TMC429 – DATA SHEET

Intelligent Triple Stepper Motor Controller with Serial Peripheral Interfaces and Step Direction

– Full Compatible Successor of the TMC428 –

TRINAMIC® Motion Control GmbH & Co. KG
Waterloohain 5
D – 22769 Hamburg
GERMANY

www.trinamic.com

1 Features

The TMC429 is a miniaturized high performance stepper motor controller. It controls up to three 2-phase stepper motors. All motors can operate independently. The TMC429 allows up to 6 bit micro step resolution via SPI™ – corresponding to 64 micro steps per full step – individually selectable for each motor. Once initialized, it performs all real time critical tasks autonomously based on target positions and velocities, which can be altered on-the-fly. So, an inexpensive microcontroller together with the TMC429 forms a complete motion control system. The microcontroller is free to do application specific interfacing and high level control functions. Both, the communication with the microcontroller and with one to three daisy chained stepper motor drivers take place via two separate 4 wire serial peripheral interfaces. The TMC429 directly connects to SPI™* smart power stepper motor drivers or via step-direction interface.

- Controls up to three stepper motors
- Serial 4-wire interface for µC with easy-to-use protocol
- Configurable interface for SPI™ motor drivers
- Step / Direction (S/D) interface
- Different types of SPI™ stepper motor driver chips may be mixed within a single daisy chain
- Communication on demand minimizes traffic to the SPI™ stepper motor driver chain
- Programmable SPI™ data rates up to 1 Mbit/s
- Wide range for clock frequency – can use CPU clock up to 32 MHz
- Internal 24 bit wide position counters
- Full step frequencies up to 20 kHz
- Read-out facility for actual motion parameters (position, velocity, acceleration) and driver status
- Individual micro step resolution of {64, 32, 16, 8, 4, 2, 1} micro steps via built-in sequencer
- Programmable 6 bit micro step table with up to 64 entries for a quarter sine-wave period
- Built-in ramp generators for autonomous positioning and speed control
- On-the-fly change of target motion parameters (like position, velocity, acceleration)
- Automatic acceleration dependent current control (power boost)
- Low power operation: Only 1.25mA @ 4 MHz (typ.)
- Power down mode with transparent wake-up for normal operation
- 3.3V or 5V operation with CMOS / TTL compatible IOs (all inputs Schmitt-Trigger)
- Available in ultra small 16 pin SSOP package, small 24 pin SOP package, and 32 pin QFN 5x5mm

Table of contents, table of figures, table of tables are located at the end of this datasheet.

* *SPI is Trademark of Motorola, Inc.*

Life support policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

2 TMC429 vs. TMC428

The TMC429 is the 100% functional compatible and pin compatible successor of the TMC428. The TMC429 can be used as a TMC428 in existing hardware / software environments. There are some additional functions of the TMC429 that are mapped in register address ranges that where indicated reserverd addresses for later versions. Without access to these additional registers of the TMC429, the TMC429 behaves identically as a TMC428. The TMC429 has a step/direction interface that perfect fits to the TRINAMIC stepper motor driver family TMC260, TMC261, and TMC262. The TMC428 can be clocked up to 16MHz where the TMC429 can be clocked with up to 32MHz. Registers, that are available for the TMC429 only are labled with _429 (e.g. register if_configuration_429). The additional TMC429 registers are described in sections 10.4, 10.5, 10.6,10.7, 10.9 and their sub-sections.

3 General Description

The TMC429 is a miniaturized high performance stepper motor controller with a unique price / performance ratio for both, high volume automotive and for demanding industrial motion control applications. Once initialized, the TMC429 controls up to three 2-phase stepper motors. Its low price makes it attractive also for applications, where only one or two stepper motors have to be controlled simultaneously.

The TMC429 performs all real time critical tasks autonomously. Thus a low cost microcontroller is sufficient to perform the tasks of initialization, application specific interfacing, and to specify target positions and velocities. The TMC429 allows on-the-fly change of all motion target parameters also during motion. Any other parameter may be changed at any time– also during motion –which does not make sense in any case, but this uniform access to any TMC429 register simplifies application programming. Read-back option for all internal registers simplifies programming. With its internal position counters, the TMC429 can perform up to 2^{23} steps respectively micro steps fully independent from the microcontroller. The step resolution– individually programmable for each stepper motor – ranges from full step (1 ‘’micro step’’ is one full step), half step (2 ‘’micro steps’’ per full step), up to 6 bit micro stepping (64 micro steps per full step) for precise positioning and noiseless stepper motor rotation (Table 9-8, page 29). Optionally, the micro step table– common for all motors –can be adapted to motor characteristics to further reduce torque ripple.

The TMC429 has got serial interfaces for communication with the microcontroller and for the stepper motor drivers. The serial interface for the microcontroller uses a fixed length of 32 bits with a simple protocol, directly connecting to SPI™ interfaces. The serial interface to the stepper motor drivers is flexibly configurable for different types– even from different vendors –with up to 64 bit length for the SPI daisy chain. TRINAMIC smart power stepper motor drivers TMC236, TMC239 and TMC246, TMC249 perfectly fit to the TMC429. Without additional hardware, drivers with same serial interface polarities of chip select and clock signals may be mixed in a single chain. To mix drivers with different serial interface polarities, additional inverters (e.g. 74HC04, 74HC14) are required. For those driver chips without serial data output, two additional variants of the TMC429 with two additional chip select outputs are available. The TMC429 sends data to the driver chain on demand only, which minimizes the interface traffic and reduces the power consumption.

Hint: Unused reference switch inputs (REF1, REF2, and REF3) should be pulled to ground (Figure 3-3). With this one can connect reference switches that connect to +5V resp. +3.3V when pushed. Concerning different reference switch configurations please refer to Figure 10-4, Figure 10-5, Figure 10-7.

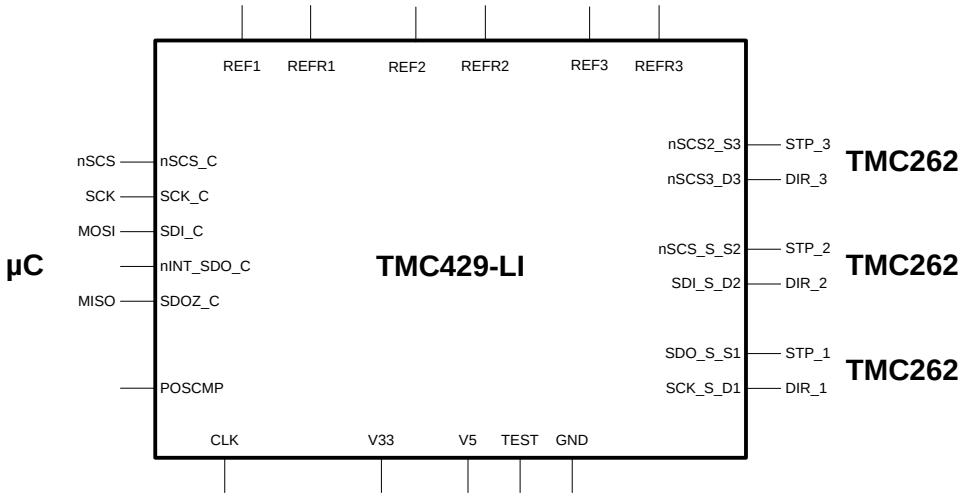

Figure 1: TMC429 within QFN32 package.

All signals of the TMC429 are available for the QFN32 package. This package is recommended, especially for the TMC260, TMC261, and TMC262 stepper motor drivers using step/direction interface.

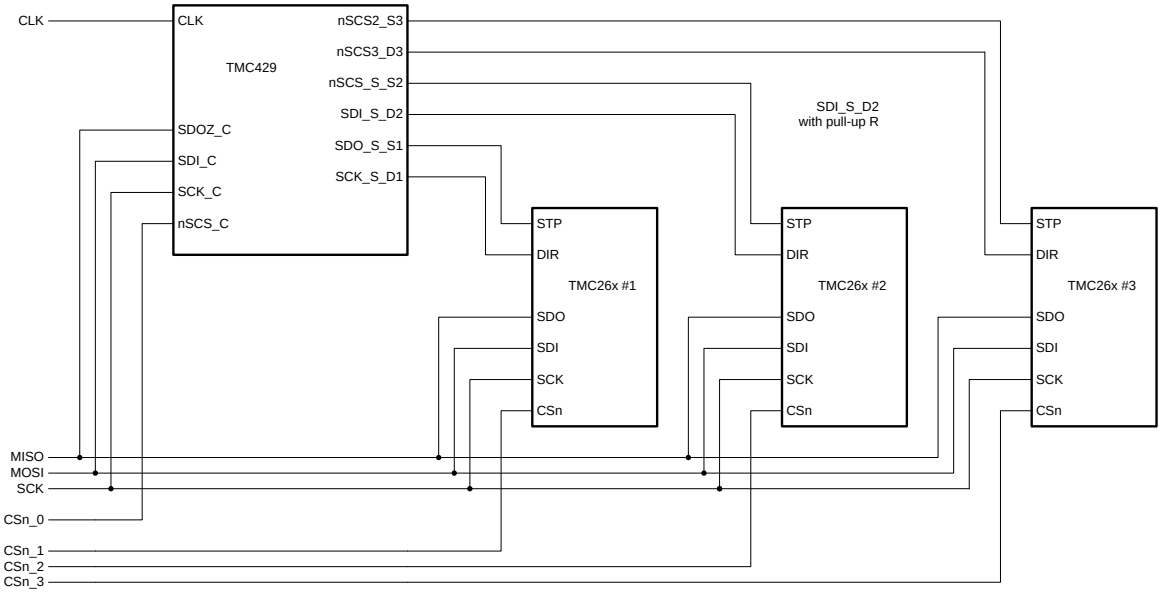

Figure 2: TMC429 / TMC26x outline for configuration via SPI and step direction for motion (only SPI signals and Step/Direcction signals drawn).

