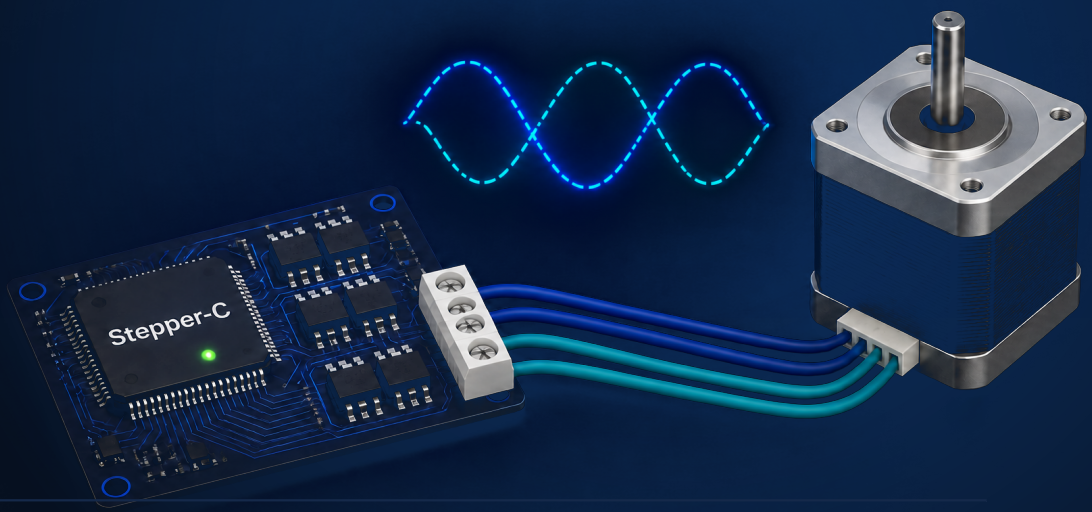
SystemFromScratch

OPEN-SOURCE EMBEDDED LIBRARY

Stepper-C

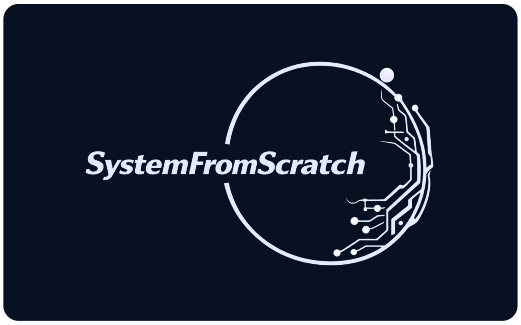
Stepper Motor Control Library for Embedded Systems

USER MANUAL



Version 1.1.0 · November 2026
Anton Vynohradov
systemfromscratch.com

Imprint



Stepper-C User Manual
Library version 1.1.0
Edition: November 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://antonvynohradov.github.io/Stepper-C/
Source code: https://github.com/AntonVynohradov/Stepper-C

Copyright © 2026 Anton Vynohradov.

The Stepper-C library 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 B. 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 is a registered trademark of STMicroelectronics International N.V. Motor and board names are used only as examples of hardware the library can drive; Stepper-C is not affiliated with or endorsed by STMicroelectronics or any motor manufacturer. 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 library version stated above; later versions may differ.

Contents

1 Introduction 6
- 1.1 What the library does 6
- 1.2 What the library does not do 6
- 1.3 Who it is for 7
- 1.4 Key properties 7
- 1.5 Terminology 7
- 1.6 Conventions in this documentation 8

2 Getting Started 9
- 2.1 Get the library 9
- 2.2 Add it to your project 9
- 2.3 Prerequisites for development 9
- 2.4 Build with presets 9
- 2.5 Unit tests and coverage 9
- 2.6 Static analysis 10
- 2.7 Formatting 10
- 2.8 Versioning and releases 10
- 2.9 Documentation 10

3 How It Works 11
- 3.1 Modules 11
- 3.2 The two loops 12
- 3.3 Driver states 13
- 3.4 Memory model 14
- 3.5 Error handling 14
- 3.6 Threading and reentrancy 15
- 3.7 Design decisions 15

4 Using the Library 16
- 4.1 Setting up a driver 16
- 4.2 Commanding motion 16
- 4.3 Reading the status 17
- 4.4 Detecting a stall 18
- 4.5 Handling faults 18
- 4.6 Several motors 19
- 4.7 Return values 20
- 4.8 Common mistakes 20

5 Extending the Library 22
- 5.1 Where to extend 22
- 5.2 Homing example 22
- 5.3 Changing the configuration 23
- 5.4 Contributing to the library 23

6 Configuration 24
- 6.1 Build-time options 24
- 6.2 Run-time configuration 24
- 6.3 Memory footprint 27

6.4 Stack usage ... 28
6.5 CMake options ... 28

7 Motor Control ... 30
7.1 Motor model ... 30
7.2 Per-unit scaling ... 30
7.3 Microstepping ... 31
7.4 Current control ... 32
7.5 Bus voltage compensation ... 33
7.6 Voltage limits ... 33
7.7 Motion profile ... 34
7.8 Hold current ... 35
7.9 Stall detection ... 35
7.10 Numeric backends ... 40

8 Integration Guide ... 41
8.1 PWM ... 41
8.2 Current and voltage measurement ... 41
8.3 Timing ... 42
8.4 Bare metal ... 42
8.5 RTOS ... 42
8.6 Adding the library to a CMake build ... 43
8.7 Bring-up checklist ... 43

9 API Reference ... 44
9.1 Version — sstep.h ... 44
9.2 Configuration — sstep_config.h ... 44
9.3 Definitions — sstep_defs.h ... 45
9.4 Types — sstep_ctrl.h ... 46
9.5 Functions — sstep.h ... 47
9.6 Return codes summary ... 52

10 MISRA C:2012 compliance ... 53
10.1 Checking ... 53
10.2 Current status ... 53
10.3 Manual review notes ... 53
10.4 Deviation register ... 54

Author & Company ... 55
Author ... 55
SystemFromScratch® ... 56
Contact ... 56
Support and feedback ... 57
Trademarks and acknowledgements ... 57

Appendix A Changelog ... 58
A.1 1.2.0 - 2026-11-06 ... 58
A.2 1.1.0 - 2026-11-01 ... 58
A.3 1.0.0 - 2026-10-31 ... 59

Appendix B License ... 62
B.1 What the license allows ... 62
B.2 What it requires ... 62

B.3 What it does not give 62
B.4 Trademarks 63
B.5 Full license text 63

CHAPTER 1

Introduction

Stepper-C is a C11 library that drives a two-phase hybrid stepper motor with closed-loop sinusoidal current control. Given the measured phase currents and bus voltage, it computes the PWM compare values of the power stage, and it turns motion commands (move to a position, run at a velocity, stop) into a smooth microstep trajectory. It runs on any MCU: the hardware is reached only through callbacks, and the numeric backend is either fixed point (no FPU needed) or single-precision float.

1.1 What the library does

- **Controls the phase currents.** One PI controller per phase with back-EMF feed-forward and anti-windup, gains derived from the motor's R and L and the requested bandwidth. An open-loop voltage mode is available for boards without current sensing.
- **Generates the microstep angle.** Sine/cosine current references from a 32-bit electrical angle, 1 to 256 microsteps per full step.
- **Plans the motion.** Trapezoidal profile with separate acceleration and deceleration, no overshoot, smooth replacement of a running command; constant-velocity mode; stop.
- **Fits the voltage to the power stage.** Two full H-bridges (4 legs) or three half-bridges with a shared leg (3 legs); the request is limited to what the topology and the measured bus voltage can deliver, keeping its direction.
- **Protects the hardware.** Overcurrent, overvoltage and undervoltage trips in the fast loop, hardware-callback errors, external faults; outputs off within the same period.
- **Detects a stall without a sensor.** A back-EMF observer reports a rotor that stops, falls out of synchronism or slips at standstill (`SSTEP_FAULT_STALL`, optional).
- **Reduces the current at standstill.** Hold current after a configurable delay.

1.2 What the library does not do

Concern	Why it is not in the library
Timer, PWM and ADC drivers	Hardware-specific; you start the conversions and pass raw ADC counts in, and write the compare values out in a callback.
Interrupt and task setup	Every system has its own; the library only defines which function runs at which rate. See the integration guide.
Position feedback (encoder)	The library is an open-loop positioner with closed-loop current; the rotor position is not measured. Sensorless stall detection reports a lost synchronism but does not correct the position.
Homing, limit switches, G-code, multi-axis coordination	Application logic on top of `sstep_move_to()`, `sstep_set_position()` and the status.
Motor parameter identification	R, L and Kt come from the datasheet or a measurement.

Concern	Why it is not in the library
Communication (UART, CAN, Modbus...)	Out of scope; call the API from your protocol handler.

1.3 Who it is for

- **Firmware for stepper drives** on a general-purpose MCU with gate drivers, for example STMicroelectronics B-G431B-ESC1 or a custom board with two H-bridges.
- **Machines with several axes,** one driver instance per motor, no shared state.
- **Projects that must be reviewed or certified:** MISRA C:2012, 100 % coverage, no heap, bounded execution time.

1.4 Key properties

Property	Value
Language	C11, no warnings with `-Wall -Wextra -Wpedantic -Wconversion`
Dependencies	`<stddef.h>`, `<stdint.h>`, `<stdbool.h>` only
Numeric backends	Fixed point Q15/Q31 (integer only) or single-precision float, chosen at build time
Memory	No heap, no global or static mutable data (`.data` and `.bss` are empty)
RAM per instance	432 B fixed point, 456 B float on Cortex-M4 (details)
Flash	9.0 KiB fixed point, 9.7 KiB float (Cortex-M4, `-Os`, both topologies)
Stack	`sstep_fast_loop()` 216 B, `sstep_slow_loop()` 152 B worst case (Cortex-M4 `-Os` , details)
Coding standard	MISRA C:2012, 0 Cppcheck addon violations, 1 deviation
Tests	100 % line and branch coverage of `src/` in both backends; closed-loop tests on a motor model
Thread safety	One fast-loop context and one slow-loop context per instance, by design; see Threading
License	Apache License 2.0

1.5 Terminology

Full step

1/200 of a revolution for a 1.8° motor; a quarter of an electrical period.

Microstep

A full step divided by `microsteps` (power of two, 1 to 256). Positions and velocities of the API are in microsteps.

Electrical angle

Angle of the current vector in the two phases; one electrical period is four full steps. Held as a 32-bit unsigned number, one turn = 2^{32}.

Per unit (pu)

Internal signals are relative to a base value: currents to `i_full_scale_ma`, voltages to `vbus_nominal_mv`. 1.0 pu is the base.

Fast loop

`sstep_fast_loop()`, called from the PWM/ADC interrupt at `fast_loop_hz` (typically 20 kHz): current control and PWM.

Slow loop

`sstep_slow_loop()`, called at `slow_loop_hz` (typically 1 kHz): trajectory, angle, currents, events.

Leg

One half-bridge (high-side and low-side switch) of the power stage, driven by one PWM compare value.

Load angle

Electrical angle between the commanded current vector and the rotor; it grows with the load torque and the motor loses steps when it exceeds 90° (180° to the next stable position).

1.6 Conventions in this documentation

- Engineering units in the API are integers with the unit in the field name: `_mohm`, `_uh`, `_ma`, `_mv`, `_hz`, `_ms`, `_ns`.
- "Fixed point" and "float" name the two numeric backends; numbers are given for both when they differ.
- The example motor is a 17HS19-2004S1 (1.6 Ω, 3.0 mH, 2.0 A, 0.21 N·m/A, 200 steps) on a 24 V supply; the example board is a B-G431B-ESC1 (three half-bridges).

CHAPTER 2

Getting Started

This chapter shows how to get Stepper-C, add it to your project, and build and test it.

2.1 Get the library

```
git clone https://github.com/AntonVynohradov/Stepper-C.git
```

Each release (github.com/AntonVynohradov/Stepper-C/releases) provides the source code, a prebuilt static library for Linux x86-64 with its headers, the MISRA report and the PDF user manual.

2.2 Add it to your project

CMake.

```
include(path/to/Stepper-C/sstep.cmake)
target_link_libraries(app PRIVATE sstep)
```

Any other build system. Compile `src/*.c` and add the repository root and `inc/` to the include path.

Prebuilt package. Link `lib/libsstep.a` and add `include/` to the include path.

2.3 Prerequisites for development

- C11 compiler, CMake 3.20+
- Optional: GoogleTest and gcovr, Python 3, Cppcheck, clang-format, MkDocs (requirements-dev.txt), Doxygen, WeasyPrint with Pango

2.4 Build with presets

Preset	Purpose
`SSTEP[Debug]`	Static library with debug information
`SSTEP[Release]`	Optimized static library
`SSTEP[UnitTests]`	GoogleTest suite, fixed-point backend, with coverage; target `coverage`
`SSTEP[UnitTestsFloat]`	The same suite with the float backend

2.5 Unit tests and coverage

The `coverage` target fails below `SSTEP_COVERAGE_MIN` (100 %) line or branch coverage.

2.6 Static analysis

Target	Tool
`cppcheck-analyze`	Cppcheck, all checks
`misra-analyze`	Cppcheck MISRA C:2012 addon; fails on any violation
`codeql-analyze`	CodeQL `cpp-queries`

2.7 Formatting

```
clang-format -i src/*.c src/*.h inc/*.h sstep.h tests/unit/*.cpp tests/unit/*.hpp
```

2.8 Versioning and releases

Stepper-C follows Semantic Versioning (semver.org): the public API (`sstep.h` and `inc/`) is stable within a major version. The version comes only from the latest `vX.Y.Z` Git tag: `cmake/version.cmake` sets it in CMake builds and the release workflow stamps it into the packaged `sstep.h`. A header taken from the repository and built without CMake reports `0.0.0` (unknown). Every change is recorded under `[Unreleased]` in `CHANGELOG.md`.

A release tag runs `release.yml`: changelog check, version stamp, unit tests with the coverage gate, MISRA gate, release build, and a GitHub Release with the static library, the headers, the MISRA report and the PDF manual.

2.9 Documentation

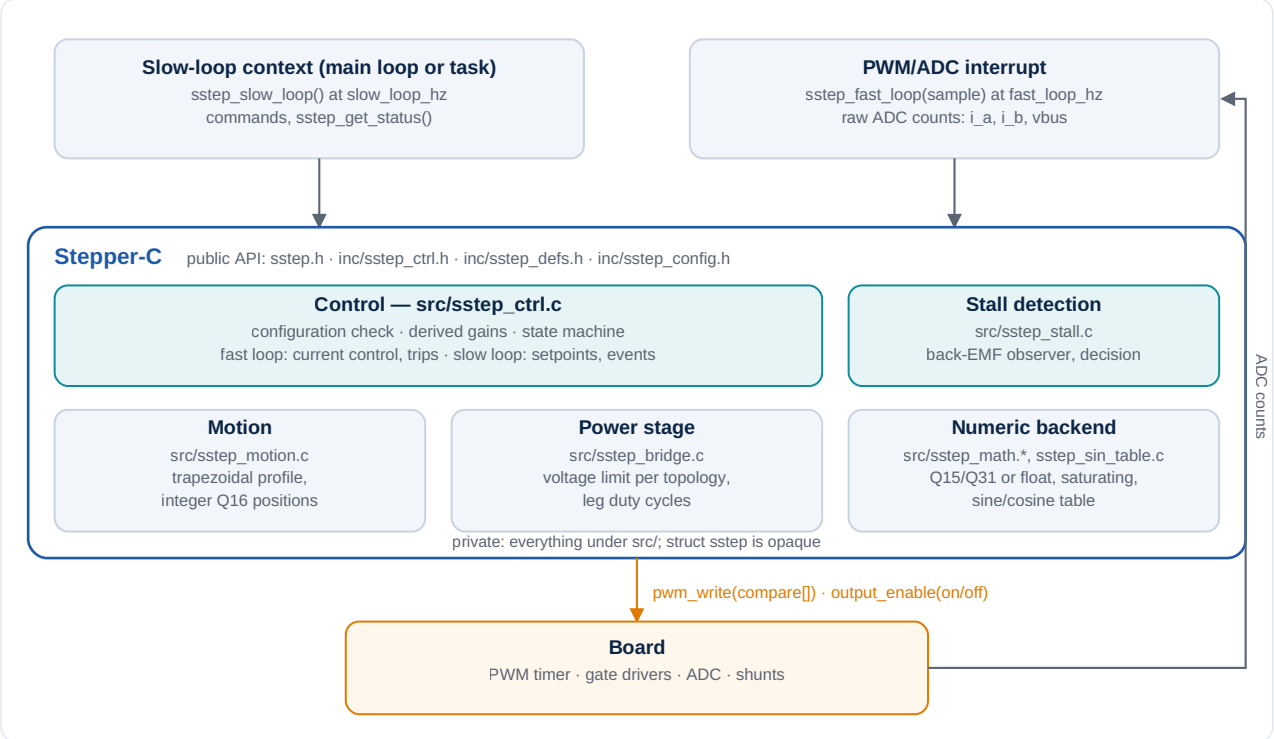
Task	Command	VS Code task
Live preview	`mkdocs serve`	*Docs: Serve on localhost*
Build the website	`mkdocs build`	*Docs: Build site*
Build the PDF manual	`python tools/docs/build_manual.py`	*Docs: Build PDF manual*
Regenerate diagrams	`python tools/docs/diagrams.py`	*Docs: Generate diagrams*

CHAPTER 3

How It Works

This chapter explains how Stepper-C is built: its modules, the two loops and how they exchange data, the driver states, the memory model, error handling and the rules for calling it from interrupts. The control algorithms themselves (current control, microstepping, trajectory, voltage limits) are described in Motor Control (p. 30).

