PicoMite Stepper Motor Control Reference

1. Overview

The STEPPER command provides a comprehensive system for controlling up to 4 stepper motor axes (X, Y, Z, and a rotary A axis) with support for G-code execution, acceleration planning, and hardware limit switches. It uses a dedicated 100kHz interrupt timer for smooth pulse generation.

2. Initialization & Configuration

2.1 Initialization

```
STEPPER INIT [arc_tolerance] [,buffer_size] [,estop_pin] [,estop_keep_enabled]
```

Initializes the stepper subsystem. Must be called before any other STEPPER commands.
- arc_tolerance: (Optional) Tolerance for arc segmentation in mm (default: 0.05).
- buffer_size: (Optional) Size of the G-code lookahead buffer (default: 32, max: 1024).
- estop_pin: (Optional) Hardware emergency-stop input pin (active low).
- estop_keep_enabled: (Optional) 0 (default) = disable drivers on E-STOP, 1 = keep drivers enabled on E-STOP.

2.2 Axis Configuration

```
STEPPER AXIS axis, step_pin, dir_pin [, enable_pin] [, dir_invert] [, steps_per_unit] [, max_vel] [, max_accel] [, home_backoff_mm]
```

Configures a specific axis (X, Y, Z, or A).
- axis: X, Y, Z, or A. A is a rotary slave axis (units are degrees).
- step_pin/dir_pin: GPIO pins for Step and Direction signals.
- enable_pin: (Optional) GPIO pin for Enable signal (active low).
- dir_invert: (Optional) 1 to invert direction, 0 otherwise.
- steps_per_unit: (Optional) Steps required to move 1mm (X/Y/Z) or 1 degree (A).
- max_vel: (Optional) Maximum velocity in mm/min (X/Y/Z) or deg/min (A).
- max_accel: (Optional) Maximum acceleration in mm/s^2 (X/Y/Z) or deg/s^2 (A).
- home_backoff_mm: (Optional) Homing switch clear/backoff distance in mm (X/Y/Z only; ignored for A).

Step-rate limit: the emitted step rate is capped at 50 kHz - one step every two 100 kHz ISR ticks, so the low interval between pulses is always a full ISR period. If max_vel x steps_per_unit exceeds 50000 steps/s the axis cannot reach the requested speed. This is not an error: the axis is configured and runs at the capped speed, and a warning is printed reporting the actual maximum achievable (50000 / steps_per_unit, shown in mm/min, or deg/min for A). To raise an axis' top speed, reduce its steps_per_unit (e.g. coarser microstepping).

2.3 Live Re-tuning

```
STEPPER TUNE axis [, dir_invert] [, steps_per_unit] [, max_vel] [, max_accel] [, home_backoff_mm]
```

Adjusts the motion parameters of an already-configured axis without closing and re-initialising the stepper subsystem - useful for interactively tuning an axis. Unlike STEPPER AXIS, it never changes the step/dir/enable pins.
Only permitted when no motion is executing and the G-code buffer is empty.

Any subset of parameters may be supplied; omit a value (leave the comma) to keep the current setting. At least one parameter must be given.
- axis: X, Y, Z, or A.
- dir_invert: (Optional) 1 to invert direction, 0 otherwise.
- steps_per_unit: (Optional) Steps to move 1mm (X/Y/Z) or 1 degree (A).
- max_vel: (Optional) Maximum velocity in mm/min (X/Y/Z) or deg/min (A).
- max_accel: (Optional) Maximum acceleration in mm/s^2 (X/Y/Z) or deg/s^2 (A).
- home_backoff_mm: (Optional) Homing switch clear/backoff distance in mm (X/Y/Z only).

max_vel, max_accel, dir_invert and home_backoff_mm take effect on the next move.
Changing steps_per_unit preserves the raw step count (the motor has not physically moved), so the known machine position is invalidated: the configured soft limits are rescaled to keep the same working envelope, and you must re-home (G28) or re-issue STEPPER POSITION before further motion can be queued.
As with STEPPER AXIS, the 50 kHz step-rate cap applies: if the resulting max_vel x steps_per_unit exceeds 50000 steps/s a non-fatal warning reports the actual maximum achievable speed.
Example: STEPPER TUNE X,,80 ' change only steps/mm
Example: STEPPER TUNE Y,,,3000,750 ' change only max velocity and acceleration

2.4 Limits & Safety

```
STEPPER HWLIMITS x_min, y_min, z_min [,x_max, y_max, z_max [,invert_mask]]
```

Configures hardware limit switch pins. Pins must be mutually exclusive with AXIS/SPINDLE/E-STOP, except min and max may share the same pin on the same axis. Use 0 in any max-pin slot to leave it unassigned.