Application environment of TMC429 in QFN32 package and 3 x TMC26x stepper motor driver; one SPI chip select signal CSN_0 selects the TMC429 SPI micro controller interface, and (up to) three SPI chip select signals (CSN_3, CSN_2, CSN1) select the (up tp) three TMC262 SPI for configuration; the TMC429 SDOZ_C is high impedance when nSCS_C is '1'.

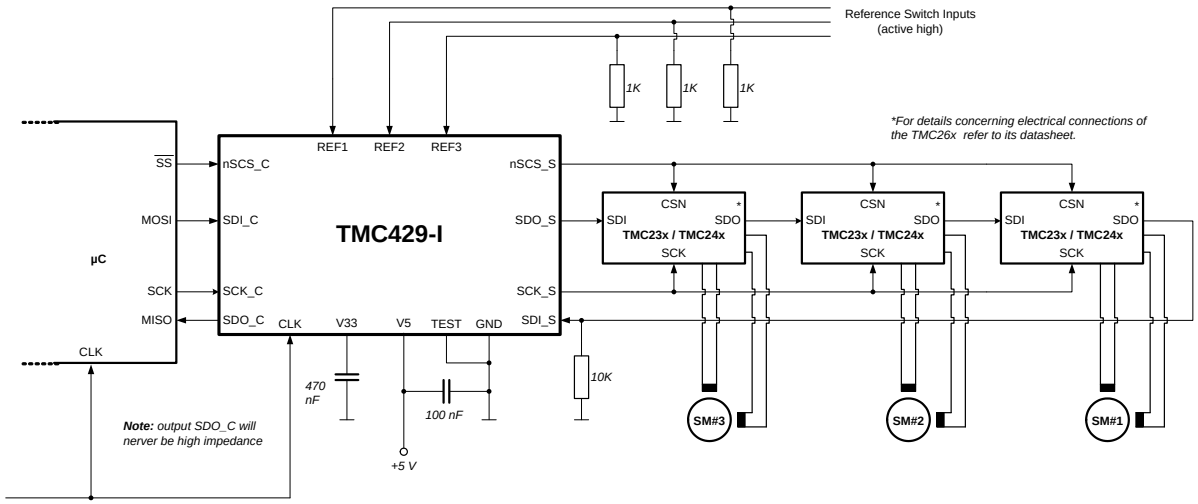

Figure 3-3: TMC429 application environment with TMC429 in SSOP16 package

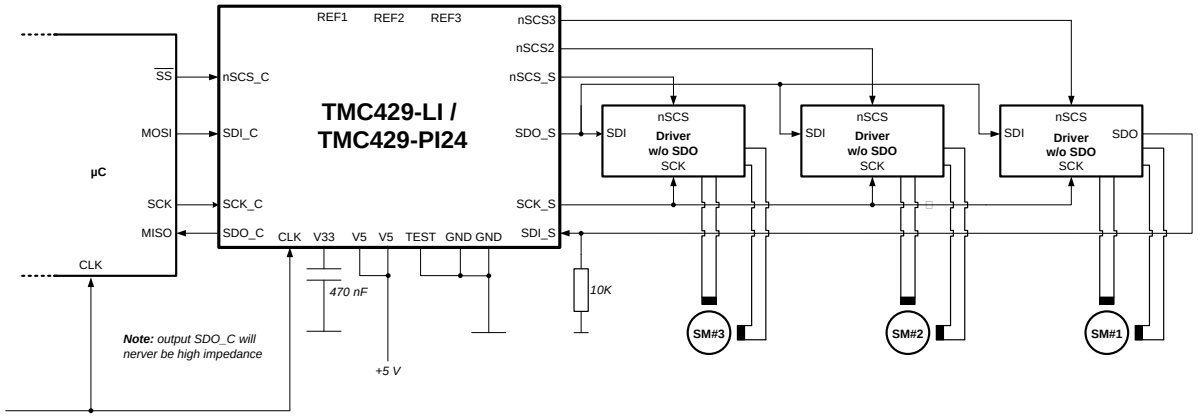

Figure 3-4: Usage of drivers without serial data output (SDO) with TMC429 in larger packages

3.1 Step Frequencies

The maximum SPI™ data rate is the clock frequency divided by 16. The maximum step frequency depends on the total length of the datagrams sent to the SPI™ stepper motor driver chain. At a clock frequency of 16 MHz, with a daisy chain of three SPI™ stepper motor drivers of 16 bit datagram length each, the maximum *full step* frequency is 16 MHz / 16 / (3 * 16). This is approximately 20 kHz and that is much higher than needed for typical stepper motors. But, the micro step rate may be higher, even if the stepper motor driver does not see all micro steps due to SPI™ data rate limit, as long as the number of skipped micro steps is less than a full step. In this respect, one should remember, that at high step rates– respectively pulse rates –the differences between micro stepping and full step excitation vanishes.

3.2 Modes of Motion

The TMC429 has four different modes of motion, programmable individually for each stepper motor, named RAMP_MODE, SOFT_MODE, VELOCITY_MODE, and HOLD_MODE. For positioning applications the RAMP_MODE is most suitable, whereas for constant velocity applications the VELOCITY_MODE is. In RAMP_MODE, the user just sets the position and the TMC429 calculates a trapezoidal velocity profile and drives autonomously to the target position. During motion, the position may be altered arbitrarily. The SOFT_MODE is similar to the RAMP_MODE, but the decrease of the

velocity during deceleration is done with a soft, exponentially shaped velocity profile. In VELOCITY_MODE, a target velocity is set by the user and the TMC429 takes into account user defined limits of velocity and acceleration. In HOLD_MODE, the user sets target velocities, but the TMC429 ignores any limits of velocity and acceleration, to realize arbitrary velocity profiles, controlled completely by the user. The TMC429 has capabilities to generate interrupts depending on different stepper motor conditions chosen by an interrupt mask. However, status bits sent back automatically to the microcontroller each time it sends data to the TMC429 are sufficient for polling techniques.
The TMC429 provides different modes for reference switch handling. In the default reference switch mode, the three reference switch inputs (**REF1**, **REF2**, and **REF3**) are defined as left side reference switches, one for each stepper motor. In another mode, the 1st reference input (**REF1**) is defined as left reference switch input of motor number one, the 2nd reference input (**REF2**) is defined as left reference switch input of motor number two, and the 3rd reference input (**REF3**) is defined as right reference switch of stepper motor number one. In that mode, there is no reference switch input available for stepper motor three. With an external multiplexer 74HC157 any stepper motor may have a left and a right reference switch.
Many serial stepper motor drivers provide different status bits (driver active, inactive, ...) and error bits (short to ground, wire open, ...). To have access to those error bits, datagramms with a total length up to 48 bits sent back from the stepper motor driver chain to the TMC429 are buffered within two 24 bit wide registers. The microcontroller has direct access to these registers. Although, the TMC429 provides datagramms with up to 64 bits, only the last 48 bits sent back from the driver chain are buffered for read out by the microcontroller. This is because buffering of 3 times 16 bits is sufficient for a chain of three stepper motor drivers (see Figure 3-3, page 5) and most other drivers sending back up to 16 bits. For a chain of three TMC236 / TMC239 / TMC246 / TMC249 all status bits are accessible.
From the software point of view, the TMC429 provides a set of registers, accessed by a microcontroller (µC) via a serial interface in an uniform way. Each datagram contains address bits, a read-write selection bit, and data bits, to access the registers and the on-chip memory. Each time, the µC sends a datagram to the TMC429, it simultaneously receives a datagram from the TMC429. This simplifies the communication with the TMC429 and makes the programming easy. Most microcontrollers have an SPI™ hardware interface, which directly connects to the serial four wire microcontroller interface of the TMC429. For microcontrollers without SPI™ hardware, a software doing the serial communication is completely sufficient and can easily be implemented.

3.3 Notation of Number Systems & Notation of Two to the Power of n

Decimal numbers are used as usual without additional identification. Binary numbers are identified by a prefixed % character. Hexadecimal numbers are identified by a prefixed $ character. So, for example the decimal number 42 in the decimal system is written as %101010 in the binary number system, and it is written as $2A in the hexadecimal number system. With this, TMC429 datagramms are written as 32 bit numbers (e.g. $1234ABCD = %00010010000110100101010111001101). In addition to the basic arithmetic operators (+, -, *, */*) the operator *two to the power of n* is required at different sections of this data sheet. For better readability instead of 2^n the notation 2^n is used.