3.1 Modules

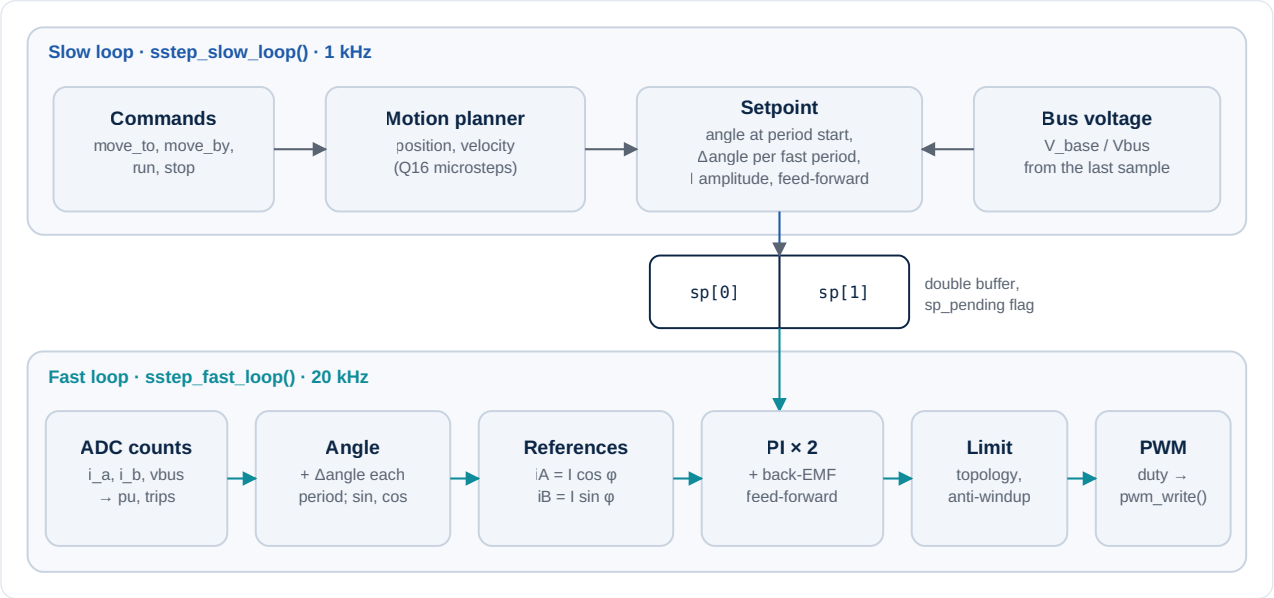


Module	Files	Responsibility
Public API	`sstep.h`, `inc/sstep_ctrl.h`, `inc/sstep_defs.h`, `inc/sstep_config.h`	Functions, configuration and status types, status codes, fault flags, build-time options
Control	`src/sstep_ctrl.c`	Configuration check, derived constants, fast loop (current control, protection, PWM), slow loop (setpoints, hold current, events), commands, state machine
Stall detection	`src/sstep_stall.c`	Back-EMF observer (fast loop), stall decision (slow loop), estimates of the status
Motion	`src/sstep_motion.c`	Trapezoidal trajectory in integer arithmetic (both backends)
Power stage	`src/sstep_bridge.c`	Voltage limit of the topology, phase voltages to leg compare values

Module	Files	Responsibility
Numeric backend	`src/sstep_math.h`, `src/sstep_math.c`, `src/sstep_sin_table.c`	Saturating Q15/Q31 or float operations, gains, sine/cosine table

Only the public API is visible to the application. Everything under `src/` is private: the driver instance `struct sstep` is an opaque block of memory whose layout may change in any release.

3.2 The two loops



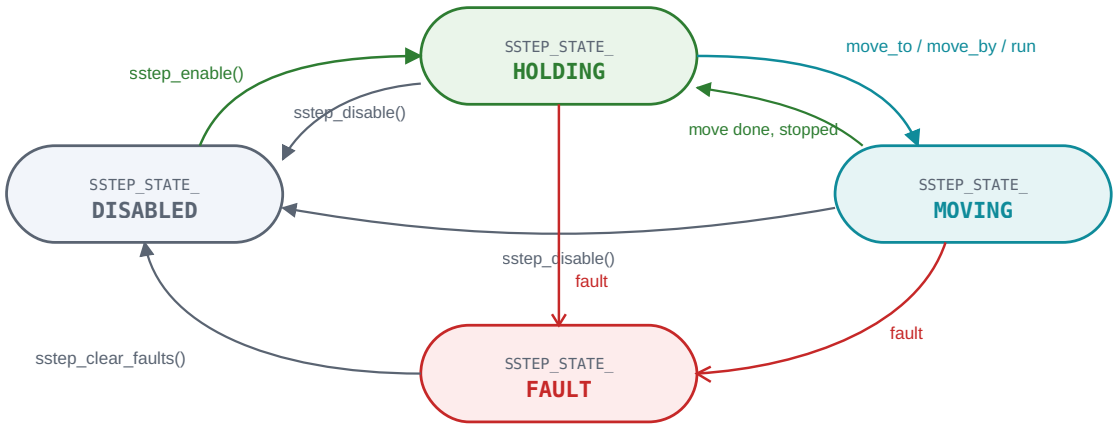
The application calls two functions periodically. Both work on the same instance.

	Fast loop `sstep_fast_loop()`	**Slow loop `sstep_slow_loop()`**
Rate	`fast_loop_hz`, typically the PWM frequency (20 kHz)	`slow_loop_hz`, typically 1 kHz; `fast_loop_hz` must be an integer multiple
Context	PWM/ADC interrupt	Main loop or a timer task
Input	Raw ADC counts: phase A and B currents, bus voltage	Commands given since the last call
Work	Convert the measurements, check the trip levels, interpolate the electrical angle, current references, PI controllers, voltage limit, leg duty cycles; back-EMF observer (stall detection)	Advance the trajectory one step, compute the setpoint of the next slow period, hold current, bus voltage compensation, state, events, stall decision
Output	`pwm_write()` with one compare value per leg	A setpoint for the fast loop; `on_event()`
Time	Bounded: no loops over data; the only loops have fixed limits (voltage pre-scale at most 18, division at most 16 iterations); two divisions, only while the voltage is limited	Bounded: integer square root at most 32 iterations; one gain computation for the bus voltage

Hand-off. The slow loop computes a *setpoint* for the next slow period: the electrical angle at its start, the angle increment per fast period, the current amplitude, the back-EMF feed-forward and the bus voltage correction. It writes it into the free half of a double buffer and marks it pending. The fast loop takes a pending setpoint at the start of its next period and from then on advances the angle by the increment once per period. If the slow loop is late, the fast loop stops advancing after `fast_loop_hz / slow_loop_hz` periods and holds the angle, so a late slow loop never makes the motor run ahead of the trajectory.

The fast loop never reads the half being written: `sp_pending` is cleared before the write and set after it. This relies on the fast loop interrupting the slow loop on the same core, never running in parallel with it.

3.3 Driver states



Faults: overcurrent, over/undervoltage, callback error, sstep_report_fault(). Outputs go off at once.

State	Outputs	Entered by	Left by
`SSTEP_STATE_DISABLED`	Off (all legs high impedance)	`sstep_init()`, `sstep_disable()`, `sstep_clear_faults()`	`sstep_enable()`
`SSTEP_STATE_HOLDING`	On, holding the position	`sstep_enable()`; a move or stop completes	A command; a fault; `sstep_disable()`
`SSTEP_STATE_MOVING`	On, following the trajectory	`sstep_move_to()`, `sstep_move_by()`, `sstep_run()` (seen at the next slow loop)	Move or stop completes; a fault; `sstep_disable()`
`SSTEP_STATE_FAULT`	Off	Any fault, in any active state	`sstep_clear_faults()` only

The state is updated by the slow loop; a command changes it at the next `sstep_slow_loop()`. Faults switch the outputs off immediately, in the context that detects them.

3.4 Memory model

Object	Owner	Size	Lifetime
`struct sstep` (driver instance)	Application (static, stack or pool)	432 B fixed point, 456 B float on Cortex-M4	From `sstep_init()` as long as the motor runs
`struct sstep_config`	Application; copied by `sstep_init()`	152 B on Cortex-M4	Only during `sstep_init()`
`struct sstep_sample`	Application	6 B	Only during `sstep_fast_loop()`
`struct sstep_status`	Application	48 B on Cortex-M4	Filled by `sstep_get_status()`
Sine table	Library, `static const` (flash)	514 B fixed point, 1028 B float (default 257 entries)	Program

There is no heap, no global or static mutable variable and no hidden buffer: everything the library writes is in the instance or in the caller's structures. Several instances (axes) are independent.

The instance size is fixed at compile time by `SSTEP_INSTANCE_WORDS` in `sstep_ctrl.h` (from `SSTEP_CONFIG_BYTES` and `SSTEP_STATE_BYTES`, plain constants so that static analysis can evaluate them) and checked against the private state by `_Static_assert`.

3.5 Error handling

Situation	Result
`NULL` instance, configuration or sample	`SSTEP_ERR_PARAM`; nothing is changed
A configuration field outside its range or a missing required callback	`sstep_init()` returns `SSTEP_ERR_PARAM`
Fields that are valid alone but not together (trip above full scale, bandwidth above `fast_loop_hz / 4`, a gain that does not fit)	`sstep_init()` returns `SSTEP_ERR_CONFIG`
A topology that is not compiled in	`sstep_init()` returns `SSTEP_ERR_UNSUPPORTED`
Any call before a successful `sstep_init()` (zeroed instance, or the last `sstep_init()` failed in the derivation)	`SSTEP_ERR_STATE`; no callback is called
A motion command while disabled or faulted	`SSTEP_ERR_STATE`
A phase current above `i_trip_ma` (current drive)	`SSTEP_FAULT_OVERCURRENT`, outputs off in the same fast period
Bus voltage above `vbus_max_mv` or below `vbus_min_mv`	`SSTEP_FAULT_OVERVOLTAGE` / `SSTEP_FAULT_UNDERVOLTAGE`, outputs off
`pwm_write()` or `output_enable()` returns a negative value	`SSTEP_FAULT_HAL`, outputs off
The rotor does not follow the commanded angle for `stall_time_ms` (stall detection on)	`SSTEP_FAULT_STALL`, outputs off in the slow loop
`sstep_report_fault()`	The given flags, outputs off

Every fault calls `output_enable(false)` and then `on_fault()` (if set) with the new flags, in the context that detected it. Faults accumulate until `sstep_clear_faults()`. Arithmetic never overflows: every operation of the numeric backend saturates.

3.6 Threading and reentrancy

- **Per instance, two contexts:** `sstep_fast_loop()` from the PWM/ADC interrupt; every other function from one task or the main loop (the slow-loop context).
- `sstep_report_fault()` may also be called from an interrupt.
- **No locking inside.** The hand-off between the loops is lock-free on a single core (see above). Do not call the slow-loop functions from two tasks without your own mutex.
- **Multi-core:** both loops of one instance must run on the same core.
- **Different instances** share nothing and may run on different cores.
- **Callbacks** run in the context of the function that calls them: `pwm_write()` and `on_fault()` may run in the interrupt.

3.7 Design decisions

Decision	Reason
Two loops at different rates	Current control needs the PWM rate; the trajectory and the square root of the motion planner do not. The fast loop stays short and bounded.
Configuration in integer engineering units	Values from the datasheet and the schematic are entered as they are (`r_mohm = 1600`); no floating point is needed to configure a fixed-point build.
Gains derived by the library	Kp and Ki follow from R, L and one bandwidth value; there is nothing to tune by hand for a first run.
Fixed point and float backends with the same ranges	The same code and the same tests run on MCUs with and without FPU; the results match within the quantization.
Motion planner in integer arithmetic in both backends	Positions must be exact over long runs; Q16 microsteps in 64 bits never drift.
Callbacks given once at initialization	No link-time coupling to a HAL; several instances can drive different timers.
Raw ADC counts as input	The library owns the scaling (zero offset, span, polarity) and the trip levels in counts, so the protection does not depend on a conversion done elsewhere.
Opaque instance	The application cannot depend on internal fields; the layout can change without breaking the API (MISRA deviation DEV-01).

CHAPTER 4

Using the Library

This chapter shows how to bring a motor up with Stepper-C: fill the configuration, start the driver, command motion, read the status and handle faults. The hardware side (timer, ADC, interrupts) is in the Integration Guide (p. 41); the meaning of every configuration field is in Configuration (p. 24).

4.1 Setting up a driver

One `struct sstep` controls one motor. Allocate it statically (or in any memory that lives as long as the motor runs), fill a `struct sstep_config` and call `sstep_init()`:

```
static struct sstep motor;

int motor_start(const struct sstep_config* cfg)
{
    int result = sstep_init(&motor, cfg);
    if (result == SSTEP_OK)
    {
        result = sstep_enable(&motor);
    }
    return result;
}
```

`sstep_init()` checks every field, derives the controller gains, the ADC scaling, the trip levels and the motion constants, and leaves the outputs off (`SSTEP_STATE_DISABLED`). It does not call any callback. Start the PWM timer and the ADC before `sstep_enable()`: the fast loop must already run when the outputs are switched on.

`sstep_enable()` calls `output_enable(true)` and holds the current position with the run current (the hold current follows after `hold_delay_ms`). The rotor jumps to the nearest position of the commanded electrical angle when the current appears: up to two full steps on the first enable after power-up.

4.2 Commanding motion

All motion commands take positions in microsteps and return immediately; the slow loop executes them. They are accepted in `SSTEP_STATE_HOLDING` and `SSTEP_STATE_MOVING`.

Command	Effect
`sstep_move_to(&motor, 3200)`	Move to the absolute position 3200 with the trapezoidal profile
`sstep_move_by(&motor, -800)`	Move 800 microsteps back from the current *target*
`sstep_run(&motor, 6400)`	Accelerate to 6400 microsteps/s and keep running (negative: other direction)
`sstep_stop(&motor)`	Decelerate to standstill and hold

Command	Effect
`sstep_set_position(&motor, 0)`	Call the current position 0 (homing); only at standstill, the rotor does not move
`sstep_set_motion_limits(&motor, v_max, accel, decel)`	New limits, used from the next slow period
`sstep_set_current(&motor, run_ma, hold_ma)`	New run and hold currents

A command replaces the running one without a velocity step: a new target during a move, a run during a move, a reversal during a run. `sstep_move_by()` adds to the target, so several relative moves given in a row add up.

