Application Note AN018:

TMC429 & TMC24x Getting Started Motion Control via SPI

Including stallGuard™ of TMC246/249

TRINAMIC® Motion Control GmbH & Co. KG
GERMANY

www.trinamic.com

1 Focus of this Application Note

This application describes how to initialize a TMC429 together with a TMC246/TMC249 with basic parameters to run a stepper motor. It explains how to initialize the SPI chain of TMC246/TMC249 stepper motor drivers, how to read out diagnosis information via the TMC429 from the SPI driver chain, and how to use stallGuard of the TMC246/TMC249. Additionally, a stallGuard profiler for rotational and for linear moving applications is included as C source code example. The C source code examples are freely available for the TRINAMIC evaluations boards TMC429+TMC24X-EVAL and TMC429+TMC26X-EVAL module for practical tests.

The TMC429 is the drop-in successor for the TMC428. So, existing examples written for the TMC428 are still valid for the TMC429.

2 Table of contents

Table of Figures

3 Introduction

The TMC429 datasheet gives a detailed description of all its registers and its functionality. In addition to the TMC429 datasheet, this application note describes the basic steps how to initialize the TMC429 and how to run a motor. It gives two practical examples written in C language – 'tmc429demo.zip' and 'sample428_236.zip'. The complete sources are available as archives named 'tmc429demo.zip' and 'sample428_236.zip' for download from www.trinamic.com. The C examples can be used as a base for own applications. The code and information is provided "as is" without warranty of any kind, either expressed or implied.

The TMC428 datasheet is recommended as a base for this application note. The sample C code included within 'tmc429demo.zip' is a Win32 console application that communicates with a TMC429 evaluation board via RS232. TMC429 evaluation boards are offered by TRINAMIC distributors. The compact example 'sample428_236.zip' is intended to outline how to realize an application for running on a stand-alone micro controller. Routines for communication with the TMC429 via SPI™ are specific for each type of micro controller and have to be added.

This application note has been updated concerning the use of datagram_low_word and datagram_high_word that build the interface registers to read out diagnosis and status information from the stepper motor driver chain. This mechanism is also need to read the StallGuard information – names load indicator bits - from TRINAMIC stepper motor drivers TMC246 and TMC249. In addition, the implementation of StallGuard profilers is described as now implemented as exemplary C code within the 'tmc429demo.zip'

In contrast to low level C programming, TRINAMIC credit card size modules come with TMCL™ (TRINAMIC Motion control Language) together with an IDE (Integrated Development Environment running under *Win2K*, *XP*, *W7*). TMCL allows rapid prototyping and building user applications without the need of low level C programming. Please refer to the TMCL user manual resp. the user manuals of the different modules for details.

4 Initialization of the TMC429

On first sight, the sample C code included within the ZIP archive 'tmc429demo.zip' might give the impression that the initialization of the TMC429 is a complicated task due to a couple of different routines used. The intent of this sample C code and its different routines is to demonstrate how to access the registers of the TMC429. All together, these routines just perform a sequence of SPI datagramms that perform the initialization of the TMC429. So, for those who are familiar with the TMC429 and its SPI datagramms, the program structure might become more compact and the initialization code might look like

```
for (i=0; i<(128-2); i+=2)    // initialize TMC429 RAM table (SPI conf. & quarter sine wave LUT)
{
    spo = 0x80000000 | (i<<(25-1)) | (tmc429_ram_tab[i+1]<<8) | (tmc429_ram_tab[i]); // RRS=1, RW=0
    spi = tmc429spi( spo );
}

spi = tmc429spi(0x7E010701);  // initialize stepper motor global parameter register, LSMD=1 for EvalKit
spi = tmc429spi(0x........);  // initialize . . .
```

where tmc429_ram_tab[] is an array representing the content of the TMC429 configuration RAM (see section 0) and tmc429spi(long int spo) represents a routine that sends a 32 bit wide SPI datagram to the TMC429 and receives a 32 bit wide SPI datagram from the TMC429. The TMC429 needs to be initialized after each power on. A micro controller is suitable to do the initialization of the TMC429 by just sending a sequence of SPI datagramms to the TMC429 after power on. The TMC429 itself performs a power on reset (POR). Almost any register of the TMC429 is set to zero by the POR. The RAM of the TMC429 is not initialized by the power on reset, and so the RAM has a more or less random content after power on. In contrast to power on, there is no need for re-initialization of the TMC429 after it has been set into power-down by writing the power_down (JDX=%1000) register. The initialization of the TMC429 enfolds three main phases:

1. initializing the TMC429 configuration RAM
2. configuring the stepper motor global parameter register
3. parameterizing individual stepper motor registers
   - choose micro step resolution / step pulse pre-divider / acceleration pre-divider
   - choose step velocities v_min and v_max and the step acceleration a_max
   - calculate pmul & pdiv for the chosen set of parameters
   - set the reference switch configuration and the ramp mode
   - set the automatic current scaling [optional]