3.4 Signal Polarities

Per default, signals– external and internal –are high active, but the polarity of some signals is programmable to be inverted. A pre-fixed lower case ‘n’ indicates low active signals (e.g. nSCS_C, nSCS_S). For example the polarity of nSCS_S can be inverted by programming, but also the polarity of datagram bits can be inverted by programming (see section 10, page 31).

3.5 Units of Motion Parameters

Motion parameters position, velocity, and acceleration are given as integer values within TMC429 specific units. Section 9.14 page 29 explains how to calculate steps, steps per second, steps per second square from given TMC429 integer values. With a given stepper motor resolution one can calculate physical units for angle, angular velocity, angular acceleration.

3.6 Representation of Signed Values by Two's Complement

Those motion parameters that have to cover negative and positive motion direction as well are processed as signed numbers represented by two's complement as usual. Signed motion parameters are x_target, x_actual, v_target, v_actual, a_actual, a_threshold (pls. refer section 9.9, page 21). Limit motion parameters as v_min, v_max, a_max, are represented as unsigned binary numbers.

3.7 Tables of Contents

A table of contents, a table of figures, and a table of tables are located at the end of the data sheet.

4 Package Variants

The TMC429 is available in three different package variants, qualified for the industrial temperature range. An additional variant is available for the automotive temperature range. The package outlines and dimensions are included within this data sheet (page 52-54.). All package variants are RoHS compilant.

part number	Package	JEDEC Drawing
TMC429-I	SSOP16 – 150 mils, 16 pins, plastic package, industrial (-40°C ... +85°C)	MO-137 (150 mils)
TMC429-PI24	SOP24 – 300 mils, 24 pins, plastic package, industrial (-40°C ... +85°C)	MS-013 (300 mils)
TMC429-LI	QFN32, 5x5mm, 32 pins, plastic package, industrial (-40°C … +85°C)	

Table 4-1: TMC429 package variants

5 Pinning

There are three package variants of the TMC429 available. The smaller SSOP16 package is sufficient for TRINAMIC stepper motor drivers (TMC236 / TMC239 / TMC246 / TMC249) with up to three drivers in a chain and for most SPI™ stepper motor drivers from other vendors. Some SPI™ stepper motor drivers from other vendors have no serial data output and cannot simply be arranged in a daisy chain to drive more than one motor. The two package variants SOP24 and QFN32 have two additional driver selection outputs (**nSCS2**, **nSCS3**) for those stepper motor drivers without serial data output. All inputs are Schmitt-Trigger. Possibly unused inputs (**REF1**, **REF2**, **REF3**, and **SDI_S**) need to be connected to ground.

Compared to the TMC428, the TMC429 is equipped with additional outputs. Some outputs have additional TMC429 specific functions. After power on-reset, the TMC429 starts in TMC428 mode. The TMC429 functions are activated by dedicated TMC429 configuration registers. The TMC429 has additional reference right side switch inputs **REF1R**, **REF2R**, and **REF3R**. With this, there is no need for an additional multiplexer 74hc157 as required for the TCM428. These right-side reference switch inputs are important for step/direction mode because the multiplexing control signal is available only in SPI stepper motor driver chain mode.

The step function outputs are named **S1**, **S2**, and **S3**. The corresponding direction function outputs are named **D1**, **D2**, and **D3**. An additional high impedance SPI output named **SDOZ_C** for the microcontroller interface is available for more comfortable SPI communication. The multiplexed output **nIND_SDO_C** can be configured in a de-multiplxed mode. An additional output named **POSCMP** is available for triggering when moving over a programmable position.

Important Hint: **POSCMP** and ***SDOZ_C*** are outputs for the TMC429. These pins of the TMC428 where not connected for the SPO24 package (TMC428-PI24). So, for replacing a TMC428 by a TMC429 in SOP24 packages the outputs **POSCMP** and **SDOZ_C** must be not connected.

Different sets of additional signals of the TMC429 are available for the diferent package varians of it. Please refer Figure 5-1: TMC429 pin out, page 8, and Table 5-1: TMC429 , page 9. Please refer this figure and this table concerning the pins 1, 12, 13, 24 of the SOP24 package.

Hint: Preferably, long wires to the reference switch inputs (**REF1**, **REF2**, and **REF3**) should be avoided. For long wires, a low pass filter for spike suppression should be provided (pls. refer the TMC429 evaluation board schematic as an example).

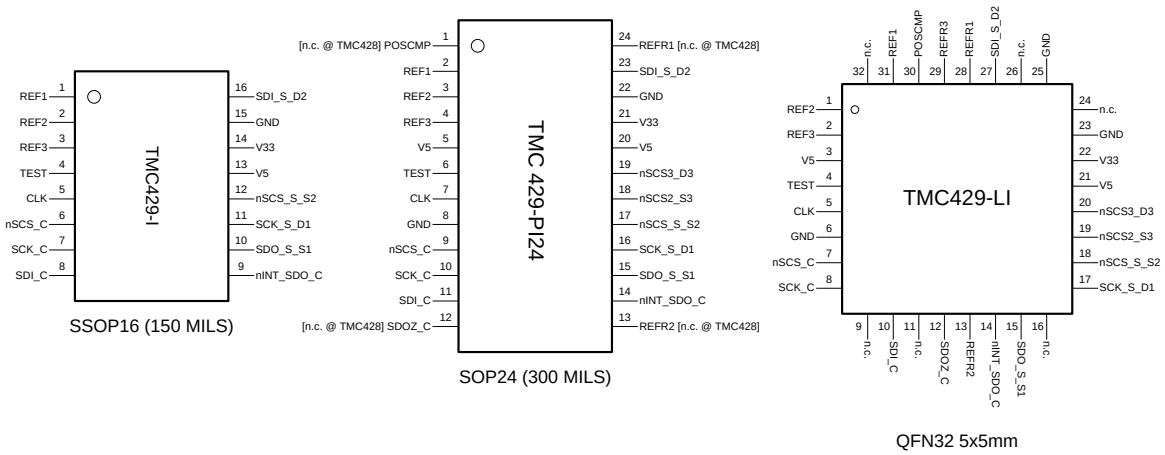

Figure 5-1: TMC429 pin out

Important Hint: To replace a TMC428 in SOP24 package by a TMC429 in SOP24 package on an existing PCB, the pin 1 needs to be open, because it is an output for the TMC429 in SOP24 package.

The TMC428-I in SSOP16 can by replanced by a TMC429-I without restrictions.

Pin [TMC428 name]	SSOP16	SOP24 [TMC428]	SOP24 [TMC429]	QFN32	In/Out	Description
Reset	-	-	-	-	-	internal power-on reset
CLK	5	7	7	5	I	clock input
nSCS_C	6	9	9	7	I	low active SPI chip select input driven from µC
SCK_C	7	10	10	8	I	serial data clock input driven from µC
SDI_C	8	11	11	10	I	serial data input driven from µC
[SDO_C / nINT] nINT_SDO_C	9	14	14	14	O	serial data output to µC input / multiplexed nINTERRUPT output if communication with µC is idle (resp. nSCS_C = 1) **Important Note:** SDO_C will never be high impedance, but this function can added with a single gate 74HCT1G125 (pls. refer Figure 7-2, page12); the TMC429 is equipped with an additional pin named SDOZ_C that becomes high impedance when nSCS_C='1';
[nSCS_S] nSCS_S_S2	12	17	17	18	O	SPI chip select signal to stepper motor driving chain / step output S2 (for motor 2) in step direction mode of TMC429
[nSCS2] nSCS2_S3	-	18	18	19	O	SPI chip select signal (SOP24 package only) / step output S3 (for motor 3) in step direction mode of TMC429
[nSCS3] nSCS3_D3	-	19	19	20	O	SPI chip select signal (SOP24 package only) / direction output D3 (for motor 3) in step direction mode of TMC429
[SCK_S] SCK_S_D1	11	16	16	17	O	serial data clock output to SPI stepper motor driver chain direction output D1 (for motor 1) in step direction mode of TMC429
[SDO_S] SDO_S_S1	10	15	15	15	O	serial data output to SPI stepper motor driver chain; step output S1 (for motor 1) in step direction mode of TMC429
SDI_S_D2 [SDI_S]	16	23	23	27	I O	serial data input from SPI stepper motor driver chain **Note:** pull-up/-down resistor at SDI_S avoids high impedance; SDI_S input is the power-on default; direction output D2 (for motor 2) in step direction mode of TMC429
REF1	1	2	2	31	I	reference switch input 1; no internal pull-up R
REF2	2	3	3	1	I	reference switch input 2; no internal pull-up R
REF3	3	4	4	2	I	reference switch input 3; no internal pull-up R
V5	13	5, 20	5, 20	3, 21		+5V supply / +3.3V supply
V33	14	21	21	22		470nF ceramic capacitor pin / +3.3V supply
GND	15	8, 22	8, 22	6, 23, 25		ground
TEST	4	6	6	4	I	must be connected to GND as close as possible to the chip
n.c.	-	-	-	9, 11, 16, 24,26, 32	-	not connected (n.c.) pins; SOP24 package of the TMC428 has n.c. pins 1, 12, 13, 24
[n.c.] POSCMP	-	**n.c.**	**1**	30	n.c. / O	*TMC428 in SOP24 the pin 1 is not connected (n.c.); TMC429 position compare output for SPO24 and QFN32; output for **pos_comp** function of TMC429*
SDOZ_C	-	**n.c.**	**12**	12	O / Z	SDOZ_C becomes high impedance (Z) when nSCS_C='1'; the nINT signal is not mapped to SDOZ_C pin; the pin **nINT_SDO_C** can be configured with TMC429 register to give the nINT signal directly without multiplexing
REFR1	-	**n.c.**	**24**	28	I	reference switch right 1 input; only available for TMC429 in SOP24 package and QFN32 package; internal pull-up R
REFR2	-	**n.c.**	**13**	13	I	reference switch right 2 input; only available for TMC429 in SOP24 package and QFN32 package; internal pull-up R
REFR3	-	-	-	29	I	reference switch right 3 input; only available for TMC429 in QFN32 package; internal pull-up R