Wait for a move with the event callback or by polling the status:

```
static volatile bool move_done;

static void on_event(void* context, enum sstep_event event)
{
    (void)context;
    if (event == SSTEP_EVENT_MOVE_DONE)
    {
        move_done = true;
    }
}

bool motor_is_idle(void)
{
    struct sstep_status st;
    return (sstep_get_status(&motor, &st) == SSTEP_OK) && (st.state ==
SSTEP_STATE_HOLDING);
}
```

4.3 Reading the status

`sstep_get_status()` fills a snapshot in engineering units:

Field	Unit	Meaning
`state`	–	`SSTEP_STATE_DISABLED`, `…_HOLDING`, `…_MOVING`, `…_FAULT`
`faults`	flags	Active `SSTEP_FAULT_…` flags
`position`	microsteps	Commanded position (rounded)
`target`	microsteps	Target of the current move
`velocity`	microsteps/s	Commanded velocity
`i_a_ma`, `i_b_ma`	mA	Last measured phase currents
`vbus_mv`	mV	Last measured bus voltage
`emf_d_mv`, `emf_q_mv`	mV	Back-EMF estimate along and 90° ahead of the commanded current (stall detection)
`rotor_velocity`	microsteps/s	Rotor velocity estimated from the back-EMF (stall detection)

Field	Unit	Meaning
`load_angle_cdeg`	0.01°	Estimated load angle in motion (stall detection)
`stall_level_pct`	%	Stall detector level; the fault comes at 100

Position and velocity are *commanded* values. There is no encoder: with stall detection the library reports a rotor that stops following the command (next section); without it a lost step goes unnoticed. Either way, keep the load angle reserve with enough current and a sensible acceleration (Motor Control (p. 30)).

4.4 Detecting a stall

Stall detection estimates the back-EMF of the motor in the fast loop and raises `SSTEP_FAULT_STALL` when the rotor stops in motion, falls out of synchronism, or turns while the command is at rest (for example a load that the hold current cannot hold). It is off unless `stall_time_ms` is set, and needs current drive and `ke_uv_s_per_rad`:

```
static struct sstep_config with_stall_detection(struct sstep_config cfg)
{
    cfg.ctrl.stall_time_ms = 50U;     /* report after 50 ms */
    cfg.ctrl.stall_filter_us = 1000U; /* back-EMF filter */
    cfg.ctrl.stall_speed = 1600U;     /* 0.5 rev/s at 16 microsteps: slip threshold */
    cfg.ctrl.stall_sync_pct = 50U;    /* in motion: back-EMF >= 50 % of Ke * speed */
    return cfg;
}
```

The fault is handled like the others (next section): the outputs are off and the commanded position no longer matches the rotor, so home the axis again after `sstep_clear_faults()`. Below `stall_speed` only a rotor faster than the command is detected; run homing moves against a hard stop faster than `stall_speed`.

While tuning, log the estimates from the slow-loop context and look at the margin: `stall_level_pct` should stay well below 100 in every normal operating point, and `load_angle_cdeg` shows how much torque reserve is left.

```
void log_stall_margin(void)
{
    struct sstep_status st;
    if (sstep_get_status(&motor, &st) == SSTEP_OK)
    {
        board_log_stall(st.velocity, st.rotor_velocity, st.load_angle_cdeg, st.stall_level
_pct);
    }
}
```

The algorithm, the choice of the four values and the measured results are in Stall detection (p. 35).

4.5 Handling faults

A fault switches the outputs off at once, sets `SSTEP_STATE_FAULT` and calls `on_fault()` with the new flags. Commands then return `SSTEP_ERR_STATE`.

Flag	Raised by	Typical cause
`SSTEP_FAULT_OVERCURRENT`	Fast loop, current drive only	Short circuit, wrong `adc_i_span` sign, gains far too high
`SSTEP_FAULT_OVERVOLTAGE`	Fast loop	Regenerated energy when braking a load; supply too high
`SSTEP_FAULT_UNDERVOLTAGE`	Fast loop	Supply off or sagging; `vbus_min_mv` too close to nominal
`SSTEP_FAULT_HAL`	Any function that calls a callback	`pwm_write()` or `output_enable()` returned a negative value
`SSTEP_FAULT_EXTERNAL` (or any flag you choose)	`sstep_report_fault()`	Gate driver fault pin, over-temperature
`SSTEP_FAULT_STALL`	Slow loop, stall detection on; fast loop, with `SSTEP_FAULT_OVERCURRENT`, when the rotor is torn away faster than `stall_time_ms`	Load above the torque, hard stop, rotor slipping at the hold current

Report a gate driver fault from its interrupt:

```
void EXTI15_10_IRQHandler(void);
void EXTI15_10_IRQHandler(void)
{
    if (board_driver_fault())
    {
        (void)sstep_report_fault(&motor, SSTEP_FAULT_EXTERNAL);
    }
}
```

To recover, remove the cause, then:

```
if ((sstep_clear_faults(&motor) == SSTEP_OK) && (sstep_enable(&motor) == SSTEP_OK))
{
    board_led(false);
}
```

`sstep_clear_faults()` returns to `SSTEP_STATE_DISABLED`; the position is kept, so the motor continues from where it was commanded (it may have been moved by hand meanwhile).

4.6 Several motors

Each motor has its own instance and its own callbacks; use `context` to tell the callbacks which hardware they drive:

```
static struct sstep axes[2];
static const uint32_t axis_ids[2] = {0U, 1U};

static int axis_pwm_write(void* context, const uint16_t compare[SSTEP_LEGS_MAX])
{
    board_pwm_set_axis(*(const uint32_t*)context, compare);
```

```
    return 0;
}
```

with `.hal.context = (void*)&axis_ids[k]` in the configuration of axis `k`. The fast loops of several axes may run in one interrupt one after the other, or in separate interrupts.

4.7 Return values

Code	Value	Returned when
`SSTEP_OK`	0	Success
`SSTEP_ERR_PARAM`	−1	`NULL` pointer, a value outside its range, `|velocity| > v_max`, a relative move that overflows
`SSTEP_ERR_CONFIG`	−2	Configuration fields that do not fit together, currents above the trip or rated current
`SSTEP_ERR_STATE`	−3	Any function but `sstep_init()` on an instance without a successful `sstep_init()`; a motion command while disabled or faulted; `sstep_set_position()` while moving; `sstep_enable()` while faulted
`SSTEP_ERR_UNSUPPORTED`	−4	A topology that is not compiled in

`sstep_fast_loop()` returns `SSTEP_ERR_PARAM` only for `NULL` arguments and only with `SSTEP_CFG_ARG_CHECKS = 1`; its faults are reported through the state and `on_fault()`.

4.8 Common mistakes

Symptom	Likely cause
`SSTEP_FAULT_OVERCURRENT` right after `sstep_enable()`	`adc_i_span` has the wrong sign (inverting amplifier) or `adc_i_zero` is wrong: the controller sees the current in the wrong direction and drives it up
`SSTEP_FAULT_UNDERVOLTAGE` right after `sstep_enable()`	The fast loop ran before the first ADC conversion (`vbus` = 0), or `adc_vbus_mv_full` is wrong
The motor hums but does not turn	Phases swapped or one phase reversed; `current_bw_hz` too low; `v_max` above what the supply can drive
The motor turns the wrong way	Swap the two wires of one phase, or negate positions in the application
Loses steps at speed	Not enough voltage for the speed (Voltage limits); shared-leg stage at its limit; acceleration too high for the load
Moves are jerky, about once per millisecond	`sstep_slow_loop()` is not called at `slow_loop_hz`, or `fast_loop_hz` does not match the real interrupt rate
Current settles at the wrong value	`i_full_scale_ma` / `adc_i_span` do not match the shunt and amplifier
`SSTEP_FAULT_STALL` at enable or at the run → hold change, rotor at rest	Observer offset above e times the slip threshold (wrong `r_mohm`, `adc_i_zero`): check `emf_d_mv` at rest, raise `stall_speed`
`SSTEP_FAULT_STALL` at speed although the motor runs	Load close to the pull-out torque, or a load angle swinging beyond about 120° near the voltage limit: check `load_angle_cdeg`, lower `stall_sync_pct` for a heavy load (not below the inductance error, Choosing the parameters), lower the speed or raise the current

Symptom	Likely cause
`sstep_init()` returns `SSTEP_ERR_CONFIG`	Check the combinations: trip ≤ full scale, run current ≤ trip and rated, bandwidth ≤ `fast_loop_hz / 4`, `fast_loop_hz` a multiple of `slow_loop_hz`

CHAPTER 5

Extending the Library

Most extensions are built on top of Stepper-C without changing it: homing, limit switches, communication, multi-axis coordination and encoders are application code that calls the public API. This chapter shows where each extension belongs and the rules for changes to the library itself.

5.1 Where to extend

You want to…	Do it in	Library change needed?
Drive another MCU or board	Your `pwm_write()`, `output_enable()`, ADC and interrupt code	No
Home the axis on a switch	Application: `sstep_run()` towards the switch, `sstep_stop()` on the edge, `sstep_set_position(0)`	No
Add soft limits	Application: clamp targets before `sstep_move_to()`	No
Accept commands over UART, CAN, Modbus	Application: protocol handler that calls the commands from the slow-loop context	No
Coordinate several axes	Application: one instance per axis, start moves in the same slow period, scale `v_max` per axis	No
Detect a stall without a sensor	Configuration: `stall_time_ms` and the other `stall_*` fields (Stall detection)	No
Detect a stall with an encoder	Application: compare the encoder with `sstep_get_status()` position, `sstep_report_fault()` on a large difference	No
Report a temperature or driver fault	`sstep_report_fault()` from the interrupt or task that detects it	No
Use another numeric format or table size	Build-time options in `sstep_user_config.h`	No
Support a new power stage topology	`src/sstep_bridge.c` (limit and duty cycles), `enum sstep_topology`, a build-time option	Yes
A different motion profile (S-curve)	`src/sstep_motion.c`	Yes
Dead-time compensation	Fast loop; the configuration already has `dead_time_ns`	Yes

5.2 Homing example

```c
enum homing_state { HOMING_SEEK, HOMING_STOP, HOMING_DONE };

static enum homing_state homing = HOMING_SEEK;

/* called from the slow-loop context after sstep_slow_loop() */
void homing_step(bool switch_pressed)
```

```
{
    struct sstep_status st;
    switch (homing)
    {
    case HOMING_SEEK:
        if (switch_pressed)
        {
            (void)sstep_stop(&motor);
            homing = HOMING_STOP;
        }
        break;
    case HOMING_STOP:
        if ((sstep_get_status(&motor, &st) == SSTEP_OK) && (st.state == SSTEP_STATE_HOLDIN
G))
        {
            (void)sstep_set_position(&motor, 0);
            homing = HOMING_DONE;
        }
        break;
    default:
        break;
    }
}
```

Start it with `sstep_run(&motor, -1600)` (towards the switch, at a low speed). The stop uses the configured deceleration, so the zero is where the motor came to rest, a few microsteps after the switch; for a repeatable zero, move back off the switch slowly and take the edge again.

5.3 Changing the configuration

Build-time options go into `sstep_user_config.h` in a folder of your project:

```
/* sstep_user_config.h */
#define SSTEP_CFG_MATH_BACKEND SSTEP_MATH_BACKEND_FLOAT
#define SSTEP_CFG_TOPOLOGY_FULL_BRIDGE 0U /* only the 3-leg stage of this board */
#define SSTEP_CFG_MICROSTEPS_MAX 64U
```

and CMake gets `-DSSTEP_CONFIG_DIR=<folder>`. Run-time values (motor, currents, limits) need no rebuild: pass another `struct sstep_config` to `sstep_init()`, or use `sstep_set_current()` and `sstep_set_motion_limits()` while running.

5.4 Contributing to the library

1. **Code.** C11, no new dependencies, no heap, no global state; the rules of `misra-analyze`; internal types stay in `src/`, `sstep.h` declares the public API only.
2. **Both backends.** Every arithmetic change works with fixed point and float; the `[UnitTests]` and `[UnitTestsFloat]` presets must both pass.
3. **Tests.** 100 % line and branch coverage of `src/`; a closed-loop test on the motor model for any change of the control behavior.
4. **Instance size.** If the private state grows, update `SSTEP_STATE_BYTES` in `inc/sstep_ctrl.h`; the build fails until it fits.
5. **Analysis.** `misra-analyze` 0 violations, or a recorded deviation in MISRA C:2012 (p. 53).
6. **Format.** `clang-format` reports no changes in the files you touched.
7. **Documentation.** The affected pages, the stack table, the diagrams and `CHANGELOG.md` `[Unreleased]`.
8. **Compatibility.** The public API is stable within a major version.

CHAPTER 6

Configuration

Stepper-C has two kinds of configuration. **Build-time options** select the code that is compiled in (numeric backend, topologies, table size); they are macros in `inc/sstep_config.h`. **Run-time configuration** describes the motor, the power stage and the control; it is a `struct sstep_config` passed to `sstep_init()`. This chapter lists both, then the memory footprint and the measured stack usage.

6.1 Build-time options

Each macro has a default, is range-checked with `#error` and can be overridden on the compiler command line or in a header `sstep_user_config.h` (included when `SSTEP_USER_CONFIG` is defined; with CMake, set `SSTEP_CONFIG_DIR` to its folder). The application must be compiled with the same values as the library: they change the size of `struct sstep`.

Macro	Default	Range	Meaning
`SSTEP_CFG_MATH_BACKEND`	`SSTEP_MATH_BACKEND_FIXED`	`…_FIXED`, `…_FLOAT`	Numeric backend (CMake: `SSTEP_MATH_BACKEND=FIXED` or `FLOAT`)
`SSTEP_CFG_TOPOLOGY_FULL_BRIDGE`	`1`	0, 1	Compile the two-H-bridge topology
`SSTEP_CFG_TOPOLOGY_SHARED_LEG`	`1`	0, 1	Compile the three-leg topology; at least one must be 1
`SSTEP_CFG_MICROSTEPS_MAX`	`256U`	power of two, 1 .. 256	Largest `microsteps` accepted by `sstep_init()`
`SSTEP_CFG_SINE_TABLE_BITS`	`8U`	6 .. 10	Quarter-wave table of $2^{\text{bits}} + 1$ entries (see the accuracy table)
`SSTEP_CFG_ARG_CHECKS`	`1`	0, 1	0 removes the `NULL` checks from `sstep_fast_loop()` only

6.2 Run-time configuration

`struct sstep_config` has four parts. All values are integers in engineering units; `sstep_init()` copies the structure, so it may be a temporary or a `const` in flash. Fields outside their range make `sstep_init()` return `SSTEP_ERR_PARAM`; combinations that do not fit return `SSTEP_ERR_CONFIG`.

Motor

Field	Unit	Range	17H-S19-2004S1	Where to get it
`r_mohm`	mΩ	1 .. 10^7	1600	Datasheet (phase resistance at 25 °C) or an ohmmeter

Field	Unit	Range	17H-S19-2004S1	Where to get it
`l_uh`	µH	1 .. 10^7	3000	Datasheet (phase inductance) or an LCR meter at 1 kHz
`i_rated_ma`	mA	1 .. 10^7	2000	Datasheet; `i_run_ma` may not exceed it
`ke_uv_s_per_rad`	µV·s/rad	0 .. 10^7	208600	$T_{\text{hold}}/(\sqrt{2}\, I_{\text{rated}})$ in µN·m/A ($0.59/(\sqrt{2}\cdot 2) = 0.2086$ N·m/A); 0 disables the feed-forward
`full_steps_per_rev`	–	≥ 1	200	200 for 1.8°, 400 for 0.9°

Power stage and measurement

Field	Unit	Range	B-G431B-ESC1 example	Meaning
`topology`	–	compiled-in value	`SSTEP_TOPOLOGY_SHARED_LEG`	See Voltage limits
`vbus_nominal_mv`	mV	1 .. 10^7	24000	Nominal supply; base of the per-unit voltage
`vbus_min_mv`	mV	0 or below nominal	10000	Undervoltage trip; 0 disables it
`vbus_max_mv`	mV	0 or above nominal	28000	Overvoltage trip; 0 disables it
`i_full_scale_ma`	mA	1 .. 10^7	4000	Phase current at the ADC reading `adc_i_zero + adc_i_span`; base of the per-unit current
`i_trip_ma`	mA	1 .. `i_full_scale_ma`	3500	Overcurrent trip (either polarity)
`pwm_period_ticks`	timer ticks	≥ 2	4250	Compare value for 100 % duty cycle (timer auto-reload)
`pwm_freq_hz`	Hz	–	20000	Reserved; not used by this version
`dead_time_ns`	ns	–	0	Reserved for dead-time compensation; not used by this version
`adc_i_zero`	counts	≤ `adc_full`	2048	Current ADC reading at 0 A (measure with the outputs off)
`adc_i_span`	counts	±1 .. ±65535	2047	Counts from 0 A to + `i_full_scale_ma`; negative for an inverting amplifier
`adc_vbus_mv_full`	mV	1 .. 10^7	34320	Bus voltage that gives the reading `adc_full` (divider ratio × ADC reference)

Field	Unit	Range	B-G431B-ESC1 example	Meaning
`adc_full`	counts	≥ 1	4095	Largest ADC reading

The B-G431B-ESC1 values above are placeholders derived from the schematic; measure `adc_i_zero`, `adc_i_span` and `adc_vbus_mv_full` on your board.