4.1 TMC429 Configuration RAM for TMC236/TMC239/TMC246/TMC249

First of all, the stepper motor driver datagram configuration has to be written into the RAM area of the TMC428. Additionally, the micro step look-up table (LUT) has to be initialized when using micro stepping. Both tables should be put into one single array for initialization of the TMC428 by the micro controller after power up. The following constant array represents the configuration concerning stepper motor driver chain and sine wave look up table according to the example given within the TMC428 datasheet (sections "Stepper Motor Driver Datagram Configuration" and "Initialization of the Microstep Look-Up-Table"):

```
unsigned char tmc429_ram_tab[128]=
{
     0x10, 0x05, 0x04, 0x03, 0x02, 0x06, 0x10, 0x0D, 0x0C, 0x0B, 0x0A, 0x2E, 0x11, 0x05, 0x04, 0x03,
     0x02, 0x06, 0x11, 0x0D, 0x0C, 0x0B, 0x0A, 0x2E, 0x07, 0x05, 0x04, 0x03, 0x02, 0x06, 0x0F, 0x0D,
     0x0C, 0x0B, 0x0A, 0x2E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11,
     0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11,

     0x00, 0x02, 0x03, 0x05, 0x06, 0x08, 0x09, 0x0B, 0x0C, 0x0E, 0x10, 0x11, 0x13, 0x14, 0x16, 0x17,
     0x18, 0x1A, 0x1B, 0x1D, 0x1E, 0x20, 0x21, 0x22, 0x24, 0x25, 0x26, 0x27, 0x29, 0x2A, 0x2B, 0x2C,
     0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x38, 0x39, 0x3A, 0x3B,
     0x3B, 0x3C, 0x3C, 0x3D, 0x3D, 0x3E, 0x3E, 0x3E, 0x3F, 0x3F, 0x3F, 0x3F, 0x3F, 0x3F, 0x3F, 0x3F
};
```

The first 64 bytes represent the stepper motor driver chain configuration for a chain of three TRINAMIC stepper motor drivers of type TMC236 / TMC239 / TMC246 / TMC249. The values ***0x2E*** values represent the primary signal code 0x0E (= mnemonic PH_B) with the NxM bit set.

For this sample configuration, the fast decay control bits of TMC428 are mapped as fixed Zero for the driver #1, fixed One for the driver #2 of the chain. For driver # 3 the FD_A is mapped to MDA and FD_B is mapped to MDB. Please refer the TRINAMIC stepper motor driver datasheets (TMC236 / TMC239 / TMC246 / TMC249) concerning the mixed decay feature. In plain text, it is

```
Zero,  A_5,  A_4,  A_3,  A_2, PH_A, Zero,  B_5,  B_4,  B_3,  B_2, PH_B,  One,  A_5,  A_4,  A_3,
 A_2, PH_A,  One,  B_5,  B_4,  B_3,  B_2, PH_B, FD_A,  A_5,  A_4,  A_3,  A_2, PH_A, FD_B,  B_5,
 B_4,  B_3,  B_2, PH_B,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,
 One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,  One,

   0,    2,    3,    5,    6,    8,    9,   11,   12,   14,   16,   17,   19,   20,   22,   23,
  24,   26,   27,   29,   30,   32,   33,   34,   36,   37,   38,   39,   41,   42,   43,   44,
  45,   46,   47,   48,   49,   50,   51,   52,   53,   54,   55,   56,   56,   57,   58,   59,
  59,   60,   60,   61,   61,   62,   62,   62,   63,   63,   63,   63,   63,   63,   63,   63
```

Please keep in mind that the DAC MSBs of the TMC428 are DAC_A_5 resp. DAC_B_5, where the DAC MSBs of the TMC236 / TMC239 / TMC246 / TMC249 drivers are CA3 resp. CB3. So, DAC_A_5 is mapped to CB3 resp. DAC_B_5 is mapped to CB3, and the other DAC bits are mapped accordingly. Once initialized, the configuration of the stepper motor driver chain is left unchanged in most cases during normal operation. In some cases, the fast decay control has to be changed. Please take into account that a write access to a RAM always overwrites a pair of TMC428 RAM values. A read access to the RAM always reads out a pair of values. Combining RAM read and RAM write allows the modification of a single value.

Use of the sensorless stall detection StallGuard (TMC246 / TMC249) requires the mixed decay feature to be switched off. At high speed, stepper motors might run better when the fast decay feature is permanently switched on. Some motors run best when mixed decay is switched on only during phases of decreasing current. The configuration can be modified also during operation by re-programming the stepper motor driver chain configuration.

The quarter sine wave look-up table RAM area is left unchanged for almost any application after it has been initialized. Changes might be wanted for fine tuning of micro stepping or for temporary changing from micro stepping to full stepping at high speed or to use a trapezoidal current shape that is more compatible with StallGuard than full stepping. Sine wave micro stepping is the best choise for most applications. Additionally, one has to take into account that changes of the quarter sine wave look-up table commonly apply to all stepper motors.

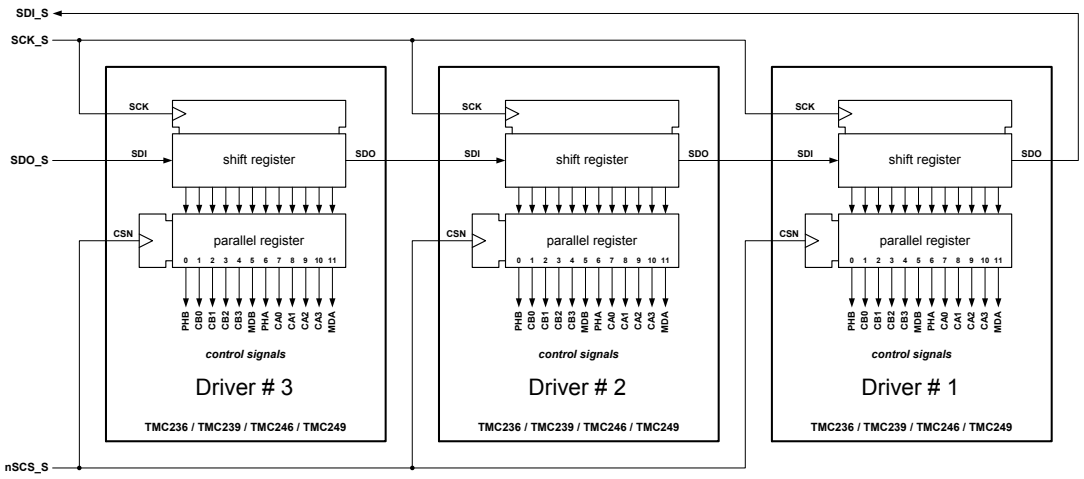

Figure 1 : SPI Chain Outline – Serial Transmitted Control Bits vs. Parallel Control Signals

All SPI datagram bits are shifted through the driver chain (Figure 1) with the SPI clock SCK_C. The actual content of the shift register chain is loaded into parallel buffer registers with the rising edge of the SPI enable signal nSCS_S. A set next motor bit forces the TMC428 to select internally the control bits for the next motor. The next motor bit is not transmitted via the SPI driver chain. The different drivers are addressed by their position within the chain.

