TMC429 & TMC26x Getting Started: Motion Control via Step/Direction

Valid for TMC260, TMC261, and TMC262

Here, TMC26x represents TMC262, TMC261, or TMC260 because they behave identical from the register communication point of view.

This application note describes how to initialize a TMC429 together with a TMC26x with basic parameters to run a stepper motor.

Additional application notes are available for more advanced parameterization of the TMC26x stepper motor driver and for parameterization of stallGuard™ and coolStep™.

Table of Contents

1 Preliminary Note

For explaining how to initialize a TMC429 together with a TMC26x the stepRocker module TMCM-1110 is used as actual example. C source code examples are freely available for this module for practical tests (www.steprocker.com).

2 Hardware Architecture Outline

From the configuration point of view, the TMC429 and TMC262 of the stepRocker have the architecture outlined in **Figure 1**. The motion control of the TMC262 takes place via step/direction by the TMC429. Both, the TMC429 and the TMC262 are parameterized via SPI by the microcontroller.

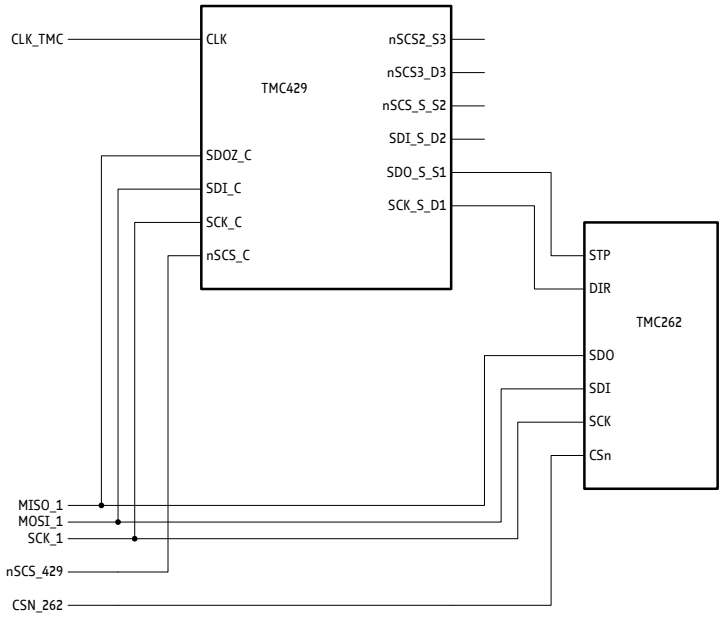

Figure 1 SPI and step/dir wiring scheme for TMC429 with TMC262 on stepRocker module

3 Source Code Examples and Software Architecture

The complete firmware of the stepRocker (www.steprocker.com) is available as C source code example on www.trinamic.com from the stepRocker page via a link to stepRocker™ open TMCL Library (github).

The main program is *stepRocker.c* calling all necessary initializations.

Please refer the HTML documentation generated from source code with doxygen concerning details of firmware architecture source code.

4 Initialization of the TMC26x

For configuration, the TMC26x has five control registers: DRVCTRL (Driver Control), CHOPCONF (Chopper Configuration), SMARTEN (Smart Energy), SGCSCONF (stallGuard2, Current Scale), and DRVCONF (driver configurations).

Note:
- *Smart Energy* is an early term for *coolStep*.
- An application note concerning *optimal chopper configuration* is available. For getting started a default configuration as given here is sufficient to start a motor running.
- An application note concerning the *parameterization of stallGuard2 and coolStep* is also available. *For getting started the stallGuard2 and coolStep features will not be used. This is proposed to be done later.*

LIST OF IMPORTANT CONTROL REGISTERS:

DRVCTRL: driver control parameter, here SDOFF=0 (default) ⇔ step-direction = ON
CHOPCONF: chopper configuration register
SMARTEN: coolStep. Leave untouched for getting started
SGCSCONF: stallGuard2. Ignored for getting started, but *setting current scale* is required
DRVCONF: basic driver configurations switches

