasible when	Leg duties	Largest circular amplitude
Two H-bridges, legs `[A+ A− B+ B−]`	`max(|a|, |b|) ≤ 1`	`0.5 + a/2`, `0.5 − a/2`, `0.5 + b/2`, `0.5 − b/2`	`Vbus`
Shared leg, legs `[U V W]`, A = U − W, B = V − W	`max(a, b, 0) − min(a, b, 0) ≤ 1`	`a + w`, `b + w`, `w` with `w = 0.5 − (max + min)/2`	`Vbus / √2`

When the request is not feasible, **both** components are scaled by the same factor: the voltage vector keeps its direction, so the current — and the torque — keeps its angle; only its magnitude is limited. The limited flag switches the anti-windup on.

Implementation details: the request is first halved (exact power-of-two pre-scale) until it is below 4 pu, then multiplied by `V_base / Vbus` (computed once per slow period from the bus voltage ADC), and a division happens only when the request is limited. The common leg of the shared topology is centred between the highest and the lowest leg voltage, which gives the largest circular amplitude. Duties become compare values `0 … pwm_period_ticks` (0 = low side on).

5.6 Motion generator



The motion generator runs in the slow loop and uses **integers only** in both numeric backends:

Quantity	Unit
Position `pos`, target	1/65536 microstep (`int64_t`)
Velocity `vel`, `v_max`	1/65536 microstep per slow period (`int32_t`, below 2^{30})
Acceleration `acc`, deceleration `dec`	1/65536 microstep per period²

Every period:

1. **Target velocity.** In RUN mode, the requested velocity limited to `v_max`. In MOVE mode, the largest velocity `v_new` for which the distance of this period $(v + v_{\text{new}})/2$ plus the braking distance $v_{\text{new}}^2/(2\,\text{dec})$ still fits into the distance left `rem`:

$$v_{\text{new}} = \frac{-d + \sqrt{d^2 + 8\,d\,|r| - 4\,d\,|v|}}{2}$$

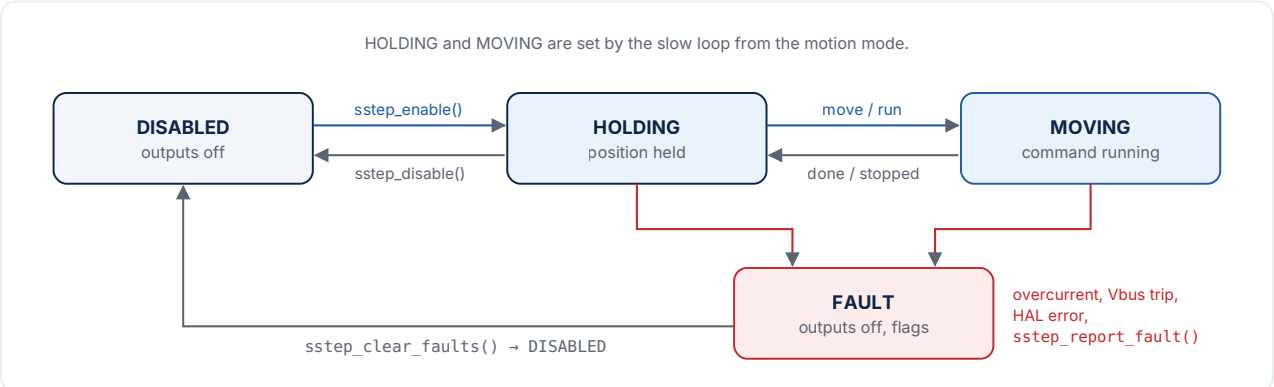
with d = `dec`, r = `rem`

(integer square root), or `v_max` when the distance left is larger than the braking distance from `v_max` plus one period. 2. **Rate limit.** The velocity moves towards the target by at most `acc` when speeding up (larger magnitude, same direction) and at most `dec` otherwise. 3. **Integration.** `pos += floor((v + v_new) / 2)` (trapezoidal rule). 4. **End of a move.** When the position crosses the target, or comes within 64/65536 microstep of it with `|v_new| ≤ dec`, the position snaps to the target, the velocity to zero, the mode to HOLD and `SSTEP_EVENT_MOVE_DONE` is reported.

Properties, all covered by unit tests: no overshoot; the move ends exactly on the target; the velocity never changes by more than `acc` or `dec` in a period, except in the last period of a move where at most `2·dec` (a few microsteps per second) drops to zero; a new target while moving changes the profile smoothly (decelerating with `dec` if it lies behind). A move of 6400 microsteps with 16 000 µsteps/s and 160 000 µsteps/s² takes 499 periods (T05). The Scilab and Python implementations are bit-exact copies.

The electrical angle increment for the fast loop is `Δ = trunc(Δpos · 2^(14 − log₂ M) / ratio)`, re-synchronised at every hand-off.

5.7 States and faults



State	Outputs	Entered by
`SSTEP_STATE_DISABLED`	off (all legs high impedance)	`sstep_init()`, `sstep_disable()`, `sstep_clear_faults()`
`SSTEP_STATE_HOLDING`	on, position held	`sstep_enable()`; slow loop when a command ends
`SSTEP_STATE_MOVING`	on, command running	slow loop while a move, run or stop executes
`SSTEP_STATE_FAULT`	off	a trip in the fast loop, a callback error, `sstep_report_fault()`

Fault flag	Raised when	Where
`SSTEP_FAULT_OVERCURRENT` (0x0001)	`|i_a|` or `|i_b|` above `i_trip_ma` (current drive only)	fast loop, same period
`SSTEP_FAULT_OVERVOLTAGE` (0x0002)	bus voltage above `vbus_max_mv`	fast loop
`SSTEP_FAULT_UNDERVOLTAGE` (0x0004)	bus voltage below `vbus_min_mv`	fast loop
`SSTEP_FAULT_HAL` (0x0008)	`pwm_write()` or `output_enable()` returned an error	any
`SSTEP_FAULT_EXTERNAL` (0x0010)	the application reported a fault	`sstep_report_fault()`

On a fault the outputs are switched off immediately (`output_enable(false)`), the flags are stored and `on_fault()` is called from the context that detected it — possibly the interrupt. Faults stay active until `sstep_clear_faults()`; the driver then returns to DISABLED and must be enabled again.

CHAPTER 6

Stepper-C Library

Stepper-C (github.com/AntonVynohradov/Stepper-C, local folder `sfs-stepper-ctrl`) is the control library of the system: C11, no heap, no RTOS, no global state, two numeric backends and two power stage topologies. This chapter explains how to add it to a firmware project, configure it for a motor and a board, call its loops from the interrupts and check that it works. The functions are listed in the API Reference (p. 34); the algorithms are explained in Control Algorithms (p. 21).

6.1 Repository layout

Path	Content
`sstep.h`	umbrella header: include only this
`inc/sstep_config.h`	build-time configuration macros
`inc/sstep_defs.h`	status codes, topologies, fault flags, states
`inc/sstep_math.h`	numeric backend: types, saturating inline operations
`inc/sstep_ctrl.h`	the API: configuration, callbacks, loops, commands
`src/sstep_ctrl.c`	initialisation, loops, commands
`src/sstep_bridge.c`	voltage limit and leg mapping
`src/sstep_motion.c`	motion generator
`src/sstep_math.c`, `src/sstep_sin_table.c`	sine/cosine, gain conversions (table generated by `tools/sstep_gen_sin_table.py`)
`sstep.cmake`, `CMakeLists.txt`, `CMakePresets.json`	build
`tests/unit/`	GoogleTest suites with a motor model (`sstep_test_plant.hpp`)
`tests/compile/sstep_api_check.c`	compile check of the API with the reference configuration

6.2 Adding the library to a project

With CMake (the library is a subdirectory or a submodule):

```
set(SSTEP_MATH_BACKEND "FLOAT" CACHE STRING "" FORCE)   # STM32G4 has an FPU
add_subdirectory(ext/sfs-stepper-ctrl)
target_link_libraries(firmware PRIVATE sstep)
```

CMake variable	Values	Effect
`SSTEP_MATH_BACKEND`	`FIXED` (default), `FLOAT`	defines `SSTEP_CFG_MATH_BACKEND`

CMake variable	Values	Effect
`SSTEP_CONFIG_DIR`	folder	adds the folder to the include path and defines `SSTEP_USER_CONFIG`, so `sstep_user_config.h` is used
`SSTEP_UNIT_TESTS`, `SSTEP_INTEGRATION_TESTS`, `SSTEP_EXAMPLES`	ON/OFF	test and example targets

Without CMake: compile the five files of `src/`, add the repository root and `inc/` to the include path, and define `SSTEP_CFG_MATH_BACKEND=SSTEP_MATH_BACKEND_FLOAT` (or leave it for fixed point). The library needs only `<stdint.h>`, `<stdbool.h>` and `<stddef.h>`.

6.3 Build-time configuration

Macro	Default	Range	Effect
`SSTEP_CFG_MATH_BACKEND`	`SSTEP_MATH_BACKEND_FIXED`	`…_FIXED`, `…_FLOAT`	Q15/Q31 integer arithmetic or single-precision float
`SSTEP_CFG_TOPOLOGY_FULL_BRIDGE`	1	0, 1	compile support for two H-bridges
`SSTEP_CFG_TOPOLOGY_SHARED_LEG`	1	0, 1	compile support for three legs with a shared leg
`SSTEP_CFG_MICROSTEPS_MAX`	256	power of two 1 … 256	largest accepted `microsteps`
`SSTEP_CFG_SINE_TABLE_BITS`	8	6 … 10	quarter-wave table size 2^bits + 1 (flash)
`SSTEP_CFG_ARG_CHECKS`	1	0, 1	0 removes the pointer checks from `sstep_fast_loop()` only

Every macro is range-checked with `#error`. Set them on the compiler command line or in `sstep_user_config.h`:

```
/* sstep_user_config.h: Stepper-C for an STM32G431 driving one ESC1 */
#define SSTEP_CFG_MATH_BACKEND SSTEP_MATH_BACKEND_FLOAT
#define SSTEP_CFG_TOPOLOGY_FULL_BRIDGE 0     /* only the shared leg is needed */
#define SSTEP_CFG_MICROSTEPS_MAX 64U
```

Which backend? Use `FIXED` on cores without a hardware FPU (Cortex-M0/M0+/M3, RISC-V RV32IMC); use `FLOAT` where single-precision float is in hardware (Cortex-M4F/M7/M33F, STM32G4). Both give the same control behaviour within the resolution of Q15 (the closed loop tests pass with the same limits in both).

6.4 Configuring a driver

The run-time configuration is one structure in integer engineering units, copied by `sstep_init()`. This is the reference configuration of the 17HS19-2004S1 on the B-G431B-ESC1 — the same values as in the simulation:

```c
#include "sstep.h"

extern int board_pwm_write(void* ctx, const uint16_t compare[SSTEP_LEGS_MAX]);
extern int board_output_enable(void* ctx, bool enable);

const struct sstep_config motor_cfg = {
    .motor = {
        .r_mohm = 1600U,             /* 1.60 Ohm */
        .l_uh = 3000U,               /* 3.0 mH */
        .i_rated_ma = 2000U,
        .ke_uv_s_per_rad = 208600U,  /* Kt = 0.2086 V*s/rad: back-EMF feed-forward */
        .full_steps_per_rev = 200U,
    },
    .stage = {
        .topology = SSTEP_TOPOLOGY_SHARED_LEG,
        .vbus_nominal_mv = 24000U,
        .vbus_min_mv = 10000U,
        .vbus_max_mv = 30000U,
        .i_full_scale_ma = 4000U,    /* ADC full scale (placeholder: from the schematic)
*/
        .i_trip_ma = 3500U,
        .pwm_period_ticks = 4250U,   /* 170 MHz, centre-aligned, 20 kHz */
        .pwm_freq_hz = 20000U,
        .dead_time_ns = 133U,        /* effective dead time from ngspice */
        .adc_i_zero = 2048U,
        .adc_i_span = 2047,
        .adc_vbus_mv_full = 34320U,
        .adc_full = 4095U,
    },
    .ctrl = {
        .drive_mode = SSTEP_DRIVE_CURRENT,
        .fast_loop_hz = 20000U,
        .slow_loop_hz = 1000U,
        .microsteps = 16U,
        .current_bw_hz = 1000U,
        .i_run_ma = 2000U,
        .i_hold_ma = 1000U,
        .hold_delay_ms = 100U,
        .v_max = 16000U,             /* 5 rev/s */
        .accel = 160000U,            /* 50 rev/s^2 */
        .decel = 0U,                 /* same as accel */
    },
    .hal = {
        .context = NULL,
        .pwm_write = board_pwm_write,
        .output_enable = board_output_enable,
        .on_fault = NULL,
        .on_event = NULL,
    },
};
```

How to get each value:

Field	How to obtain it
`r_mohm`, `l_uh`	datasheet, better measured (R with 4-wire, L at 1 kHz)
`ke_uv_s_per_rad`	back-EMF amplitude / speed of the spun motor (Motor Model); 0 disables the feed-forward
`vbus_*`	supply; `vbus_min_mv` < nominal < `vbus_max_mv` or 0 to disable a trip
`i_full_scale_ma`, `adc_i_zero`, `adc_i_span`	from the shunt, the amplifier gain and offset: `span = I_full · R_shunt · G / V_ref · adc_full`; negative `adc_i_span` for an inverting amplifier; measure the zero with the outputs off

Field	How to obtain it
`adc_vbus_mv_full`, `adc_full`	bus voltage divider: the bus voltage that gives the largest ADC reading
`pwm_period_ticks`	timer auto-reload of the centre-aligned PWM (compare for 100 %)
`dead_time_ns`	effective dead time (Power Stage Model); stored for a future compensation
`current_bw_hz`	1 kHz is a good start at 20 kHz; at most `fast_loop_hz / 4`
`v_max`, `accel`	microsteps per second (and per second²); with M = 16 one revolution is 3200 microsteps

> **Current measurement resolution**
>
> The simulation assumes a 4 A full scale (1.95 mA per ADC count). The current sense amplifiers of motor control boards are often designed for tens of amperes; compute the counts per ampere from the schematic before choosing the run current — a resolution coarser than about 1 % of the run current makes the current loop noisy.