Hint: In contrast to the TMC428, the TMC429 loads a default initialization into its configuration RAM for a TMC236/TMC239/TMC246/TMC249 stepper motor driver SPI chain (pls. refer TMC429 datasheet).

4.2 Stepper Motor Global Parameter Register (JDX=%1111)

The stepper motor global parameter register holds different flags (**mot1r**, **refmux**) to configure the references switches (REF1, REF2, REF3), the SPI interface update behavior (**continuous_update**), speed (**clk2_div**), chip select (**csCommonIndividual**), the polarities of different control bits of the SPI datagramms (**DAC_AB**, **FD_AB**, **PH_AB**) and the polarities of SPI control signals (**SCK_S**, **nSCS_S**) , and the number (**LSMD**) of stepper motor drivers within the chain. Please refer to the TMC428 data sheet concerning functionalities of the flags. ***Figure 2*** outlines an SPI datagram **0x7E010702** to initialize the stepper motor global parameter register with **LSMD**=2 for 3 stepper motor drivers.

<table>
<tr><th>Register/ Bit</th><th colspan="2">stepper motor global parameter register</th><th>function</th></tr>
<tr><td>23</td><td colspan="2"></td><td></td></tr>
<tr><td>22</td><td colspan="2"></td><td></td></tr>
<tr><td>21</td><td colspan="2">mot1r = 0</td><td>for SD_EN=1 of in if_configuration_429 this control bit is ignored</td></tr>
<tr><td>20</td><td colspan="2">refmux = 1</td><td>for SD_EN=1 of in if_configuration_429 this control bit is ignored</td></tr>
<tr><td>19</td><td colspan="2"></td><td></td></tr>
<tr><td>18</td><td colspan="2"></td><td></td></tr>
<tr><td>17</td><td colspan="2"></td><td></td></tr>
<tr><td>16</td><td colspan="2">continuous_update = 1</td><td>for SD_EN=1 of in if_configuration_429 this control bit is ignored</td></tr>
<tr><td>15</td><td rowspan="9">clk2_div[7…0]=7 (for SD_EN=0)</td><td rowspan="4">(for SD_EN=1)</td><td rowspan="9">for SD_EN=0 this defines the timing of the SPI for the SPI stepper motor driver chain, for SD_EN=1 of in if_configuration_429 this defines the timing of the step direction interface;</td></tr>
<tr><td>14</td></tr>
<tr><td>13</td></tr>
<tr><td>12</td></tr>
<tr><td>11</td><td rowspan="5">clk2_div[3…0]=0 (for SD_EN=1)</td></tr>
<tr><td>10</td></tr>
<tr><td>9</td></tr>
<tr><td>8</td></tr>
<tr><td>7</td><td colspan="2">csCommonIndividual = 1</td><td>for SD_EN=0 this is the SPI configuration addressing mode bit; for SD_EN=1 of in if_configuration_429 this part of the register is ignored</td></tr>
<tr><td>6</td><td colspan="2">DAC_AB = 0</td><td rowspan="5">polarities (0 : positive / 1 : inverted) of the SPI signals for the SPI stepper motor driver chain; these polarities are not relevant for the step direction mode of TMC429 (for EN_SD=1)</td></tr>
<tr><td>5</td><td colspan="2">FD_AB = 0</td></tr>
<tr><td>4</td><td colspan="2">PH_AB = 0</td></tr>
<tr><td>3</td><td colspan="2">SCK_S = 0</td></tr>
<tr><td>2</td><td colspan="2">nSCS_S = 0</td></tr>
<tr><td>1</td><td rowspan="2" colspan="2">LSDM = 00, 01, 10</td><td rowspan="2">last stepper motor driver; this is important for the SPI driver chain; it is not relevant for step direction mode of TMC429</td></tr>
<tr><td>0</td></tr>
</table>

Figure 2: Stepper motor global parameter register (SPI datagram ⇔ 0x7E010702)

4.3 Parameterizing Individual Stepper Motors

Each stepper motor has its associated set of registers for motion control. Before running a motor, some parameters have to be initialized once. For many applications, there is no need to re-program settings done once during initialization. Once initialized, the motion control becomes quite easy. For ramp_mode the micro controller just sends desired target positions and the TMC429 autonomously takes care of positioning. For velocity_mode, the micro controller just sends the desired target velocity to the TMC429 to run a stepper motor continuously.

On first sight, the determination of the required parameters might look a little bit complicated but it is simple. These parameters allow the adjustment to a very wide range of applications. The 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.

From the stepper motor application point of view, motion control parameters within units of full steps (FS) for position, full steps per second (FS/s) for velocity, and full steps per second square (FS/s^2) for acceleration are natural units for stepper motors. The formulas to calculate into these units are given within the TMC429 data sheet section pulse_div & ramp_div & usrs (IDX=%1100)). A spread sheet named 'tmc429_frequencies.xls' that calculates between physical motion units (rad, rad/s, …) and stepper specific units (FS, FS/s, …) and a stand-alone program 'TMC429Calc.exe' are available on www.trinamic.com

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

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

4.3.2 Choosing Micro Step Resolution / Step Pulse Pre-Divider / Acceleration Pre-Divider

First, one has to choose the micro step resolution. Following, the highest micro step resolution is chosen be setting **usrs** = 6. Then, the pulse pre-divider has to be determined. It allows to scale the step frequencies in a very wide range. Based on the formula R[Hz] = f_clk[Hz] * velocity / (2^pulse_div * 2048 * 32) given within the TMC428 datasheet one can determine

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

setting **v_max** = 2047 (resp. 2048 for simplified calculation) and R[Hz] to the maximum desired micro step frequency. The full step frequency can be calculated based on the formula R_{FS}[Hz] = R[Hz] / 2^usrs given within the TMC428 data sheet. With this, the micro step 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.