Table 5-1: TMC429 pinning (TMC428 SOP24: pin 1 ⇔ n.c. / TMC429 SOP24: pin 1 ⇔ output)

6 Functional Description and Block Diagram

From the software point of view, the TMC429 provides a set of registers of different units and on-chip RAM (see Figure 6-1), accessed via the serial µC interface in an uniform way. The serial interface uses just a simple protocol with fixed length datagram for read access and write access. The serial interface to the stepper motor driver chain has to be configured by an initialization sequence which writes the configuration into the on-chip RAM. Once configured the serial driver interface works autonomously. The internal multiple port RAM controller of the TMC429 takes care of access scheduling. So, the user may read and write registers and on-chip RAM at any time. The registers hold global configuration parameters and the motion parameters. The on-chip RAM stores the configuration of the serial driver interface and the micro step table. During power-on reset, the TMC429 initializes a default configuration within the on-chip RAM for a SPI driver chain for TMC236/TMC239/TMC246/TMC249 stepper motor drivers.

The ramp generator monitors the motion parameters stored in its registers and calculates velocity profiles controlling the pulse generator. The pulse generator then generates step pulses taking into account user defined motion parameter limits. The serial driver interface sends datagrams to the stepper motor driver chain whenever a step pulse comes. The micro step unit (including sequencer) processes step pulses from the pulse generator– representing micro steps, half steps, or full steps depending on the selected step resolution –and makes the results available to the serial driver interface. The ramp generator also interfaces the reference switch inputs. Unused reference switches have to be connected to ground. A pull-down resistor is necessary at the SDI_S input of the TMC429 for those serial peripheral interface stepper motor drivers that set their serial data output to high impedance ‘Z’ while inactive.

The interrupt controller continuously watches reference switches and ramp generator conditions and generates an interrupt if required. To save pins, the interrupt signal is multiplexed to the SDO_C signal. This output becomes the low active interrupt signal called **nINT** while **nSCS_C** is high (see Figure 7-1, page 12). So, if the microcontroller disables the interrupt during access to the TMC429 and enables the interrupt otherwise, the multiplexed interrupt output of the TMC429 behaves like a dedicated interrupt output. For polling, the TMC429 sends the status of the interrupt signal to the microcontroller with each datagram.

To drive a stepper motor to a new target position, one just has to write the target position into the associated register by sending a datagram to the TMC429. To run a stepper motor with a target velocity, one just has to write the velocity into the register assigned to the stepper motor.

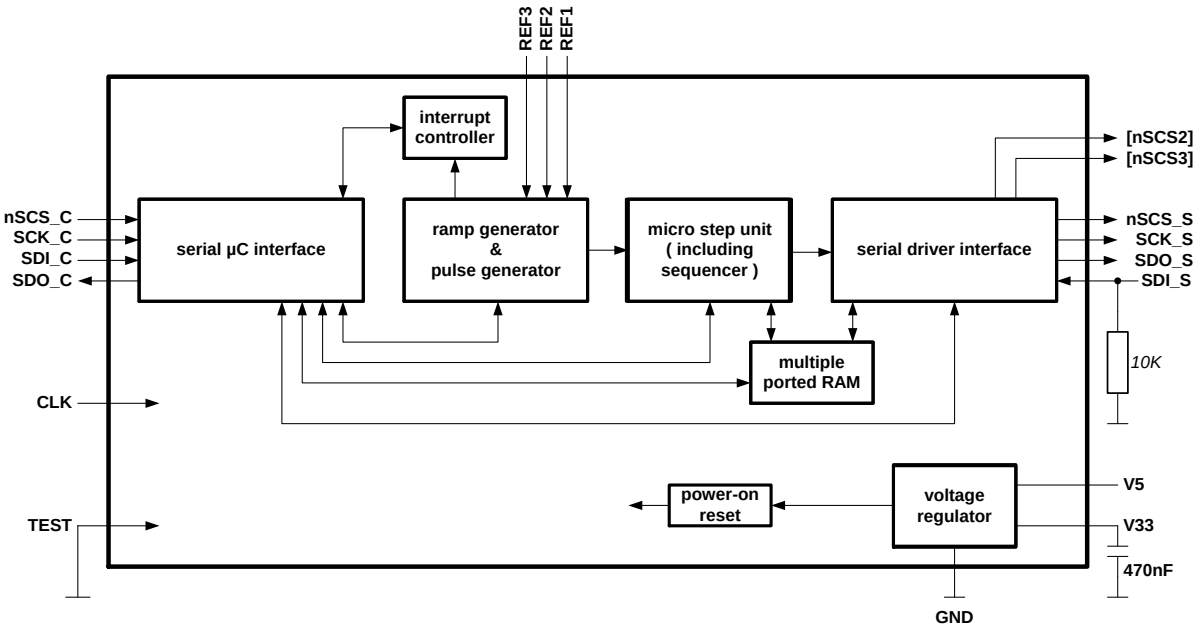

Figure 6-1: TMC428 functional block diagram

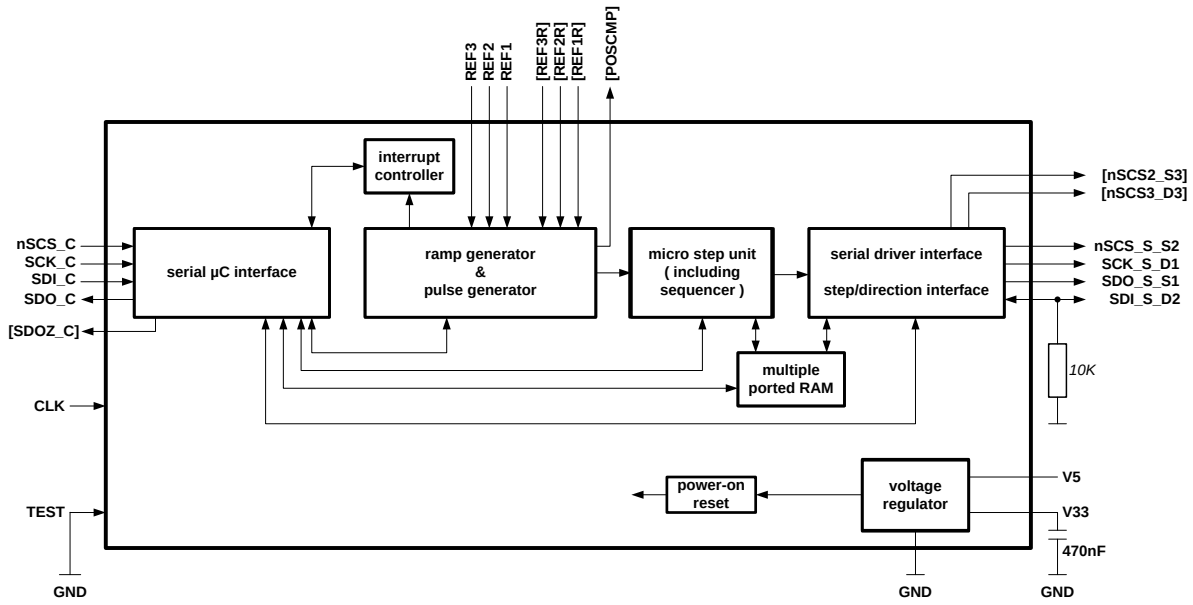

Figure 6-2: TMC429 functional block diagram

7 Serial Peripheral Interfaces

The four pins named SCS_C, SCK_C, SDI_C, and SDO_C form the serial microcontroller interface of the TMC429. The communication between the microcontroller and the TMC429 takes place via 32 bit datagrams of fixed length. Concerning communication, the µC is the master and the TMC429 is the slave, with the TMC429 in turn being the master for the stepper motor driver daisy chain. Similar to the microcontroller interface, the TMC429 uses a four wire serial interface for communication with the stepper motor driver daisy chain. The four pins named SCS_S, SCK_S, SDO_S, SDI_S form the serial stepper motor driver interface. Stepper motor drivers with parallel inputs can be used in connection with the TMC429 with some additional glue logic.

7.1 Automatic Power-On Reset

The TMC429 performs an automatic power-on reset. For details see section Power-On-Reset, page 57. The TMC429 cannot be accessed before the power-on-reset is completed and the clock is stable. All register bits are initialized with ‘0’ during power on reset, except the SPI clock pre-divider **clk2_div** (see section 10.11, page 35) that is initialized with 15.