`sstep_init()` checks every field and returns `SSTEP_ERR_PARAM` for a value out of range or a missing required callback, `SSTEP_ERR_CONFIG` for inconsistent values (for example `i_run_ma` above `i_trip_ma` or `i_rated_ma`, `fast_loop_hz` not a multiple of `slow_loop_hz`, `current_bw_hz` above `fast_loop_hz / 4`, a bus voltage trip on the wrong side of the nominal value) and `SSTEP_ERR_UNSUPPORTED` for a topology that is not compiled in.

6.5 Connecting the loops to the hardware

The library expects:

1. A **centre-aligned PWM** with one compare channel per leg. Compare 0 means the low side is on for the whole period, `pwm_period_ticks` the high side. Entries above the leg count of the topology are 0.
2. **Current sampling at the counter peak**, the middle of the period where all low-side switches conduct: with low-side shunts the phase current flows through the shunt only then, and the sample equals the period average of the ripple.
3. **`sstep_fast_loop()` once per PWM period**, right after the ADC conversion, with the raw counts of phase A, phase B and the bus voltage. The computed compare values are written by `pwm_write()` and take effect at the next timer update — the one-period delay the controller is designed for.
4. **`sstep_slow_loop()` at `slow_loop_hz`**, from a timer task or the main loop, never concurrently with the commands (they run in the same context).

```c
#include "sstep.h"

/* Board layer (your HAL): timer and ADC are configured elsewhere */
extern void timer_set_compare(uint32_t channel, uint16_t value);
extern void gate_driver_enable(bool enable);
extern uint16_t adc_read_injected(uint32_t rank);

static struct sstep drv;

static int pwm_write(void* ctx, const uint16_t compare[SSTEP_LEGS_MAX])
{
    (void)ctx;
```

```
    timer_set_compare(1U, compare[0]);   /* leg U: phase A+ */
    timer_set_compare(2U, compare[1]);   /* leg V: phase B+ */
    timer_set_compare(3U, compare[2]);   /* leg W: common   */
    return 0;
}

static int output_enable(void* ctx, bool enable)
{
    (void)ctx;
    gate_driver_enable(enable);          /* also: timer main output enable */
    return 0;
}

/* ADC end-of-conversion interrupt, triggered at the counter peak (20 kHz) */
void adc_injected_isr(void)
{
    const struct sstep_sample s = {
        .i_a = adc_read_injected(1U),
        .i_b = adc_read_injected(2U),
        .vbus = adc_read_injected(3U),
    };
    (void)sstep_fast_loop(&drv, &s);
}

/* 1 kHz timer task */
void control_task_1khz(void)
{
    (void)sstep_slow_loop(&drv);
}

int motor_start(struct sstep_config* cfg)
{
    cfg->hal.pwm_write = pwm_write;
    cfg->hal.output_enable = output_enable;
    int r = sstep_init(&drv, cfg);
    if (r == SSTEP_OK)
    {
        r = sstep_enable(&drv);
    }
    return r;
}
```

On the ESC1 the three legs are driven by the advanced timer TIM1 (one complementary channel per leg); phase A of the motor is connected between U and W, phase B between V and W. The exact channel assignment, the dead time register and the ADC trigger are documented with the port (P3).

6.6 Commanding motion

```
#include "sstep.h"

static void on_event(void* ctx, enum sstep_event ev)
{
    (void)ctx;
    if (ev == SSTEP_EVENT_MOVE_DONE)
    {
        /* target reached: start the next move, signal the host... */
    }
}

void demo(struct sstep* m)
{
    struct sstep_status st;

    (void)sstep_set_position(m, 0);              /* homing: here is zero */
    (void)sstep_move_to(m, 6400);                /* two revolutions forward (M = 16) */
    (void)sstep_set_motion_limits(m, 32000U, 160000U, 0U);  /* from the next slow period
*/
```

```
    (void)sstep_run(m, -12800);                 /* turn backwards at 4 rev/s, smoothly */
    (void)sstep_stop(m);                        /* decelerate and hold */
    (void)sstep_get_status(m, &st);             /* state, position, currents, Vbus */
    (void)on_event;
}
```

Commands are accepted in HOLDING and MOVING; they return `SSTEP_ERR_STATE` when the driver is disabled or faulted. A new command replaces the running one without a velocity step. `sstep_set_position()` is only allowed at standstill and keeps the rotor where it is (only the numbering changes).

6.7 Memory, timing and reentrancy

Item	Value
RAM	one `struct sstep` per motor (configuration copy, gains, two setpoint buffers, controller and motion state); allocate it statically
Flash	8.2 kB Cortex-M4F fixed point, 9.0 kB float, 8.5 kB Cortex-M0+ (clang `-Os`, both topologies, table included)
Fast loop work	fixed: one sine/cosine, two PI, one leg mapping, at most one division; no loops over data except the bounded power-of-two pre-scale
Fast loop time on the MCU	not measured yet (P3)
Reentrancy	instances are independent; one instance must not be used from two contexts except the documented split (fast loop in the interrupt, everything else in one task)
Fault callback	may be called from the interrupt

6.8 Quality

Check	Result
Unit tests (GoogleTest)	76 tests fixed point, 75 float: math (29), bridge (5), motion (10), API (24), closed loop with a motor model (8)
Coverage	100 % lines and branches of `src/` in both backends (gcov)
Sanitizers	clean under AddressSanitizer and UndefinedBehaviorSanitizer
Against the prototype	current step overshoot 1.11 % (prototype 1.12 %), settling 0.40 ms; 8 rev/s current 1.862 A / 1.557 A (prototype identical); load angle at 0.25 N·m 37.21° (prototype 37.21°)
Motion generator	bit-exact with the Scilab/Python twin over 3 052 885 periods
MISRA C:2012	followed in the code (no dynamic memory, single exit, no recursion, explicit casts, every `if` with `else`); the Cppcheck MISRA target `misra-analyze` is configured — a full report is pending

6.9 Porting checklist

1. Measure R, L and the back-EMF constant of the motor; enter them.

2. Compute the ADC scaling of the currents and of the bus voltage from the schematic; measure the current zero with the outputs disabled.
3. Configure a centre-aligned PWM at `pwm_freq_hz` with the timer dead time; trigger the current ADC at the counter peak.
4. Call `sstep_fast_loop()` from the ADC interrupt, `sstep_slow_loop()` at 1 kHz.
5. Start with a current-limited supply, `i_run_ma` at half the rated current and a low `v_max`; check the phase currents with a current probe.
6. Run the same configuration in the SIL diagram (p. 47) and compare.

CHAPTER 7

API Reference

The public API of Stepper-C 0.1.0 is declared in `inc/sstep_ctrl.h` and included through `sstep.h`. All functions return an `int` status code, never block and never allocate. Types and members marked *private* may change in any release.

7.1 Headers

Header	Content
`sstep.h`	umbrella header and version (`SSTEP_VERSION`, fallback `"0.1.0"`)
`inc/sstep_config.h`	build-time macros (see Build-time configuration)
`inc/sstep_defs.h`	status codes, `enum sstep_topology`, fault flags, `enum sstep_state`
`inc/sstep_math.h`	numeric types (`sstep_real_t`, `sstep_acc_t`, `sstep_angle_t`, `struct sstep_gain`)
`inc/sstep_ctrl.h`	configuration, callbacks, loops, commands

7.2 Status codes

Code	Value	Meaning
`SSTEP_OK`	0	success
`SSTEP_ERR_PARAM`	−1	NULL pointer, missing required callback, value out of range
`SSTEP_ERR_CONFIG`	−2	values that are valid alone but inconsistent together
`SSTEP_ERR_STATE`	−3	command not allowed in the current state (disabled, faulted, moving)
`SSTEP_ERR_UNSUPPORTED`	−4	topology not compiled in

7.3 Enumerations and flags

Name	Values
`enum sstep_topology`	`SSTEP_TOPOLOGY_FULL_BRIDGE` = 0 (4 legs), `SSTEP_TOPOLOGY_SHARED_LEG` = 1 (3 legs); `SSTEP_LEGS_MAX` = 4
`enum sstep_drive_mode`	`SSTEP_DRIVE_CURRENT` = 0 (PI current control), `SSTEP_DRIVE_VOLTAGE` = 1 (open loop `R·I + ff`)
`enum sstep_state`	`SSTEP_STATE_DISABLED` = 0, `SSTEP_STATE_HOLDING` = 1, `SSTEP_STATE_MOVING` = 2, `SSTEP_STATE_FAULT` = 3
`enum sstep_event`	`SSTEP_EVENT_MOVE_DONE` = 0, `SSTEP_EVENT_HOLD_CURRENT` = 1
Fault flags (`uint16_t`)	`SSTEP_FAULT_NONE` 0x0000, `…_OVERCURRENT` 0x0001, `…_OVERVOLTAGE` 0x0002, `…_UNDERVOLTAGE` 0x0004, `…_HAL` 0x0008, `…_EXTERNAL` 0x0010

7.4 Callbacks

```c
typedef int  (*sstep_pwm_write_fn)(void* context, const uint16_t compare[SSTEP_LEGS_MAX]);
typedef int  (*sstep_output_enable_fn)(void* context, bool enable);
typedef void (*sstep_fault_fn)(void* context, uint16_t faults);
typedef void (*sstep_event_fn)(void* context, enum sstep_event event);
```

Callback	Required	Called from	Contract
`pwm_write`	yes	fast loop (interrupt)	write one compare value per leg (0 … `pwm_period_ticks`; 0 = low side on); the array is valid only during the call; return < 0 on error (raises `SSTEP_FAULT_HAL`)
`output_enable`	yes	enable, disable, any fault	`true` enables the gate drivers, `false` switches every leg off (high impedance); return < 0 on error
`on_fault`	no	the context that detected the fault (may be the interrupt)	newly raised flags
`on_event`	no	slow loop	move done, hold current applied

7.5 Configuration structures

`struct sstep_motor_cfg`

Member	Type	Unit	Range	Meaning
`r_mohm`	`uint32_t`	mΩ	1 … 10^7	phase resistance at 25 °C
`l_uh`	`uint32_t`	µH	1 … 10^7	phase inductance
`i_rated_ma`	`uint32_t`	mA	1 … 10^7	rated phase current (upper limit of `i_run_ma`)
`ke_uv_s_per_rad`	`uint32_t`	µV·s/rad	0 … 10^7	back-EMF constant = `K_t` in µN·m/A; 0 disables the feed-forward
`full_steps_per_rev`	`uint16_t`	—	≥ 1	200 for 1.8°

`struct sstep_stage_cfg`

Member	Type	Unit	Meaning
`topology`	`enum sstep_topology`	—	power stage topology
`vbus_nominal_mv`	`uint32_t`	mV	nominal bus voltage, base of the per-unit voltage
`vbus_min_mv`	`uint32_t`	mV	undervoltage trip, below nominal; 0 = disabled
`vbus_max_mv`	`uint32_t`	mV	overvoltage trip, above nominal; 0 = disabled
`i_full_scale_ma`	`uint32_t`	mA	phase current at ADC full scale, base of the per-unit current
`i_trip_ma`	`uint32_t`	mA	overcurrent trip, ≤ `i_full_scale_ma`
`pwm_period_ticks`	`uint16_t`	ticks	compare value for 100 % duty (≥ 2)

Member	Type	Unit	Meaning
`pwm_freq_hz`	`uint32_t`	Hz	PWM frequency
`dead_time_ns`	`uint32_t`	ns	dead time for compensation (stored; compensation planned)
`adc_i_zero`	`uint16_t`	counts	current ADC reading at 0 A (≤ `adc_full`)
`adc_i_span`	`int32_t`	counts	counts from 0 A to + `i_full_scale_ma`; negative for an inverting amplifier; ±1 … ±65535
`adc_vbus_mv_full`	`uint32_t`	mV	bus voltage at ADC reading `adc_full`
`adc_full`	`uint16_t`	counts	largest ADC reading (4095 for 12 bits)

`struct sstep_ctrl_cfg`

Member	Type	Unit	Meaning
`drive_mode`	`enum sstep_drive_mode`	—	current or voltage drive
`fast_loop_hz`	`uint32_t`	Hz	rate of `sstep_fast_loop()`; integer multiple of `slow_loop_hz`, ratio ≤ 65535
`slow_loop_hz`	`uint32_t`	Hz	rate of `sstep_slow_loop()`
`microsteps`	`uint16_t`	—	microsteps per full step, power of two ≤ `SSTEP_CFG_MICROSTEPS_MAX`
`current_bw_hz`	`uint32_t`	Hz	current loop bandwidth, ≤ `fast_loop_hz / 4`
`i_run_ma`	`uint32_t`	mA	peak phase current while moving, ≤ `i_trip_ma` and ≤ `i_rated_ma`
`i_hold_ma`	`uint32_t`	mA	peak phase current at standstill, ≤ `i_run_ma`
`hold_delay_ms`	`uint32_t`	ms	standstill time before the hold current
`v_max`	`uint32_t`	µsteps/s	maximum velocity
`accel`	`uint32_t`	µsteps/s²	acceleration
`decel`	`uint32_t`	µsteps/s²	deceleration; 0 = same as `accel`

`struct sstep_config`, `struct sstep_callbacks`

`struct sstep_config` groups `motor`, `stage`, `ctrl` and `hal` (`struct sstep_callbacks`: `context`, `pwm_write`, `output_enable`, `on_fault`, `on_event`). It is copied by `sstep_init()` and may be discarded afterwards.

`struct sstep_sample`

Member	Type	Meaning
`i_a`	`uint16_t`	phase A current, ADC counts (ignored in voltage drive)
`i_b`	`uint16_t`	phase B current, ADC counts (ignored in voltage drive)

Member	Type	Meaning
`vbus`	`uint16_t`	bus voltage, ADC counts

`struct sstep_status`

Member	Type	Unit
`state`	`enum sstep_state`	—
`faults`	`uint16_t`	active fault flags
`position`	`int32_t`	commanded position, µsteps
`target`	`int32_t`	target of the move, µsteps
`velocity`	`int32_t`	commanded velocity, µsteps/s
`i_a_ma`, `i_b_ma`	`int32_t`	last measured phase currents, mA
`vbus_mv`	`uint32_t`	last measured bus voltage, mV

`struct sstep` (private)

The driver instance, allocated by the application and initialised by `sstep_init()`. Its members (configuration copy, derived gains, double-buffered setpoint, PI integrators, motion state, faults, state) are private; use the functions only.