TMC26X REGISTER CONFIGURATION EXAMPLE

Register/ Bit	DRVCTRL (*SDOFF=1*)	DRVCTRL (***SDOFF=0***)	CHOPCONF	SMARTEN	SGCSCONF	DRVCONF
19	0	0	1	1	1	1
18	0	0	0	0	1	1
17	PHA	-	0	1	0	1
16	CA7	-	TBL1=1	0	SFILT=1	TST=0
15	CA6	-	TBL0=0	SEIMIN=0	-	SLPH1=1
14	CA5	-	CHM=0	SEDN1=0	SGT6=0	SLPH0=1
13	CA4	-	RNDTF=0	SEDN0=0	SGT5=0	SLPL1=1
12	CA3	-	HDEC1=0	-	SGT4=0	SLPL0=1
11	CA2	-	HDEC0=0	SEMAX3=0	SGT3=0	-
10	CA1	-	HEND3=0	SEMAX2=0	SGT2=1	DISS2G=0
9	CA0	INTPOL=0	HEND2=0	SEMAX1=0	SGT1=0	TS2G1=0
8	PHB	DEDGE=0	HEND1=1	SEMAX0=0	SGT0=1	TS2G0=0
7	CB7	-	HEND0=0	-	-	***SDOFF=0***
6	CB6	-	HSTRT2=0	SEUP1=0	-	VSENSE=1
5	CB5	-	HSTRT1=1	SEUP0=0	-	*RDSEL1=0*
4	CB4	-	HSTRT0=1	-	CS4=0	*RDSEL0=0*
3	CB3	MRES3=0	TOFF3=0	SEMIN3=0	CS3=0	-
2	CB2	MRES2=0	TOFF2=0	SEMIN2=0	CS2=1	-
1	CB1	MRES1=0	TOFF1=0	SEMIN1=0	CS1=0	-
0	CB0	MRES0=0	TOFF0=1	SEMIN0=0	CS0=1	-

Table 4.1 TMC26x register configuration table example

SPI DATAGRAMS TO WRITE THE TMC26X REGISTER CONFIGURATION GIVEN IN THE EXAMPLE

```
DRVCTRL   = 0x00000;   // set interpolation = OFF, micro step resolution = 256x
CHOPCONF  = 0x90131;   // set TBL=3, CHM=0, RNDTF=0, TOFF=1
SMARTEN   = 0xA0000;   // disable smart energy function (coolStep)
SGCSCONF  = 0xD0505;   // set CS=0x05 that is 0x5/0x1F = 5/31 = 16% of max. current
DRVCONF   = 0xEF040;   // set driver configuration & step/direction control
```

The initialization of the driver is handled by the routine TMC26x.c: void InitMotorDrivers(void);

Within the C source code example, the TMC26x control bits are stored within a data record that holds all control bits. The access is handled in a way that one can read back the actual settings from that data record. This is because the TMC26x allows reading back status information but written configuration bits are write only and cannot be read back.

5 Initialization of the TMC429

5.1 TMC429 Configuration RAM – for SPI Stepper Driver Only

This application note describes how to control a TMC26x via step-direction signals. For step/direction control the TMC429 configuration RAM is unused and the internal sine wave look-up table (SinLUT) of the TMC26x driver is used instead. So, there is no need for initialization of the TMC429 configuration RAM.

5.2 TMC429 Configuration Step/Direction Interface & Timing

ACTIVATING THE STEP/DIRECTION INTERFACE

The step/direction interface of the TMC429 is activated by setting the *ENable StepDirection* control bit EN_SD = 1 of the TMC429 configuration register named if_configuration_429.

TIMING OF THE STEP/DIRECTION INTERFACE

The timing of the step/direction interface of the TMC429 is programmed via the nibble *clk2_div* [3… 0]. This are the bits [11… 9] of the stepper motor global parameter register.

The programming of the timing is intended for use of external step/direction power stages. For local communication the timing needs to be programmed to the fastest setting that is clk2_div = 0000.

5.2.1 if_configuration_429 (JDX=%0100) and Step/Direction Timing via CLK2_DIV

The register **if_configuration_429** is the interface configuration register for the TMC429. This register is used for

- configuration of the additional reference inputs,
- de-multiplexed interrupt output,
- step/direction interface, and for
- association of the position compare output signal to one stepper motor.