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 (resp. 2048 for simplified calculation) and ΔR[Hz/s] to the maximum desired micro step acceleration. The calculation result of **ramp_div** then has to be chosen close to the next integer value.

4.3.3 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 TMC428 data sheet for details). The **v_max** parameter determines the maximum velocity and has to be set depending on the application. Once set, the a_max parameter can be left untouched for many applications. Change of the parameter **a_max** requires recalculation of **p_mul** and **p_div**.

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 TMC datasheet for details). The velocity does not change with a_max = 0.

4.3.4 Calculate p_mul & p_div for a Chosen Set of Parameters

Two parameters named **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**. So, they have to be determined for a set of **pulse_div**, **ramp_div**, **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 TMC428 is given as a C program included within the TMC428 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. That **p_div**, that 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**.

Except reference switch configuration, the most important register part is the rm to set the mode of motion 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 = 0	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		

Figure 3: REF_CONF & RAMP_MODE setting for stepper motor # 0 (smda = %00) ⇔ 0x14000300

4.3.5 Set the Reference Switch Configuration and the Ramp Mode

Both, the reference switch configuration and the ramp mode are configured by access to a single register. Normally, this kind of initialization is done once. The switch configuration **ref_conf** together with the ramp mode **rm** has to be chosen. Unused reference switch inputs REF1, REF2, REF3 should be pulled down to ground or disabled by setting **ref_conf**. Otherwise, the REF1, REEF2, REF3 inputs might detect a switch signal and stop a motor.

4.3.6 Set the Automatic Current Scaling [optional]

With the power-on reset settings, the full current is driven when the motor is at rest, during acceleration, and during motion. Automatic down scaling when the motor is at rest reduces power dissipation. This makes sense if the application allows lower holding torque for a motor at rest.

For automatic current scaling, it is necessary to switch on the **continuous_update**, to force the TMC429 to send the current scaling datagram, even if all motors are at rest. That does not cause the TMC429 to sent datagramms to the stepper motor driver chain.

Hint: The automatic current scaling is for SPI control of stepper motor driver chain only.

Register/ Bit	stepper motor global parameter register	function
23		
22	is_agtat = 000	
21		
20		
19		
18	is_aleat = 000	
17		
16		
15		
14	is_v0 = 001	
13		
12		
11		
10	a_threshold = 0	
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		

Figure 4: Current Scaling Register for stepper motor # 0 (smda = %00) ⇔ 0x10001000

4.4 Running a Motor

With all these settings described before, one can simply run a stepper motor. In **RAMP_MODE**, one just has to write the desired target position into the register **x_target** of the associated motor. In **VELOCITY_MODE**, one just has to write the desired target velocity **v_target** of the associated motor.

4.5 Communication Outline of tmc429demo and sample428_236

The tmc429demo.exe is a Win32 console application. Its communication takes place over a RS232 interface byte by byte. The windows based RS232 communication is used to perform a 32 bit wide SPI communication with the TMC428 on a evaluation kit.

For an embedded application running stand alone on a µC, the user just has to write an SPI routine for the µC used for communication between µC and TMC429. For the code example sample428_236, the SPI routine spi429_uc() for communication with the TMC429 is named Send428(). This has to be completed for a given type of micro controller.

The communication is outlined by Figure 5 on page 11.