7.6 Functions

`sstep_init()`

```
int sstep_init(struct sstep* m, const struct sstep_config* cfg);
```

Validates the configuration, copies it, derives the per-unit gains (`Kp = L ωc`, `Ki = R ωc`, `Kb = Ts R / L`), the ADC scaling, the trip levels and the motion limits. No floating point is used with the fixed point backend. The outputs stay disabled.

Parameter	Meaning
`m`	driver instance
`cfg`	configuration; copied

Returns	When
`SSTEP_OK`	state `SSTEP_STATE_DISABLED`
`SSTEP_ERR_PARAM`	NULL pointer, missing `pwm_write` / `output_enable`, a field out of range
`SSTEP_ERR_CONFIG`	inconsistent fields (trip above full scale, run current above trip or rated, loop rates, bandwidth, bus thresholds, gains that do not fit)
`SSTEP_ERR_UNSUPPORTED`	topology not compiled in

`sstep_enable()`

```
int sstep_enable(struct sstep* m);
```

Enables the outputs and holds the current position with the run current (the hold current follows after `hold_delay_ms`). Resets the integrators. No effect when already enabled.

Returns	When
`SSTEP_OK`	state `SSTEP_STATE_HOLDING`
`SSTEP_ERR_PARAM`	`m` is NULL
`SSTEP_ERR_STATE`	faulted, or `output_enable()` failed (then `SSTEP_FAULT_HAL`)

`sstep_disable()`

```
int sstep_disable(struct sstep* m);
```

Stops immediately and switches the outputs off: the motor coasts without holding torque. A faulted driver stays faulted. Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

`sstep_fast_loop()`

```
int sstep_fast_loop(struct sstep* m, const struct sstep_sample* sample);
```

Call once per PWM period from the PWM/ADC interrupt. Converts the sample, checks the overcurrent and bus voltage trips (when the outputs are on), runs the current controllers, limits and maps the voltage and calls `pwm_write()`. Bounded execution time. With the outputs off it only stores the measurements.

Returns	When
`SSTEP_OK`	always, except:
`SSTEP_ERR_PARAM`	NULL pointer (only with `SSTEP_CFG_ARG_CHECKS` = 1)

`sstep_slow_loop()`

```
int sstep_slow_loop(struct sstep* m);
```

Call at `slow_loop_hz`. Advances the motion generator, selects run or hold current, updates the bus voltage scale, publishes the setpoint for the fast loop, sets the state (HOLDING/MOVING, or FAULT if the fast loop tripped meanwhile) and reports events. Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

`sstep_move_to()`, `sstep_move_by()`

```
int sstep_move_to(struct sstep* m, int32_t position);
int sstep_move_by(struct sstep* m, int32_t delta);
```

Start a move to an absolute position, or relative to the current *target*, with the trapezoidal profile. A running command is replaced smoothly (no velocity step).

Returns	When
`SSTEP_OK`	move started
`SSTEP_ERR_PARAM`	`m` is NULL; `move_by` target outside `int32_t`
`SSTEP_ERR_STATE`	disabled or faulted

`sstep_run()`

```
int sstep_run(struct sstep* m, int32_t velocity);
```

Run at a constant signed velocity [µsteps/s] until another command, accelerating with the configured limits. `|velocity| ≤ v_max`. Returns `SSTEP_OK`, `SSTEP_ERR_PARAM` (NULL or velocity above `v_max`) or `SSTEP_ERR_STATE` (disabled or faulted).

`sstep_stop()`

```
int sstep_stop(struct sstep* m);
```

Decelerates to standstill with the deceleration limit, then holds. No effect when nothing moves. Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

`sstep_set_position()`

```
int sstep_set_position(struct sstep* m, int32_t position);
```

Redefines the current position (homing). The rotor does not move: the electrical angle is kept and only the numbering changes. Returns `SSTEP_OK`, `SSTEP_ERR_PARAM` or `SSTEP_ERR_STATE` (a command is running).

`sstep_set_current()`

```
int sstep_set_current(struct sstep* m, uint32_t run_ma, uint32_t hold_ma);
```

Changes the run and hold currents; used from the next slow loop period.

Returns	When
`SSTEP_OK`	accepted
`SSTEP_ERR_PARAM`	NULL, `run_ma` = 0, a value above 10^7
`SSTEP_ERR_CONFIG`	`hold_ma > run_ma`, `run_ma` above `i_trip_ma` or `i_rated_ma`

`sstep_set_motion_limits()`

```
int sstep_set_motion_limits(struct sstep* m, uint32_t v_max, uint32_t accel, uint32_t
decel);
```

Changes the velocity [µsteps/s], acceleration and deceleration [µsteps/s²] limits (`decel` = 0: same as `accel`). The motion generator uses them from the next slow loop period, also for a running move. Returns `SSTEP_ERR_PARAM` for zero values or values that become zero or too large in the internal units (for example an acceleration below `slow_loop_hz² / 65536`).

`sstep_report_fault()`

```
int sstep_report_fault(struct sstep* m, uint16_t faults);
```

Raises application faults (a driver fault pin, over-temperature), typically `SSTEP_FAULT_EXTERNAL`: the outputs switch off and the state becomes FAULT. May be called from an interrupt. Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

`sstep_clear_faults()`

```
int sstep_clear_faults(struct sstep* m);
```

Clears all faults; the driver goes to DISABLED and must be enabled again. Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

`sstep_get_status()`

```
int sstep_get_status(const struct sstep* m, struct sstep_status* status);
```

Fills a snapshot in engineering units (state, faults, position, target, velocity, currents, bus voltage). Returns `SSTEP_OK` or `SSTEP_ERR_PARAM`.

7.7 Calling contexts

Function	Context
`sstep_fast_loop()`	PWM/ADC interrupt only
`sstep_report_fault()`	any (interrupt allowed)
all others	the slow loop context (one task), never concurrently with each other

CHAPTER 8

Simulation

sfs-stepper-sim (github.com/AntonVynohradov/sfs-stepper-sim) is the Scilab/Xcos environment in which every algorithm of Stepper-C is first developed and later verified: the motor model, a Scilab prototype of the control loops, an averaged and a switching power stage, the software-in-the-loop diagram and Python twins of everything. This chapter covers the Scilab side; the Xcos diagram has its own chapter, Software in the Loop (p. 47).

8.1 Repository layout

Folder / file	Content
`loader.sce`	loads every function (`getd` of `params`, `model`, `sil`, `xcos`, `xcos/blocks`) and the Xcos libraries
`params/`	parameter functions: motor, board, controller, motion, ngspice stage values
`model/`	motor equations, solvers, power stages, current loop, motion generator, references, SIL parameter builders
`sil/`	C sources of the Xcos blocks (`sil/c/sstep_xcos.c`), build script, library lookup
`xcos/`	diagram generator `sil_diagram.sci`, block interfaces `xcos/blocks/*.sci`
`start_sil.sce`	builds the library and opens the SIL diagram
`tests/`	`run_all.sce`, test scripts, helpers, reference data `tests/reference/*.csv`
`tools/`	Python twins `ref_model.py`, `ref_ctrl.py`, `ref_motion.py`, `ref_switching.py`, `ref_ext.py`
`demos/`	scripts with plots: open loop, switching stage

Start every session with

```
exec("loader.sce", -1);
```

8.2 Parameters

Parameters are Scilab structures returned by functions, so a scenario changes a field without editing a file.

Function	Content
`motor_17hs19_2004s1()`	`R`, `L`, `I_rated`, `Kt`, `Ns`, `Td`, `J`, `B`, `Tc`, `w_c` (see Motor Model)
`board_bg431b_esc1()`	`vbus` 24 V, `topology` `"shared"`, `fs` 20 kHz; trips and ADC scaling for the SIL; `stage` `"avg"` / `"switching"`, `dead_ns`, `rds_on`, `vd`, `sw_nstep`
`stage_esc1_spice()`	generated by ngspice: `rds_on`, `r_leg`, `vd`, `dead_timer_ns`, `dead_eff_ns`, `shunt_settle_ns`, `duty_max_sampling`

Function	Content
`ctrl_default()`	`bw_hz` 1000, `delay` 1 (computation delay, as on the MCU), `ff` 1 (back-EMF feed-forward), `nsub` 4 (RK4 steps per period)
`motion_default()`	`M` 16, `f_slow` 1000 Hz, `v_max` 16 000 µsteps/s, `acc` / `dec` 160 000 µsteps/s², `i_run` 2 A, `i_hold` 1 A, `hold_delay` 0.1 s

8.3 Open-loop motor simulation

`stepper_simulate(mot, src, srcp, t, x0, locked)` integrates the motor with lsoda for a voltage source `u = src(t, x, srcp)` returning `[v_A; v_B; T_L]`:

```
mot = motor_17hs19_2004s1();
srcp = struct("V", mot.R * mot.I_rated, "f_el", 5, "TL", 0);   // rotating voltage, 5 Hz el.
t = 0:1e-4:0.4;
x = stepper_simulate(mot, src_open_loop, srcp, t);              // x = [iA; iB; w; theta]
plot(t, x(1, :), t, x(2, :));
```

Sources: `src_dc` (constant voltages and load) and `src_open_loop` (rotating voltage vector). `demos/demo_open_loop.sce` plots currents, speed and load angle.

8.4 Closed-loop prototype

`sim_closed_loop` is the discrete prototype of the Stepper-C fast loop, run period by period exactly in the order of the MCU: sample the currents, compute the references and the controller, apply the voltage computed in the previous period (`c.delay = 1`) for one period, integrate the motor.

```
[rec, st, ext] = sim_closed_loop(mot, brd, c, ref, refp, T, locked, st)
```

Argument	Meaning
`mot`, `brd`, `c`	motor, board, controller; `c.mot` (optional) is the motor the controller is tuned for, if different from the plant
`ref`, `refp`, `st`	reference function `[r, st] = ref(t, refp, st)` returning `[I; φ; ω_ref; T_L]`, its parameters and state
`T`	simulated time [s]
`locked`	`%t`: rotor locked (current loop tests)
`rec`	one row per period: `t iA iB w θ iA_ref iB_ref vA vB k I φ` (`k` = power stage scale, 1 = not limited)
`ext`	one row per period: `iA_min iA_max iB_min iB_max vA_mean vB_mean` (switching stage: within the period)

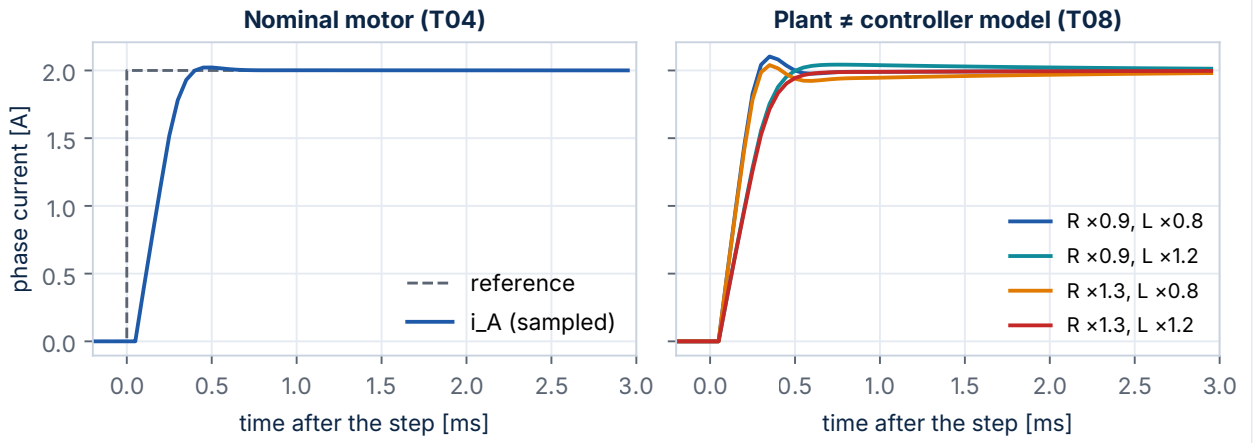
Reference functions:

Function	Reference
`ref_current_step`	constant angle, current step `I0 → I1` at `t_step`

Function	Reference
`ref_const_speed`	constant current, constant electrical frequency
`ref_speed_ramp`	constant current, speed ramp to `rev_s` in `t_ramp`, optional load `TL`
`ref_angle_step`	constant current, angle step `0 → φ1` (rotor natural frequency)
`ref_motion`	the motion generator: commands `[t kind value]` (see below)

Example — current step on a locked rotor, the test T04:

```
mot = motor_17hs19_2004s1(); brd = board_bg431b_esc1(); c = ctrl_default();
refp = struct("I0", 0, "I1", 2, "t_step", 1e-3, "phi", 0);
rec = sim_closed_loop(mot, brd, c, ref_current_step, refp, 6e-3, %t);
plot(rec(:, 1), rec(:, 2), rec(:, 1), rec(:, 6));     // iA and its reference
```