Default polarity is active-low (triggered when grounded); the firmware enables a pull-up on each such pin so an open switch reads HIGH. The optional invert_mask marks pins as active-high instead - useful when wiring a TMC2209 DIAG output directly to the input. Active-high pins are configured with a pull-down.

invert_mask bit positions:
bit 0 = X_MIN bit 1 = Y_MIN bit 2 = Z_MIN
bit 3 = X_MAX bit 4 = Y_MAX bit 5 = Z_MAX
Example: invert_mask = 7 marks all three minimum pins as active-high.

If a min and max on the same axis share one physical pin, the corresponding invert_mask bits must agree (a single GPIO can only have one polarity). Mismatched bits are rejected at parse time.

```
STEPPER LIMITS axis, min_mm, max_mm
```

Sets soft limits for an axis (X, Y, or Z) in mm. The A axis has no soft limits.

```
STEPPER ESTOP
```

Software emergency stop: halts motion immediately, clears buffer, and turns spindle off. Drivers are disabled unless estop_keep_enabled was set to 1 in STEPPER INIT.
If an INIT estop_pin is configured, hardware E-STOP is monitored in ISR and terminates processing immediately under all circumstances (including homing).

3. Motion Control

3.1 G-Code Execution

```
STEPPER GC <gcode> [words...]
```

Executes a standard G-code command string.
Supported codes: G0, G1, G2, G3, G4, G28, G90, G91, G92, M3, M5.
Supported words: X, Y, Z, A, F, I, J, K, R, P.
M3/M5 and G4 are buffered and executed in-order with motion blocks.
G28 homes specified X/Y/Z axes (A axis is not homed). The A word is ignored on G2/G3 arcs.
Feedrate semantics: combined XYZ+A moves use mm/min over the XYZ Euclidean distance with A as a slave axis. A-only moves (no XYZ change) treat F as deg/min over |dA|.
Example: STEPPER GC G1 X10 Y5 F500
Example: STEPPER GC G1 A90 F360

```
STEPPER GS string$
```

Executes a G-code command supplied as a string expression. Accepts any MMBasic string variable, literal, or expression and passes it to the same G-code parser as STEPPER GC.
This is useful when G-code lines are constructed dynamically at runtime.
Supported codes: G0, G1, G2, G3, G4, G28, G90, G91, G92, M3, M5.
Supported words: X, Y, Z, A, F, I, J, K, R, P.
Example: STEPPER GS "G1 X" + STR$(x_pos) + " Y" + STR$(y_pos) + " F500"
Example: STEPPER GS gcode$

```
STEPPER GCODE G0|G1|G2|G3|G4|G28|G90|G91|G92|M3|M5 [, X, x] [, Y, y] [, Z, z] [, A, a] [, F, feed] [, I, i] [, J, j] [, K, k] [, R, r] [, P, ms]
```

Alternative syntax for adding motion commands to the buffer. All parameters must be comma separated. The A axis is rotary; A units are degrees and the A word is ignored on G2/G3 arcs.
G4 uses P in milliseconds (for example: STEPPER GCODE G4, P, 500).
Example: STEPPER GCODE G1, X, 10, Y, 5, F, 500
Example: STEPPER GCODE G1, A, 90, F, 360

3.2 Manual Positioning

```
STEPPER POSITION HOME
```

Sets all axes to position 0 and clears G92 offsets.

```
STEPPER POSITION axis, position
```

Sets the current position of an axis (X/Y/Z in mm; A in degrees) to the specified value.

4. Advanced Features

```
STEPPER SCURVE 0|1
```

Enables (1) or disables (0) S-curve acceleration profiling for smoother motion.

```
STEPPER JERK value
```

Sets the jerk limit in mm/s^3 for S-curve planning.

```
STEPPER SPINDLE pin [,invert]
```

Configures a spindle control pin used by buffered M3/M5 commands. Pin must not conflict with AXIS/HWLIMITS/E-STOP pins.