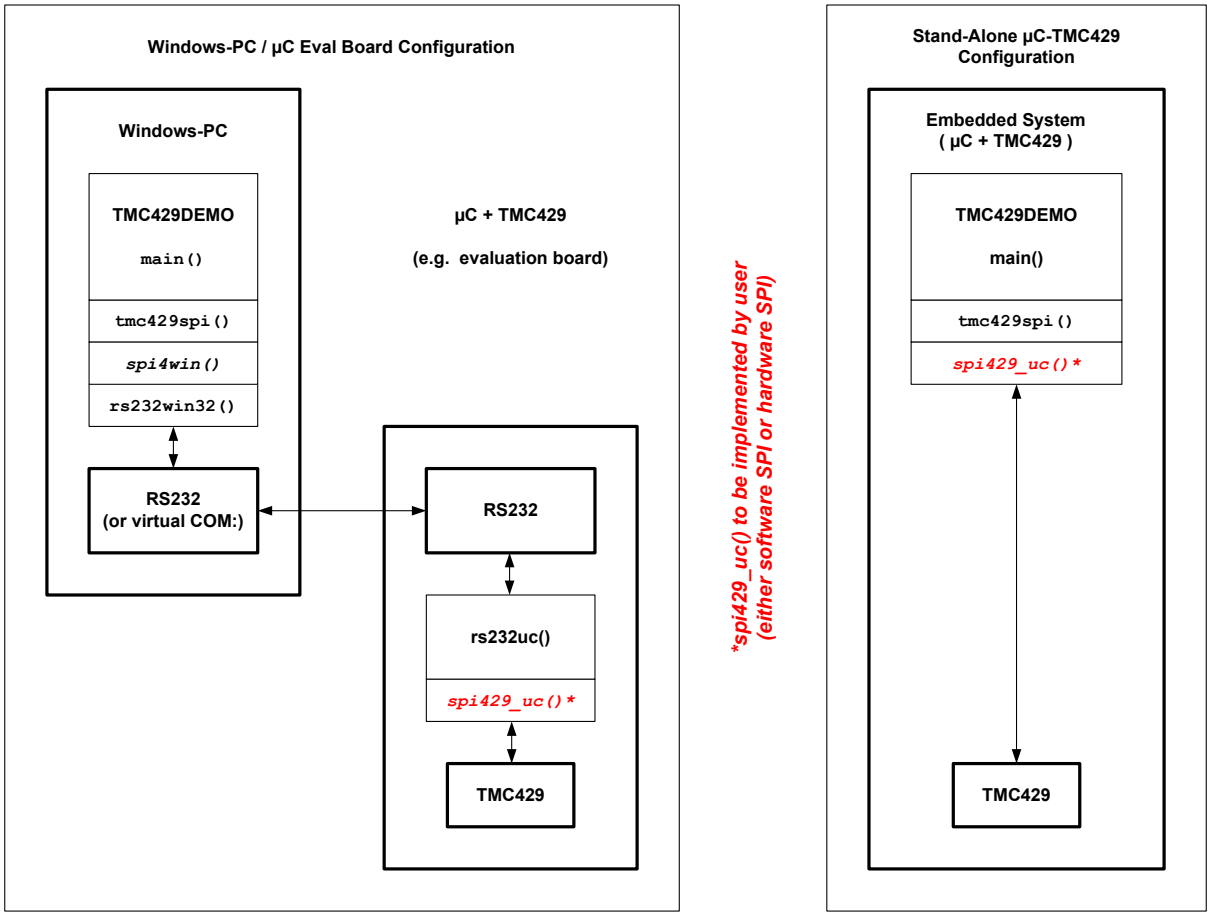

Figure 5 : Communication Outline for tmc429demo (SPI over RS232 or virtual COM port)

4.6 Reading Driver Chain Status Bits with TMC429

Beyond the basic initializations required to run stepper motors with the TMC428, the read out of status bits from the stepper motor driver chain is important for diagnosis and to use the sensorless stall detection StallGuard that is integrated within the TMC246 and TMC249 stepper motor drivers.

The status bits of TMC246 and TMC249 are: three load indicator bits (LD2, LD1, LD0), over temperature (OT), over temperature pre-warning (OTPW), under voltage (UV), over current high side (OCHS), open load bridge B (OLB), open load bridge A (OLA), over current bridge B low side (OCB), over current bridge A low side (OCA). For the TMC236 and TMC239, the load indicator bits (LD2, LD1, LD0) are permanent '0'. The status bits are shifted through the driver chain. The SDO output of the last stepper motor driver of the SPI chain has to be connected with the SDI_S of the TMC429.

To get the status bits from the stepper motor driver chain, fetching of them has to be initialized first by a write access to either **datagram_high_word** or **datagram_low_word**. Then, with the next SPI datagram the TMC429 sends to the stepper motor driver chain, it gets associated stepper motor status bits and holds them within the registers **datagram_high_word** and **datagram_low_word**.

Due to the fact, that the TMC429 autonomously updates the stepper motor driver chain, status information is required that indicates that **datagram_high_word** and **datagram_low_word** have been updated. The status bit CDGW (**C**over **D**ata**G**ram **W**aiting) provides this status information. Its name CDGW sometimes causes a little confusion. In principle, the CDGW should be named like

CDGW_or_WFDHWDLW (**C**over **D**ata**G**ram **W**aiting / Waiting for Datagram High Word Datagram Low Word). The cover datagram mechanism – to send an arbitrary datagram from the µC to the stepper motor driver chain – and the datagram high word and datagram low word both use this single status bit named CDGW. The CDGR status bit is part of each SPI datagram sent from TMC428 to µC.

To get the driver chain status bits, the following actions have to be done:

1. Initialize the fetching mechanism by a write to **datagram_high_word** or **datagram_low_word**
2. Send datagramms to the TMC428 (e.g. read out some TMC428 registers)
3. Extract the CDGW from datagramms received from the TMC428 to read the CDGW
4. Read the CDGW until it becomes '0'
5. If the CDGW is '0' read out the **datagram_high_word** and **datagram_low_word**
6. Extract the status bits of interest from **datagram_high_word** and **datagram_low_word**