7.2 Serial Peripheral Interface for µC

The serial microcontroller interface of the TMC429 behaves as a simple 32 bit shift register. It shifts serial data SDI_C in with the rising edge of the clock signal SCK_C and copies the content of the 32 bit shift register with the rising edge of the selection signal nSCS_C into a buffer register. The serial interface of the TMC429 immediately sends back data read from registers or read from internal RAM via the signal SDO_C. The signal SDO_C can be sampled with the rising edge of SCK_C, but SDO_C becomes valid at least four CLK clock cycles after SCK_C becomes low as outlined in the timing diagram Figure 7-1. For detailed timing parameters see Table 7-1, page 14. The SPI signals from the µC interface may be asynchronous to the clock signal CLK of the TMC429.

Because of on-the-fly processing of the input data stream, the serial microcontroller interface of the TMC429 requires the serial data clock signal SCK_C to have a minimum low / high time of three clock cycles. The data signal SDI_C driven by the microcontroller has to be valid at the rising edge of the

serial data clock input SCK_C. The maximum duration of the serial data clock period is unlimited. While the µC interface of the TMC429 is idle, the SDO_C signal is the (active low) interrupt status nINT of the integrated interrupt controller of the TMC429. The timing of the multiplexed interrupt status signal nINT is characterized by the parameters tIS an tSI (see Table 7-1, page 14).

Hint: If the microcontroller and the TMC429 work on different clock domains that run asynchronous to each other, the timing of the SPI interface of the microcontroller should be made conservative in the way that the length of one SPI clock cycle equals 8 or more clock cycles of the TMC429 clock CLK. This make the system robust concerning frequency drift, jitter, etc.

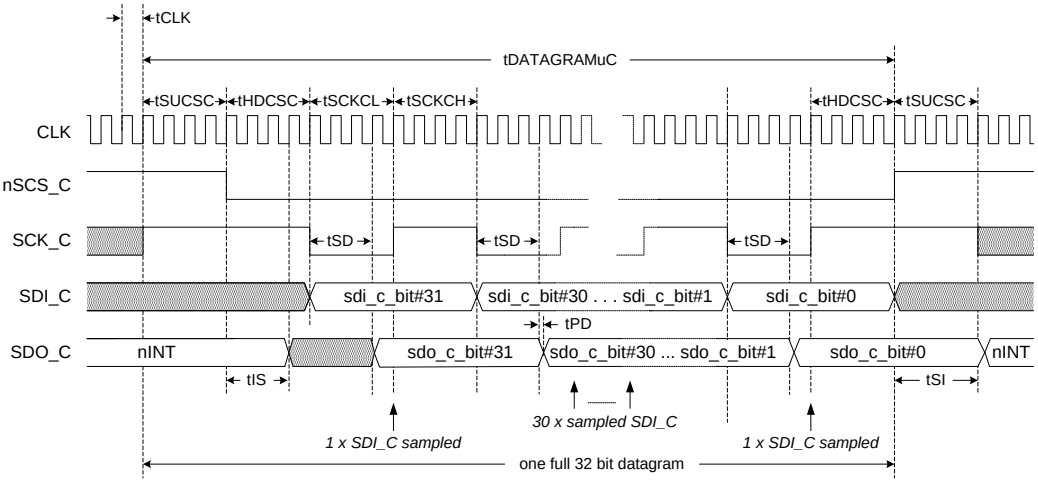

Figure 7-1: Timing diagram of the serial µC interface

A complete serial datagram frame has a fixed length of 32 bit. While the data transmission from the microcontroller to the TMC429 is idle, the low active serial chip select input nSCS_C and also the serial data clock signal SCK_C are set to high. While the signal nSCS_C is high, the TMC429 assigns the status of the internal low active interrupt signal named nINT to the serial data output SDO_C (see Figure 7-1). The serial data input SDI_C of the TMC429 has to be driven by the microcontroller.

Important Hint: In contrast to most other SPI™ compatible devices, the SDO_C signal of the TMC429 is always driven. So, it will never be high impedance 'Z'. If high impedance is required for the SDO_C connected to the microcontroller, it can simply be realized using a single gate 74HCT1G125. An additional pin named SDOZ_C is available for the TMC429 with an integrated high impedance driver.

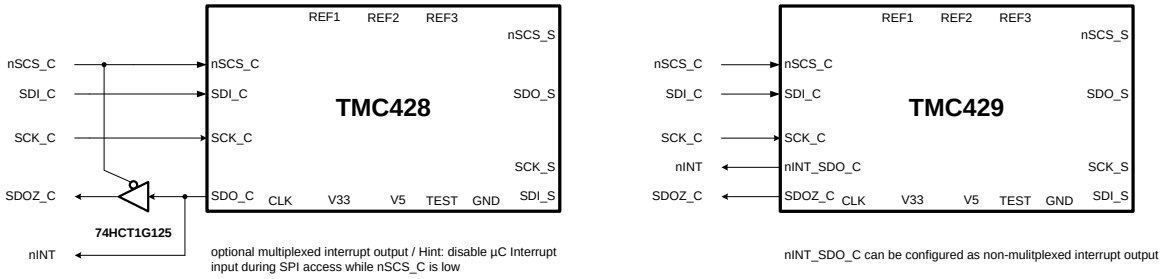

Figure 7-2 : TMC428 SDO_C output high impedance with single gate 74HCT1G125 vs. The TMC429 has a dedicated high impedance pin SDOZ_C available and the nINT_SDO_C can be nINT.

The signal nSCS_C has to be high for at least three clock cycles before starting a datagram transmission. To initiate a transmission, the signal nSCS_C has to be set to low. Three clock cycles later the serial data clock may go low. The most significant bit (MSB) of a 32 bit wide datagram comes first and the least significant bit (LSB) is transmitted as the last one. A data transmission is finished by setting nSCS_C high three or more CLK cycles after the last rising SCK_C slope. So, nSCS_C and SCK_C change in opposite order from low to high at the end of a data transmission as these signals change from high to low at the beginning. The timing of the serial microcontroller interface is outlined in Figure 7-1.

7.3 Serial Peripheral Interface to Stepper Motor Driver Chain

The timing of the serial stepper motor interface is similar to that of the microcontroller interface. It directly connects to SPI™ smart power stepper motor drivers. The SPI™ datagram is configurable individually for each stepper motor driver chip of the daisy chain. It is simply configurable by sending a fixed sequence of datagrams to the TMC429 to initialize it after power-up. Once initialized, the TMC429 autonomously generates the datagrams for the stepper motor driver daisy chain without any additional interventions of the microcontroller.

The SPI™ datagram for each stepper motor driver is composed of so called *primary signal bits* provided by the micro step unit of the TMC429 individually for each stepper motor. Each primary signal bit is represented by a five bit code word called *primary signal code*. The order of primary signal bits forming the SPI™ datagrams for the stepper motor driver daisy chain is defined by the order of primary signal code words in the configuration RAM area.

Hint: For clock frequency fCLK = 16MHz an uppder SPI clock frequency of 1MHz is recommended. For a clock frequency fCLK = 32MHz an uppder SPI clock frequency of 2MHz is recommended. This avoids possible issues concerning clock phase polarity between micro controller and TMC429.

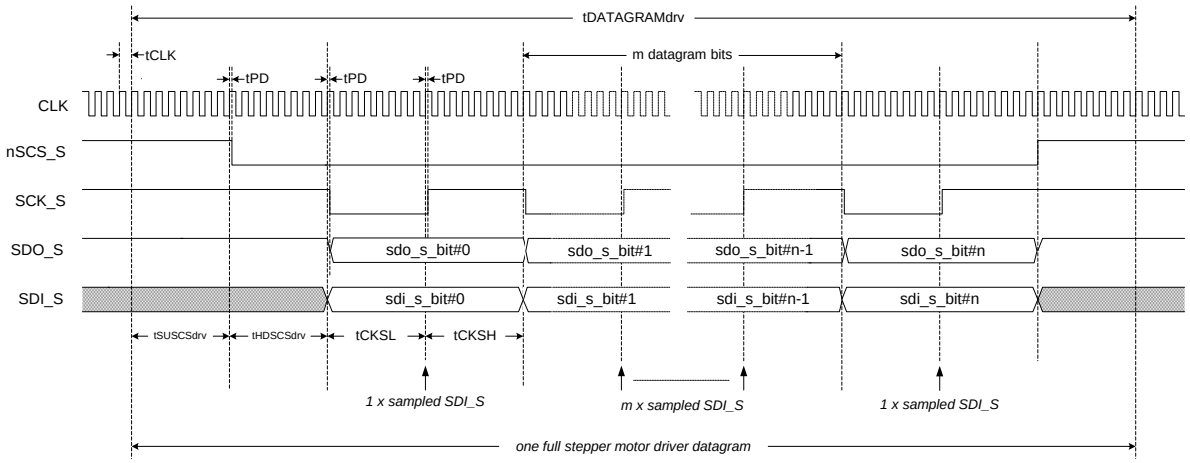

Figure 7-3: Timing diagram of the serial stepper motor driver interface

To switch to the next motor, an additional bit called *next motor bit* (NxM-Bit) is prefixed to the five bit wide primary signal code words. So, the total data word width is six bit. Each NxM-Bit effects an increment of an internal stepper motor address until the processing for all stepper motors within the daisy chain is completed. A parameter called **LSMD** (last stepper motor driver) defines the total number of stepper motors within the daisy chain. So, the codes written into the serial interface configuration RAM area represent the mapping of control signals provided by the micro step units to control bits of the drivers. It might be noted here, that configuring the serial driver interface is much easier as it might seem here. It is explained in detail, illustrated by examples below (see section 12 Stepper Motor Driver Datagram Configuration, page 41).