Control and motion

Field	Unit	Range	Example	Meaning
`drive_mode`	–	`SSTEP_DRIVE_CURRENT`, `SSTEP_DRIVE_VOLTAGE`	current	Closed-loop current or open-loop voltage drive
`fast_loop_hz`	Hz	1 .. 10^7	20000	Rate of `sstep_fast_loop()`
`slow_loop_hz`	Hz	divides `fast_loop_hz`, ratio ≤ 65535	1000	Rate of `sstep_slow_loop()`
`microsteps`	–	power of two, 1 .. `SSTEP_CFG_MICROSTEPS_MAX`	16	Microsteps per full step
`current_bw_hz`	Hz	1 .. `fast_loop_hz / 4`	1000	Current loop bandwidth; the PI gains follow from it
`i_run_ma`	mA	1 .. min(`i_trip_ma`, `i_rated_ma`)	1500	Peak phase current while moving
`i_hold_ma`	mA	0 .. `i_run_ma`	500	Peak phase current at standstill
`hold_delay_ms`	ms	any	500	Standstill time before the hold current
`v_max`	microsteps/s	see below	16000	Velocity limit
`accel`	microsteps/s²	see below	64000	Acceleration
`decel`	microsteps/s²	as `accel`, or 0	0	Deceleration; 0 = same as `accel`
`stall_time_ms`	ms	0, or 1 .. 10^7 and at least one slow period	50	Stall detection: time the condition must persist before `SSTEP_FAULT_STALL`; 0 = off (default) and the other `stall_*` fields are ignored
`stall_filter_us`	µs	≥ one fast period, ≤ 10^7	1000	Time constant of the back-EMF low-pass filter
`stall_speed`	microsteps/s	1 .. 10^7; its back-EMF 1 mV .. `vbus_nominal_mv`	1600	Slip threshold at rest and below this commanded speed; synchronism check above it
`stall_sync_pct`	%	1 .. 100	50	Smallest back-EMF along the commanded motion, relative to $K_e\,\omega$

Stall detection requires `SSTEP_DRIVE_CURRENT` and `ke_uv_s_per_rad > 0` (`SSTEP_ERR_CONFIG` otherwise); the Stall detection (p. 35) section explains the algorithm and how to choose the four values.

The motion limits are converted to 1/65 536 microstep per slow period. Each must round to at least 1 and stay below 2^{30}: with `slow_loop_hz = 1000`, `v_max` from 0.008 to below 16 384 000 microsteps/s, `accel` and `decel` from 8 to below $1.638 \cdot 10^{10}$ microsteps/s².

Callbacks

Field	Required	Called from	Contract
`context`	–	–	Passed unchanged to every callback
`pwm_write`	Yes	`sstep_fast_loop()`	Load one compare value per leg (`compare[0..3]`, `compare[3]` is 0 for the shared leg) for the next PWM period; return 0, or a negative value on error
`output_enable`	Yes	Any function that enables, disables or faults	true: enable the gate drivers; false: all legs high impedance. Return 0, or a negative value on error
`on_fault`	No	The context that detects the fault (may be the interrupt)	Informs the application of new fault flags; outputs are already off
`on_event`	No	`sstep_slow_loop()`	`SSTEP_EVENT_MOVE_DONE`, `SSTEP_EVENT_HOLD_CURRENT`

6.3 Memory footprint

Measured with `arm-none-eabi-gcc` 15.2.1 for Cortex-M4 (`-mthumb`, float backend with `-mfpu=fpv4-sp-d16 -mfloat-abi=hard`), default build-time options, both topologies.

Object	Fixed point	Float
`struct sstep` (RAM per motor)	432 B	456 B
Code and constants, `-Os` (flash)	9216 B	9894 B
of which the sine table	514 B	1028 B
`.data` + `.bss`	0 B	0 B

`struct sstep` is the configuration copy (`SSTEP_CONFIG_BYTES`: 152 B on Cortex-M4, 176 B on 64-bit hosts) plus the private state (`SSTEP_STATE_BYTES`: 280 B fixed point, 304 B float on Cortex-M4; 288 B and 312 B on 64-bit hosts); on a 64-bit host the instance is 464 B and 488 B. Stall detection accounts for 96 B of it on Cortex-M4, and for 2236 B (fixed point) or 2448 B (float) of the flash whether it is enabled or not.

6.4 Stack usage

Worst-case stack of each public function including the library functions it calls, measured with `-fstack-usage -fcallgraph-info=su` and `tools/docs/stack_usage.py` (Cortex-M4, toolchain as above). Calls through the callbacks are not included: add the stack of your `pwm_write()`, `output_enable()`, `on_fault()` and `on_event()` to the functions that call them.

Function	Fixed `-Os`	Fixed `-O0`	Float `-Os`	Float `-O0`
`sstep_init()`	224	1544	200	1432
`sstep_enable()`	48	80	64	88
`sstep_disable()`	24	40	24	40
`sstep_fast_loop()`	216	304	128	312
`sstep_slow_loop()`	152	696	140	624
`sstep_move_to()`	0	48	0	48
`sstep_move_by()`	0	112	0	112
`sstep_run()`	16	104	16	104
`sstep_stop()`	0	40	0	40
`sstep_set_position()`	16	72	16	72
`sstep_set_current()`	52	216	36	128
`sstep_set_motion_limits()`	72	224	72	224
`sstep_report_fault()`	24	40	24	40
`sstep_clear_faults()`	0	40	0	40
`sstep_get_status()`	92	536	92	496

The interrupt stack must hold the frame of `sstep_fast_loop()` plus `pwm_write()` (and `output_enable()` + `on_fault()` for the fault path) plus the interrupt entry. There is no recursion; every frame is static.

To re-measure:

```
mkdir stack && cd stack
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Os -std=c11 -fstack-usage -fcallgraph-info=su \
    -I.. -I../inc -c ../src/*.c
python ../tools/docs/stack_usage.py . --markdown
```

6.5 CMake options

Option	Default	Meaning
`SSTEP_MATH_BACKEND`	`FIXED`	Numeric backend: `FIXED` or `FLOAT`
`SSTEP_CONFIG_DIR`	empty	Folder with `sstep_user_config.h`

Option	Default	Meaning
`UNIT_TESTS`	`OFF`	GoogleTest suite and `coverage` target
`SSTEP_COVERAGE_MIN`	`100`	Coverage gate (lines and branches, %)
`SSTEP_MISRA_FAIL_ON_VIOLATION`	`ON`	`misra-analyze` fails on any violation
`SSTEP_MISRA_RULE_TEXTS`	empty	Licensed MISRA rule texts for readable reports

CHAPTER 7

Motor Control

This chapter describes the algorithms of Stepper-C and the equations behind them: the motor model, per-unit scaling, the current controller, microstepping, the motion profile, the voltage limits of the two power stage topologies, the stall detection and the numeric backends. Every number in this chapter comes from the code or from the closed-loop tests on the motor model (`tests/unit/gtest_sstep_loop.cpp`).

7.1 Motor model

A two-phase hybrid stepper motor with N_s full steps per revolution has $p = N_s/4$ pole pairs (50 for a 1.8° motor). With the rotor angle θ and its speed ω, each phase is a resistance, an inductance and a back-EMF:

$$v_A = R\,i_A + L\,\frac{di_A}{dt} - K_t\,\omega\sin(p\theta)$$
$$v_B = R\,i_B + L\,\frac{di_B}{dt} + K_t\,\omega\cos(p\theta)$$
$$T_e = K_t\left(i_B\cos(p\theta) - i_A\sin(p\theta)\right)$$

K_t is the torque constant [N·m/A]; in SI units it equals the back-EMF constant [V·s/rad], the value entered as `ke_uv_s_per_rad`.

The library imposes the phase currents

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

where φ is the electrical angle commanded by the trajectory. The torque becomes

$$T_e = K_t\,I\sin(\varphi - p\theta)$$

The rotor follows φ with a lag (the *load angle*) that grows with the load. The motor holds up to K_tI (0.42 N·m for 2 A with the 17HS19-2004S1) at a load angle of 90° and loses steps above it.

The closed-loop test with a 0.25 N·m load at 1 rev/s measures a load angle of 37.21°, against $\arcsin(0.25/0.4172) = 36.83°$ predicted by the equation; with 0.50 N·m the motor slips, as expected.

7.2 Per-unit scaling

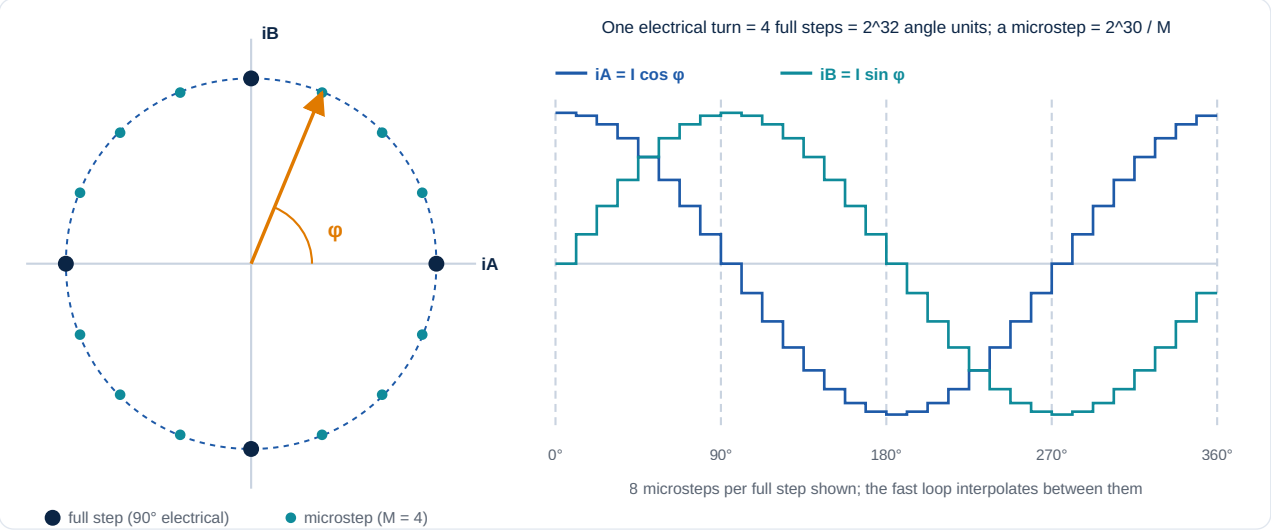
Internally every signal is a fraction of a base value, so that the same arithmetic works for a 0.5 A and a 10 A motor:

Quantity	Base	Range inside the loops
Current	`i_full_scale_ma`: the current at ADC full scale	-1 .. +1 pu

Quantity	Base	Range inside the loops
Voltage	`vbus_nominal_mv`: the nominal bus voltage	-1 .. +1 pu for signals; up to ±65 536 pu in accumulators
Bus voltage	`vbus_nominal_mv`	0 .. 2 pu
Angle	One electrical turn = 2^{32}	`uint32_t`, wraps

`sstep_init()` converts every engineering value of the configuration into per-unit gains with integer arithmetic ($2\pi \approx 710/113$, error $8.5 \cdot 10^{-8}$), so a fixed-point build needs no floating point at all.

7.3 Microstepping



One full step is a quarter of an electrical turn (2^{30}). With `microsteps` $= M$, one microstep is $2^{30}/M$ of the 32-bit angle; positions are kept in Q16 microsteps (1/65 536 microstep) in 64 bits, so the angle is exact and never drifts.

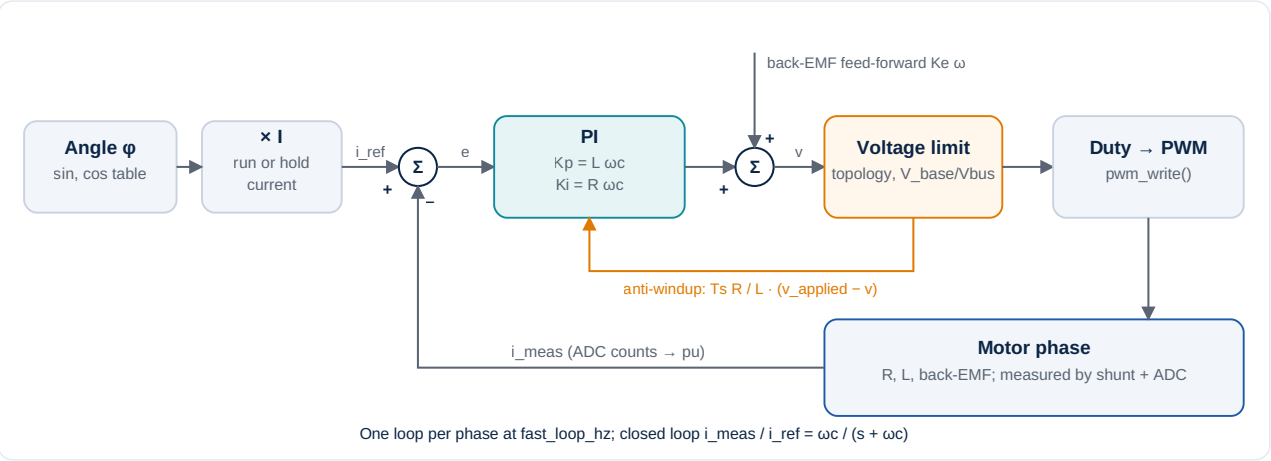
The sine and cosine come from a quarter-wave table with linear interpolation (`SSTEP_CFG_SINE_TABLE_BITS`, default 2^8 intervals, 257 entries in flash). Measured worst-case error over a full turn:

Table bits	Fixed point	Float
6	$1.08 \cdot 10^{-4}$ (3.5 LSB of Q15)	$7.5 \cdot 10^{-5}$
8 (default)	$3.1 \cdot 10^{-5}$ (1.0 LSB of Q15)	$4.8 \cdot 10^{-6}$
10	$3.1 \cdot 10^{-5}$ (1.0 LSB of Q15)	$3.5 \cdot 10^{-7}$

With the fixed-point backend the default table is already at the resolution of Q15; a larger table only helps the float backend.

Between two slow-loop periods the fast loop advances the angle by a constant increment, so the current vector turns smoothly at the PWM rate instead of jumping once per millisecond.

7.4 Current control



Each phase has its own PI controller. The reference comes from the angle and the amplitude, the measurement from the ADC:

$$e = i_{\text{ref}} - i_{\text{meas}} \qquad v = K_p\, e + K_i \int e\, dt + v_{\text{ff}}$$

Gains. `sstep_init()` places the zero of the controller on the pole of the winding (R/L), which leaves an integrator with the crossover frequency $\omega_c = 2\pi \cdot$ `current_bw_hz`:

$$K_p = L\,\omega_c \qquad K_i = R\,\omega_c \qquad \Longrightarrow \qquad \frac{i_{\text{meas}}}{i_{\text{ref}}} = \frac{\omega_c}{s + \omega_c}$$

The closed loop is a first-order low-pass with the requested bandwidth; no manual tuning is needed. The bandwidth must not exceed `fast_loop_hz / 4` (checked by `sstep_init()`).

Back-EMF feed-forward. At speed the motor generates a voltage that the PI would otherwise have to compensate with an error. The library adds it in advance from the commanded speed:

$$v_{A,\text{ff}} = -K_e\,\omega \sin\varphi \qquad v_{B,\text{ff}} = K_e\,\omega \cos\varphi \qquad \omega = \frac{2\pi\, v}{N_s M}$$

with v in microsteps/s. With `ke_uv_s_per_rad = 0` the feed-forward is off.