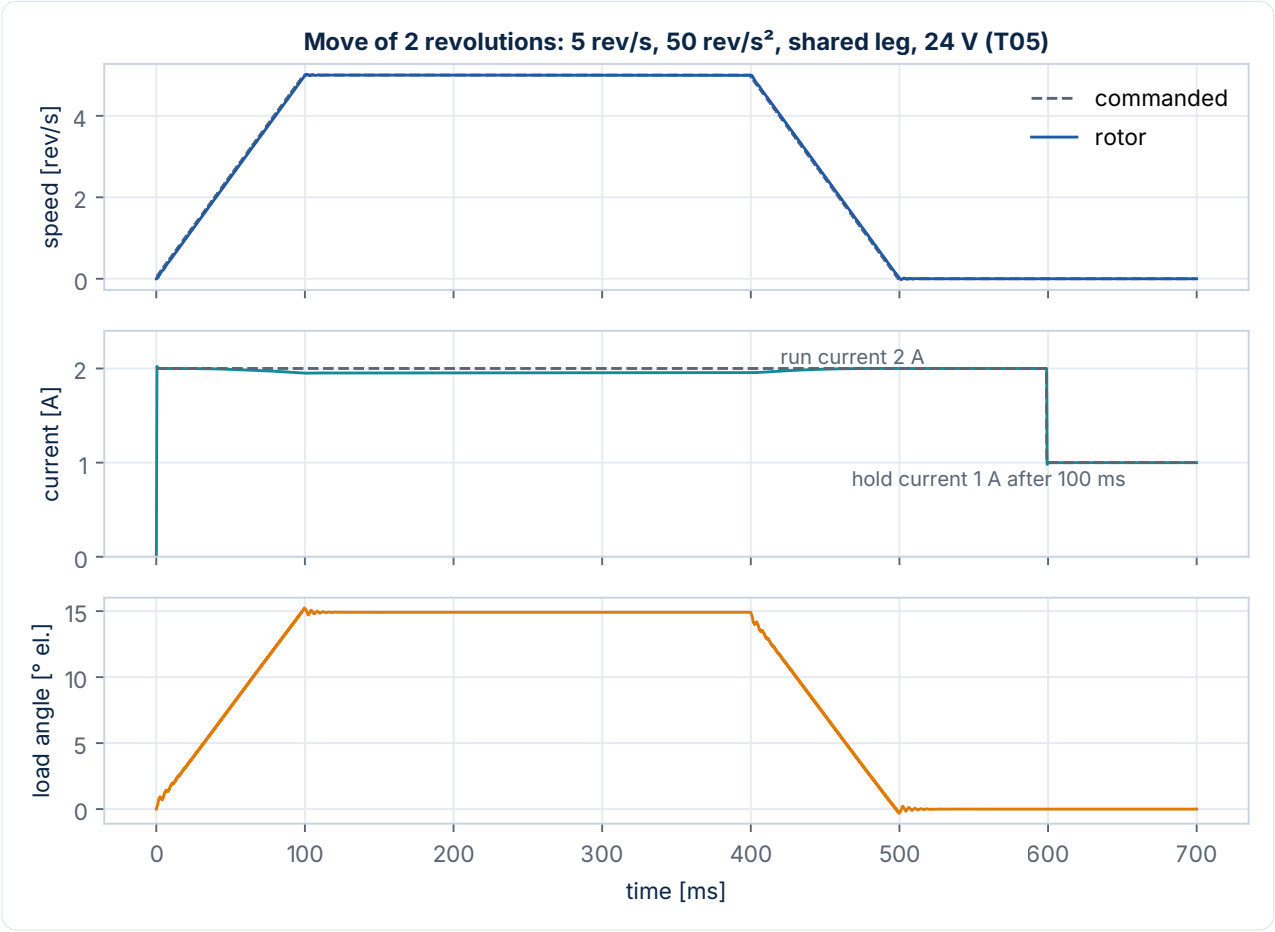


8.5 Motion generator twin

`motion_init`, `motion_slow` and `motion_step` reproduce the integer motion generator of Stepper-C bit for bit (`model/sim_int.sci` provides C-style integer division and square root). `ref_motion` turns it into a reference: it calls the slow loop every `ratio` fast periods and interpolates the angle exactly like the library.

Command `[t kind value]`	Meaning
kind 1 MOVE	move to `value` microsteps
kind 2 RUN	run at `value` microsteps/s
kind 3 STOP	decelerate to standstill
kind 4 LOAD	load torque `value` N·m from time `t` (simulation only)

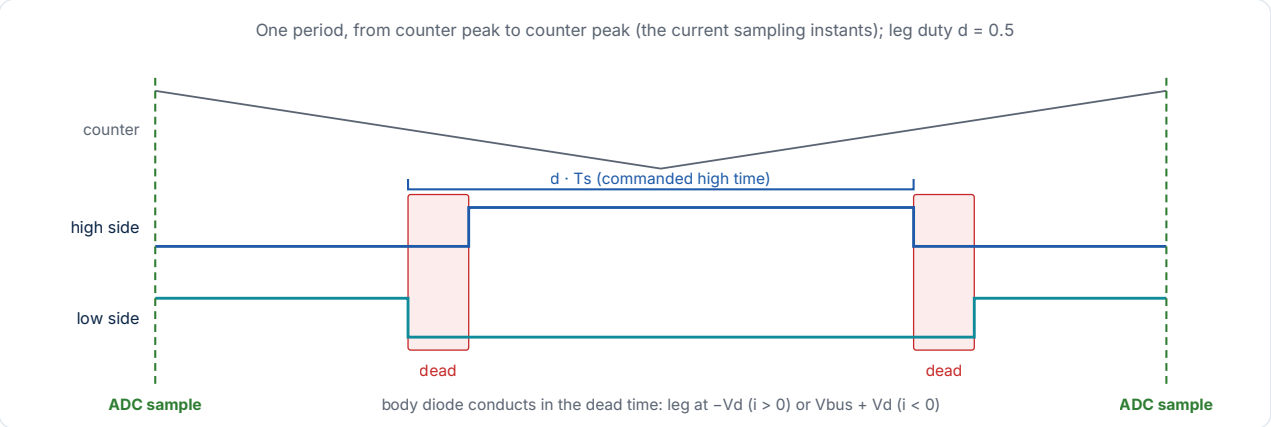
```
mp = motion_default();
st = motion_init(mot, brd, mp, [0, 1, 6400]);          // move 2 revolutions at t = 0
[rec, st] = sim_closed_loop(mot, brd, c, ref_motion, [], 0.6, %f, st);
// st.hist: [t pos v vel_q16 mode] per slow period
```



The figure shows the test T05: the rotor follows the commanded speed with a load angle of 15° at 50 rev/s² (inertia of the rotor only), the run current drops to the hold current 100 ms after the move, and the move ends exactly on 6400 microsteps after 499 ms.

8.6 Power stage models

`brd.stage`	Model	Use
`"avg"` (default)	each leg delivers `duty · Vbus` (`bridge_avg`)	fast; all control tests
`"switching"`	`bridge_switching.sci`: centre-aligned legs, turn-on delayed by the dead time, body diode in the dead time (`−Vd` or `Vbus + Vd` by current direction), `Rds(on)`; RK4 between the switching instants (`stepper_period_sw`)	ripple, peak current, dead time distortion; about 5× slower



The period starts at the counter peak (all low sides on, the sampling instant); the high side of a leg with duty `d` is commanded on in `[Ts/2 (1 − d), Ts/2 (1 + d)]`. A pulse shorter than the dead time disappears. With all outputs disabled every leg is in the dead state (only the diodes conduct).

With ideal switches the switching model gives the same sampled currents as the averaged model within 0.01 mA (test T10). The dead time changes the mean phase voltage by `−2 sign(i) (Vbus + 2 Vd) td / Ts`; the stage values come from ngspice (`stage_esc1_spice()`: effective dead time 133 ns, leg resistance 3.4 mΩ, diode 0.73 V).

```
brd.stage = "switching";
sp = struct("I", 2, "rev_s", 4, "t_ramp", 0.05, "Ns", mot.Ns);
[rec, st, ext] = sim_closed_loop(mot, brd, c, ref_speed_ramp, sp, 0.15);
peak = max(abs(ext(:, 1:4)));                    // largest current including the ripple
```

`demos/demo_switching.sce` plots the sampled current with its ripple band, the ripple per period and the dead time voltage error.

8.7 Python twins

Every Scilab algorithm has a Python twin in `tools/`. The twins are independent implementations used to cross-check the Scilab code (tests X01–X04), to prototype tests quickly and to generate the reference CSV files.

Twin	Covers	Run
`ref_model.py`	motor model (scipy Radau), T00–T02, X01 reference	`python tools/ref_model.py [--write]`
`ref_ctrl.py`	averaged stage, PI loop, B01, T03, T04, X02 reference	`python tools/ref_ctrl.py [--write]`
`ref_motion.py`	integer motion generator (bit-exact with C), T05–T07, T09, X03 reference	`python tools/ref_motion.py [--write]`
`ref_switching.py`	switching stage, T10, X04 reference	`python tools/ref_switching.py [--write]`
`ref_ext.py`	T08, T11, T12	`python tools/ref_ext.py`

`--write` regenerates the reference files in `tests/reference/`. Requirements: `numpy`, `scipy`.

8.8 Running the tests

```
exec("tests/run_all.sce", -1);
```

The runner executes every test script with error catching, prints `PASS` / `FAIL` lines with the measured values and a failure count, and exits with status 1 in batch mode (`scilab-cli -f tests/run_all.sce`) when a test fails. Test variables use the prefix `rt_` so that test scripts cannot overwrite them. The complete list with the pass criteria is in the Test Catalogue (p. 56).

8.9 Adapting the model to another motor or board

1. Copy `params/motor_17hs19_2004s1.sci` to a new function with the values of your motor (`R`, `L`, `I_rated`, `Kt`, `Ns`, `J`), or change the fields in a script.
2. Copy `params/board_bg431b_esc1.sci` for another board: bus voltage, topology (`"full"` or `"shared"`), PWM frequency and period, trip levels, ADC scaling.
3. Run T01–T04 first (electrical model and current loop), then the motion tests.
4. For the switching stage, set `dead_ns`, `rds_on`, `vd` from your power stage (or adapt the ngspice model and export).

CHAPTER 9

Software in the Loop

The software-in-the-loop (SIL) diagram runs the **real C code of Stepper-C** — the same sources that go into the firmware — against the motor model in Xcos. It is the place where a user without Scilab knowledge sees the library control a motor: open one file, press Start, watch currents, speed, load angle and position. It is also a test bench: S01–S04 compare it sample by sample with the Scilab prototype.

9.1 Start here

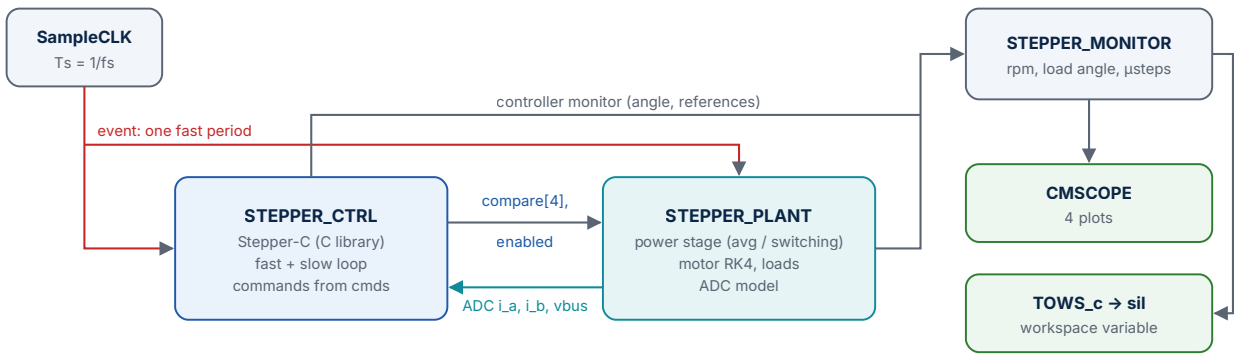
1. Install Scilab 2027 and, on Windows, Visual Studio 2022 or the free *Build Tools* with the workload *Desktop development with C++*.
2. Clone Stepper-C (github.com/AntonVynohradov/Stepper-C) into the folder `sfs-stepper-ctrl` next to `sfs-stepper-sim` (or set the environment variable `SSTEP_CTRL_DIR` to it).
3. Open **`start_sil.sce`** in Scilab and press **F5**.
4. Xcos opens `xcos/stepper_sil.zcos`. Press **Start** (▶).

The default scenario moves the motor two revolutions (6400 microsteps at 1/16) with 5 rev/s and 50 rev/s² on the B-G431B-ESC1 at 24 V. The scope shows four plots: the phase currents with their references, the speed in rpm, the load angle in electrical degrees (above 90° the motor is close to losing steps) and the commanded and real rotor positions in microsteps.

`start_sil.sce` has two settings at the top:

Variable	Values	Effect
`sil_backend`	`"FIXED"` (default), `"FLOAT"`	numeric backend of the compiled library; switching needs a Scilab restart
`sil_regenerate`	`%f` (default), `%t`	`%t` builds the diagram again from `xcos/sil_diagram.sci` (needed after a change of the block interfaces)

9.2 The diagram



Scenario (motor, board, commands, loads, T_end) in Simulation → Set Context; END_c stops the run at T_end.

Block	Computational function (C, type 4)	Inputs → outputs	Parameters
`SampleCLK`	Xcos	→ event every `Ts = 1/fs`	`Ts`
`STEPPER_CTRL`	`sstep_xcos_ctrl`	ADC `[i_a i_b vbus]` (counts) → `[compare(4) enabled]`, monitor `[φ position velocity state faults iA* iB*]`	`sil_ctrl_rpar(mot, brd, ctl, mp, cmds)`
`STEPPER_PLANT`	`sstep_xcos_plant`	`[compare(4) enabled]` → ADC sample, `[iA iB ω θ vA vB |iA|max |iB| max]`	`sil_plant_rpar(mot, brd, loads, locked, 4)`
`STEPPER_MONITOR`	`sstep_xcos_monitor`	plant + controller monitor → currents, rpm, load angle, positions, 12 values for the workspace	`[mot.Ns, mp.M]`
`CMSCOPE`	Xcos	4 plots	ranges from the context
`TOWS_c`	Xcos	→ workspace variable `sil`	buffer `N_log`
`END_c`	Xcos	stops at `T_end`	`T_end`

What the controller block does. At initialisation it builds a `struct sstep_config` from the parameter vector, calls `sstep_init()` (a configuration error stops the simulation with the library's status text) and `sstep_enable()`. On every clock event it applies the commands whose time has come, calls `sstep_slow_loop()` every `ratio` events, converts the ADC input to a `struct sstep_sample`, calls `sstep_fast_loop()` and outputs the compare values the library wrote through its `pwm_write` callback.

What the plant block does. It integrates the motor over one PWM period with the leg voltages of the *previous* period (the one-period computation delay of the MCU), using the averaged or the switching power stage (`brd.stage`), applies the load torque steps, and models the ADC: `counts = round(zero + i / i_full · span)`, clamped to `0 … adc_full`, and the bus voltage reading.