5. System Management

```
STEPPER RUN [,0|1]
```

Arms the system and begins executing buffered commands.
Optional mode argument controls behavior when the queue drains:
- 0 (default): remain armed with drivers enabled while idle. The driver continues to apply its programmed standstill / hold current (for example, IHOLD on a TMC2209 configured via TMC22xx). Use this mode when the load could shift, drift, or back-drive if torque is removed.
- 1: when idle and no buffered work remains, drivers are disabled and then re-enabled automatically when new queued work arrives. With the enable pin de-asserted, current drops to zero - the motor freewheels and the driver runs cool, but holding torque is lost. Use this mode for axes that can sit unpowered between moves (rotary tool changers, indexers, demo rigs).

```
STEPPER CLEAR
```

Clears the G-code buffer (only when motion is idle).

```
STEPPER STATUS
```

Displays detailed system status, including axis positions, buffer state, and configuration (including auto-disable-on-idle mode).

```
STEPPER CLOSE
```

Shuts down the stepper subsystem, releases resources, and deconfigures stepper-owned GPIO pins.

```
PEEK(STEPPER X)
PEEK(STEPPER Y)
PEEK(STEPPER Z)
PEEK(STEPPER A)
PEEK(STEPPER ACTIVE)
PEEK(STEPPER STATUS)
PEEK(STEPPER BUFFER)
```

Returns current workspace axis position in mm (X/Y/Z) or degrees (A), or active state (ACTIVE returns 1 when stepper is actively processing queued work, 0 when idle, or -1 if the stepper subsystem has not been initialized).
STATUS returns a bitmap of safety and coordinate state:
- bit0: X_MIN limit asserted
- bit1: X_MAX limit asserted
- bit2: Y_MIN limit asserted
- bit3: Y_MAX limit asserted
- bit4: Z_MIN limit asserted

- bit5: Z_MAX limit asserted
- bit6: E-STOP asserted
- bit7: position known (machine coordinates established).
BUFFER returns the number of free slots in the G-code circular buffer. This can be used to throttle G-code submission and avoid overflowing the buffer.

6. Rotary A Axis

The A axis is an optional fourth axis intended for rotary motion (degrees). It runs as a slave axis of the same Bresenham planner as X/Y/Z, but with rotary semantics:
- Units are degrees. steps_per_unit is steps/degree; max velocity input is deg/min (stored as deg/s); max acceleration is deg/s^2.
- The A axis has no soft limits, no homing (G28 ignores A), and no hardware limit switch input. It is therefore not subject to STEPPER LIMITS or STEPPER HWLIMITS.
- For combined moves with XYZ, A travels alongside as a slave; the F word is mm/min over the XYZ Euclidean distance and A is clamped against its max velocity if it would otherwise exceed it.
- For A-only moves (no change in X, Y, or Z), the F word is treated as deg/min over |dA|.
- The A word is ignored on G2/G3 arc commands.
- G92 A<value> is supported and sets the A workspace offset.

7. TMC22xx Driver Configuration

The TMC22xx command configures a Trinamic TMC2208 or TMC2209 stepper motor driver over its single-wire UART interface. It is a one-shot initialisation command: it writes five configuration registers to the driver and returns. The driver retains its configuration until power is removed or the command is re-issued. The TMC22xx command is independent of the STEPPER subsystem and may be called before or after STEPPER INIT.

7.1 Syntax

```
TMC22xx tx_pin, chip_type, slave_addr, i_run_mA, i_hold_pct, microsteps [, r_sense_ohm [, stealthchop]]
TMC22xx SGTHRS slave_addr, value
TMC22xx TCOOLTHRS slave_addr, value
TMC22xx CLOSE
```

Parameters:
- tx_pin: GPIO pin number connected via a 1 k-ohm resistor to the PDN_UART pin of the driver module. The pin is driven as a bit-banged UART transmitter at 115200 baud and is reserved until TMC22xx CLOSE is called.
- chip_type: Driver chip model - either 2208 (TMC2208) or 2209 (TMC2209).
- slave_addr: UART slave address. Must be 0 for the TMC2208. For the TMC2209, may be 0, 1, 2, or 3 to match the MS1/MS2 pin strapping on the module.
- i_run_mA: RMS run current in milliamps (0.0 to 2000.0). The driver's current-scale register (CS) is computed from this value and r_sense_ohm.
- i_hold_pct: Holding current as a percentage of the run current (0 to 100). The motor is held at this fraction of i_run_mA when idle after the TPOWERDOWN delay expires.

- microsteps: Microstep resolution. Must be one of: 1, 2, 4, 8, 16, 32, 64, 128, or 256. Interpolation to 256 microsteps is always enabled in hardware.
- r_sense_ohm: (Optional) Sense resistor value in ohms (0.08 to 0.2). Defaults to 0.11, which is correct for most common breakout modules.
- stealthchop: (Optional) 1 (default) to enable StealthChop2 (quiet, constant-current PWM mode). 0 to use SpreadCycle (high-performance chopper, louder).

TMC22xx CLOSE releases the tx_pin reservation.