Anti-windup. When the requested voltage exceeds what the power stage can apply (see Voltage limits (p. 33)), the integrator is corrected by the difference between the applied and the requested voltage (back-calculation with the tracking time constant $T_t = L/R$, gain $T_s R/L$). The integrator does not wind up while the voltage is limited.

Measured response (17HS19-2004S1, 24 V, 20 kHz, 1 kHz bandwidth, locked rotor, step 0 → 2 A, ADC 12 bits):

Topology	Overshoot	Rise time 0-90 %	Settling ±2 %	Final value
Full bridge	1.11 %	0.350 ms	0.400 ms	2.0002 A
Shared leg	1.11 %	0.350 ms	0.400 ms	1.9999 A

The float backend gives the same numbers within 0.01 %. A first-order system with 1 kHz bandwidth rises to 90 % in $2.3/\omega_c = 0.37$ ms; the small overshoot comes from the one-period delay between measurement and PWM update.

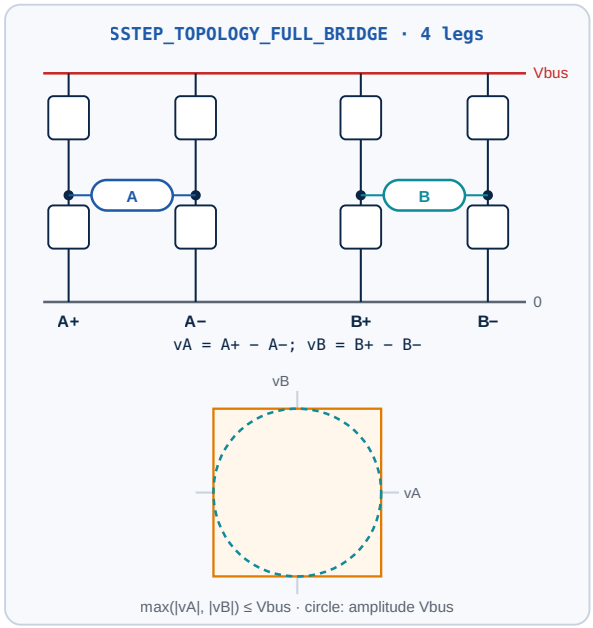
Voltage drive

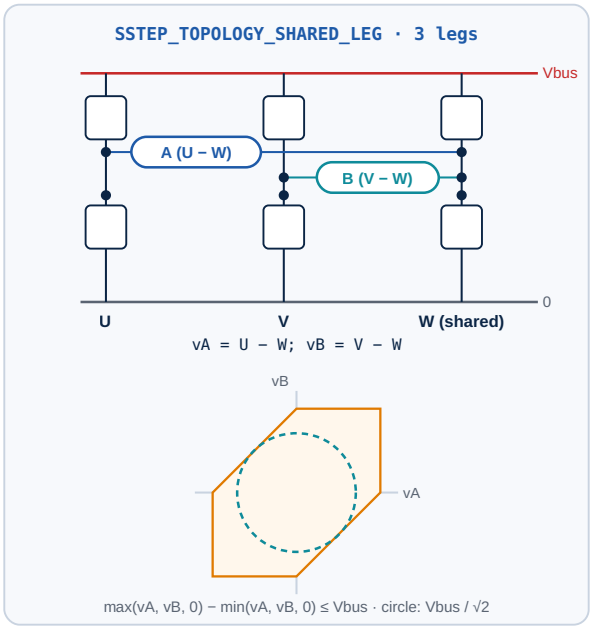
Boards without current sensing use `SSTEP_DRIVE_VOLTAGE`: the controller is replaced by $v = R\, i_{\mathrm{ref}} + v_{\mathrm{ff}}$. The current settles at the reference at standstill (test: 2.0 A within 10 mA) but the torque drops at speed, where the inductance and the error of the back-EMF estimate are not corrected. The overcurrent trip is not active in this mode (there is no current measurement).

7.5 Bus voltage compensation

The controllers compute voltages in per unit of the *nominal* bus voltage. The fast loop converts them to duty cycles with the *measured* bus voltage ($V_{\mathrm{base}}/V_{\mathrm{bus}}$, updated every slow period), so the phase voltage, and with it the current, does not depend on the supply. Test: with the supply at half its nominal value, the current still settles at 2.0 A within 4 mA.

7.6 Voltage limits





The two supported topologies differ in how much phase voltage they can apply:

Topology	Legs	Phase voltages	Limit (relative to Vbus)	Sinusoidal amplitude
`SSTEP_TOPOLOGY_FULL_BRIDGE`	4: A+, A−, B+, B−	$v_A = A^+ - A^-$, $v_B = B^+ - B^-$	$\max(\lvert v_A\rvert, \lvert v_B\rvert) \le 1$	V_{bus}
`SSTEP_TOPOLOGY_SHARED_LEG`	3: U, V, W (shared)	$v_A = U - W$, $v_B = V - W$	$\max(v_A, v_B, 0) - \min(v_A, v_B, 0) \le 1$	about $V_{\mathrm{bus}}/\sqrt{2}$

When the request is beyond the limit, both phase voltages are scaled down by the same factor: the direction of the voltage vector is kept, only its length is reduced, and the anti-windup takes the difference into account.

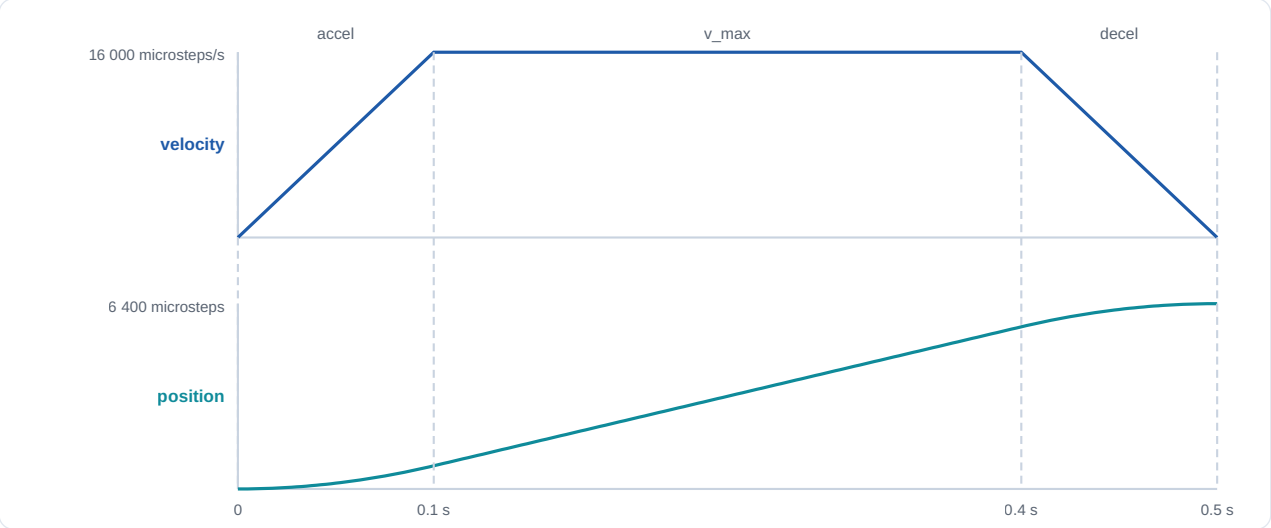
Duty cycles. Full bridge: each phase is centred at 50 %. Shared leg: the shared leg W is placed in the middle of the available range:

$$A^{\pm} = \frac{1}{2} \pm \frac{v_A}{2}, \qquad B^{\pm} = \frac{1}{2} \pm \frac{v_B}{2} \qquad\qquad W = \frac{1}{2} - \frac{\max(v_A,\, v_B,\, 0) + \min(v_A,\, v_B,\, 0)}{2}, \qquad U = v_A + W, \qquad V = v_B + W$$

The fourth compare value is 0 for the shared leg.

Consequence at speed. At 8 rev/s (2 A requested, 24 V) the measured current amplitude is 1.862 A with a full bridge and 1.557 A with a shared leg: the three-leg stage reaches its voltage limit first. Choose the supply voltage for the highest speed of the application.

7.7 Motion profile



The slow loop advances an integer trajectory once per period; positions are Q16 microsteps (64 bits), velocities Q16 microsteps per period, so a position is exact after any number of moves.

Move (`sstep_move_to()`, `sstep_move_by()`). In each period the planner chooses the largest velocity from which it can still stop exactly on the target with the deceleration limit:

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

with the deceleration d and the remaining distance r, both per slow period,

then capped at `v_max` and changed by at most `accel` (speeding up) or `decel` (slowing down) per period. The result is a trapezoid (or a triangle for short moves) that ends on the target with zero velocity, without overshoot. A new target during a move is followed smoothly: there is no velocity step, and a target behind the motor makes it decelerate, reverse and approach from the other side.

Run (`sstep_run()`). Accelerate or decelerate with the limits to the requested signed velocity and keep it. A reversal passes through zero with `decel` and then `accel`.

Stop (`sstep_stop()`). Decelerate to zero with `decel`; the target becomes the position where the motor stops.

Measured (16 microsteps, `v_max` 16 000 microsteps/s, `accel` 160 000 microsteps/s²): a move of 6 400 microsteps (2 revolutions) ends after 499 ms with the rotor at 6400.00 microsteps and a largest load angle of 10.8°; a reversal from +4 to −4 rev/s stays below a load angle of 8.7°.

Resolution. With `slow_loop_hz = 1000` the velocity resolution is $1000/65536 = 0.015$ microsteps/s. `sstep_set_motion_limits()` rejects values that round to zero or reach 2^{30} Q16 per period.

7.8 Hold current

After `hold_delay_ms` without motion the current amplitude drops from `i_run_ma` to `i_hold_ma` and `on_event(SSTEP_EVENT_HOLD_CURRENT)` is reported. Any motion switches back to the run current in the same period. The hold current reduces heating at standstill; the holding torque is $K_t\, i_{\text{hold}}$.

7.9 Stall detection

Without an encoder the library does not know where the rotor is. When the load exceeds the torque, the rotor stops following the commanded angle: it stalls at a hard stop, falls behind at speed, or slips under a load that the hold current cannot hold, while the commanded position keeps its value. The stall detector estimates the back-EMF from the quantities the fast loop already has and raises `SSTEP_FAULT_STALL` when the rotor does not move as commanded. It is off by default (`stall_time_ms = 0`) and needs `SSTEP_DRIVE_CURRENT` and `ke_uv_s_per_rad > 0`.

Reference case. A 17HS19-2004S1 on a three-leg stage at 24 V, 1.5 A run current and 0.5 A hold current after 500 ms, holds a constant 0.12 N·m load at the run current ($K_t \cdot 1.5 = 0.31$ N·m) but not at the hold current (0.10 N·m). At the run → hold change the rotor slips and spins away; without stall detection the driver stays in `SSTEP_STATE_HOLDING` with no fault. With the settings below it reports `SSTEP_FAULT_STALL` 48 ms after the load angle passes 90°.

Principle

The winding equation gives the back-EMF from the applied voltage and the measured current. With the convention of the library ($i_A = I\cos\varphi$, $i_B = I\sin\varphi$), the back-EMF of a rotor at the angle θ and the speed ω is

$$e_A = -K_e\,\omega\sin(p\theta) \qquad e_B = K_e\,\omega\cos(p\theta)$$

Rotated into the frame of the commanded current (d along the current, q 90° ahead):

$$e_d = e_A\cos\varphi + e_B\sin\varphi = K_e\,\omega\sin\delta \qquad e_q = -e_A\sin\varphi + e_B\cos\varphi = K_e\,\omega\cos\delta \qquad \delta = \varphi - p\theta$$

The magnitude of the vector is the rotor speed, its angle the load angle δ.

- **In motion,** a synchronous rotor has $\omega = \omega_c$ and a moderate load angle, so $e_q \approx K_e\,\omega_c\cos\delta$ is a large constant of the sign of the motion. A stalled rotor gives no back-EMF, so e_q falls far below $K_e\,\omega_c$. A rotor that slips turns relative to the commanded frame: the vector rotates, e_q changes its sign, and the negative half is the sign of the slip. A rotor under load keeps its speed, so a low e_q comes with a large $e_d = K_e\,\omega\sin\delta$; that tells it from a blocked rotor.

- **At rest,** the commanded frame stands still, so any lasting back-EMF means that the rotor turns. A slipping rotor makes the vector rotate at its electrical speed.

Equations

These are the exact operations of the library, in their order, so that a model can reproduce them (Scilab, Python). $T_s = 1/$ `fast_loop_hz`, $p = N_s/4$, $M =$ `microsteps`; voltages are per unit of `vbus_nominal_mv` in the fast loop and millivolts in the slow loop.

Fast loop, every period k, after the PI and the voltage limit (phase $x \in \{A, B\}$):

$$\widehat{e}_x[k] = \mathrm{sat}_{\pm 1\,\mathrm{pu}}\big(v_x[k-2] - a\,i_x[k] + b\,i_x[k-1]\big) \qquad a = \frac{L}{T_s} + \frac{R}{2} \qquad b = \frac{L}{T_s} - \frac{R}{2}$$

This is $v - R\,\frac{i[k]+i[k-1]}{2} - L\,\frac{i[k]-i[k-1]}{T_s}$: the winding equation integrated over the period from sample $k-1$ to k with the trapezoidal rule, written with two gains computed by `sstep_init()`. $v_x[k-2]$ is the phase voltage computed two calls earlier: with the one-period computation delay, the compare values written in call $k-2$ are applied from sample $k-1$ to sample k. It is the voltage after the limit of the power stage (what the bridge applies, not the PI output), so the estimate stays valid at the voltage limit.

$$\widehat{e}_d = \widehat{e}_A \cos\varphi[k] + \widehat{e}_B \sin\varphi[k] \qquad \widehat{e}_q = -\widehat{e}_A \sin\varphi[k] + \widehat{e}_B \cos\varphi[k]$$

$$\bar{e}_{d,q}[k] = \bar{e}_{d,q}[k-1] + \frac{T_s}{\tau}\left(\widehat{e}_{d,q}[k] - \bar{e}_{d,q}[k-1]\right)$$

with $\tau =$ `stall_filter_us`. $\varphi[k]$ is the commanded angle of call k. The voltage was averaged over the period before it, so the estimate refers the rotor at the middle of that period to $\varphi[k]$: the load angle reads $\Omega T_s/2$ more than $\varphi[k] - p\theta(t_k)$, 0.45° per rev/s at 20 kHz with 50 pole pairs.

Slow loop, every period n, against the command of the slow period that just ended: velocity v_c [microsteps/s] and its back-EMF $e_c = K_e\,\omega_c = K_e\,2\pi v_c/(N_s M)$ (the feed-forward value, saturated at 1 pu, converted to mV). $\bar{e}_d$, $\bar{e}_q$ are the filter outputs rounded to mV.

$$\text{in motion } (|v_c| \ge v_s): \qquad a = \mathrm{sgn}(v_c)\,\bar{e}_q \qquad \text{bad} = \left(a < \frac{s}{100}\,|e_c|\right) \wedge \left(-\frac{|e_c|}{3} - \frac{RI_r}{10} \le \bar{e}_d \le \frac{|e_c|}{3} + \frac{2RI_r}{5}\right)$$

$$a < -\left(\frac{|e_c|}{2} + e_s\right) \;\Rightarrow\; h \leftarrow \left\lfloor N/2 \right\rfloor$$

$$\text{at rest or slow } (|v_c| < v_s): \qquad \Delta = (\bar{e}_d,\ \bar{e}_q - e_c) \qquad o \leftarrow o + \frac{\Delta - o}{N} \qquad \text{bad} = |\Delta - o| > e_s$$

$$h > 0 \;\Rightarrow\; h \leftarrow h - 1, \quad \text{bad} \leftarrow \text{true} \qquad c \leftarrow \begin{cases} c+1 & \text{bad} \\ \max(c-1,\ 0) & \text{otherwise} \end{cases}$$

and `SSTEP_FAULT_STALL` when $c \ge N$, with $v_s =$ `stall_speed`, $s =$ `stall_sync_pct`, $N =$ `stall_time_ms` · `slow_loop_hz` / 1000, $I_r =$ `i_run_ma` and the slip threshold $e_s = K_e\,2\pi\,v_s/(N_s M)$ rounded to mV. In integers: RI_r is in mV, the divisions truncate, o is held in mV·2^{16}, the update divides with truncation, $\Delta - o$ uses o truncated to mV, and the magnitude is compared as a sum of squares. `sstep_enable()` clears $\bar{e}$, o, c, h and the voltage history ($v[k-1] = v[k-2] = 0$, $i[k-1] =$ the last measurement).