Symbol	Parameter	Min	Typ	Max	Unit
tSUCSC	Setup Clocks for nSCS_C	3		∞	CLK periods
tHDCSC	Hold Clocks for nSCS_C	3		∞	CLK periods
tSCKCL	Serial Clock Low	3		∞	CLK periods
tSCKCH	Serial Clock High	3		∞	CLK periods
tSD	SDO_C valid after SCK_C low	2.5		3.5	CLK periods
tIS	nINTERRUPT status valid after nSCS_C low	2.5			CLK periods
tSI	SDO_C valid after nSCS_C high			4.5	CLK periods
tDAMAGRAMuC	Datagram Length	3+3 + 32*6 = 198		∞	CLK periods
tDAMAGRAMuC	Datagram Length	12.375		∞	µs
fCLK	Clock Frequency	0		32	MHz
tCLK	Clock Period tCLK = 1 / fCLK	31.25		∞	ns
tPD	CLK-rising-edge-to-Output Propagation Delay		5		ns

Table 7-1: Timing characteristics of the serial microcontroller interface

Symbol	Parameter	Min	Typ	Max	Unit
tSUSCSdrv		8	16	256	CLK periods
tHDSCSdrv		8	16	256	CLK periods
tCKSL		8	16	256	CLK periods
tCKSH		8	16	256	CLK periods
tDAMAGRAMdrv	Datagram Length	8+8+1*16+8+8=48		512+64*512+512= 33792	CLK periods
tDAMAGRAMdrv	Datagram Length @ fCLK = 16 MHz	3		2112	µs
tDAMAGRAMdrv	Datagram Length @ fCLK = 32 MHz	3		1056	µs
tPD	CLK-rising-edge to Outputs Delay		5		ns

Table 7-2: Timing characteristics of the serial stepper motor driver interface

The timing of the serial driver interface is programmable in a wide range. The clock divider provides 16 up to 512 clock cycles (tCLK) for a serial driver interface data clock period. The default duration of a clock period (tSCKCL+tSCKCH) of the signal nSCS_S is 16+16=32 clock periods of the clock signal CLK. The minimal duration of a serial interface clock period (tSCKCL+tSCKCH) is 8+8=16 clock cycles of signal CLK as outlined in Figure 7-3. Also, the polarities of the signals nSCS_S and SCK_S are programmable to use driver chips from other vendors with inverted polarities without additional glue logic. The input SDI_S of the serial driver interface must always be driven to a defined level. So, to avoid high impedance ('Z') at that input pin while the stepper motor driver chain is idle, a pull-up resistor or a pull-down resistor of 10 KΩ is required at that input.

7.4 Datagram Structure

The microcontroller (µC) communicates with the TMC429 via the four wire (nSCS_C, SCK_C, SDI_C, SDO_C) serial interface. Each datagram sent to the TMC429 via the pin SDI_C and each datagram received from the TMC429 via the pin SDO_C is 32 bits long. The first bit sent is the MSB (most significant bit named sdi_c_bit#31 at Figure 7-1). The last bit sent is the LSB (least significant bit named sdi_c_bit#0 in Figure 7-1). During reception of a datagram, the TMC429 immediately sends back a datagram of the same length to the microcontroller. This datagram is the result of the request from the microcontroller.

With each 32 bit wide datagram the microcontroller sends to the TMC429, it simultaneously receives a 32 bit wide datagram. A read request is distinguished from a write request by one datagram bit named **RW**. The TMC429 immediately sends back requested read data in the lower 24 datagram bits. Status bits are sent back in the higher 8 datagram bits. Datagrams sent from the microcontroller to the TMC429 have the form:

MSB	32 bit DATAGRAM sent from µC to the TMC429 via pin SDI_C																													LSB
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRS	ADDRESS						RW	DATA																							

Table 7-3 : 32 bit DATAGRAM structure sent from µC (MSB sent first)

The 32 bit wide datagrams sent to the TMC429 are assorted in four groups of bits: **RRS** (register RAM select) selecting either registers or on-chip RAM; **ADDRESS** bits addressing memory within the register set or within the RAM area; **RW** (read / not write (**RW**=**1** : **read** / **RW**=**0** : **write**)) bit distinguishing between read access and write access; **DATA** bits for write access– for read access these bits are *don't care* and should be set to '0'. Different internal registers of the TMC429 have different lengths. So, for some registers only a subset of these 24 data bits is used. Unused data bits should be set to '0' for clearness. Some addresses select more than a single register mapped together into the 24 data bit space.

The 32 bit wide datagrams received by the µC from the TMC429 contain two groups of bits: **STATUS BITS** and **DATA BITS**. The status bits, sent back with each datagram, carry the most important information about internal states of the TMC429 and the settings of the reference switches. These datagrams have the form:

MSB	32 bit DATAGRAM sent back from the TMC429 to µC via pin SDO_C																													LSB
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STATUS BITS								DATA BITS																							
INT	CDGW	SM3		SM2		SM1																									
		RS3	xEQt3	RS2	xEQt2	RS1	xEQt1																								

Table 7-4: 32 bit DATAGRAM structure received by µC (MSB received first)

The status bit **INT** is the internal high active interrupt controller status output signal. Handling of interrupt conditions without using interrupt techniques is possible by polling this status bit. The interrupt signal is also directly available at the SDO_C pin of the TMC429 if nSCS_C is high. The pin SDO_C may directly be connected to an interrupt input of the microcontroller. Since the SDO_C / nINT output is multiplexed, the microcontroller has to disable its interrupt input while it sends a datagram to the TMC429, because the SDO_C signal– driven by the TMC429 –alternates during datagram transmission. For initialization purposes, the TMC429 enables direct communication between the microcontroller and the stepper motor driver chain by sending a so called *cover datagram* (see sections 10.2 and 10.3). The position **cover_position** and actual length **cover_len** of a cover datagram is specified by writing them into a common register. Writing an up to 24 bit wide cover datagram to the register **cover_datagram** will fade in that cover datagram into the next datagram sent to the stepper motor driver chain. As a default setting, the TMC429 only sends datagrams on demand. Optionally, continuous update– periodic sending of datagrams to the stepper motor driver chain –is also possible. So, the status bit named **CDGW** (cover datagram waiting) is a handshake signal for the

microcontroller in regard to the datagram covering mechanism. This feature is necessary to enable direct data transmission from a microcontroller to the stepper motor driver chips for initialization purposes. The **CDGW** status bit also gives the status of the **datagram_high_word** and **datagram_low_word** (see section 10.1).

The status bits **RS3, RS2, RS1** represent the settings of the reference switches. But, the reference switch inputs REF3, REF2, REF1 are not mapped directly to these status bits. Rather, the reference switch inputs may have different functions, depending on programming (see pages 25 – 27). The three status bits **xEQt3, xEQt2, xEQt1** indicate individually for each stepper motor, if it has reached its target position. The status bits **RS3, RS2, RS1** and bits **xEQt3, xEQt2, xEQt1** can trigger an interrupt or enable simple polling techniques.

7.5 Simple Datagram Examples

The % prefix– normally indicating binary representation in this data sheet –is omitted for the following datagram examples. Assuming, one would like to write (RW=**0**) to a register (RRS=0) at the address %001101 the following data word %0000 0000 0000 0001 0010 0011, one would have to send the following 32 bit datagram

0001101**0**00000000000000000100100011

to the TMC429. With inactive interrupt (**INT**=0), no cover datagram waiting (**CDGW**=0), all reference switches inactive (RS3=0, RS2=0, RS1=0), and all stepper motors at target position (xEQt3=1, xEQt2=1, xEQt1=1) the status bits would be %10010101 the TMC429 would send back the 32 bit datagram:

10010101000000000000000000000000

To read (RW=**1**) back the register written before, one would have to send the 32 bit datagram

0001101**1**00000000000000000000000000

to the TMC429 and would get back from it the datagram

10010101000000000000000100100011.

Write (RW=**0**) access to on-chip RAM (RRS=1) to an address %111111 occurs similar to register access, but with RRS=1. To write two 6 bit data words %100001 and %100011 to successive pair-wise RAM addresses %1111110 and %1111111 (%100001 to %1111110 and %100011 to %1111111) which are commonly addressed by one datagram (see pages 17 and 40), one would have to send the datagram

1111111**0**00000000001000110010000 1.

To read (rw=**1**) from that on-chip memory address, one would have to send the datagram

1111111**1**00000000000000000000000.