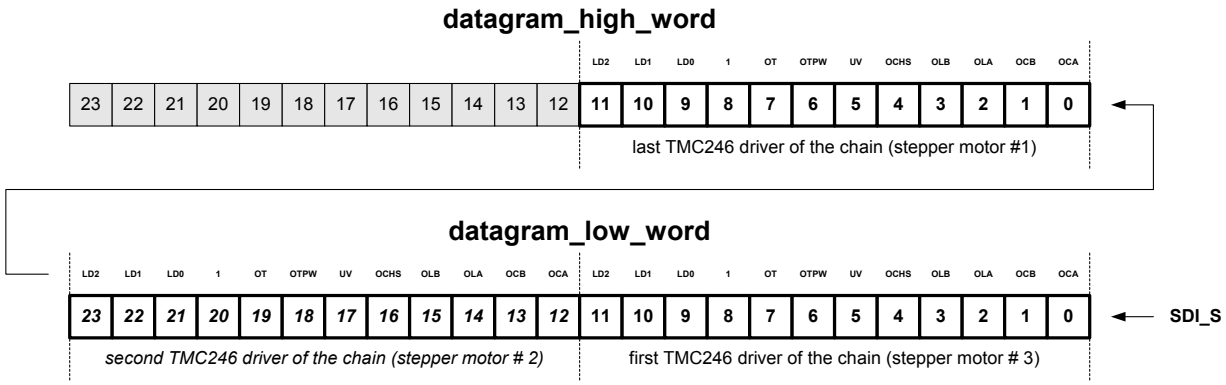

Figure 6: Example of status bit mapping for a chain of three TMC246 or TMC249

4.7 How to Get the Driver Chain Status Information – Code Example

The following C code example (part of tmc429demo.zip) outlines how to get the raw status bits from a SPI stepper motor driver chain into the TMC428 registers **datagram_high_word** and **datagram_low_word.**

```
void tmc429_get_datagram_words( long int *datagram_low_word, long int *datagram_high_word )
{
       long int spi, spo;

    // write datagram_low_word (or high_word) for initialization, sets CDGW to '1'
       spi = spi429glue( 0, 0x3, JDX_DATAGRAMM_LOW, WRITE, 0);
       spo = tmc429spi( spi );

    // wait until CDGW becomes '0'
       do
       {
              Sleep(1);  // (wait 1ms, allows the operating system to focus on other tasks)

              spi = spi429glue( 0, 0x3, JDX_DATAGRAMM_LOW, READ, 0);
              spo = tmc429spi( spi );

       } while (!(0x40000000 & (~spo))); // mask CDGW

       spi = spi429glue( 0, 0x3, JDX_DATAGRAMM_LOW, READ, 0);
       spo = tmc429spi( spi );

       *datagram_low_word = 0x00FFFFFF & spo;

       spi = spi429glue( 0, 0x3, JDX_DATAGRAMM_HIGH, READ, 0);
       spo = tmc429spi( spi );

       *datagram_high_word = 0x00FFFFFF & spo;
}
```

4.8 How to Get the Status Bits of TMC246 / TMC249 – Code Example

The following C code example (part of tmc429demo.zip) outlines how to get the raw status bits of each TMC246 or TMC249 driver. This is just a little data shifting.

```
void tmc429_get_tmc24x_status_bits( int *smo_status, int *sm1_status, int *sm2_status )
{
	long int datagram_low_word, datagram_high_word;
	int      lsmd;

	// this routine assumes a chain of up to three TMC246 / TMC249 drivers
	// pls. refer TMC428 datasheet v. 2.02 / April 26, 2007 or newer

	tmc429_get_datagram_words( &datagram_low_word, &datagram_high_word );

    // 1st, determine the number of TMC246 resp. TMC249 stepper motor drivers

	tmc429_get_lsmd( &lsmd );


	switch (((char)lsmd))
	{
		case 3 : ; // TMC428 interprets LSMD=3 as LSMD = 2 // NO "break;"
		case 2 : *smo_status = 0x0fff & ( datagram_high_word >>  0 ); // motor #1
			  *sm1_status = 0x0fff & ( datagram_low_word  >> 12 ); // motor #2
			  *sm2_status = 0x0fff & ( datagram_low_word  >>  0 ); // motor #3
			  break;

		case 1 : *smo_status = 0x0fff & ( datagram_low_word  >> 12 ); // motor #1
			  *sm1_status = 0x0fff & ( datagram_low_word  >>  0 ); // motor #2
			  *sm2_status = 0x0000;                                // motor #3
			 break;

		case 0 : *smo_status = 0x0fff & ( datagram_low_word >>   0 ); // motor #1
			  *sm1_status = 0x0000;                                // motor #2
			  *sm2_status = 0x0000;                                // motor #3
			  break;
	}
}
```

4.9 How to Get the StallGuard Load Indicator (LD) Bits of TMC246 / TMC249

The following C code example (part of tmc429demo.zip) outlines how to get the load indicator bits of each TMC246 or TMC249 driver to be used for StallGuard.

```
void tmc429_get_tmc24x_stallguard_bits( int sm, int *ld )
{
	int smo_status, sm1_status, sm2_status;

	tmc429_get_tmc24x_status_bits( &smo_status, &sm1_status, &sm2_status );

	switch (((char)sm))
	{
		case 0 : *ld = 0x7 & ( smo_status >> 9 ); break;
		case 1 : *ld = 0x7 & ( sm1_status >> 9 ); break;
		case 2 : *ld = 0x7 & ( sm2_status >> 9 ); break;
	}
}
```

At a given speed, a stall of the motor is detected by comparing load indicator bits (LD2, LD1, LD0) forming a three bit vector LD with a threshold. The threshold has to be determined for a given stepper motor within its application. The stallGuard signal (load indicator bits) depends on the speed of the motors.

5 StallGuard (1) Profiler

StallGuard (1) is primarily intended for noiseless reference search with a mechanical reference position. How well StallGuard works primarily depends on three constrains from the stepper motor and its application:

- efficiency of a stepper motor in terms of mechanical power vs. power dissipation
- difference in mechanical load between free running and stall on barrier
- velocity of the stepper motor

If a given stepper motor and its application fit well to StallGuard, the optimal velocity has to be determined for StallGuard. Generally, there is not only one optimal velocity for StallGuard. There are ranges of velocities that are sufficient for StallGuard.

The goal of a StallGuard profiler is to determine the ranges of velocities that fit well for the sensorless stall detection. Each velocity of a given stepper motor is associated with an individual StallGuard level. Mechanical load – as occurs on a stall – changes the StallGuard level.

There are two kinds of StallGuard profilers: One for continuous motion and one for motion within a limited range of motion.

Because StallGuard is able to sense oscillations of the rotor, after acceleration it is necessary to wait a while until the StallGuard load indicator are valid to detect a stall.

So, first one has to accelerate to the velocity that is to be profiled. When the acceleration phase is finished, one has to wait a while (typical 100 ms for a free running stepper motor) before processing of the StallGuard load indicator bits. Then one measures n (e.g. n=10) times the value of the load indicator bits an calculates and calculates the mean value and the standard deviation for each velocity.

The resolution of StallGuard concerning detection of a mechanical reference point is one full step.

5.1 StallGuard Demo – Source Code Example

A source code example of a stallGuard demonstration (routine tmc429_stallguard_demo(...)) is part of the tmc429demo.zip (pls. refer 'tmc429_misc.c). The parameters are

sm : stepper motor (0, 1, 2)
vmax : velocity of the stepper motor number **sm**
sgl : StallGuard level (0, 1, 2, 3, 4, 5, 6, 7)

The read stallGuard values (load indicator bit vectors **ld**) are continuously printed on the console. If a stall is detected (**ld** < **sgl**) it additionally prints "stallGuard @ x_actual" where x_actual is the actual position where a stall has been detected. Other parameters (a_max, ...) are use as they currently set.