Register/ Bit	if_configuration_429	Function
0	INV_REF = 0	Invert polarity of reference switches (common polarity for all reference switches.
1	SDO_INT = 1	Map internal non-multiplexed interrupt status to nINT_SDO_C (needs SDOZ_C as SDO_C for read back information from the TMC429 to the micro controller); with SDO_INT='1' the nINT_SDO_C is a non-multiplexed nINT output to the micro controller
2	STEP_HALT = 0	Toggle on each step pulse (this halfs the step frequency, both pulse edges represent steps); this function can be used for the TMC262; STEP_HALF reduces the required step pulse bandwidth and is use full if one used e.g. low-bandwidth opto-couples;
3	INV_STP = 0	Invert step pulse polarity; this is for adaption of the step polarity to external diver stages
4	INV_DIR = 0	Invert step pulse polarity; this is for adaption to external diver stages; alternatively, this can be used as a shaft bit to adjust the direction of motion for a motor, but do not use this as a direction bit because it has no effect on the internal handling of signs (x_actual, v_actual, …)
5	EN_SD = 1	ENable StepDirection. Important Hint: The Step Pulse Timing (length) must be compatible with step frequency; the Step Pulse Timing is determined by the 4 LSBs of CLK2_DIV for when step/direction mode is selected by EN_SD='1';

Register/ Bit	if_configuration_429	Function
6	POS_COMP_SEL_0 = 0	Select one motor out of three motors (%00, %01, %10) for the position compare function output of the TMC429 named **poscmp**.
7	POS_COMP_SEL_1 = 0	
8	EN_REFR = 1	Enable new TMC429 reference inputs REFR1, REFR2, REFR3. EN_REFR=0 is the default. This is important because the REFRx input have internal pull-up resistors and this might cause trouble if these in-out are not-connected (for the SSOP16 these REFRx cannot be connected).

Table 5.1 Example of if_configuration_429 interface configuration register setting for TMC429

SPI DATAGRAMS TO WRITE THE TMC26X REGISTER CONFIGURATION GIVEN IN THE EXAMPLE:

if_configuration_429 = 0x68000122; // write JDX=4 with SDO_INT=1, EN_SD=1, EN_REFR=1

The initialization of the TMC429 is handled by the routine TMC429.c : void Init429(void);

5.3 TMC429 Step/Direction Timing – Set clk2_div in Global Parameter Register

The step/direction mode is enabled while the *ENable StepDirection* control bit EN_SD of the if_configuration_429 register is set to 1.

The timing of the step/direction interface is controlled by the four LSBs [3... 0] of the clk2_div of the global parameter register.

The clk2_div [3... 0] is named stpdiv_429. For a given clock frequency fCLK [unit: MHz] of the TMC429, the length tSTEP [unit: µs] of a step pulse is

$$tSTEP\ [\mu s] = 16 * (1 + stpdiv\_429) / fCLK\ [MHz].$$

For a clock frequency fCLK [MHz] of 16MHz the step pulse length can be programmed by stpdiv_429 in integer multiple of 1 µs.

The **stpdiv_429** must be set that it is compatible to the upper step frequency fSTEP = 1 / tSTEP that is used.

The first step pulse after a change of direction is delayed by tDIR2STP that is equal to tSTEP to avoid setup time violations of the step/direction power stage.

Note:
- The maximum step pulse frequency is fSTEP_MAX [MHz] = fCLK [MHz] / 32.
- For a clock frequency fCLK [MHz] = 16MHz the maximal possible step pulse frequency fSTEP_MAX is 500kHz.
- For a clock frequency fCLK [MHz] = 32MHz the maximal step pulse frequency fSTEP_MAX is 1MHz.

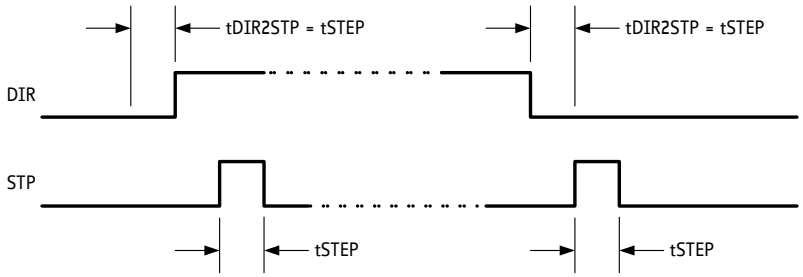

Figure 2: TMC429 Step/direction timing (EN_SD='1' & STEP_HALF='0')

For SD_EN = 1 the clk2_div [3… 0] is named stpdiv_429 within the TMC429 datasheet (available on www.trinamic.com). In step/direction mode the other control bits of the stepper motor global parameter register are ignored. For setting the fastest step/direction timing write 0 in the stepper motor global parameter register with

```
stepper_motor_global_parameter_register = 0x7E000000; // clk2_div = 0 resp. stpdiv_429 = 0
```

This initialization of the TMC429 is also handled by the routine TMC429.c : void Init429(void);

Generally:
For short wire link between TMC429 and TMC26x of the step/direction signals (e.g. on TMCM-1110 stepRocker board) one can set the shortest tSTEP be setting stpdiv_429 = 0.