7.2 Hardware Notes

The TMC2208 and TMC2209 PDN_UART pin is open-drain and must be pulled up to VIO (3.3 V). Connect the PicoMite GPIO pin to PDN_UART through a 1 k-ohm series resistor to limit current when the pin drives low. Most commercial breakout modules include the pull-up resistor.

For the TMC2209, the UART slave address (0-3) is set by the MS1 and MS2 strapping pins on the module. When multiple TMC2209 drivers share a single UART line, each must be given a unique address and the same tx_pin may be wired to all modules. Only the addressed device responds.

The TMC2208 has a fixed hardware slave address of 0 and does not support multi-drop addressing.

7.3 Configured Registers

TMC22xx writes five registers in sequence with a 200 us inter-frame gap:
- GCONF (0x00): enables PDN_UART mode, register-based microstep selection, and microstep filter; selects StealthChop2 or SpreadCycle chopper.
- IHOLD_IRUN (0x10): sets hold current CS, run current CS, and a power-down delay of 8 clock cycles.
- TPOWERDOWN (0x11): sets the power-down delay to 20 (approx. 327 ms at 12 MHz internal clock) before the hold current takes effect.
- CHOPCONF (0x6C): sets chopper timing (TOFF=3, HSTRT=5, HEND=2, TBL=2), vsense flag, MRES for the requested microstep count, and enables 256-step interpolation.
- PWMCONF (0x70): configures the StealthChop2 PWM parameters with automatic scaling and gradient adjustment (PWM_OFS=36, PWM_GRAD=14, pwm_autoscale=1, pwm_autograd=1).

7.4 StallGuard / CoolStep Tuning (TMC2209)

Two runtime subcommands write the StallGuard4 / CoolStep threshold registers used for sensorless homing and load-aware current control. They send a single UART datagram to the addressed driver and require a prior TMC22xx init to have claimed the TX pin.

TMC22xx SGTHRS slave_addr, value
- slave_addr: 0 to 3 (TMC2209 multi-drop address). For TMC2208, only 0 is valid.
- value: 0 to 255. Stall detection threshold for StallGuard4. Lower values are more sensitive (fire on lighter loads). Tune empirically against the mechanics; good starting values are 100-150 for typical CNC axes. TMC2209 only.

TMC22xx TCOOLTHRS slave_addr, value
- slave_addr: 0 to 3 (TMC2209) or 0 (TMC2208).
- value: 0 to 1048575 (20-bit). Lower-velocity threshold expressed in TSTEP units. StallGuard4 fires only while TSTEP <= TCOOLTHRS, i.e. while the motor is moving fast enough. A value of 0 disables StallGuard. CoolStep also activates at and above this speed.

Both subcommands are write-only; reading SG_RESULT for live tuning would require a UART receiver, which is not implemented.

7.5 Examples

```
' Configure a TMC2209 on GPIO 16, slave address 0,
' 800 mA run, 30% hold, 16 microsteps, default r_sense:
TMC22xx 16, 2209, 0, 800, 30, 16

' TMC2208 with SpreadCycle and a 0.15 ohm sense resistor:
TMC22xx 16, 2208, 0, 600, 50, 32, 0.15, 0

' Two TMC2209 drivers on the same wire, addresses 0 and 1:
TMC22xx 16, 2209, 0, 1000, 25, 16
TMC22xx 16, 2209, 1, 1000, 25, 16

' Set StallGuard4 sensitivity and enable it for speeds above the
' equivalent of TSTEP <= 200 (i.e. fast moves):
TMC22xx TCOOLTHRS 0, 200
TMC22xx SGTHRS 0, 120

' Disable StallGuard4 again:
TMC22xx TCOOLTHRS 0, 0

' Release the pin when done:
TMC22xx CLOSE
```

8. Sensorless Homing (TMC2209)

The TMC2209 can detect a stall by monitoring back-EMF (StallGuard4). When a stall is detected the chip drives its DIAG pin HIGH. With suitable wiring and configuration this can be used in place of physical limit switches for homing - no extra mechanical parts, no debounce noise, and the home reference is set by the mechanical stop itself.

PicoMite supports this without any sensorless-specific code: the existing G28 homing and ISR limit-trip path are reused, with DIAG simply wired to the limit input pin. The active-high invert option on STEPPER HWLIMITS handles the polarity difference.

8.1 Wiring

Per axis being homed sensorlessly:
- TMC2209 DIAG pin -> a free PicoMite GPIO. No series resistor needed; DIAG is push-pull.
- TMC2209 PDN_UART pin -> PicoMite TX GPIO (via 1 k-ohm series resistor) for the existing TMC22xx command. The same TX pin can drive multiple drivers on different slave addresses.
- TMC2209 ENN tied low (or to a dedicated enable GPIO controlled via STEPPER AXIS).