Overcurrent: when the trip of the fast loop finds $c > 0$, the fault is `SSTEP_FAULT_OVERCURRENT | SSTEP_FAULT_STALL`.

Diagnostics in `sstep_get_status()`: `emf_d_mv` and `emf_q_mv` are $\bar{e}_d$, $\bar{e}_q$; `rotor_velocity` is $\mathrm{sgn}(\bar{e}_q)\,|\bar{e}|\,N_s M/(2\pi K_e)$; `load_angle_cdeg` is $\mathrm{atan2}(\sigma\bar{e}_d,\ \sigma\bar{e}_q)$ with $\sigma = \mathrm{sgn}(v_c)$ (at rest $\mathrm{sgn}(\bar{e}_q)$), in 0.01°, positive when the rotor lags in the positive direction (the sign of the motor torque); `stall_level_pct` is $100\,c/N$.

Design decisions

Decision	Reason
Filter in the commanded frame	A synchronous back-EMF is constant there at any speed, so a millisecond filter neither attenuates nor delays it. In the phase frame the back-EMF at 10 rev/s is a 500 Hz sine that such a filter would suppress.
In motion, e_q against $K_e\,\omega_c$	A resistance error (R of a hot winding) adds $\Delta R\,I$ along the current, the d axis, and does not change e_q. An inductance error adds $\Delta L\,I\,p\,\omega$ along q, so the ratio is $\cos\delta + \Delta L\,I\,p/K_e$ at every speed (the second term is −0.22 for L −20 % at 1.5 A, −0.29 at 2 A).
Window on e_d	The L error alone brings a rotor under load down to the threshold: at 2 A and L −20 % a ratio of 0.30 means only $\delta \approx 54°$. A blocked rotor has only the resistance error on d, mostly positive and, for R between −10 % and +40 %, inside the window. A rotor under load is outside it at speed; at low speed the outcome depends on the sign of the motion and on ΔR, so the margin is wider going forward on a hot winding.
Reversed back-EMF holds the condition	The vector of a slipping rotor is above any threshold on e_q for about half of each turn, so the up/down counter alone stays low. $a < -(\lvert e_c\rvert/2 + e_s)$ means δ beyond about 120°: no equilibrium exists there. It keeps the condition true for $N/2$ periods, a turn of the vector faster than that accumulates at full rate, and a single event ends at 50 % of the trip.
Threshold relative to $K_e\,\omega_c$	The synchronous value $\cos\delta$ does not depend on the speed; a fixed fraction separates a loaded synchronous rotor from a stall (about 0, ±0.29 with the L error at 2 A).
Stall flag on the overcurrent trip	A light rotor torn away by a large load reaches the trip in less than `stall_time_ms`, because the back-EMF of a rotor spinning backwards exceeds the bus. The flag tells the application that the position is lost. It only needs the detector to have started ($c > 0$), so it can also appear with an overcurrent of another cause during a transient.
At rest, learned offset	A resting rotor has no back-EMF, so a constant estimate is an observer error: $\Delta R\,I$, the dead-time voltage, current-sensor offsets (0.91 V at 1.5 A in the worst case below, more than the threshold). Its running mean is subtracted. A slipping rotor makes the vector rotate, which averages out of the mean and stays in $\Delta - o$.
Offset time constant = trip time	An offset that appears at once (enable, run → hold) decays below e_s after $N\ln(\mathrm{offset}/e_s)$ periods. That is less than N for an offset below $e \cdot e_s$ (2.7 times the threshold), so it never trips. A slip at $\Omega \gg 1/T_N$ (T_N = `stall_time_ms`) passes the mean unchanged.
Below v_s, deviation from $(0,\ e_c)$	The motor may move slowly right after `sstep_enable()`, without a rest period to learn the offset: the deviation from the expected back-EMF has the same zero mean at rest and in slow motion.
Up/down counter	A stall keeps the condition true: the fault comes N slow periods after it begins. Short events (enable, current steps, resonance, reversal through zero) count down again; a condition true less than half of the time does not accumulate.
Decision in the slow loop	The fast loop only estimates and filters; the comparison, the offset tracking and the counter run at 1 kHz on integer millivolts, identical in both numeric backends.

Choosing the parameters

Field	Example (17HS19, ESC1)	How to choose
`stall_time_ms`	50	Detection delay after the stall; longer is more robust. Keep an offset at rest below $e \cdot e_s$ (see above).
`stall_filter_us`	1000	0.5 to 2 ms. Shorter follows faster slips but passes more current quantization noise: one ADC count changes $\widehat{e}$ by $L\,\Delta i/\tau$ (87 mV for 29 mA at 1 ms). A rotating vector is filtered to at most $K_e/(p\,\tau)$ (4.2 V at 1 ms), which must stay well above e_s.
`stall_speed`	1600 (0.5 rev/s, e_s = 655 mV)	Above the back-EMF noise at rest; the commanded speed from which the synchronism check runs.
`stall_sync_pct`	50	Above $\lvert\Delta L\rvert\, I\, p/K_e$, the value a blocked rotor shows with a larger inductance than configured (29 % at L +20 %, 2 A). A rotor under load with e_d inside the window must stay above it: at 50 %, with the nominal R and L, that ends at δ = 60°, 87 % of the pull-out torque. The window accepts the lag at the voltage limit, so this value also runs at 11 rev/s on a 24 V three-leg stage.

Measured on the motor model

Tests `SstepLoop.Stall*` and `SstepStall.*` with the example settings, fixed point and float backend alike. The worst-case plant differs from the configuration as a real board does: R +30 % (hot winding), L −20 % (saturation), 70 mV dead-time error per leg (24 V, 20 kHz), ESC1 current quantization (29 mA per count instead of 2 mA) and a detent torque of 0.02 N·m.

Case	Result
Reference case, 0.12 N·m at the hold current (nominal and worst-case plant)	`SSTEP_FAULT_STALL` 48 ms after the load angle passes 90°
Rotor blocked at 3 rev/s (hard stop), either direction, both topologies, worst case	`SSTEP_FAULT_STALL` after 50.0 ms
Rotor turning at 1 rev/s with the command at rest	`SSTEP_FAULT_STALL` after 50.0 ms
Rotor imposed at 3 rev/s	$\lvert\bar{e}\rvert$ 3936 mV against $K_e\,\omega$ = 3932 mV; `rotor_velocity` 9609 microsteps/s for 9600; load angle 10.27° (model 10.28°)
Enable, run → hold, 0.07 N·m through run → hold, 6400-microstep move, reversal ±4 rev/s and stop, runs at 0.125, 0.25, 0.47, 1, 2, 8, 9.5 and 11 rev/s	No fault. Largest detector level 6 % (nominal plant), 32 % (worst case)
Heavy axis, 0.5 N·m load step above the pull-out torque, J ×20 and ×100 at 1 and 3 rev/s, either direction, nominal and worst case	`SSTEP_FAULT_STALL` 43 to 60 ms after the load angle passes 90°, 23 to 158 full steps lost
Bare rotor and J ×5, the same load step at 1 rev/s: the overcurrent trip comes first	`SSTEP_FAULT_OVERCURRENT | SSTEP_FAULT_STALL`, 14 and 78 full steps lost
0.30 N·m (72 % of the pull-out torque) at ±1, ±2 and ±3 rev/s, R +30 %, L −20 %; 0.25 N·m at ±1 rev/s, worst case	No fault. Largest detector level 40 % and 58 %

Case	Result
Voltage limit: 8 rev/s at 18 V, 9 rev/s at 20 V, 11 rev/s at 24 V with L +20 %	No fault. Largest detector level 0 %

The motor model of this repository shows no sustained mid-band oscillation between 8 and 11 rev/s; those runs check the voltage limit and the current lag. A 0.09 N·m load at the run → hold change is a real slip in the model (the undamped rotor overshoots past 90°) and is reported.

Cost

	Fixed point	Float
Fast loop, added per period	4 gain multiplications, 4 Q15 multiplications, 2 filter updates, 10 saturating additions	Same operations in float
Fast loop, executed instructions without / with stall detection (`-Os`)	1030 / 1502	726 / 940
Same with `-O2`	664 / 909	683 / 923
Slow loop, added	367 instructions (`-Os`)	318 instructions
`sstep_get_status()`	1202 instructions (`-Os`; integer CORDIC and square root)	1148
RAM per instance, added	96 B (Cortex-M4)	96 B

Instructions counted on a Cortex-M4 image (GCC 15.2) in an instruction-set emulator, mean over 600 fast-loop periods while accelerating to 3 rev/s; on Cortex-M4 most of them take one cycle. At 170 MHz and 20 kHz the observer takes about 1.5 µs (`-O2`) to 2.8 µs (fixed point, `-Os`) of the 50 µs period. With `stall_time_ms = 0` the fast loop skips the observer (one comparison).

Limits

- Below `stall_speed`, a rotor that stops while the command creeps slowly is not seen: its back-EMF is close to zero either way. Only a rotor faster than the command is.
- The fault comes `stall_time_ms` after the stall begins; the position is lost from the beginning of the stall.
- A rotor under load near the voltage limit whose load angle repeatedly swings beyond about 120° is reported, even if it has not lost a step yet (11 rev/s at 20 V: fault 74 to 78 ms after the angle first passes 90°, largest angle 137° to 140° until then). This operating point has no margin; lower the speed or raise the voltage.
- On a hot winding with L −20 % at low speed, going backward, e_d of a rotor under load is close to that of a blocked rotor. There the margin comes from `stall_sync_pct` alone (the formula above).
- During a slip at rest `rotor_velocity` is low (the filter attenuates the rotating vector: about 10 000 for 27 000 microsteps/s); it is accurate in synchronism. An inductance error biases it too (26 700 for 31 400 microsteps/s with L −20 % at 10 rev/s).
- The observer needs the phase currents (`SSTEP_DRIVE_CURRENT`) and K_e.
- Verified on the motor model of the tests; the closed-loop validation with the sfs-stepper-sim plant and on hardware follows.

7.10 Numeric backends

	Fixed point	Float
Select	`SSTEP_MATH_BACKEND=FIXED` (default)	`SSTEP_MATH_BACKEND=FLOAT`
Signals	Q15 in `int16_t`	`float`
Accumulators	Q15 in `int32_t` (±65 536)	`float`, clamped to the same range
Integrators	Q27 in `int32_t` (±16 pu)	`float`, clamped to ±16
Gains	Q15 mantissa and power-of-two exponent (−16 .. 15)	`float`
Arithmetic	Saturating; rounding half up; no undefined shifts	Saturating to the same ranges
Typical target	Cortex-M0/M0+/M3, RISC-V without F extension	Cortex-M4F/M7/M33 with FPU, STM32G4

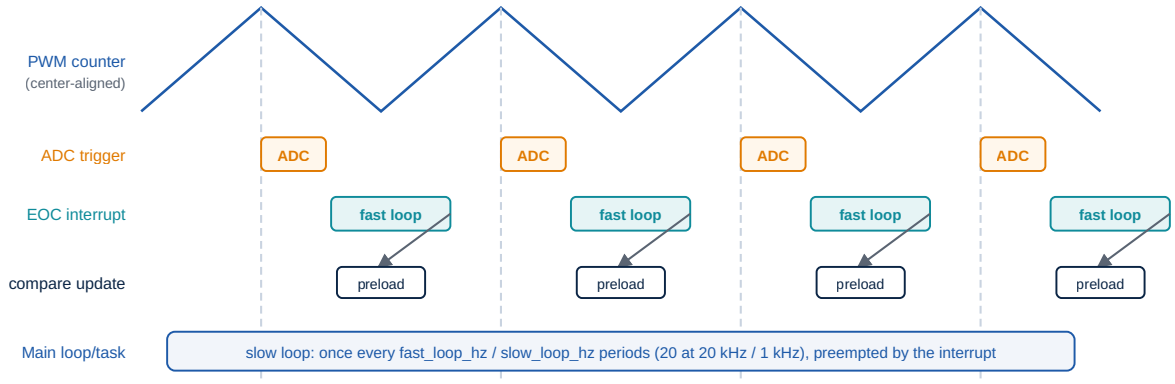
Both backends saturate at the same limits, so the control behaves the same; the unit tests and the closed-loop tests run against both. The motion planner and the configuration are integer code in both backends.

CHAPTER 8

Integration Guide

Stepper-C does not touch hardware. An integration has four parts:

1. **PWM:** a timer with one compare channel per leg, loaded by `pwm_write()`.
2. **Measurement:** ADC conversions of the two phase currents and the bus voltage, synchronized to the PWM, passed to `sstep_fast_loop()` as raw counts.
3. **Fast loop interrupt:** at the end of each conversion sequence, call `sstep_fast_loop()`.
4. **Slow loop:** call `sstep_slow_loop()` and the commands from one task or the main loop at `slow_loop_hz`.



Sample at the center of the period (low-side switches on); the new duty cycles take effect at the next period boundary.

8.1 PWM

- Use **center-aligned** PWM with the period `pwm_period_ticks`; one compare value per leg, in the order of the topology: full bridge `[A+ A– B+ B–]`, shared leg `[U V W]`. A compare value of `pwm_period_ticks` means the high-side switch is on for the whole period.
- Load the compare values with **preload** (shadow registers) so that `pwm_write()` takes effect at the next period boundary, not in the middle of a period.
- Dead time is inserted by the timer or the gate driver; the library does not compensate it in this version.
- `output_enable(false)` must put **every** leg in high impedance (for example the timer's main output enable bit and the gate drivers' enable pins), not only set 50 % duty.

8.2 Current and voltage measurement

- Trigger the ADC from the PWM timer at the **center of the period**, where the low-side switches conduct and the switching noise is furthest away; with low-side shunts this is the only point where the current is visible.
- Convert phase A, phase B and the bus voltage in one sequence, and call `sstep_fast_loop()` from its end-of-conversion interrupt.

- In the shared-leg topology the two measured currents are the phase currents A and B; the shared leg carries their sum.
- Measure `adc_i_zero` with the outputs off after power-up; the library uses the value from the configuration, so re-initialize with the measured value if it differs from the default.

8.3 Timing

Item	Requirement
`fast_loop_hz`	The real rate of the end-of-conversion interrupt; usually the PWM frequency or a fraction of it (every second period at 40 kHz PWM)
`slow_loop_hz`	The real rate of `sstep_slow_loop()`; `fast_loop_hz` must be a multiple of it
Jitter of the slow loop	Tolerated: the fast loop holds the angle if the slow loop is late, and catches up with the next setpoint
Priority	The fast-loop interrupt preempts the slow-loop context, never the other way round
Execution time of the fast loop	Must end well before the next conversion; measure it on your MCU (toggle a pin or read the cycle counter)

The library measures nothing about time: if the configured rates do not match the real ones, the velocities, the accelerations and the hold delay scale accordingly.

8.4 Bare metal

```
int main(void)
{
    board_init(); /* clocks, PWM timer, ADC triggered by the timer, interrupts */
    if ((sstep_init(&motor, &motor_config) != SSTEP_OK) || (sstep_enable(&motor) != SSTEP_
OK))
    {
        for (;;)
        {
        }
    }
    for (;;)
    {
        if (board_tick_1ms()) /* set by SysTick or a timer */
        {
            (void)sstep_slow_loop(&motor);
            /* commands and the application's state machine here */
        }
    }
}
```

8.5 RTOS

Run the slow loop in a periodic task with a fixed period (not a delay after the work), and call every command from that task, or from another task through a queue:

```
#include "FreeRTOS.h"
#include "task.h"

static void motor_task(void* arg)
{
```

```
    TickType_t wake = xTaskGetTickCount();
    (void)arg;
    for (;;)
    {
        vTaskDelayUntil(&wake, pdMS_TO_TICKS(1));
        (void)sstep_slow_loop(&motor);
        /* take commands from a queue and call sstep_move_to() etc. here */
    }
}
```

The fast loop stays in the ADC interrupt; give it a priority above the RTOS kernel's interrupt priority limit only if it calls no RTOS function (the library calls none; check your callbacks).