EXCERPT OF STEPPER MOTOR GLOBAL PARAMETER REGISTER

Register/ Bit	Stepper motor global parameter register	Function
23		
22		
21	mot1r	For SD_EN = 1 of in if_configuration_429 this control bit is ignored.
20	refmux	For SD_EN = 1 of in if_configuration_429 this control bit is ignored.
19		
18		
17		
16	continuous_update	For SD_EN = 1 of in if_configuration_429 this control bit is ignored.
15	clk2_div [7…4]	For SD_EN = 1 of in if_configuration_429 this part of the register is ignored.
14		
13		
12		
11	clk2_div [3…0] = 0	For SD_EN = 1 of in if_configuration_429 this defines the timing of the step/direction interface; setting to 0 for fastest timing.
10		
9		
8		
7	csCommonIndividual	For SD_EN=1 of in if_configuration_429 this part of the register is ignored
6	polarities	Polarities of the SPI signals for the SPI stepper motor driver chain. Not relevant for step/direction mode of TMC429
5		
4		
3		
2		
1	LSDM = 00, 01, 10	Last stepper motor driver. Not relevant for step/direction mode of TMC429
0		

Table 5.2 Example of stepper motor global parameter register for TMC429 step/direction timing

5.4 Parameterizing Individual Stepper Motors for Motion

The preceding basic initializations of interfacing normally need to be done only once. Now, proceed as follows:

- Choose and set the motion parameters **v_min** and **v_max**.
- Choose and set the clock pre-dividers **pulse_div** and **ramp_div**.
- Choose and set the microstep resolution **usrs**.
- Set **a_max** with a valid pair of **pmul** and **pdiv**.
- Choose the switch configuration **ref_conf** with the ramp mode rm.
- Pull down to ground or disable the reference switch inputs REF1, REF2, REF3 plus REF1R, REF2R, and REF3R by setting **ref_conf**.

5.4.1 Choose *x_target* (for Ramp Mode) or *v_target* (for Velocity Mode)

With the above mentioned settings the TMC429 runs a motor if one writes either x_target or v_target, depending on the choice of the ramp mode rm.

Each stepper motor has its associated set of registers for motion control. Before running a motor, parameters have to be initialized. For many applications there is no need to re-program settings done once during initialization.

MODES OF OPERATION:

The parameters mentioned here allow the adjustment for a wide range of applications.

- For **ramp_mode** the microcontroller sends desired target positions **x_target** and the TMC429 autonomously takes care of positioning.
- For **velocity_mode**, the micro controller sends the desired target velocity **v_target** to the TMC429 to run a stepper motor continuously with that speed.

All motion control parameters are represented as integer resp. signed integer values within units that are specific for the TMC429 (depending on the clock frequency used for the TMC429).

5.4.2 Real World Units, Units of Stepper Motors, and TMC429 Internal Units

From the stepper motor application point of view, motion control parameters within units of fullsteps (FS) for position, fullsteps per second (FS/s) for velocity, and fullsteps per second square (FS/s^2) for acceleration are natural units for stepper motors. The formulas to calculate into these units are given in the TMC429 datasheet section *pulse_div & ramp_div & usrs (IDX=%1100)*.

The following calculation files are available on www.trinamic.com (on TMC429 product page):

- *tmc429_frequencies.xls* spread sheet, which calculates between physical motion units (rad, rad/s…) and stepper specific units (FS, FS/s, …)*[1]
- *TMC429Calc.exe* standalone program*[2]

*[1] The link between real world units and stepper motor units in full step units is an application specific gear ration that defines how distances (in meters, inches…) or angels (radians, grad…) match with one full step.

*[2] The link between time in seconds and TMC429 units is done via lengths of internal counter and the clock frequency fCLK [MHz] of the TMC429.