5.2 StallGuard Profiler – Source Code Example

A source code example of a StallGuard profiler (routine tmc429_stallguard_profiler(...)) is are part of the tmc429demo.zip (pls. refer 'tmc429_misc.c). This StallGuard Profiler performs both, StallGuard profiling for continuous motion (VELOCITY_MODE) and StallGuard profiling for motion within limited range of motion. The parameters are

sm	: stepper motor (0, 1, 2)
v_traget_min	: minimum target velocity of the stepper motor for profiling
v_target_max	: maximum target velocity of the stepper motor for profiling
v_step	: target velocity increment width
x_min	: start position for profiling
x_**max**	: end position for profiling

Other parameters (a_max, ...) are used as they currently set.

5.3 StallGuard Profiler for continuous motion (VELOCITY MODE)

For **x_min==x_max** the StallGuard profiler runs in VELOCITY_MODE for StallGuard profiling within continuous motion.

5.4 StallGuard Profiler for limited motion range (RAMP_MODE)

For **x_min!=x_max** the stallGuard profiler runs in RAMP_MODE moving between position **x_min** and position **x_max**. For each actual target velocity, the stallGuard profiler calculates the number of full steps required to perform stallGuard profiling (pls. refer tmc429_stallguard_profiler(...)for details). If the number of full steps are sufficient to perform stallGuard profiling it does it, if not it skips the profiling for the target velocity and prints an error message concerning this.

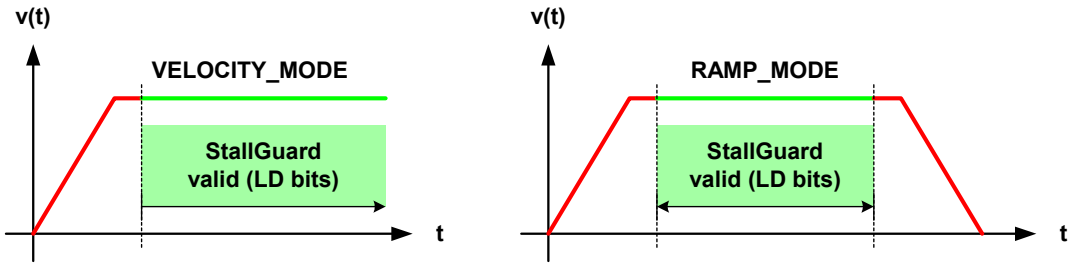

Figure 7 : StallGuard profiling (left: continuous motion / right: motion within limited range)

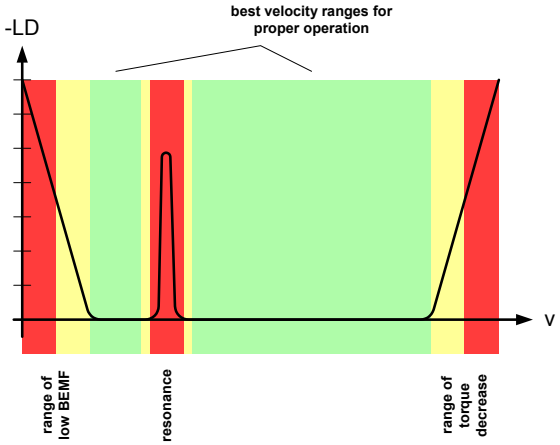

Figure 8 : Outline of a typical StallGuard Profile

5.5 StallGuard – General Constrains

5.5.1 Velocity Ranges for Proper Operation of StallGuard

For stallGuard, on should run the stepper motor within the velocity ranges that are best for proper operation of stallGuard. For a given stepper motor, these ranges can be determined using the stallGuard Profiler. Although it is not the goal of the stallGuard profiler, it can be used to determine the resonance frequency of a stepper motor.

Proper velocities of operation for StallGuard are those with low LD bit vector value and low standard deviation. Without load, one gets load indicator bit vector LD. Under mechanical load at the axis of the stepper motor, one gets a lower value of the load indicator bit vector LD.

So, velocities with high load indicator values are good for stallGuard. Low standard deviation means, that one gets stable load indicator bits. The load indicator values might vary by one if they are close to one of the internal thresholds of the TMC246 / TMC249 stepper motor drivers at a given velocity.

5.5.2 Mixed-Decay to be set OFF for StallGuard

The mixed-decay (MD) feature of the TMC246 and TMC249 has to be set off (MDA='0', MDB='0') when using stallGuard. This is because the mixed-decay forces a better current regulation but it regulation disadvantageously interferes the measurements of stallGuard. Under special conditions, stallGuard might work together with mixed-decay, but with restrictions.

5.5.3 Full Step vs. Half Step vs. Micro Step

StallGuard is compatible with full stepping, but it gives the best performance with micro stepping due to lower mechanical resonances of the stepper motor when driven by micro stepping [Larsson2003].

5.5.4 Resistances

The coil resistance of a stepper motor and the resistance of the sense resistors should be of the same order of magnitude – not of same value. Stepper motors of low coil resistance are advantageous in most cases, because those kinds of stepper motors mostly have a higher efficiency. On the other hand, a higher resistance of the sense resistors improves the stallGuard by higher signal amplitude of the sense signal and by lower signal noise ratio. For higher resistance sense resistors one can use a higher external reference voltage of up to 3V ((please refer TMC246 / TMC249 datasheets for details).

5.6 How to Compare Different Motors Concerning StallGuard Qualification

One could directly compare stepper motors concerning the qualification for StallGuard based on the efficiency. The efficiency η naturally is not found within stepper motor datasheets. This is because stepper motors are not intend to be most efficient in terms of mechanical efficiency – stepper motors are efficient in terms of pricing, precise control, reliability, torque at low speed, cost concerning mechatronic systems.