8.6 Adding the library to a CMake build

```
set(SSTEP_MATH_BACKEND FLOAT)                          # MCU with FPU; default FIXED
include(path/to/Stepper-C/sstep.cmake)
target_link_libraries(app PRIVATE sstep)
target_link_libraries(sstep PUBLIC zephyr_interface)  # Zephyr applications only
```

The backend and the other build-time options are compile definitions of the `sstep` target, so the application sees the same values as the library.

Without CMake: compile `src/*.c`, add the repository root and `inc/` to the include path, and pass the same `-DSSTEP_CFG_…` definitions to the library and the application.

8.7 Bring-up checklist

1. **Outputs off:** read the ADC and check `adc_i_zero` and the bus voltage reading against a multimeter (`sstep_get_status()` shows `vbus_mv` after a fast loop).
2. **Polarity:** start with `drive_mode = SSTEP_DRIVE_VOLTAGE` and a small `i_run_ma`; after `sstep_enable()`, the measured current of phase A must be positive and close to the reference. If it is negative, negate `adc_i_span`.
3. **Current loop:** switch to `SSTEP_DRIVE_CURRENT` with a low run current; look at the phase current with a current probe or the status: it must settle at `i_run_ma`.
4. **Motion:** run slowly (`sstep_run()` with a few hundred microsteps/s), then raise the speed and the acceleration to the application's values.
5. **Protection:** check the trips with a lowered `i_trip_ma` and `vbus_max_mv`, and the external fault input with `sstep_report_fault()`.
6. **Stall detection (optional):** enable it with the example values, log `stall_level_pct`, `emf_d_mv` and `load_angle_cdeg` through every operating point (enable, hold, the fastest move, the heaviest load), then block the shaft by hand at speed and at standstill to see `SSTEP_FAULT_STALL` (choosing the parameters (p. 38)).

CHAPTER 9

API Reference

This chapter documents every public function, type and macro of Stepper-C. Include one header; it declares every function and includes the public type headers:

```
#include "sstep.h"
```

Header	Contents
`sstep.h`	All public functions; library version; includes the headers below
`sstep_config.h`	Build-time configuration
`sstep_defs.h`	Status codes, topologies, fault flags, driver states
`sstep_ctrl.h`	Configuration, callbacks, measurements, status, driver instance

Everything else (`src/`) is private and not installed with the library.

9.1 Version — `sstep.h`

Macro	Value	Meaning
`SSTEP_VERSION_MAJOR`, `…_MINOR`, `…_PATCH`	`0`, `0`, `0`	Version numbers; set by CMake from the latest `vX.Y.Z` tag and stamped into the release package; `0.0.0` means unknown (a header from the repository built without CMake)
`SSTEP_VERSION`	`"0.0.0"`	Version string

9.2 Configuration — `sstep_config.h`

Macro	Default	Meaning
`SSTEP_MATH_BACKEND_FIXED`, `SSTEP_MATH_BACKEND_FLOAT`	`0`, `1`	Values of `SSTEP_CFG_MATH_BACKEND`
`SSTEP_CFG_MATH_BACKEND`	`SSTEP_MATH_BACKEND_FIXED`	Numeric backend
`SSTEP_CFG_TOPOLOGY_FULL_BRIDGE`	`1`	Compile the full-bridge topology
`SSTEP_CFG_TOPOLOGY_SHARED_LEG`	`1`	Compile the shared-leg topology
`SSTEP_CFG_MICROSTEPS_MAX`	`256U`	Largest microstep division
`SSTEP_CFG_SINE_TABLE_BITS`	`8U`	Size of the sine table
`SSTEP_CFG_ARG_CHECKS`	`1`	`NULL` checks in `sstep_fast_loop()`

Ranges and override rules: Configuration (p. 24).

9.3 Definitions — `sstep_defs.h`

Status codes

Macro	Value	Meaning
`SSTEP_OK`	`0`	Success
`SSTEP_ERR_PARAM`	`-1`	`NULL` pointer or a value outside its documented range
`SSTEP_ERR_CONFIG`	`-2`	Configuration consistent field by field but not as a whole
`SSTEP_ERR_STATE`	`-3`	Call not allowed in the current state
`SSTEP_ERR_UNSUPPORTED`	`-4`	Topology not compiled in

Topologies

Value	Legs	Meaning
`SSTEP_TOPOLOGY_FULL_BRIDGE` (0)	4	Two H-bridges: phase A between legs 0 and 1, phase B between legs 2 and 3
`SSTEP_TOPOLOGY_SHARED_LEG` (1)	3	Three half-bridges: phase A between legs 0 and 2, phase B between legs 1 and 2

`SSTEP_LEGS_MAX` (`4U`) is the length of the compare array of `pwm_write()`.

Fault flags

Macro	Value	Raised when
`SSTEP_FAULT_NONE`	`0x0000U`	No fault
`SSTEP_FAULT_OVERCURRENT`	`0x0001U`	A phase current above `i_trip_ma` (current drive)
`SSTEP_FAULT_OVERVOLTAGE`	`0x0002U`	Bus voltage above `vbus_max_mv`
`SSTEP_FAULT_UNDERVOLTAGE`	`0x0004U`	Bus voltage below `vbus_min_mv`
`SSTEP_FAULT_HAL`	`0x0008U`	A callback returned an error
`SSTEP_FAULT_EXTERNAL`	`0x0010U`	Reported by the application with `sstep_report_fault()`
`SSTEP_FAULT_STALL`	`0x0020U`	The rotor does not follow the commanded angle (stall detection, `stall_time_ms` > 0); since 1.1.0. Also together with `SSTEP_FAULT_OVERCURRENT` when the trip comes while the detector is counting

Driver states — `enum sstep_state`

Value	Meaning
`SSTEP_STATE_DISABLED` (0)	Outputs off

Value	Meaning
`SSTEP_STATE_HOLDING` (1)	Outputs on, position held
`SSTEP_STATE_MOVING` (2)	A move or a velocity command is executing
`SSTEP_STATE_FAULT` (3)	Outputs off after a fault; leave with `sstep_clear_faults()`

9.4 Types — `sstep_ctrl.h`

Callbacks

Type	Signature	Called from
`sstep_pwm_write_fn`	`int (*)(void* context, const uint16_t compare[SSTEP_LEGS_MAX])`	`sstep_fast_loop()`
`sstep_output_enable_fn`	`int (*)(void* context, bool enable)`	Enable, disable and fault paths
`sstep_fault_fn`	`void (*)(void* context, uint16_t faults)`	The context that detects a fault
`sstep_event_fn`	`void (*)(void* context, enum sstep_event event)`	`sstep_slow_loop()`

`pwm_write` and `output_enable` return 0 on success and a negative value on error (`SSTEP_FAULT_HAL`). Events (`enum sstep_event`): `SSTEP_EVENT_MOVE_DONE` (0, target reached with zero velocity), `SSTEP_EVENT_HOLD_CURRENT` (1, current reduced at standstill).

Configuration

Type	Members
`struct sstep_config`	`motor`, `stage`, `ctrl`, `hal`
`struct sstep_motor_cfg`	`r_mohm`, `l_uh`, `i_rated_ma`, `ke_uv_s_per_rad`, `full_steps_per_rev`
`struct sstep_stage_cfg`	`topology`, `vbus_nominal_mv`, `vbus_min_mv`, `vbus_max_mv`, `i_full_scale_ma`, `i_trip_ma`, `pwm_period_ticks`, `pwm_freq_hz`, `dead_time_ns`, `adc_i_zero`, `adc_i_span`, `adc_vbus_mv_full`, `adc_full`
`struct sstep_ctrl_cfg`	`drive_mode`, `fast_loop_hz`, `slow_loop_hz`, `microsteps`, `current_bw_hz`, `i_run_ma`, `i_hold_ma`, `hold_delay_ms`, `v_max`, `accel`, `decel`, `stall_time_ms`, `stall_filter_us`, `stall_speed`, `stall_sync_pct` (since 1.1.0)
`struct sstep_callbacks`	`context`, `pwm_write`, `output_enable`, `on_fault`, `on_event`
`enum sstep_drive_mode`	`SSTEP_DRIVE_CURRENT` (0), `SSTEP_DRIVE_VOLTAGE` (1)

Units, ranges and examples of every member: Run-time configuration (p. 24).

Measurement and status

Type	Member	Unit
`struct sstep_sample`	`i_a`, `i_b`	Phase currents, ADC counts (ignored in voltage drive)
	`vbus`	Bus voltage, ADC counts
`struct sstep_status`	`state`, `faults`	`enum sstep_state`, fault flags
	`position`, `target`	microsteps
	`velocity`	microsteps/s (commanded)
	`i_a_ma`, `i_b_ma`	mA (last measured)
	`vbus_mv`	mV (last measured)
	`emf_d_mv`, `emf_q_mv`	mV, filtered back-EMF estimate along and 90° ahead of the commanded current (since 1.1.0)
	`rotor_velocity`	microsteps/s, estimated from the back-EMF (since 1.1.0)
	`load_angle_cdeg`	0.01°, estimated load angle, commanded minus rotor electrical angle (since 1.1.0)
	`stall_level_pct`	%, stall detector level, 100 = trip (since 1.1.0)

The five estimates are 0 while stall detection is off; their equations are in Stall detection (p. 35).

Driver instance

`struct sstep` is an opaque array of `SSTEP_INSTANCE_WORDS` 64-bit words (on Cortex-M4 432 B fixed point, 456 B float; on 64-bit hosts 456 B and 480 B). Allocate it, pass its address, never read or write its content. `SSTEP_CONFIG_BYTES` is the size of the configuration copy (152 B with 32-bit pointers, 176 B with 64-bit pointers) and `SSTEP_STATE_BYTES` the size of the private state behind it.

9.5 Functions — `sstep.h`

All functions return a status code. Positions are in microsteps, velocities in microsteps/s, currents in mA. Every function except `sstep_init()` returns `SSTEP_ERR_STATE` for an instance without a successful `sstep_init()` and then calls no callback (`sstep_fast_loop()` only with `SSTEP_CFG_ARG_CHECKS = 1`); the tables below list the other cases.

sstep_init

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

Validate a configuration and initialize a driver; the outputs stay off. Derives the per-unit bases, the PI gains ($K_p = L\,\omega_c$, $K_i = R\,\omega_c$), the trajectory constants and, with `stall_time_ms` > 0, the stall detector constants without floating point. Calls no callback.

Parameter	Description
`drv`	Driver instance
`cfg`	Configuration; copied

Return	Meaning
`SSTEP_OK`	Success, state `SSTEP_STATE_DISABLED`
`SSTEP_ERR_PARAM`	`NULL` pointer, missing required callback or a field out of range
`SSTEP_ERR_CONFIG`	Inconsistent fields (for example `i_run_ma > i_trip_ma`, stall detection with voltage drive or without `ke_uv_s_per_rad`) or a gain that does not fit
`SSTEP_ERR_UNSUPPORTED`	Topology not compiled in

sstep_enable

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

Switch the outputs on and hold the current position with the run current; resets the current controllers. Does nothing when already enabled.

Return	Meaning
`SSTEP_OK`	Enabled (`SSTEP_STATE_HOLDING`)
`SSTEP_ERR_PARAM`	`drv` is `NULL`
`SSTEP_ERR_STATE`	Faulted, or `output_enable()` failed (then `SSTEP_FAULT_HAL`)

sstep_disable

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

Switch the outputs off at once (the motor coasts, no holding torque) and cancel the command. A faulted driver stays faulted.

Return	Meaning
`SSTEP_OK`	Disabled
`SSTEP_ERR_PARAM`	`drv` is `NULL`

sstep_fast_loop

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

Convert the measurements, check the trip levels, run the current controllers, limit the voltage, compute the leg duty cycles and call `pwm_write()`. Call from the PWM/ADC interrupt at `fast_loop_hz`. The measurements are stored in every state; control and protection run only while the outputs are on.

Parameter	Description
`drv`	Driver instance
`sample`	Raw ADC readings of this period

Return	Meaning
`SSTEP_OK`	Done (faults are reported through the state and `on_fault()`)
`SSTEP_ERR_PARAM`	`NULL` argument; only with `SSTEP_CFG_ARG_CHECKS = 1`

sstep_slow_loop

```c
int sstep_slow_loop(struct sstep* drv);
```

Advance the trajectory by one period, prepare the setpoint of the fast loop, switch between run and hold current, compensate the bus voltage, update the state and report events. Call at `slow_loop_hz`.

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`

sstep_move_to

```c
int sstep_move_to(struct sstep* drv, int32_t position);
```

Move to an absolute position with the trapezoidal profile. Replaces the running command without a velocity step.

Return	Meaning
`SSTEP_OK`	Accepted
`SSTEP_ERR_PARAM`	`drv` is `NULL`
`SSTEP_ERR_STATE`	Disabled or faulted

sstep_move_by

```c
int sstep_move_by(struct sstep* drv, int32_t delta);
```

Move by `delta` relative to the current target.

Return	Meaning
`SSTEP_OK`	Accepted
`SSTEP_ERR_PARAM`	`drv` is `NULL`, or the new target is outside `int32_t`
`SSTEP_ERR_STATE`	Disabled or faulted

sstep_run

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

Accelerate or decelerate to a signed constant velocity and keep it until another command.

Return	Meaning
`SSTEP_OK`	Accepted
`SSTEP_ERR_PARAM`	`drv` is `NULL`, or `|velocity| > v_max`
`SSTEP_ERR_STATE`	Disabled or faulted

sstep_stop

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

Decelerate to standstill with the configured deceleration and hold there. Does nothing when the motor is not moving.

Return	Meaning
`SSTEP_OK`	Accepted or nothing to stop
`SSTEP_ERR_PARAM`	`drv` is `NULL`

sstep_set_position

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

Give the current position a new value (homing). The electrical angle is kept, so the rotor does not move.

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`
`SSTEP_ERR_STATE`	A command is executing

sstep_set_current

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

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

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`, `run_ma` is 0 or a value above 10^7
`SSTEP_ERR_CONFIG`	`hold_ma > run_ma`, or `run_ma` above `i_trip_ma` or `i_rated_ma`

sstep_set_motion_limits

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

Change the velocity limit, the acceleration and the deceleration (`decel = 0`: same as `accel`). They apply from the next slow period, also to a running command.

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`, or a value that rounds to zero or is too large (ranges)

sstep_report_fault

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

Raise fault flags detected by the application (gate driver fault pin, temperature): the outputs go off at once and `on_fault()` is called. May be called from an interrupt. `SSTEP_FAULT_NONE` does nothing.

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`

sstep_clear_faults

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

Clear all fault flags; the driver goes to `SSTEP_STATE_DISABLED` and can be enabled again.

Return	Meaning
`SSTEP_OK`	Done
`SSTEP_ERR_PARAM`	`drv` is `NULL`

sstep_get_status

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

Read a snapshot of the state, faults, commanded motion, the last measurements and the stall detector estimates in engineering units. With stall detection on it also converts the back-EMF estimate (a square root and a CORDIC arctangent, about 1200 instructions on Cortex-M4): call it from the slow-loop context, not from the fast loop.

Parameter	Description
`drv`	Driver instance
`status`	Filled on success

Return	Meaning
`SSTEP_OK`	`status` filled
`SSTEP_ERR_PARAM`	A `NULL` pointer

9.6 Return codes summary

Function	OK	ERR_PARAM	ERR_CONFIG	ERR_STATE	ERR_UNSUPPORTED
`sstep_init()`	✓	✓	✓		✓
`sstep_enable()`	✓	✓		✓	
`sstep_disable()`	✓	✓		✓[1]	
`sstep_fast_loop()`	✓	✓		✓[1]	
`sstep_slow_loop()`	✓	✓		✓[1]	
`sstep_move_to()`, `sstep_move_by()`, `sstep_run()`	✓	✓		✓	
`sstep_stop()`	✓	✓		✓[1]	
`sstep_set_position()`	✓	✓		✓	
`sstep_set_current()`	✓	✓	✓	✓[1]	
`sstep_set_motion_limits()`	✓	✓		✓[1]	
`sstep_report_fault()`, `sstep_clear_faults()`, `sstep_get_status()`	✓	✓		✓[1]	

[1] Only for an instance without a successful `sstep_init()`.