5.4.3 Velocity R [Hz] and Acceleration ΔR [Hz/s]

The desired microstep frequency R [Hz] and the desired microstep acceleration ΔR [Hz] depend on the application. Typical stepper motors can go up to fullstep frequencies of some thousand fullsteps per second. Without load, they can accelerate to those fullstep frequencies within a couple of fullsteps.

5.4.3.1 Choosing Microstep Resolution / Step Pulse Pre-Divider / Acceleration Pre-Divider

Microstep resolution, step pulse pre-divider, and acceleration pre-divider have to be set according to the following procedure.

1. CHOOSE THE MICROSTEP RESOLUTION

First, set a microstep resolution. It is assumed that the highest microstep resolution **usrs** = 6 for the driver motion control of the SPI driver chain is set. In step/direction mode the microstep resolution is controlled within the driver chip and should be taken into account.

2. SET THE STEP PULSE PRE-DIVIDER

Then, the pulse pre-divider has to be determined. It allows scaling the step frequencies in a very wide range. Therefore take into account the maximum desired velocity v_max. Based on the formula R [Hz] = f_clk [Hz] * velocity / (2^pulse_div * 2048 * 32) given within the TMC429 datasheet one can determine

pulse_div := log(f_clk [Hz] * v_max / (R [Hz] * 2048 * 32)) / log(2)

setting **v_max** = 2047 (or 2048 for simplified calculation) and R [Hz] to the maximum desired microstep frequency.

The fullstep frequency can be calculated based on the formula R_{FS} [Hz] = R [Hz] / 2^usrs given within the TMC429 data sheet. With this, the microstep frequency is R [Hz] = R_{FS} [Hz] * 2^usrs. The quotient of logarithms comes from the relation $\log_2(x) = \log(x) / \log(2)$ to calculate the logarithm to the basis of two which is the number of bits need to represent x. The calculation result of pulse_div then has to be chosen close to the next integer value.

Hint:
For step/direction control via TMC429, the microstep frequency R [Hz] of the TMC429 is relevant. For calculation of full step frequency the microstep resolution of the TMC26x is relevant. This is because each step pulse of the TMC429 is a microstep for the TMC26x.

3. SET THE RAMP PRE-DIVIDER

After determination of **pulse_div**, the parameter **ramp_div** can be calculated. Based on the formula ΔR[Hz/s] = f_clk[Hz] * f_clk[Hz] * a_max / (2^(pulse_div+ramp_div+29)) given within the TMC429 datasheet one can determine

ramp_div := log(f_clk [Hz] * f_clk [Hz] * a_max / (ΔR [Hz/s] * 2^(pulse_div+29))) / log(2)

setting **a_max** = 2047 (or 2048 for simplified calculation) and ΔR [Hz/s] to the maximum desired microstep acceleration. The calculation result of **ramp_div** then has to be chosen close to the next integer value.

5.4.3.2 Choosing Step Velocities *v_min* and *v_max* and the Step Acceleration *a_max*

The **v_min** parameter should be set to 1 (please refer the TMC429 datasheet for details).

The **v_max** parameter determines the maximum velocity and has to be set depending on the application.

The change of the parameter **a_max** requires a recalculation of **p_mul** and **p_div**. Once set, the a_max parameter can be left untouched for many applications.

Hints for **a_max** setting:

If the parameters **pulse_div** and **ramp_div** are equal, the parameter **a_max** can be set to any value within the range of 0 … 2047.

If the parameters **pulse_div** and **ramp_div** differ, the limits **a_max_lower_limit** and **a_max_upper_limit** have to be checked (please refer to the TMC429 datasheet).

The velocity does not change with a_max = 0.

5.4.3.3 Calculate p_mul and p_div for a Chosen Set of Parameters

The two parameters **p_mul** and **p_div** have to be calculated for positioning in **RAMP_MODE**. These parameters depend on **pulse_div**, **ramp_div**, and **a_max**. The parameters **p_mul** and **p_div** have to be recalculated if one of the parameters **pulse_div**, **ramp_div**, **a_max** changes.

An example for calculation of **p_mul** and **p_div** for the TMC429 is given as a C program included within the TMC429 datasheet. This C program source code can be copied directly out of the PDF document. Additionally, a spread sheet named **tmc429_pmulpdiv.xls** demonstrating the calculation of pmul and pdiv is available on www.trinamic.com for download.