9.3 Scenario: the diagram context

The scenario is edited in **Simulation → Set Context** and evaluated before each run:

Variable	Meaning	Example
`mot`	motor parameters	`mot.Td = 0.02; mot.Tc = 0.01;` (detent, dry friction)
`brd`	board and power stage	`brd.topology = "full"; brd.stage = "switching"; brd.vbus = 36;`
`ctl`	current controller	`ctl.bw_hz = 500;`
`mp`	microsteps, speed and acceleration limits, run / hold current	`mp.v_max = 25600; mp.i_run = 1.5;`
`cmds`	commands `[t kind value]`: 1 MOVE to (µsteps), 2 RUN (µsteps/s), 3 STOP	`[0 2 12800; 0.15 2 -12800]`
`loads`	load torque steps `[t TL]` (N·m)	`[0 0; 0.15 0.25]`

Variable	Meaning	Example
`locked`	1: rotor locked (current loop test)	`1`
`T_end`	simulated time (s)	`1.0`
`scope_i`, `scope_rpm`, `scope_pos`	scope ranges	`3`, `400`, `7000`

Lines below `// derived values (do not edit)` compute `Ts` and the workspace buffer size. A command the library rejects (for example a RUN above `v_max`) stops the simulation with the command number, its time and the status text.

9.4 Results in the workspace

After a run the variable **`sil`** holds `sil.time` and `sil.values` with one row per PWM period:

Column	Quantity	Unit
1, 2	`iA`, `iB` (sampled)	A
3, 4	`iA*`, `iB*` (references)	A
5	speed	rpm
6	load angle	electrical °
7, 8	commanded position, rotor position	µsteps
9, 10	`vA`, `vB` (averaged voltage of the next period)	V
11, 12	largest `|iA|`, `|iB|` within the period (with the ripple when switching)	A

```
plot(sil.time, sil.values(:, 6));      // load angle
xtitle("Load angle", "t [s]", "deg el.");
```

9.5 Build and rebuild

`sil_build(backend, force)` copies the library sources and `sil/c/sstep_xcos.c` into `sil/build/<backend>/`, compiles them with the compiler Scilab finds (`tbx_build_src`; Visual Studio on Windows, gcc on Linux) with `-DSSTEP_CFG_MATH_BACKEND=…`, and links the shared library. The build happens once; it is repeated automatically when any source file differs from its copy in the build folder. `sil/build/` is ignored by Git.

The library is searched in this order: `SSTEP_CTRL_DIR`, the submodule `ext/sfs-stepper-ctrl`, the sibling folder `../sfs-stepper-ctrl`.

9.6 Agreement with the prototype

Test	Scenario	Criterion	Result
S01	current step 0 → 2 A, locked rotor	currents within 5 mA of the prototype	fix of an empty-command case applied; result of the re-run pending

Test	Scenario	Criterion	Result
S02	move 640 µsteps	currents < 5 mA, commanded angle identical, rotor < 0.05 µstep	1.05 mA, angle identical, rotor 0.0075 µstep
S03	8 rev/s, both topologies	current amplitude as the prototype (T09) ± 30 mA	1.862 A (two H-bridges), 1.557 A (shared leg), equal to the prototype
S04	run 4 rev/s with the switching stage	currents and peak current within 5 mA	new in this version; first run pending (the C plant equals the Python twin to $4{\cdot}10^{-16}$)

The remaining difference comes from the quantisation the prototype does not have: ADC counts, Q15 arithmetic and integer compare values.

9.7 Troubleshooting

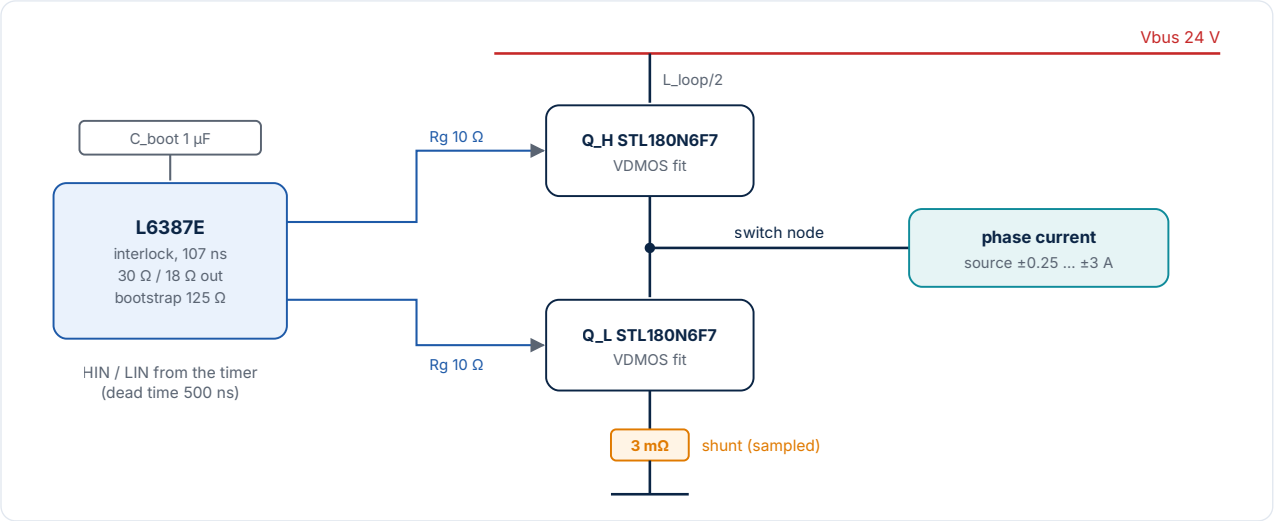
Symptom	Cause	Remedy
"No C compiler found by Scilab"	Visual Studio / Build Tools missing	install the C++ workload, restart Scilab
"Stepper-C sources not found"	library not next to the simulation	set `SSTEP_CTRL_DIR`
"wrong parameter vector (use sil_plant_rpar)"	diagram older than the blocks	set `sil_regenerate = %t` once, or delete `xcos/stepper_sil.zcos`
Changes in the C code have no effect	the old DLL is still loaded	restart Scilab (the build is redone automatically)
"The FIXED library is already linked ... Restart Scilab to switch to FLOAT"	backend changed in a running session	restart Scilab
"Stepper-C: sstep_init failed: ..."	configuration rejected by the library	the message names the status; check the context values
Link warnings when the diagram is generated	Xcos versions before the fix of the scope ports	regenerate the diagram with the current `sil_diagram.sci`

CHAPTER 10

Power Stage Model

sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice) models one half-bridge leg of the B-G431B-ESC1 in ngspice: the L6387E gate driver, two STL180N6F7 MOSFETs, gate resistors, the bootstrap supply and the low-side current shunt. It answers the questions an averaged model cannot: how long the transistors really take to switch, how much of the timer dead time is left, what the phase voltage loses in the dead time, and when the shunt voltage is valid for sampling. The results are exported to the simulation.

10.1 The leg circuit



File	Content
`models/stl180n6f7_sfs.lib`	STL180N6F7 as an ngspice VDMOS model, fitted by this project to the datasheet
`models/l6387_behav.lib`	L6387E behavioural model
`circuits/esc1_leg.lib`	subcircuit `ESC1_LEG`: driver, MOSFETs, gate resistors, bootstrap capacitor, shunt, layout inductances
`tools/run_tests.py`	test benches SP01–SP03, fit, export
`tools/plot_leg.py`	switch node waveforms (`docs/leg_edges.png`)

The test bench drives the leg with the timer signals of the simulation: a centre-aligned period from counter peak to counter peak, the high side commanded on in $\left[\frac{T_s}{2}(1-d),\ \frac{T_s}{2}(1+d)\right]$, every turn-on delayed by the timer dead time. The phase is a constant current source (the 3 mH phase inductance keeps the current almost constant within 50 µs). Six periods are simulated and the last one is evaluated.

10.2 Device models

STL180N6F7 (VDMOS). The ST SPICE model is not part of the repository (its licence is ST's). The project model uses the ngspice VDMOS device — smooth, with a body diode and a non-linear gate-drain capacitance — fitted to the datasheet values:

Quantity	Model (SP01)	Datasheet (typical)
Rds(on), VGS 10 V, 16 A, 25 °C	1.903 mΩ	1.9 mΩ
Rds(on) at 100 °C	2.466 mΩ (×1.30)	—
Body diode VSD at 32 A	0.997 V	≤ 1.2 V (max)
Body diode at 2 A	0.727 V	—
Ciss at VDS 25 V	4801 pF	4825 pF
Coss at VDS 25 V	2189 pF	2240 pF
Crss at VDS 25 V	207 pF	216 pF
Qg, VGS 0 … 10 V, 30 V, 32 A	78.3 nC	79.5 nC

Assumptions: VGS(th) 3.0 V, internal gate resistance 1.2 Ω, diode 0.70 V at 1 A and 1.0 V at 32 A, transit time 2 ns. The switching times of the datasheet (td(on) 33.9 ns, tr 35.6 ns, td(off) 68.9 ns, tf 42.2 ns) are measured with a stronger gate drive and are not used as fit targets.

L6387E (behavioural). Interlocking (HIN and LIN both high → both outputs low; the L6387E inserts no dead time of its own), a propagation delay of 107 ns (mean of ton 110 ns and toff 105 ns), output stages with 30 Ω source and 18 Ω sink resistance (from the 400 mA / 650 mA short-circuit currents), and the bootstrap DMOS (125 Ω) with its diode. The logic edges are smoothed and delayed with matched lossless lines, so no XSPICE is needed.

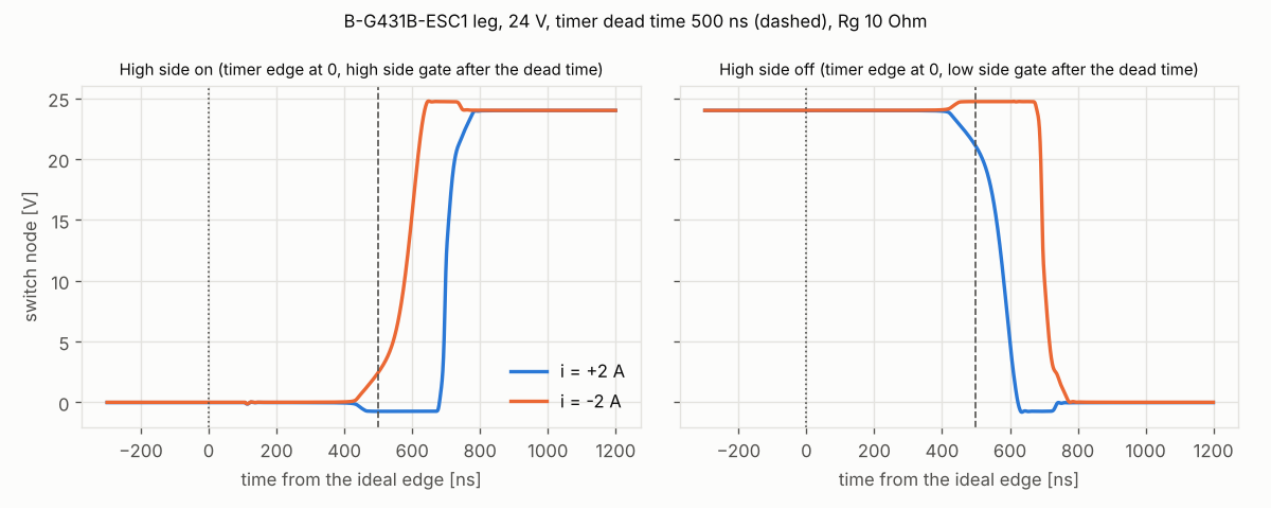
10.3 Board values and assumptions

The ESC1 user manual does not list the passive values of the power stage. Until they are read from the schematic (MB1419), the model uses:

Value	Assumed	Influence
Timer dead time	500 ns	the main input; set in the firmware
Gate driver supply	12 V	gate drive strength
Gate resistors	10 Ω, no turn-off diode	turn-off delay, Miller bump
Bootstrap capacitor	1 µF	high-side gate voltage
Shunt	3 mΩ, 1 nH	leg resistance, shunt settling
Commutation loop	3 nH, damped by 1 Ω across each inductance	ringing

They are the `BOARD` dictionary at the top of `tools/run_tests.py`; change them there and rerun.

10.4 What happens in the dead time



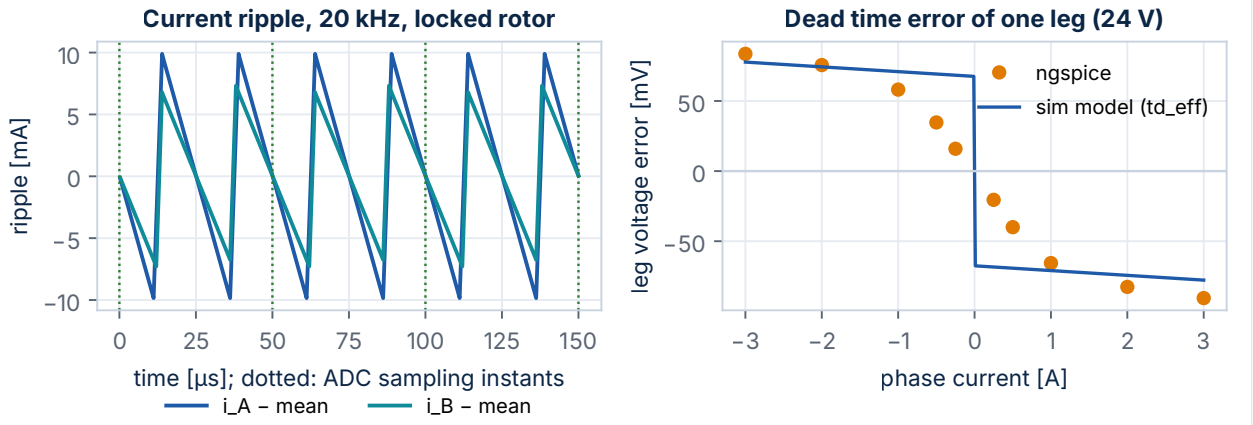
The figure shows the switch node at both edges for +2 A (current out of the leg) and −2 A:

- **The high side turns on 700 ns after the ideal edge** for +2 A: 500 ns timer dead time
- 107 ns driver delay + the time to charge the gate to the Miller plateau. Until then the low-side diode holds the node at −0.73 V.
- **For −2 A the node rises already in the dead time**: the low side switches off and the phase current itself charges the switch node to the bus (soft switching).
- **The high side turns off slowly**: the gate discharges through 18 Ω + 10 Ω + 1.2 Ω from 11 V to the Miller plateau and stays there while the drain voltage changes — about 400 ns. The switch node falls 571 ns after the ideal edge for +2 A.