8 Address Space Partitions

The address space is partitioned in different ranges. Each of the up to three stepper motors has a set of registers individually assigned to it, arranged within a contiguous address space. An additional set of registers within the address space holds some global parameters common for all stepper motors. One dedicated global parameter register is essential for the configuration of the serial four wire stepper motor driver interface. One half of the on-chip RAM address space holds the configuration parameters for the stepper motor driver chain. The other half of the on-chip RAM address space is provided to store a micro step table if required. The first seven datagram bits (*sdi_c_bit#31* and *sdi_c_bit#30* ... *sdi_c_bit#25*, respectively *RRS* and *ADDRESS*) address the whole address space of the TMC429.

address ranges (incl. RRS)			assignment	
%000 0000	. . .	%000 1111	16 registers for stepper motor #1	registers with up to 24 bits
%001 0000	. . .	%001 1111	16 registers for stepper motor #2	
%010 0000	. . .	%010 1111	16 registers for stepper motor #3	
%011 0000	. . .	%011 1110	15 common registers	
		%011 1111	1 global parameter register	
%100 0000	. . .	%101 1111	32 addresses of 2x6 bit for driver chain configuration	RAM 128x6 bit
%110 0000	. . .	%111 1111	32 addresses of 2x6 bit for micro step table	

Table 8-1: TMC429 address space partitions

The stepper motors are controlled directly by writing motion parameters into associated registers. Only one register write access is necessary to change a target motion parameter. E.g. to change the target position of one stepper motor, the microcontroller has to send only one 32 bit datagram to the TMC429. The same is true for changing a target velocity. Some parameters are packed together in a single data word at a single address. Those parameters– initialized once and unchanged during operation –have to be changed commonly. Access to on-chip RAM addresses concern two successive RAM addresses. So, always two data words are modified with each write access to the on-chip RAM. Once initialized after power-up, the content of the RAM is usually left unchanged.

8.1 Read and Write

Read and write access is selected by the RW bit (sdi_c_bit#24) of the datagram sent from the µC to the TMC429. The on-chip configuration RAM and the registers are writeable with read-back option. Some addresses are read-only. Write access (**RW**=0) to some of those read-only registers triggers additional functions, explained in detail later.

8.2 Register Set

The register address mapping is given in Table 8-2 on page 18. These registers are initialized internally during power-up. During power-up initialization, the TMC429 sends no datagrams to the stepper motor driver chain.

Note: The RAM has to be initialized before writing target parameters to the register set of the TMC428. The TMC429 loads a default RAM configuration for a TMC236 / TMC239 / TMC246 / TMC249 driver chain on power-on reset.

Note: There are unused addresses within the address space of the TMC428. Access to these addresses has no effect. How ever, access should be avoided, because this address space may be used for future devices. The TMC429 has some new registers (ustep_count_429, if_configuration_429, pos_comp_429, pos_comp_int_429, type_version_429, stpdiv_429 [if EN_SD='1']).

8.3 RAM Area

The RAM address mapping is given in Table 11-1 page 41. The on-chip RAM of the TMC429 is initialized internally during power-up. It can be modified done by the microcontroller.

Important Hint: All register bits are initialized with '0' during power on reset, except the SPI clock pre-divider **clk2_div** (see section 10.11, page 35) that is initialized with 15.

32 bit DATAGRAM sent from a µC to the TMC428 / TMC429 via pin SDI_C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRS	ADDRESS						RW	DATA																							
0	smda		IDX				RW=0 : WRITE access / RW=1 : READ access	three stepper motor register sets (SMDA={00, 01, 10})																									
0	0		0	0	0	0		x_target																							
			0	0	0	1		x_actual																							
			0	0	1	0										v_min														
			0	0	1	1										v_max														
			0	1	0	0										v_target														
			0	1	0	1										v_actual														
			0	1	1	0										a_max														
	1		0	1	1	1										a_actual														
			1	0	0	0		is_agtat			is_aleat					is_v0								a_threshold							
	0		1	0	0	1										1	pmul											pdiv			
			1	0	1	0									lp					ref_conf										rm	
			1	0	1	1										interrupt_mask								interrupt_flags							
			1	1	0	0										pulse_div				ramp_div										usrs	
			1	1	0	1										dx_ref_tolerance															
			1	1	1	0		x_latched																							
			1	1	1	1																		ustep_count_429							
0	1		JDX					common registers (SMDA=11)																							
	1		0	0	0	0		datagram_low_word																							
			0	0	0	1		datagram_high_word																							
			0	0	1	0		cw										cover_position										cover_len			
			0	0	1	1		cover_datagram																							
			0	1	0	0																		if_configuration_429							
			0	1	0	1		pos_comp_429																							
			0	1	1	0		pos_comp_int_429 :															M								I
			1	0	0	0		power-down																							
			1	0	0	1		type_version_429 (= $429101 for TMC429 version 1.01, read only)																							
			1	1	1	0																							I3	r3	I2	r2	I1	r1
	1		1	1	1	1				mot1r	refmux				cont_update	clk2_div					cs_ComInd	polarities					LSMD				
																0	0	0	0	stpdiv_429 (if EN_SD='1')		DAC_AB	FD_AB	PH_AB	SCK_S	nSCS_S					
RRS	ADDRESS						RW	DATA																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 8-2: TMC428 / TMC429 register mapping (additional TMC429 registers marked with _429)

9 Register Description

The registers hold binary coded numbers. Some are unsigned (positive) numbers, some are signed numbers in two's complement, and some are control bits or single flags. The functionality of different registers depends on the ramp mode (s. page 25).

9.1 x_target (IDX=%0000)

This register holds the current target position in units of full steps, respectively micro steps. The unit of the target position depends on the setting of the associated micro step resolution register **usrs**. If the difference **x_target – x_actual** is unequal zero, the TMC429 moves the stepper motor in that direction of **x_target** so that the difference becomes zero. The condition | **x_target – x_actual | < 2^{23}** must be satisfied for motion into the correct direction. Both, target position **x_target** and current position **x_actual** may be altered on the fly. Usually **x_target** is modified to start a positioning. To move from one position to another, the ramp generator of TMC429 automatically generates ramp profiles in consideration of the velocity limits **v_min** and **v_max** and acceleration limit **a_max**.

Note: The registers **x_target, x_actual, v_min, v_max, and a_max** are initialized with zero after power up. Thus, no step pulses are generated because motion is prohibited.

9.2 x_actual (IDX=%0001)

The current position of each stepper motor is available by read out of the registers called **x_actual**. The actual position can be overwritten by the microcontroller. This feature is for reference switch position calibration under control of the microcontroller.

9.3 v_min (IDX=%0010)

This register holds the absolute value of the velocity at or below which the stepper motor can be stopped abruptly. The parameter **v_min** is relevant only for deceleration while reaching a target position. It should be set greater than zero. This control value allows to reach the target position faster because the stepper motor is not slowed down below **v_min** before the target is reached. Also consider, that due to the finite numerical representation of integral relations, the target position can not be reached exactly, if the calculated velocity is less than one, before the target is reached. So, setting **v_min** to at least one assures reaching each target position exactly. The unit of velocity parameters (**v_max**, **v_target**, and **v_actual**) is steps per time unit. The scale of velocity parameters (**v_min**, **v_max**, **v_target**, **v_actual**) is defined by the parameter **pulse_div** (see page 29 for details) and depends on the clock frequency of the TMC429.

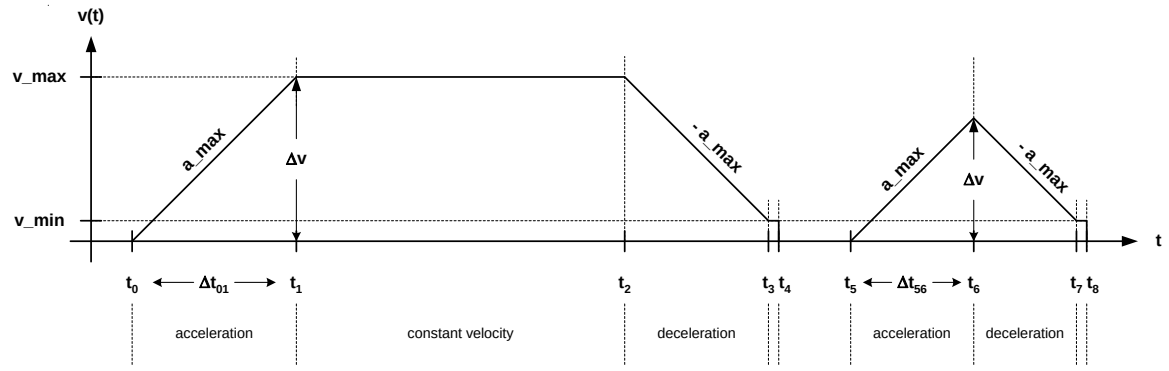

Figure 9-1: Velocity ramp parameters and velocity profiles

9.4 v_max (IDX=%0011)

This parameter sets the maximum motor velocity. The absolute value of the velocity will not exceed this limit, except if the limit **v_max** is changed during motion to a value below the current velocity.

Note: To set target position **x_target** and current position **x_actual** to an equivalent value (e.g. to set both to zero at a reference point), the assigned stepper motor should be stopped first, and the parameter **v_max** should be set to zero to hold the assigned stepper motor at rest before writing into the register **x_target** and **x_actual**.