The principle of calculation of **p_mul** and **p_div** is simple: the routine CalcPMulPDiv(...) gets the parameters **a_max**, **ramp_div**, **pulse_div**, with a reduction factor p_reduction. With these parameters, a **pmul** is calculated for any allowed **p_div** ranging from 0 to 13. The **p_div**, which results in a valid **pmul** that is in the range of 0 to 127 (resp. **p_mul** that is in range 128… 255) selects a valid pair of **p_mul** and **p_div**.

p_mul and **p_div** have to be determined for each set of **pulse_div**, **ramp_div**, and **a_max**.

5.4.4 Set the Reference Switch Configuration and the Ramp Mode

Both, the *reference switch configuration* (ref_conf) and the *ramp mode* (rm) are configured by access to a single register. Normally, this kind of initialization is done only once. Please proceed as follows:

- Choose the switch configuration **ref_conf** together with the ramp mode **rm**.
- Pull unused reference switch inputs REF1, REF2, REF3 down to ground or disable them by setting **ref_conf**. (Otherwise, the REF1, REF2, and REF3 inputs might detect a switch signal and stop a motor.)

The most important register part (except reference switch configuration) is the **rm** for setting RAMP_MODE for positioning applications or VELOCITY_MODE for constant speed applications.

Register/ Bit	Stepper motor global parameter register	Function
23		
22		
21		
20		
19		
18		
17		
16	LP	Latched position waiting (read only status bit)
15		
14		
13		
12		
11	REF_RnL = 0	Reference switch right not left (to change assignment of left/right switch)
10	SOFT_STOP = 1	Soft stop with deceleration a_max during for active stop switch
9	DISABLE_STOP_R = 1	Disable stop switch right.
8	DISABLE_STOP_L = 1	Disable stop switch left.
7		
6		
5		
4		
3		
2		
1	RM = %00, %01, %10, %11	Ramp mode: 00 = RAMP_MODE, 01 = SOFT_MODE, 10 = VELOCITY_MODE, 11 = HOLD_MODE
0		

Table 5.3 Example of REF_CONF and RAMP_MODE setting for stepper motor #0 (smda = %00)

5.5 Running a Motor

All necessary settings for getting started are done now. Run your stepper motor as follows:

RAMP_MODE: write the desired target position into the register **x_target** of the associated motor.

VELOCITY_MODE: write the desired target velocity **v_target** of the associated motor.

6 Summary

This application note explains how to initialize a TMC429 motion controller and a TMC26x stepper motor driver. The initialization of the TMC429 is done via a sequence of a couple of SPI datagrams. The initialization of the stepper motor driver TMC26x is done by up to five SPI datagrams. So, altogether the initialization is represented by a sequence of a couple of SPI datagrams. After initialization, motion commands can simply be executed by writing motion parameters (target positions, target velocities…) into TMC429 registers by sending SPI datagrams.

Based on the low cost motion controller/driver board TMCM-1110 stepRocker, this application note shows with a practical example – available as C open source code - how to get a stepper motor running. The sample C code is intended to be used as a basis for own developments of customers.

7 Disclaimer

8 Revision History

8.1 Document Revision

Version	Date	Author LL – Lars Larsson SD – Sonja Dwersteg	Description
1.00	2012-FEB-03	LL	Initial version
1.01	2012-MAR-26	SD	New design
1.02	2012-AUG-02	SD	Chapter 5.2.1: - Column *register/bit* in Table 5.1 corrected - Column *if_configuration_429* in Table 5.1 corrected - example for *SPI datagram to write in the TMC26x register configuration* corrected.
1.03		LL	Section 5.2.1 example if_configuration = ~~0x680122~~ corrected to 0x68000122, and ~~ED_SD~~='1'; corrected to EN_SD='1';

9 References

- TMC260/TMC261/TMC262 datasheet, www.trinamic.com
- TMC429 datasheet, www.trinamic.com
- stepRocker (TMCM-1110) Hardware Manual, www.trinamic.com
- stepRocker (TMCM-1110) Getting Started (www.trinamic.com)
- stepRocker (TMCM-1110) Schematic (www.trinamic.com)

- Open Source Motion Control Community
- www.motioncontrol-community.org or www.steprocker.com