As a result most of the 500 ns timer dead time is consumed by the turn-off of the other transistor:

Result	Value	Meaning
Gap between the driven gates (both below VGS(th) 3 V)	**154 ns**	margin of the dead time; reducing the timer dead time below about 350 ns risks shoot-through
Effective dead time of the phase voltage (1–3 A)	**133 ns**	what the step model of the simulation and a future compensation must use
Gate of the off switch while the other turns on	**up to 3.73 V**	Cgd·dv/dt against the gate resistor: above VGS(th), a short cross conduction at every edge with the assumed 10 Ω

10.5 Dead time voltage error



The mean leg voltage over a period differs from the ideal $d\,V_{\mathrm{bus}}$. The simulation uses the step model

$$\Delta v_{\mathrm{leg}} = -\mathrm{sign}(i)\,(V_{\mathrm{bus}} + 2V_d)\frac{t_{d,\,\mathrm{eff}}}{T_s} - R_{\mathrm{leg}}\,i$$

with $t_{d,\,\mathrm{eff}}$ fitted for 1 A ≤ $|i|$ ≤ 3 A (largest deviation 12.8 mV) and $R_{\mathrm{leg}} = (R_{ds} + R_{ds} + R_{\mathrm{shunt}})/2$ = 3.4 mΩ. Below 1 A the real error is smaller: the phase current charges the switch node capacitance (about 2 × Coss) too slowly to finish the swing within the dead time — 50 % of the step model at 0.5 A, about 25 % at 0.25 A. The measured table (`leg_error_table`) is exported for a better compensation later.

Current	ngspice error	Step model
−3 A	+83.6 mV	+77.6 mV
−2 A	+75.6 mV	+74.2 mV
−1 A	+58.1 mV	+70.8 mV
−0.5 A	+34.7 mV	+69.1 mV
+0.5 A	−40.0 mV	−69.1 mV
+1 A	−65.6 mV	−70.8 mV
+2 A	−82.6 mV	−74.2 mV
+3 A	−90.6 mV	−77.6 mV

10.6 Shunt sampling

Case	Shunt at the sampling instant	Ideal $-i\,R_{\mathrm{sh}}$	Settled (2 %) after the low-side timer edge
d = 0.5, +2 A	−6.000 mV	−6.000 mV	345 ns
d = 0.5, −2 A	+6.000 mV	+6.000 mV	344 ns
d = 0.9, +2 A	−6.000 mV	−6.000 mV	345 ns

The low-side shunt carries the phase current also in the dead time (through the body diode of the low switch), so the shunt voltage is valid from 345 ns after the low-side command. With an ADC sampling time of 250 ns before the centre, a centred sample stays valid up to a **leg duty of 0.956**. With the shared leg at full voltage the largest leg duty approaches 1; the library does not limit it yet (planned).

10.7 Export to the simulation

`python tools/run_tests.py --export` writes:

File	Content
`export/stage_esc1.json`	everything: board values, Rds(on) at 25 and 100 °C, Vd, timer and effective dead time, leg resistance, edge delays, gate gap and bump, error table, shunt settling, largest duty
`../sfs-stepper-sim/params/stage_esc1_spice.sci`	Scilab function `stage_esc1_spice()` (generated, do not edit)
`../sfs-stepper-sim/params/stage_esc1_spice.json`	the same values for the Python twins

`board_bg431b_esc1()` takes `dead_ns`, `rds_on` and `vd` from there, and the same effective dead time is passed to Stepper-C as `dead_time_ns`.

10.8 Tests

Test	Check	Result
SP01	Rds(on) ±5 %, VSD(32 A) ≤ 1.2 V, Ciss/Coss/Crss ±10 %, Qg ±10 %	pass (table above)
SP02	step model within 15 mV for 1–3 A; driven gates never overlap	12.8 mV; gap 154 ns; gate bump 3.73 V reported
SP03	shunt at the sampling instant within 1 % of `−i·R_sh`	pass, settling 345 ns

Run time about 40 s (`--quick`: fewer operating points). ngspice options: absolute tolerances 1 nA / 10 µV (the defaults stall the solver on the quiet gate nodes), maximum time step 2 ns.

CHAPTER 11

Test Catalogue

Every claim in this manual is backed by a test that prints the measured value. This chapter lists all tests of the three repositories with their stimulus, their pass criterion and the last result. The **Status** column says where the result comes from: *Scilab* (run in Scilab 2027), *Python* (run in the Python twin, the Scilab run of this version is pending), *C* (GoogleTest), *ngspice*.

11.1 Naming

Prefix	Meaning	Where
`T`	test of the model or of the prototype against theory or limits	`sfs-stepper-sim/tests/test_T*.sce`, twins in `tools/`
`X`	cross-check: Scilab equals its Python twin	`tests/test_X*.sce` + `tests/reference/*.csv`
`B`	power stage mapping	`tests/test_B01_bridge.sce`
`S`	software in the loop: the C library in Xcos equals the prototype	`tests/test_S01_sil.sce`
`SP`	ngspice power stage	`sfs-stepper-power-spice/tools/run_tests.py`
GoogleTest suites	library unit tests	`sfs-stepper-ctrl/tests/unit/`

11.2 Motor model

Test	Stimulus	Pass criterion	Result	Status
T00	1000 random states; current vector at φ = 0.7	power balance `e·i = T·ω` within 10^{-9} W; zero and restoring torque at `p θ = φ`	$1.1 \cdot 10^{-13}$ W; $-1.0 \cdot 10^{-5}$ N·m restoring	Scilab
T01	3.2 V on phase A, locked and free rotor	steady current `V/R` within 10^{-6}; free rotor stays at θ = 0	2.000000 A, error $2 \cdot 10^{-9}$	Scilab
T02	voltage step, locked rotor	`i(t) = V/R (1 − e^(−t/τ))`, t63 = L/R within 0.1 %	t63 = 1.8750 ms	Scilab
X01	rotating voltage 5 Hz el., free rotor	Scilab (lsoda) equals Python (Radau)	$3.6 \cdot 10^{-8}$ A, $8.8 \cdot 10^{-7}$ rad/s	Scilab

11.3 Power stage and current loop

Test	Stimulus	Pass criterion	Result	Status
B01	20 000 random voltage requests, both topologies	duties 0 … 1, direction kept, exact when feasible; largest circle `Vbus` / `Vbus/√2`	direction error $2 \cdot 10^{-16}$; 24.0000 V / 16.9706 V	Scilab

Test	Stimulus	Pass criterion	Result	Status
T03	microstepping 2 A, 10 Hz el.	A leads B by 90° ± 0.5°, amplitude ± 10 mA, rotor follows	90.000°, 2.0000 A, lag 0.57°	Scilab
T04	current step 0 → 2 A, locked, both topologies	overshoot < 5 %, settling (2 %) < 1 ms, error < 0.1 %	1.12 %, rise 0.35 ms, settling 0.40 ms	Scilab
X02	current step, shared leg	Scilab equals Python	$5 \cdot 10^{-13}$ A	Scilab
T08	controller for the nominal motor; plant R ×0.9 … 1.3, L ×0.8 … 1.2 (current step) and R, L, J ×5, load 0.15 N·m (ramp to 4 rev/s)	step: overshoot < 10 %, settling < 2 ms, error < 1 %; ramp: load angle < 90°	worst settling 1.6 ms (R ×1.3, L ×0.8); worst load angle 46.3°	Python
T09	8 rev/s, both topologies	shared leg limited earlier than full bridge, both synchronous	1.862 A / 1.557 A, load angle 24.5° / 39.3°	Scilab
T10	switching stage: hold 2 A; ±2 A with dead time; 4 rev/s	ideal switches = averaged (< 1 mA); ripple = analytic ± 5 %; dead time error = formula ± 1 %; synchronous, peak < trip	0.01 mA; 23.11 mA p-p (analytic 23.11); 0.07 mV deviation; peak 2.03 A	Python
X04	switching stage, shared leg, 4 rev/s	Scilab equals Python (10^{-6} A)	—	pending

11.4 Motion

Test	Stimulus	Pass criterion	Result	Status
T05	move 6400 µsteps (2 rev), 5 rev/s, 50 rev/s²	target exact, `|Δv|` ≤ acc, monotonic, load angle < 45°	499 ms, load angle 15.24°, final 6400.00 µsteps	Scilab
T06	load step 0.25 and 0.5 N·m at 1 rev/s	synchronous below `K_t I` with angle `asin(T/(K_t I))`; loss of synchronism detected above	37.21° (theory 37.21°); slip at 151.9 ms	Scilab
T07	reversal +4 → −4 rev/s	synchronous, deceleration limit kept, final speed	load angle 12.25°, −240.0 rpm	Scilab
X03	move 160 µsteps	motion generator + loop equal the Python twin	$4.9 \cdot 10^{-13}$ A	Scilab

11.5 Resonance and friction

Test	Stimulus	Pass criterion	Result	Status
T11a	angle step 0.1 rad at 2 A	natural frequency `√(p K_t I / J)/2π` ± 5 %	248.1 Hz (theory 253.8 Hz)	Python
T11b	speed sweep 0.8 … 1.8 rev/s, detent 0.02 N·m	largest oscillation at `4 p n = f_n` ± 10 %, peak > 3× minimum	peak at 1.2 rev/s (theory 1.27), ±13.6°	Python
T11c	5, 8, 11 rev/s, shared leg, 24 V	oscillation < 1° / < 5° below the voltage limit, > 20° beyond	0.0° / 1.3° / 47.6°	Python
T12	unpowered rotor without back-EMF, `T_c` 0.01 N·m, 50 rad/s	spin-down time `(J/B) ln((ω0 + T_c/B)/(ω + T_c/B))` ± 1 %	39.60 ms (theory 39.60 ms)	Python

11.6 Software in the loop

Test	Scenario	Pass criterion	Result	Status
S01	current step, locked rotor	currents < 5 mA from the prototype	—	pending re-run
S02	move 640 µsteps	currents < 5 mA, commanded angle identical, rotor < 0.05 µstep	1.05 mA, 0, 0.0075 µstep	Scilab
S03	8 rev/s, both topologies	amplitude as T09 ± 30 mA	1.862 A / 1.557 A	Scilab
S04	run 4 rev/s, switching stage	currents and peak < 5 mA from the prototype	—	pending

11.7 Library unit tests (GoogleTest)

Suite	Tests	Covers
`SstepMath`	29	saturating arithmetic, gains from ratios (brute force over 200 000 cases), sine/cosine accuracy, integrator, duty to ticks
`SstepBridge`	5	random requests, circular amplitude, bus voltage scaling, very large requests keep the direction, zero request centred
`SstepMotion`	10	integer square root, two-revolution move (499 ms), short and negative moves, random targets without velocity step, retarget and reversal, run and reverse, hold
`SstepApi`	24	configuration validation, NULL arguments, state rules, trips, faults, hardware errors, late slow loop, bus voltage corner cases, feed-forward saturation
`SstepLoop`	8	closed loop with a motor model: current step, move, load step, reversal and stop, voltage limit by topology, bus voltage compensation, voltage drive, other microstep settings

Result: 76 tests pass with the fixed point backend, 75 with float (one test is backend-specific); 100 % line and branch coverage of `src/`; no finding under ASan and UBSan. Status: C.

11.8 ngspice

Test	Pass criterion	Result	Status
SP01	device against the datasheet (Rds ±5 %, capacitances and Qg ±10 %, VSD ≤ 1.2 V)	all within	ngspice
SP02	step model within 15 mV for 1–3 A; driven gates never overlap	12.8 mV; 154 ns gap	ngspice
SP03	shunt sample within 1 %	exact, settling 345 ns	ngspice

11.9 Running everything

Repository	Command
Stepper-C (github.com/AntonVynohradov/Stepper-C)	`ctest --test-dir "build/[UnitTests]Stepper Motor Control"` (and the Float preset)
sfs-stepper-sim (github.com/AntonVynohradov/sfs-stepper-sim)	`scilab-cli -f tests/run_all.sce`; Python: `python tools/ref_*.py`
sfs-stepper-power-spice (github.com/AntonVynohradov/sfs-stepper-power-spice)	`python tools/run_tests.py`

CHAPTER 12

Results and Design Guidance

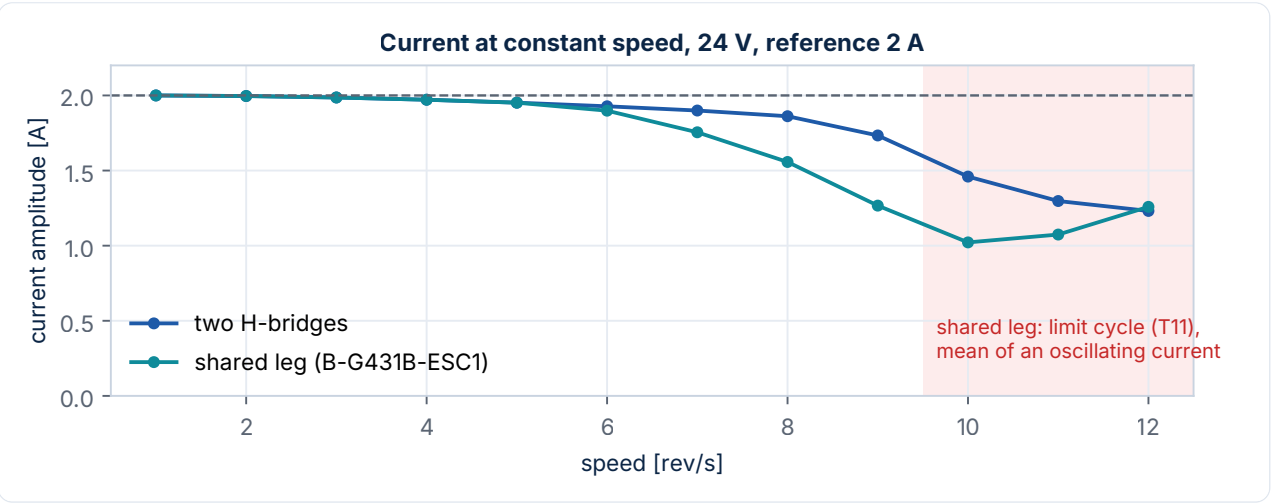
This chapter collects what the models say about the 17HS19-2004S1 on the B-G431B-ESC1 at 24 V, and turns it into guidance for configuring the library and the board. All values come from the tests in the Test Catalogue (p. 56); the assumptions behind them (datasheet motor parameters, detent and friction values, board passives) are listed in Motor Model (p. 17) and Power Stage Model (p. 51).