5.6.1 Comparison by Hand

A simple test to compare stepper concerning their efficiency is to short both coils an to turn the axis. A stepper motor of high efficiency breaks stronger than a stepper motor of lower efficiency.

5.6.2 Comparison Based on Resonance Frequency

Normally, the resonance frequency ν_0 is not found within stepper motor datasheets. The torque almost vanishes at a speed near resonance. It can be avoided by micro stepping or by fast going through the resonance frequency range. Nevertheless, the resonance frequency of a stepper motor can relatively easy be measured and characterizes it concerning its efficiency.

A torque τ proportional to a displacement ϕ between rotor and magnetic field forms a harmonic oscillator with a resonance frequency

$$\nu_0 = \frac{1}{2\pi} \cdot \sqrt{\frac{\kappa_I \cdot I}{J_M}},$$

where $I = (I_x^2 + I_y^2)^{1/2}$ is the absolute value of the coil currents I_x and I_y, and J_M is the moment of inertia of rotor axis. This expression can be transformed to

$$4\pi^2 \cdot J_M \cdot \nu_0^2 = \kappa_I \cdot I$$

In other words, at a given current I the coupling constant

$$\kappa_I \propto J_M \cdot \nu_0^2$$

is proportional to the moment of inertia J_M times the square of the resonance frequency. This allows comparing stepper motors of different sizes concerning their qualification for stallGuard.

"A stepper motor with a high $J_M \cdot \nu_0^2$ is good for stallGuard"

A stepper motor might have higher harmonics. So, the $J_M \cdot \nu_0^2$ has to be on the basis of the first harmonic, the resonance with the highest amplitude.

5.6.3 Why to Take Both, the Torque and the Moment of Inertia in into Account

A high torque itself does not imply a high efficiency. Generally, the torque of stepper motors scales with their size and the moment of inertia scales with their size.

5.6.4 Competition Based on Torque vs. Speed Diagram

If a torque over speed diagram is available for a stepper motor one can compare stepper motors by its efficiency, calculating $\eta = \tau$[Nm] * ω[rad/s] / (I [A]² * R[Ω] + τ[Nm] * ω[rad/s]) for a given angular velocity.

5.7 Overview of 'tmc429demo'

These routines all together build a simple example application to control the TMC429. One can directly run it together with a TMC429 evaluation kit. These sources are distributed in the hope that they will be useful. They might be a base for your own application. The C code has been compiled using MS Visual C++ 6.0 to build the tmc429demo.exe Win32 console application.

The software runs with evaluation boards TMC428-EVAL, TMC429-EVAL, TMC429+TMC24x-EVAL, and with TMC429+TMC26x-EVAL. This application note is primarily intended to show how to program the TMC429 with TMC246/TMC249 drivers in a SPI stepper motor driver chain architecture. Running the software with with a TMC429+TMC26x-EVAL shows how use the TMC429 for motion control – but for that board the initialization of the TMC26x is done by the firmware of the board.

rs232win.c	:	routines to access RS232 under Win32 (Win9x, NT4, Win2k, XP, W7...)
rs232win.h	:	header file for rs232win.c
tmc429spi.c	:	SPI routine tmc428spi() - calling tmc429spi4win() for the Win32 application
tmc429spi.h	:	header file of tmc429spi.c
tmc429spi4win.c	:	routines to access a TMC429 on evaluation board via RS232 under Win32
tmc429spi4win.h	:	header file of tmc429spi4win.c
tmc429base.c	:	basic TMC429 access routines (basic register IO, ...) using tmc429spi.c
tmc429base.h	:	header file for tmc429_base.c
tmc429misc.c	:	Miscellaneous routines (dump of internal TMC429 configuration RAM, calculation of full step frequency from TMC429 parameters, stallGuard demonstration and stallGuard profiler)
tmc429misc.h	:	header file for tmc429misc.c
tmc429demo.c	:	MAIN program demo using routines of 'tmc429spi.c', 'tmc429base.c', ...
tmc429demo.exe	:	Win32 executable - run this program from console (execute: cmd.exe)

The LSMD is set to 1 (=2 drivers) for the TMC429 evaluation board (TMC429+TMC246-EVAL).

5.8 Overview of 'sample428_236'

This example can be used as a frame for an own micro controller based application. It is written in a compact form. In contrast to the 'tmc429demo' - where a separate routine is available for each type of register – the single routine send428() is used to handle directly the communication with the TMC429. The SPI datagramms are composed directly – supported by a set of macros. This code was written for the old TMC428-EVAL board.

sample428_236.c	:	frame for a stand-alone micro controller application
sample428_236.h	:	header file for sample428_236.c

6 Literature & Links

[Larsson2003] Lars Larsson, Micro Step vs. Full Step – a Quantitative Competition, SAE'2003 World Congress, Detroit, Michigan, USA, March 3-6, 2003, SAE Technical Paper Series,
Paper # 2003-01-0093, SAE International, Warrendale, USA, March 2003

[TMC428 – Data Sheet Version 2.02 (last version)]
[TMC428 Evaluation Kit V3.0 (or higher) Manual]

TMC429 Datasheet
TMC429+TMC24X-EVAL, evaluation board manual
TMC429+TMC26X-EVAL, evaluation board manual

TMC236 Datasheet
TMC239 Datasheet
TMC246 Datasheet
TMC249 Datasheet

TMCL – Reference and Programming Manual

Up to date documentation of all TRINAMIC products is available on www.trinamic.com Up to date documentation of all TRINAMIC products is available on www.trinamic.com

7 Revision history

7.1 Documentation revision

Version	Date	Author LL = Dr. Lars Larsson	Description
1.0	2012-FEB-10	LL	Initial Version, based on Application Note "TMC428 Getting Started including stallGuard"

Table 1: Documentation Revisions