CHAPTER 10

MISRA C:2012 compliance

This page records how the library is checked against MISRA C:2012 and every deviation from it. It covers the library code only: `src/`, `inc/` and `sstep.h`.

10.1 Checking

Check	Where	Gate
Cppcheck MISRA C:2012 addon	`misra-analyze`, CI *MISRA C:2012 Analysis*	Fails on any violation
Cppcheck, all checks	`cppcheck-analyze`, CI *Cppcheck Analysis*	Report
CodeQL `cpp-queries`	`codeql-analyze`, CI *CodeQL Analysis*	Report
Compiler	`-std=c11 -Wall -Wextra -Wpedantic -Wconversion`	No warnings
Manual review	Pull request review	Reviewer

10.2 Current status

Item	Result
Cppcheck MISRA addon	0 violations (Cppcheck 2.16.2)
Unit test coverage of `src/`	100 % lines, branches and functions, fixed point and float (gcovr 8.6)
Deviations	1 (DEV-01)

10.3 Manual review notes

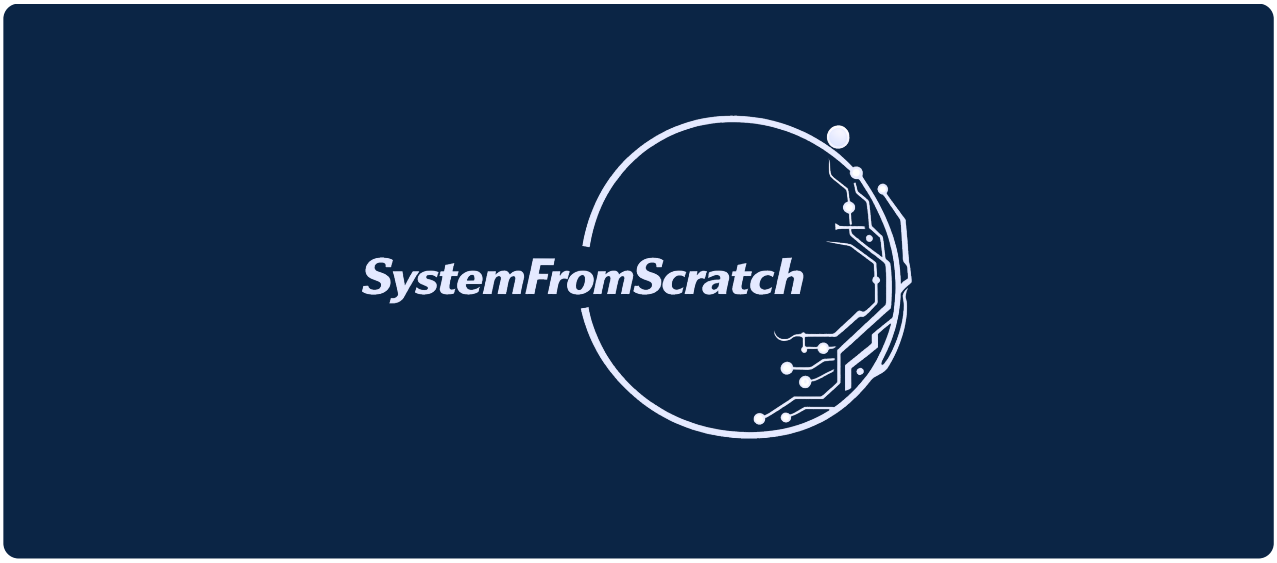
Guideline	How the library complies
Dir 4.1 (run-time failures)	Every arithmetic operation of the numeric backend saturates; shifts of signed values use `sstep_asr()`; divisions are by validated non-zero values; the gains are range-checked by `sstep_init()`.
Dir 4.11 (validity of arguments)	Every public function checks its pointers and ranges and returns `SSTEP_ERR_PARAM`; `SSTEP_CFG_ARG_CHECKS = 0` removes only the `NULL` checks of `sstep_fast_loop()`.
Dir 4.12, Rule 21.3 (dynamic memory)	No heap.
Rule 15.5 (single point of exit)	One `return` per function.
Rule 17.2 (recursion)	No recursion.
Rule 21.6 (standard I/O)	No `<stdio.h>`.

10.4 Deviation register

DEV-01: Rule 11.3 (required), opaque driver instance

- **Location:** `sstep_ctx_of()` and `sstep_ctx_of_const()` in `src/sstep_internal.h`, the only casts from `struct sstep *` to `struct sstep_ctx *`. Marked with `cppcheck-suppress misra-c2012-11.3`.
- **Rationale:** The application allocates the driver instance (no heap, no global state), so it must know its size, but not its layout. The public `struct sstep` in `sstep_ctrl.h` is an array of `uint64_t` of that size; the private `struct sstep_ctx` holds the state. This keeps the internal types (gains, integrators, setpoints, numeric backend) out of the application code and lets them change without breaking the API.
- **Risk control:** `sstep_ctrl.c` checks at compile time that `struct sstep` is at least as large and as aligned as `struct sstep_ctx`; a layout change that outgrows `SSTEP_STATE_BYTES` does not compile. The application never accesses `struct sstep::opaque`, so the storage is only ever accessed as `struct sstep_ctx`. The cast is applied to every public entry point before any member access, and the NULL check is done on its result (a NULL pointer converts to NULL).

Author & Company



Stepper-C 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 Stepper-C, Anton wrote the library, its unit tests and closed-loop tests on a motor model, and this documentation.

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 on GitHub (github.com/AntonVynohradov/Stepper-C/issues). Please include the library version, the numeric backend, your compiler and target, the `struct sstep_config` values, and — for control problems — the status or a phase-current capture.
- **Contributions:** see Contributing to the library (p. 23).
- **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.

Motor and board names (17HS19-2004S1, B-G431B-ESC1, STM32) appear only as examples of hardware the library can drive. STM32 is a registered trademark of STMicroelectronics International N.V.; Stepper-C is not affiliated with or endorsed by STMicroelectronics or any motor manufacturer.

APPENDIX A

Changelog

All notable changes to this project 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).

A.1 1.2.0 - 2026-11-06

Fixed

- **Stall detection missed a slipping rotor at low speed.** The back-EMF vector of a slipping rotor turns in the commanded frame and stayed above `stall_sync_pct` about half of the time, so the up/down counter did not accumulate: a heavy axis (load inertia ×100) lost hundreds of full steps without a fault. A reversed back-EMF now keeps the condition true for half of the trip time: `SSTEP_FAULT_STALL` 43 to 60 ms after the load angle passes 90°.
- **False stall under a heavy load.** The inductance error adds to $\cos\delta$ (it does not scale it, as the documentation said), so a synchronous rotor at 72 % of the pull-out torque with R +30 %, L −20 % tripped. The condition in motion now also requires the d axis inside the window of a rotor that does not turn. The same window removes the false trip at the voltage limit: 11 rev/s at 24 V now runs with the example `stall_sync_pct` of 50.

Changed

- `SSTEP_FAULT_OVERCURRENT` comes **together with** `SSTEP_FAULT_STALL` when the trip finds the stall detector counting: a light rotor reaches the trip before `stall_time_ms`, and the position is lost. An application that compares `faults` with a single flag must test the bits. Without stall detection nothing changes.
- A load angle swinging repeatedly beyond about 120° near the voltage limit is reported as a stall even before a step is lost (documented in Limits).
- `SSTEP_STATE_BYTES` on 64-bit hosts: 288 B fixed point, 312 B float (Cortex-M4 unchanged: 280 B, 304 B). Flash on Cortex-M4 (`-Os`): 9216 B fixed point, 9894 B float. Worst-case stack of `sstep_slow_loop()` 152 B fixed point (float 140 B); the slow loop executes about 145 more instructions per period with stall detection on (`-Os`); the fast loop is unchanged.
- The version comes only from the `vX.Y.Z` Git tag. `sstep.cmake` now applies it to the applications that include it, and the headers of the release package carry it. A header taken from the repository and built without CMake reports `0.0.0` (unknown). `CITATION.cff` no longer states a version or a release date.

A.2 1.1.0 - 2026-11-01

Sensorless stall detection. Compatible with 1.0.0: the new configuration fields are zero in existing initializers, which keeps stall detection off and the behaviour unchanged; rebuild the application with the new headers (the instance size changed).

Added

- **Stall detection** (`src/sstep_stall.c`): a back-EMF observer in the fast loop (e = v − R·i − L·di/dt from the applied voltage, the measured currents and the commanded angle, filtered in the commanded current frame) and a decision in the slow loop. In motion it checks that the back-EMF along the commanded motion stays above `stall_sync_pct` of Ke·ω; at rest and at low speed it checks that the rotor does not turn faster than the command by more than `stall_speed`, with the observer offset (resistance, dead time) learned while the rotor follows. An up/down counter raises the new fault `SSTEP_FAULT_STALL` (`0x0020U`) after `stall_time_ms`, with the usual fault behaviour (outputs off, `SSTEP_STATE_FAULT`, `on_fault()`, cleared by `sstep_clear_faults()`).
- **Configuration** (`struct sstep_ctrl_cfg`): `stall_time_ms` (0 = off, the default), `stall_filter_us`, `stall_speed`, `stall_sync_pct`. `sstep_init()` returns `SSTEP_ERR_CONFIG` for stall detection with voltage drive or without `ke_uv_s_per_rad`.
- **Status** (`struct sstep_status`): `emf_d_mv`, `emf_q_mv`, `rotor_velocity`, `load_angle_cdeg`, `stall_level_pct`.
- **Documentation:** the stall detection section of Motor Control (principle, the exact equations for a model, design decisions, parameter choice, measured results, cost, limits), configuration, usage, integration and API updates. New hero illustrations (`docs/assets/images/hero-dark.png` for the website and the PDF cover, `hero-light.png` for the README).

Changed

- `struct sstep` grows by 96 B (432 B fixed point, 456 B float on Cortex-M4): `SSTEP_CONFIG_BYTES` 152 / 176 B, `SSTEP_STATE_BYTES` 280 / 304 B. `struct sstep_status` grows from 32 B to 48 B.
- Flash on Cortex-M4 (`-Os`): 9040 B fixed point, 9718 B float. Worst-case stack of `sstep_fast_loop()` 216 B fixed point (128 B float), of `sstep_get_status()` 92 B.
- With stall detection on, the fast loop executes about 240 more instructions per period (`-O2`, both backends; 475 fixed point and 213 float with `-Os`).

Quality

- 93 GoogleTest tests (fixed point) and 92 (float), 100 % line, branch and function coverage of `src/` in both backends. The motor model of the tests gained a detent torque, a dead-time voltage error, a configurable current quantization and an imposed rotor motion for synthetic tests.
- Stall detection on the motor model: the reference case (17HS19-2004S1, three-leg stage, 24 V, 0.12 N·m held at 1.5 A but not at the 0.5 A hold current) reports `SSTEP_FAULT_STALL` 48 ms after the load angle passes 90°; a hard stop at 3 rev/s after 50 ms. No fault through enable, run → hold, moves, reversal and runs from 0.125 to 11 rev/s, also with R +30 %, L −20 %, 70 mV dead time per leg, 29 mA current quantization and detent torque (largest detector level 32 %).
- MISRA C:2012: 0 violations, no new deviation.

A.3 1.0.0 - 2026-10-31

First release: closed-loop microstepping current control for two-phase hybrid stepper motors in portable C11.

Added

- **Public API** (`sstep.h`, 15 functions): `sstep_init()`, `sstep_enable()`, `sstep_disable()`, `sstep_fast_loop()`, `sstep_slow_loop()`, `sstep_move_to()`, `sstep_move_by()`, `sstep_run()`, `sstep_stop()`, `sstep_set_position()`, `sstep_set_current()`, `sstep_set_motion_limits()`, `sstep_report_fault()`, `sstep_clear_faults()`, `sstep_get_status()`.
- **Types** (`inc/sstep_ctrl.h`, `inc/sstep_defs.h`): configuration in integer engineering units (motor, power stage and measurement, control and motion), hardware callbacks given at initialization (`pwm_write`, `output_enable`, optional `on_fault`, `on_event`), raw ADC sample, status in engineering units, status codes, fault flags, driver states. `pwm_freq_hz` and `dead_time_ns` are reserved for later versions.
- **Opaque driver instance:** `struct sstep` is `uint64_t opaque[SSTEP_INSTANCE_WORDS]`, allocated by the application (336 B fixed point, 360 B float on Cortex-M4); the driver state and every numeric type stay private in `src/`.
- **Current control:** PI controller per phase with gains derived from R, L and the bandwidth (`Kp = L wc`, `Ki = R wc`), back-EMF feed-forward, back-calculation anti-windup, bus voltage compensation; open-loop voltage drive for boards without current sensing.
- **Microstepping:** 1 to 256 microsteps per full step, 32-bit electrical angle, quarter-wave sine table with linear interpolation (generated by `tools/sstep_gen_sin_table.py`), angle interpolated at the fast-loop rate.
- **Motion planner:** trapezoidal moves without overshoot in integer arithmetic (positions in 1/65536 microstep, 64 bits), separate acceleration and deceleration, smooth retargeting, constant-velocity runs, stop, homing with `sstep_set_position()`, hold current after a standstill delay, `SSTEP_EVENT_MOVE_DONE` and `SSTEP_EVENT_HOLD_CURRENT`.
- **Power stages:** two H-bridges (4 legs) and three half-bridges with a shared leg (3 legs); the voltage request is limited to the topology with its direction kept.
- **Protection:** overcurrent, overvoltage and undervoltage trips in the fast loop, callback errors and external faults switch the outputs off in the same period. Every function except `sstep_init()` returns `SSTEP_ERR_STATE` for an instance that was not initialized successfully, without touching the hardware.
- **Numeric backends:** fixed point (Q15/Q31, integer only) and single-precision float, selected with `SSTEP_MATH_BACKEND` (CMake) or `SSTEP_CFG_MATH_BACKEND`.
- **Build-time options** (`inc/sstep_config.h`, overridable in `sstep_user_config.h`): topologies, largest microstep division, sine table size, argument checks of the fast loop.
- **Build:** CMake library target in `sstep.cmake`, presets for Debug, Release and the unit tests of both backends; `misra-analyze`, `cppcheck-analyze` and `codeql-analyze` targets.
- **Documentation:** website and PDF manual: introduction, getting started, architecture, usage, extending, configuration with the measured stack and memory footprint, motor control theory with measured results, integration guide, API reference, MISRA compliance.

Quality

- 78 GoogleTest tests (fixed point) and 77 (float): numeric backend, power stage, motion planner, API and closed loop on a motor model (`tests/unit/sstep_test_plant.hpp`); 100 % line, branch and function coverage of `src/` in both backends.
- Closed loop on the motor model (17HS19-2004S1, 24 V, 20 kHz, 1 kHz bandwidth), matching the Scilab prototype: current step overshoot 1.11 %, settling 0.40 ms; 8 rev/s current 1.862 A with two H-bridges and 1.557 A with a shared leg; load angle at 0.25 N·m 37.21°.
- MISRA C:2012: 0 violations (Cppcheck 2.13 and 2.16), one documented deviation (DEV-01, Rule 11.3, opaque instance).

- No warnings with `-std=c11 -Wall -Wextra -Wpedantic -Wconversion` (GCC on x86-64 and `arm-none-eabi-gcc` for Cortex-M4); clean under AddressSanitizer and UndefinedBehaviorSanitizer.
- Cortex-M4 (GCC 15.2, `-Os`): 6980 B flash fixed point, 7446 B float; worst-case stack of `sstep_fast_loop()` 184 B / 128 B; `.data` and `.bss` empty.
- Sine table error (default 8 bits): 1.0 LSB of Q15 (fixed point), 4.8·10^-6 (float).

APPENDIX B

License

Stepper-C is 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 (`sstep.h`, `inc/`, `src/`), tests, tools and this documentation

B.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.

B.2 What it requires

When you distribute Stepper-C 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 (Stepper-C has 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.

B.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 System-FromScratch® to endorse your product.

B.4 Trademarks

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. Motor and board names (17HS19-2004S1, B-G431B-ESC1, STM32) appear only as examples of hardware the library can drive. STM32 is a registered trademark of STMicroelectronics International N.V.; Stepper-C is not affiliated with or endorsed by STMicroelectronics or any motor manufacturer.

B.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.
```