12.1 Current loop

Quantity	Value	Test
Bandwidth (design)	1 kHz	—
Step 0 → 2 A: overshoot / rise (90 %) / settling (2 %)	1.12 % / 0.35 ms / 0.40 ms	T04
Same with R ×0.9 … 1.3, L ×0.8 … 1.2	overshoot ≤ 5.2 %, settling ≤ 1.6 ms	T08
Phase lag at 1 rev/s	about 4°	T03, T05
Current ripple at 2 A, locked (20 kHz, 3 mH)	23 mA p-p	T10

The pole-zero cancellation keeps the loop well damped for the whole tolerance range of the datasheet; a hot winding (R +30 %) with a low inductance slows the last 2 % of the step to 1.6 ms but does not destabilise it. **Guidance:** 1 kHz bandwidth at 20 kHz PWM is a safe default; do not exceed `fast_loop_hz / 4` (enforced by `sstep_init()`).

12.2 Voltage limit and speed



Speed	Two H-bridges: current, scale, load angle	Shared leg (ESC1): current, scale, load angle
2 rev/s	2.00 A, 1.00, 6°	2.00 A, 1.00, 6°
4 rev/s	1.97 A, 1.00, 12°	1.97 A, 1.00, 12°

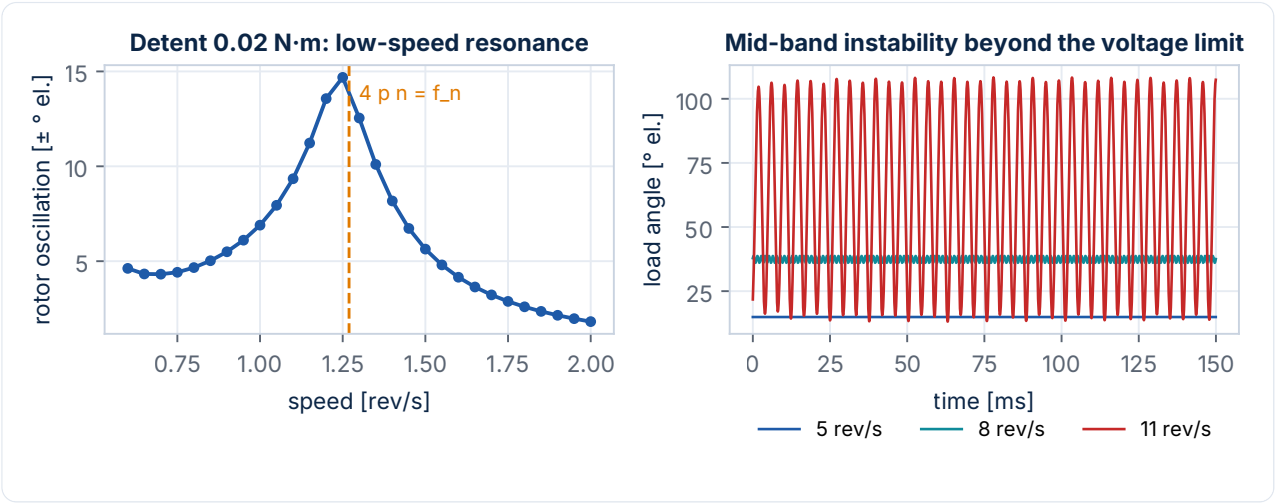
Speed	Two H-bridges: current, scale, load angle	Shared leg (ESC1): current, scale, load angle
6 rev/s	1.93 A, 1.00, 18°	1.90 A, 0.81, 19–20°
8 rev/s	1.86 A, 0.97, 24°	1.56 A, 0.51, 36–39°
9 rev/s	1.73 A, 0.83, 31°	1.27 A, 0.46, 48–50°
10 rev/s	1.46 A, 0.71, 42°	1.02 A, 0.41, 51–63° (oscillating)
12 rev/s	limit cycle, 16–95°	limit cycle, 3–134°

(Unloaded rotor, 2 A reference, back-EMF feed-forward on; *scale* is the smallest factor the power stage applied to the requested voltage, 1 = not limited.)

With three legs the largest circular voltage is `Vbus/√2`: the ESC1 reaches the voltage limit at about 6 rev/s at 24 V, two H-bridges at about 8 rev/s. Beyond the limit the current — and the torque — drops with speed, and the load angle grows even without load.

Guidance: at 24 V on the ESC1 plan for **at most 8 rev/s** with useful torque reserve; the torque available at speed `n` is roughly `K_t · I(n)`, minus the angle already used for acceleration. A higher bus voltage moves every speed in the table up proportionally (the ESC1 accepts up to 6S LiPo voltages).

12.3 Rotor resonance



Low-speed resonance. The rotor on the current vector is a spring-mass system with `f_n = √(p K_t I / J) / 2π` = 254 Hz at 2 A (248 Hz with the current loop). Any torque ripple at that frequency excites it. The detent torque has four periods per electrical period, so it excites the rotor at `4 p n = f_n`, i.e. **about 1.25 rev/s** with the rotor alone: with an assumed detent torque of 0.02 N·m the oscillation reaches ±14° there against ±3–5° elsewhere. A load inertia lowers `f_n` (∝ 1/√J) and with it the resonant speed.

Mid-band instability. As long as the current loop controls the current, it damps the rotor: no oscillation at 5 rev/s. Once the bridge limits the voltage, the drive behaves like a voltage source and the back-EMF coupling undamps the rotor: ±1.3° at 8 rev/s, a limit cycle of ±48° at 11 rev/s on the ESC1 (from about 12 rev/s with two H-bridges).

Guidance:

- Do not run continuously at the low-speed resonance; accelerate through it. Microstepping (M ≥ 16) already removes the full-step excitation; the detent torque remains.
- Keep the operating speed below the voltage limit (previous section) — that is also the region where the rotor is damped.
- Measure `f_n` on the real motor (a small angle step, as in T11a) and the detent torque; the resonant speed depends on both.
- Active damping in the library (from the back-EMF estimate) is a possible extension.

12.4 Robustness

Variation	Effect	Test
R ×0.9 … 1.3, L ×0.8 … 1.2 in the plant, nominal controller	current loop stable, settling ≤ 1.6 ms	T08
5× load inertia	synchronous at 4 rev/s, 50 rev/s², load angle 13°	T08
5× inertia + 0.15 N·m load + R ×1.3, L ×0.8	synchronous, load angle 46°	T08
Load 0.25 N·m at 1 rev/s (below `K_t I` = 0.417 N·m)	synchronous, load angle 37.2° = `asin(T/(K_t I))`	T06
Load 0.5 N·m (above `K_t I`)	loss of synchronism after 152 ms	T06

Guidance: keep the static load below about 60 % of `K_t I` (0.25 N·m at 2 A) to leave angle for acceleration; the load angle in the SIL diagram shows the reserve directly.

12.5 Power stage

Quantity	Value	Consequence
Effective dead time (timer 500 ns)	133 ns	phase voltage error ±0.135 V; distortion near the current zero crossing
Dead time margin	154 ns	do not set the timer dead time below about 350 ns without measuring
Gate bump of the off switch	3.7 V with 10 Ω (assumed)	check the real gate resistors / turn-off diodes
Shunt valid	345 ns after the low-side command	sample centred; leg duty ≤ 0.956
Current peak at 4 rev/s incl. ripple	2.03 A	trip at 3.5 A leaves margin

12.6 Open points

Topic	Status	Where
Measure R, L, `K_t`, detent torque, `f_n` of the real motor	to do (P3)	Hardware Bring-up

Topic	Status	Where
Board passives (Rg, shunt, amplifier gain) from the schematic	to do	Power Stage Model
Dead time compensation in the library	planned; value and error table available	`dead_time_ns`, `leg_error_table`
Delay compensation (angle advance)	planned	Control Algorithms
Limit of the leg duty for valid shunt samples	planned	0.956 from SP03
ADC zero auto-calibration at enable	planned	—
Fast loop execution time on the STM32G431	to measure (P3)	—
Scilab run of T08, T10–T12, X04, S01, S04	pending	Test Catalogue

CHAPTER 13

Hardware Bring-up

The next step of the project (P3) ports Stepper-C to the B-G431B-ESC1 and runs the 17HS19-2004S1. This chapter is the plan and the checklist for that step: how to connect the motor, what to measure first and how to compare the hardware with the models. It describes work that has not been done yet; results will be added in the next edition.

> **Safety**
>
> A two-phase motor on a three-phase board is a non-standard connection; a wrong firmware setting can short a leg. Use a laboratory supply with a current limit (start at 1 A), keep a hand on its output switch, and never connect a battery before the current loop is verified.

13.1 Connection

ESC1 output	Motor wire	Phase
U	A+	phase A between U and W
V	B+	phase B between V and W
W	A− and B− joined	common leg

Identify the two coils of the motor with an ohmmeter (1.6 Ω within a pair, open between pairs). The polarity of a coil only changes the direction of rotation; swapping the phases A and B reverses it too.

The common wire W carries `−(i_A + i_B)`, up to √2 times the phase current amplitude (2.83 A at 2 A); the ESC1 leg and its shunt handle it easily.

13.2 Measurements before the first run

Quantity	Method	Enters
Phase resistance R	4-wire measurement, or a DC current from the supply and a voltmeter, at room temperature	`r_mohm`, `mot.R`
Phase inductance L	LCR meter at 1 kHz; or the current rise time `L/R` with a voltage step	`l_uh`, `mot.L`
Back-EMF constant K_t	spin the unpowered shaft with a drill, measure the open-circuit phase voltage amplitude and the speed: `K_t = E/ω`	`ke_uv_s_per_rad`, `mot.Kt`
Detent torque	torque gauge or a lever and a scale on the unpowered shaft	`mot.Td`
Current sense	outputs off: ADC zero of both phases; known DC current: counts per ampere	`adc_i_zero`, `adc_i_span`, `i_full_scale_ma`
Bus voltage sense	ADC reading at a known supply voltage	`adc_vbus_mv_full`

Quantity	Method	Enters
Timer dead time	the value programmed in the timer break/dead-time register	ngspice `BOARD["dead_ns"]`
Gate resistors, shunt value	schematic MB1419	ngspice `BOARD`

Enter the measured values into the simulation first and rerun T01–T05: the SIL diagram then predicts what the hardware should do.

13.3 Firmware configuration (STM32G431)

Item	Setting
PWM	advanced timer, centre-aligned, 20 kHz: with a 170 MHz timer clock the auto-reload is 4250 (`pwm_period_ticks`)
Dead time	the timer's dead-time generator (start with 500 ns, the value of the models)
Outputs	complementary outputs of the three legs; break input for a hardware trip
ADC	injected conversions of the two phase shunts used by A and B and of the bus voltage, triggered at the counter peak (all low sides on)
Interrupt	ADC end of injected conversion → `sstep_fast_loop()`
Slow loop	1 kHz timer or SysTick → `sstep_slow_loop()`
Backend	`SSTEP_CFG_MATH_BACKEND_FLOAT` (the Cortex-M4F has an FPU), shared leg only

Phase A is measured on the U shunt and phase B on the V shunt; the W shunt (common leg) can be used as a plausibility check: `i_W = −(i_A + i_B)`.

13.4 First power-up sequence

1. Outputs disabled: check the ADC zero readings and the bus voltage reading.
2. `sstep_enable()` with `i_run_ma` = 500, locked shaft: the rotor snaps to a full step and holds. Measure the phase currents with a current probe; they must match the reference within the current sense resolution.
3. Current step test as in T04 (enable, move by one microstep): compare overshoot and settling with the SIL run of the same configuration.
4. Slow move (0.5 rev/s): the rotor follows; listen for the low-speed resonance around 1.2 rev/s.
5. Raise the speed in steps up to 6 rev/s; log the current amplitude and compare with Results (p. 60).
6. Measure the execution time of `sstep_fast_loop()` (GPIO toggle or cycle counter).

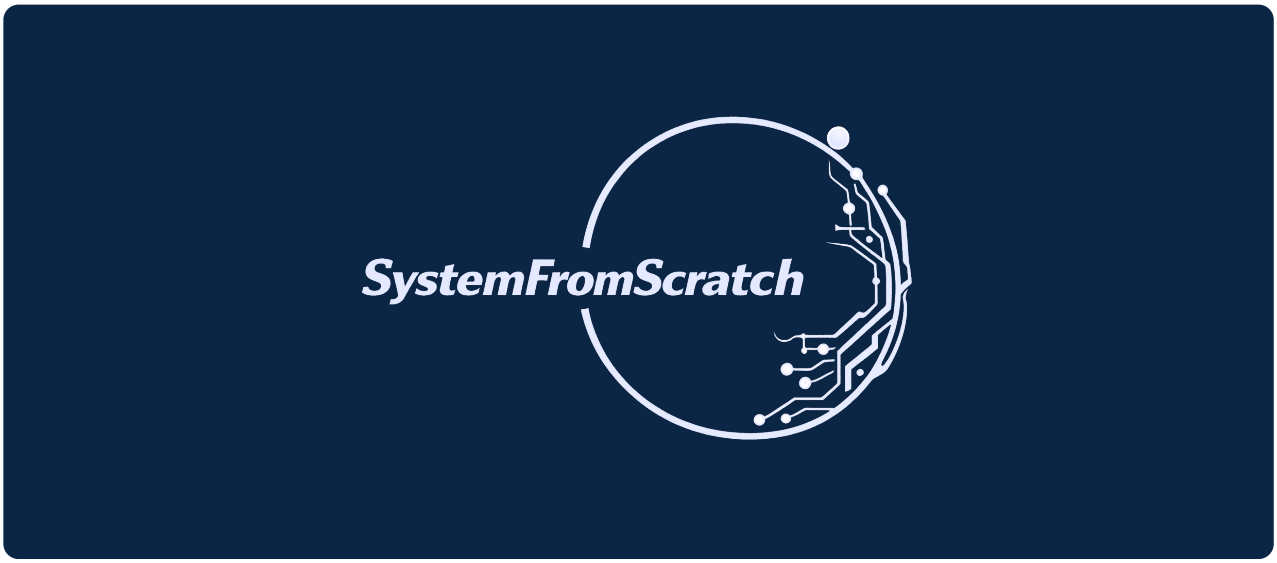
13.5 What to compare with the models

Hardware measurement	Model counterpart	Test
Current step response	prototype / SIL	T04, S01

Hardware measurement	Model counterpart	Test
Current amplitude vs. speed	prototype	T09, voltage limit table
Resonant speed	detent resonance	T11b
Switch node edges, gate voltages	ngspice	SP02
Phase voltage error near the zero crossing	switching stage	T10

Differences beyond the tolerances point to a wrong parameter (most often `K_t`, the ADC scaling or the dead time) and are fed back into the models.