The DIAG pin replaces the limit switch wire entirely. There is no mechanical switch.

8.2 Configuration

1. Configure the driver itself with TMC22xx (StealthChop must be enabled - it is by default).
2. Register the DIAG pins as limit pins with STEPPER HWLIMITS, using the invert_mask to mark them as active-high. For three sensorless minimum-end homes:

```
STEPPER HWLIMITS x_diag, y_diag, z_diag, 0, 0, 0, 7
```

3. Tune StallGuard. SGTHRS controls sensitivity (0-255, lower = more sensitive). TCOOLTHRS is a velocity gate: StallGuard4 only fires while TSTEP <= TCOOLTHRS, so the homing speed must be fast enough for the chosen TCOOLTHRS. 200 is a reasonable starting point.

8.3 The arming pattern

StallGuard must be armed only during homing. If it is left enabled during normal cutting, any high-load move - heavy cut, sharp deceleration into a hard part, even a high-jerk corner - will fire DIAG and look exactly like a limit-switch trip to the firmware. The current move is aborted, drivers disabled, position marked unknown.

The pattern is therefore to arm StallGuard immediately before G28 and disarm it afterwards. Disarming is a single-register write of TCOOLTHRS=0, which gates StallGuard off.

```
' One-time setup (typically in startup):
TMC22xx GP5, 2209, 0, 800, 30, 16     ' driver init
STEPPER INIT
STEPPER AXIS X, ...
STEPPER AXIS Y, ...
STEPPER AXIS Z, ...
STEPPER HWLIMITS GP10, GP11, GP12, 0, 0, 0, 7  ' DIAG inputs, all active-high
STEPPER RUN

' Each home cycle:
TMC22xx TCOOLTHRS 0, 200             ' arm StallGuard for slave 0
TMC22xx SGTHRS 0, 120                ' set sensitivity (tune empirically)
STEPPER GC G28                       ' run the homing cycle
TMC22xx TCOOLTHRS 0, 0               ' disarm so normal moves can't trip
```

8.4 Caveats

- Minimum speed: StallGuard4 requires the motor to be moving above some minimum velocity (set by TCOOLTHRS). The G28 routine performs a fast approach followed by a slow refine pass; the slow pass will be below TCOOLTHRS and DIAG will not fire on it. In practice you either keep the refine fast enough by tightening SGTHRS, or accept the fast-approach hit as your home position and skip the refine.
- StallGuard sensitivity is highly mechanics- and load-dependent. SGTHRS that works on a lightly-loaded axis will misfire when the carriage is heavier, and vice versa. Treat it as a per-axis tuning value, not a fixed constant.
- StealthChop must remain enabled (it is the default for TMC22xx). SpreadCycle disables StallGuard4 on the TMC2209.
- DIAG remains armed any time TCOOLTHRS is non-zero. The arming pattern above keeps that window confined to the homing cycle. If you intentionally want DIAG-as-collision-detection during normal moves, you can leave it armed - just be ready for false trips on heavy cuts.

- Sensorless homing is a TMC2209 feature only. The TMC2208 does not implement StallGuard.

8.5 What actually drives DIAG

DIAG itself is always physically active - it cannot be disabled. The pin is the OR of two internal sources, only one of which is gated by the registers above:

1. Driver fault (always live). Overtemperature shutdown, overtemperature pre-warning, short to ground / supply, undervoltage, and similar fault conditions drive DIAG high regardless of SGTHRS or TCOOLTHRS. There is no way to suppress this.

2. Stall (StallGuard4 - gated). For a stall to fire DIAG, both gates must be open at the same time:
   - velocity gate: TSTEP <= TCOOLTHRS (the motor is moving fast enough), AND
   - sensitivity gate: SG_RESULT < 2 * SGTHRS.
   After driver init both registers default to 0. With TCOOLTHRS = 0 the velocity gate is closed (TSTEP <= 0 is never true), so the stall path cannot fire. With SGTHRS = 0 the sensitivity comparison can never succeed either. Either zero is sufficient to suppress stall reporting.

Implication: leaving DIAG wired to a HWLIMITS input permanently is safe. Between homing cycles, set TCOOLTHRS = 0 and the stall path is off. A real driver fault during normal motion will still drive DIAG high, which the firmware then treats as a limit-switch trip - motion halts, drivers disable, position is marked unknown. That is the desired response to a hardware fault, so the coupling is intentional rather than something to work around.

There is no need to clear SGTHRS when disarming. Setting TCOOLTHRS to 0 alone is enough.