Author & Company



The Stepper-C system is designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification.

Author

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

Embedded Systems Engineer · Secure Product Architect · Founder of SystemFromScratch®

Embedded systems built from scratch — from bare metal to Linux, from first schematic to certification.

Anton designs and builds reliable, secure embedded products end to end: firmware, board support packages and drivers, RTOS and Embedded Linux platforms, hardware bring-up, and the security and compliance work needed to ship them. He is based in Poland and works remotely worldwide.

Area	Focus
Embedded software	Bare-metal, RTOS and Linux firmware in C/C++, built to be maintained for years
BSP & drivers	Bootloaders, Zephyr drivers and Linux kernel work for MCUs, SoCs and SoC FPGAs
Security & compliance	Secure boot, threat modelling, MISRA C/C++, EU Cyber Resilience Act (CRA) readiness
Industries	Industrial IoT, automotive, robotics, medical and consumer electronics

Technology: C · C++ · Python · Zephyr RTOS · FreeRTOS · Embedded Linux · Yocto · Buildroot · ARM Cortex-M · STM32 · Microchip FPGA · RISC-V · CAN / ISO-TP · ROS 2 · Docker · Git · CI/CD · MISRA C/C++ · EU CRA

For the Stepper-C system, Anton wrote the control library and its tests, the Scilab/Xcos motor model and software-in-the-loop environment, the ngspice power stage model and this manual.

SystemFromScratch®

SystemFromScratch® was founded by Anton Vynohradov. The company combines deep embedded engineering with cybersecurity and certification-focused product development, for teams in industrial IoT, automotive, robotics, medical and consumer electronics that need a partner who understands the whole stack — silicon, firmware, operating system and the regulations that govern them.

Service	What it covers
Embedded software development	Bare-metal, RTOS and Linux firmware 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
RTOS & Embedded Linux	Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot): porting, configuration, integration, reproducible builds
Hardware bring-up & integration	Schematic review, first boot, validation and production support, including SoC FPGA and motor-control platforms
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

Stepper-C is one of the open-source projects in the SystemFromScratch® portfolio: public engineering work that can be inspected line by line, built with the same rules — MISRA C, full test coverage, documented decisions — that the company applies to customer products.

Contact

Channel	Address
Website	systemfromscratch.com
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.com or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue in the repository concerned (library, simulation, power stage model or this manual). Please include the versions, your compiler and target or your Scilab / ngspice version, and the output of the failing test.
- **Contributions:** keep the rule of the project — a change of an algorithm in the library is ported to the Scilab prototype and its Python twin in the same change, and every test passes before a pull request.
- **Commercial support, integration and custom development:** contact SystemFromScratch® through systemfromscratch.com.

Trademarks and acknowledgements

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

STM32, STM32G4, B-G431B-ESC1, STL180N6F7 and L6387E are trademarks or products of STMicroelectronics; 17HS19-2004S1 is a product of StepperOnline. They are named only to identify the hardware this manual refers to; Stepper-C is not affiliated with or endorsed by their owners. The device parameters come from the public datasheets of these parts. Scilab and Xcos are developed by the Scilab community; ngspice by the ngspice project.

APPENDIX A

Parameter Reference

All parameters of the system in one place: the library configuration, the Scilab parameter structures and the ngspice board values, with the reference values for the 17HS19-2004S1 on the B-G431B-ESC1 at 24 V.

A.1 Stepper-C configuration

Field	Reference value	Unit	Constraint
`motor.r_mohm`	1600	mΩ	1 … 10^7
`motor.l_uh`	3000	µH	1 … 10^7
`motor.i_rated_ma`	2000	mA	1 … 10^7
`motor.ke_uv_s_per_rad`	208600	µV·s/rad	0 … 10^7 (0 = no feed-forward)
`motor.full_steps_per_rev`	200	—	≥ 1
`stage.topology`	`SSTEP_TOPOLOGY_SHARED_LEG`	—	compiled in
`stage.vbus_nominal_mv`	24000	mV	1 … 10^7
`stage.vbus_min_mv`	10000	mV	0 or < nominal
`stage.vbus_max_mv`	30000	mV	0 or > nominal
`stage.i_full_scale_ma`	4000 (placeholder)	mA	1 … 10^7
`stage.i_trip_ma`	3500	mA	≤ full scale
`stage.pwm_period_ticks`	4250	ticks	≥ 2
`stage.pwm_freq_hz`	20000	Hz	—
`stage.dead_time_ns`	133 (effective)	ns	—
`stage.adc_i_zero`	2048	counts	≤ `adc_full`
`stage.adc_i_span`	2047	counts	±1 … ±65535
`stage.adc_vbus_mv_full`	34320	mV	1 … 10^7
`stage.adc_full`	4095	counts	≥ 1
`ctrl.drive_mode`	`SSTEP_DRIVE_CURRENT`	—	—
`ctrl.fast_loop_hz`	20000	Hz	multiple of the slow rate
`ctrl.slow_loop_hz`	1000	Hz	ratio ≤ 65535
`ctrl.microsteps`	16	—	power of two ≤ `SSTEP_CFG_MICROSTEPS_MAX`
`ctrl.current_bw_hz`	1000	Hz	≤ fast rate / 4
`ctrl.i_run_ma`	2000	mA	≤ trip, ≤ rated

Field	Reference value	Unit	Constraint
`ctrl.i_hold_ma`	1000	mA	≤ run
`ctrl.hold_delay_ms`	100	ms	—
`ctrl.v_max`	16000	µsteps/s	> 0
`ctrl.accel`	160000	µsteps/s²	> 0
`ctrl.decel`	0 (= accel)	µsteps/s²	—

A.2 Simulation parameters

Structure.field	Value	Unit	Source
`mot.R`, `mot.L`, `mot.I_rated`	1.60, $3.0 \cdot 10^{-3}$, 2.0	Ω, H, A	datasheet
`mot.Kt`, `mot.Ns`	0.2086, 200	N·m/A, —	derived, datasheet
`mot.J`, `mot.B`	$8.2 \cdot 10^{-6}$, $1 \cdot 10^{-5}$	kg·m², N·m·s/rad	datasheet, assumption
`mot.Td`, `mot.Tc`, `mot.w_c`	0, 0, 0.05	N·m, N·m, rad/s	not in datasheet (tests: 0.02, 0.01)
`brd.vbus`, `brd.topology`, `brd.fs`	24, `"shared"`, 20000	V, —, Hz	laboratory supply, ESC1
`brd.vbus_min`, `brd.vbus_max`	10, 30	V	trips
`brd.i_full`, `brd.i_trip`	4.0, 3.5	A	ADC model
`brd.period`, `brd.adc_zero`, `brd.adc_span`	4250, 2048, 2047	ticks, counts	ADC model
`brd.adc_vbus_full`, `brd.adc_full`	34.32, 4095	V, counts	ADC model
`brd.stage`, `brd.sw_nstep`	`"avg"`, 16	—, steps	power stage model
`brd.dead_ns`, `brd.dead_timer_ns`	132.6, 500	ns	ngspice export
`brd.rds_on`, `brd.vd`	$3.40 \cdot 10^{-3}$, 0.727	Ω, V	ngspice export
`ctl.bw_hz`, `ctl.delay`, `ctl.ff`, `ctl.nsub`	1000, 1, 1, 4	Hz, periods, —, steps	controller
`mp.M`, `mp.f_slow`	16, 1000	—, Hz	motion
`mp.v_max`, `mp.acc`, `mp.dec`	16000, 160000, 160000	µsteps/s, µsteps/s²	motion
`mp.i_run`, `mp.i_hold`, `mp.hold_delay`	2.0, 1.0, 0.1	A, A, s	motion

A.3 ngspice board values

Key	Value	Unit	Status
`vbus`	24	V	operating point
`fs`	20000	Hz	operating point
`dead_ns`	500	ns	assumption (firmware)
`vcc`	12	V	assumption
`rg`	10	Ω	assumption
`cboot`	1	µF	assumption
`rsh`, `lsh`	3, 1	mΩ, nH	assumption
`lloop`	3	nH	assumption
`tj`	25	°C	operating point

A.4 Derived values

Quantity	Formula	Value
`Kp`	`L · 2π · bw`	18.85 V/A
`Ki`	`R · 2π · bw`	10 053 V/(A·s)
`Kb`	`Ts · R / L`	0.0267
Electrical time constant	`L / R`	1.875 ms
Natural frequency at 2 A	`√(p K_t I / J) / 2π`	254 Hz
Microsteps per revolution	`N_s · M`	3200
Largest phase voltage, shared leg	`Vbus / √2`	16.97 V
ADC current resolution (placeholder scaling)	`i_full / span`	1.95 mA/count

APPENDIX B

Glossary

Anti-windup (back-calculation)
: Feedback of the difference between the applied and the requested voltage into the PI integrator while the power stage limits the voltage, so the integrator does not grow without bound.

Back-EMF
: The voltage induced in a phase by the moving rotor, $K_t\,\omega$ in amplitude.

Bootstrap
: The supply of the high-side gate driver: a capacitor charged from VCC through a diode (in the L6387E a DMOS) while the low side conducts.

Centre-aligned PWM
: PWM whose counter counts up and down; the pulses of all legs are centred in the period, and the counter peak is a common instant at which all low sides conduct.

Dead time
: The interval in which both transistors of a leg are off, inserted to prevent a short circuit. The *timer* dead time is programmed; the *effective* dead time is its net effect on the phase voltage after the driver and transistor delays.

Detent torque
: The torque of the permanent magnet on the unpowered motor, periodic with one full step.

Electrical angle
: $p\,\theta$: one electrical turn equals four full steps.

Feed-forward
: A control signal computed from a model (here the back-EMF from the commanded speed) and added to the controller output.

Hold current
: The reduced phase current applied after a standstill time, to limit heating.

Leg
: A half-bridge: two transistors in series between the bus and ground, with the output between them.

Load angle
: $\delta = \varphi - p\,\theta$, the electrical angle between the current vector and the rotor.

Mid-band instability
: A rotor oscillation that appears at medium speeds when the drive cannot control the current any more (voltage limit) and the back-EMF coupling undamps the rotor.

Miller plateau
: The gate voltage at which a MOSFET switches; the gate current then only charges the gate-drain capacitance while the drain voltage changes.

Per unit (pu)
: A quantity divided by its base value (current: ADC full scale; voltage: nominal bus voltage).

Pole-zero cancellation

Choosing the PI zero at the pole of the R-L phase (`Ki / Kp = R / L`), so the closed loop is first order.

Q15, Q27

Fixed point formats with 15 or 27 fractional bits.

Shoot-through

Both transistors of a leg conducting at the same time; a short circuit of the bus.

SIL

Software in the loop: the target code running in a simulation.

Slow loop, fast loop

The two periodic functions of the library: motion at 1 kHz, current control at the PWM rate.

Synchronism

The rotor following the rotating field; lost when the load angle exceeds about 180°.

Twin

An independent implementation of the same algorithm used for cross-checking.

VDMOS

The ngspice model of a vertical power MOSFET with body diode and non-linear capacitances.

APPENDIX C

Changelog

All notable changes to this manual are documented in this file. The format is based on Keep a Changelog (keepachangelog.com/en/1.1.0/), and this project adheres to Semantic Versioning (semver.org/spec/v2.0.0.html).

C.1 [Unreleased]

Added

- First edition (0.1.0) of the Stepper-C system user manual, covering Stepper-C 0.1.0, `sfs-stepper-sim` 0.1.0 and `sfs-stepper-power-spice` 0.1.0: introduction, system architecture, getting started, motor model, control algorithms, library guide, API reference, simulation, software in the loop, power stage model, test catalogue, results and design guidance, hardware bring-up plan, parameter reference, glossary.
- Website (MkDocs Material, SystemFromScratch® style) and print-ready A4 PDF (`tools/docs/build_manual.py`, WeasyPrint) from the same Markdown.
- Typeset formulas (LaTeX through `pymdownx.arithmatex`): MathJax on the website, SVG rendered with matplotlib mathtext in the PDF.
- Generated diagrams (`tools/docs/diagrams.py`) and data figures from the Python twins of the simulation (`tools/docs/figures.py`).
- GitHub Actions workflow: builds the PDF and deploys the website to GitHub Pages.

APPENDIX D

License

The Stepper-C system and this manual are free and open-source software, licensed under the **Apache License, Version 2.0** (apache.org/licenses/LICENSE-2.0).

Item	Value
Copyright	© 2026 Anton Vynohradov
License	Apache License 2.0
SPDX identifier	`Apache-2.0`
Applies to	The library (`sfs-stepper-ctrl`), the simulation (`sfs-stepper-sim`), the power stage model (`sfs-stepper-power-spice`) and this manual

D.1 What the license allows

In short — the full text below is authoritative:

- **Use** the library in any project, open source or proprietary, commercial or not.
- **Modify** it and **distribute** it, in source or binary form, alone or as part of a product.
- **Patent grant:** each contributor grants you a license to any of their patents that the contribution uses.

D.2 What it requires

When you distribute any part of the system or a modified version of it:

- Include a copy of the license.
- Keep the copyright, patent, trademark and attribution notices in the source files.
- Mark the files you changed.
- If you redistribute a NOTICE file, keep its attribution notices (there is none at the moment).

Firmware that links the library in binary form only needs to make the license and the copyright notice available, for example in its documentation or "about" screen.

D.3 What it does not give

- No warranty: the software is provided "as is".
- No trademark rights: the license does not let you use the names of the author or of SystemFrom-Scratch® to endorse your product.

D.4 Trademarks

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. STM32, B-G431B-ESC1, STL180N6F7 and L6387E are trademarks or products of STMicroelectronics; they are used only to identify the hardware. Stepper-C is not affiliated with or endorsed by STMicroelectronics.

D.5 Full license text


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

   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.
```