9.5 v_target (IDX=%0100)

In modes RAMP_MODE and SOFT_MODE this register holds the current target velocity calculated internally by the ramp generator. In mode VELOCITY_MODE a target velocity can be written into this register. Then the associated stepper motor accelerates until it reaches the target velocity specified. In VELOCITY_MODE the velocity is changed according to the motion parameter limits if the register **v_target** is changed. In HOLD_MODE the register **v_target** is ignored.

9.6 v_actual (IDX=%0101)

This read-only register holds the current velocity of the associated stepper motor. Internally, the ramp generator of the TMC429 processes with 20 bits while only 12 bits can be read out as **v_actual**. So, an actual velocity of zero *read out* by the microcontroller means that the current velocity is in an interval between zero and one. Because of this, the actual velocity should not be used to detect a stop of a stepper motor. Writing zero to register **v_actual**, which is possible in HOLD_MODE only, immediately stops the associated stepper motor, because hidden bits are set to zero with each write access to the register **v_actual**. In HOLD_MODE only, this register is a read-write register. In HOLD_MODE, motion parameters are ignored and the microcontroller has the full control to generate a ramp. In that mode, the TMC429 only handles the microstepping and datagram generation for the associated stepper motor of the daisy chain.

9.7 a_max (IDX=%0110)

The absolute value of the maximum acceleration is defined by this register. It ranges from 0 to 2047. The unit of the acceleration is change of step frequency per time unit divided by 256. The scale of acceleration parameters (**a_max**, **a_actual**, **a_threshold**) is defined by the parameter **ramp_div** (see section 9.14, page 29 for details) and depends on the clock frequency of the TMC429. Setting **a_max** to zero during motion of the stepper motor results in the inability of the stepper motor to stop, because it cannot change its velocity.

9.7.1 a_max_lower_limit & a_max_upper_limit for ramp_div ≠ pulse_div

Under special conditions, the parameter **a_max** might have a lower limit (>1) and might an upper limit (<2047) concerning ***deceleration*** in **RAMP_MODE** and **SOFT_MODE** if the difference between **ramp_div** and **pulse_div** is more than one. This is because the deceleration ramp is internally limited to 2^19 steps respectively micro steps, which is sufficient for most applications. The lower limit concerning the deceleration is given by

a_max_lower_limit = 2^(**ramp_div** – **pulse_div** –1)

With **v_max** set to 2048 / √2 (≈ 1448) or lower, the **a_max_lower_limit** is half of this value. If **ramp_div** – **pulse_div** – 1 ≤ 0 the limit **a_max_lower_limit** is 1 and the parameter **a_max** may be set to down to 1 and of course to 0. On the other side, the upper limit of **a_max** is given by

a_max_upper_limit = 2^(**ramp_div** – **pulse_div + 12**) – 1

So, if **ramp_div** – **pulse_div +** 1 ≥ 0 the **a_max_upper_limit** is > 2048 and the parameter **a_max** might be set to any value up to 2047.

Important Note: So, **a_max** can be set without restrictions within its range of 0 to 2047 for those combinations of **ramp_div** and **pulse_div** with | **ramp_div – pulse_div | ≤ 1**.

The parameter **a_max** must not be set below **a_max_lower_limit** except **a_max** is set to 0. The condition **a_max** ≥ **a_max_lower_limit** as well as **a_max** ≤ **a_max_upper_limit** must be satisfied to reach any target position without oscillations. If that condition is not satisfied, oscillations around a target position may occur. For description of the parameters **ramp_div** and **pulse_div** see page 29.

So, **a_max_lower_limit** and **a_max_upper_limit** restrict the allowed range of **a_max** only for those cases where **ramp_div** is non-equal to **pulse_div** and differ more than one. These both limits of **a_max** concern the deceleration phase for **RAMP_MODE** and **SOFT_MODE** only. As long as **ramp_div ≥ pulse_div – 1** is valid, any value of **a_max** within its range (0,1, ..., 2047) is allowed and there exists a valid pair {**pmul, pdiv**} for each **a_max**. Qualitative verbalized, this is because the acceleration scaling determined by **ramp_div** is compatible with the step velocity scaling determined by **pulse_div**. In other words, large **ramp_div** stands for low acceleration where large **pulse_div** stands for low velocity and low acceleration is compatible with low speed and high speed as well, but high acceleration is more compatible with high speed.

Important Note: Changing at least one parameter out of the triple {**a_max**, **ramp_div**, **pulse_div**} requires re-calculation of the parameter pair {**pmul, pdiv**} to update the associated register. For description of the parameters **pmul** and **pdiv** see section 9.10, page 22.

9.8 a_actual (IDX=%0111)

The actual acceleration, which the TMC429 actually applies to a stepper motor, can be read out by the microcontroller from this read-only register for monitoring purposes. The actual acceleration is used to select scale factors for the coil currents. Internally, it is updated with each clock. The returned value **a_actual** is smoothed to avoid oscillations of the readout value. Thus, returned **a_actual** values should not be used directly for precise calculations.

9.9 is_agtat & is_aleat & is_v0 & a_threshold (IDX=%1000)

These parameters represent current scaling values I_s and are applied to the motor depending on the ramp phase: The parameter **is_agtat** is applied if the acceleration (a) is greater than (gt) a threshold acceleration (a_t). This is to increase current during acceleration phases. The parameter **is_aleat** is applied if the acceleration is lower than or equal to (le) the threshold acceleration. This is the nominal motor current. The third parameter **is_v0** is applied if the stepper motor is at rest, to save power, to keep it cool, and to avoid noise probably caused by chopper drivers. The parameter **a_threshold** is the threshold used to compare with the current acceleration to select the current scale factor. The three parameters **is_agtat, is_aleat**, and **is_v0** are bit vectors of three bit width. One of these is selected conditionally and assigned to an interim bit vector **i_scale**. The current scaling factor I_s is defined in Table 9-1.

i_scale			I_s		
0	0	0	1	= 100	%
0	0	1	1 / 8	= 12.5	%
0	1	0	2 / 8	= 25	%
0	1	1	3 / 8	= 37.5	%
1	0	0	4 / 8	= 50	%
1	0	1	5 / 8	= 62.5	%
1	1	0	6 / 8	= 75	%
1	1	1	7 / 8	= 87.5	%

Table 9-1: Coil current scale factors

Important Notes: The maximum current scaling factor 1 is selected by **i_scale** = %000. This is the power-on default. The minimum current scaling factor 1/8 = 0.125 is selected by **i_scale** = %001. The current scaling factor I_s proportionally reduces the effective number of micro steps per full step. For example, with **i_scale** = %100 (= 4/8 = 50%) the number of effective micro steps per full step is halved.

One of the three scale factors **is_agtat, is_aleat**, and **is_v0** is selected according to Table 9-2. If the velocity is zero, the parameter **is_v0** is used for scaling. If the velocity is not zero, either **is_aleat** or **is_agtat** is used for scaling, depending on the absolute value of the acceleration and the acceleration threshold **a_threshold**.

v = 0		I_s := is_v0
v ≠ 0	$a_{threshold > 1023}$	I_s := is_aleat
	$\lvert a \rvert \le a_{threshold}$	I_s := is_aleat
	$\lvert a \rvert > a_{threshold}$	I_s := is_agtat

Table 9-2: Current scale selection scheme

The automatic motion dependent current scale feature of the TMC429 is provided primarily for micro step operation. It may also be applied for full step or half step drivers, if those provide current control bits. For those drivers, one could initialize the micro step table with a constant function, square function or sine wave using the two most significant DAC bits.

The configuration bit **continuous_update** of the **stepper motor global parameter register** (Table 10-3, page 35) must be set to '1' to make sure that the coil current is scaled for v=0 if all motors are at rest.

The current is scaled for v=0 takes place delayed to avoid mechanical step lost due to oscillations of the motor after it has been stopped. The delay time is

t_isv0_delay[s] = 255 * (32 * 2^ramp_div) / fCLK[Hz].

Important Hints: Due to signed compare of **a_threshold** with **a_max**, a setting of **a_threshold** with a value greater than 1023 results in using **is_aleat** for current scaling if **a_max** is greater than 1023 during acceleration instead of using **is_agtat** for current scaling. For most applications, setting **is_aleat** and is_agtat to the same value and using a lower value for is_v0 is the best choice. For selection of current scaling is_v0 at rest, both parameters is_agtat and is_aleat must be larger than 0.

9.10 pmul & pdiv (IDX=%1001)

The stepper motors are driven with a trapezoidal velocity profile, which may become triangular if the maximum velocity is not reached (see Figure 9-1, page 19). Depending on the difference between the target position **x_target** and the actual position **x_actual**, the ramp generator continuously calculates target velocities **v_target** for the pulse generator (see Figure 9-2, page 23). The pulse generator then generates (micro) step pulses taking into account the motion parameter limits (**v_min**, **v_max**, **a_max**). With a target velocity proportional to the difference of target position **x_target** and current position **x_actual**, the stepper motor approaches the target position. This also works, if the target position is changed during motion. The stepper motor moves to a target position until the difference between the target position **x_target** and the current position **x_actual** vanishes.

With the right proportionality factor **p**, target positions are quickly reached and without overshooting them. The proportionality factor primarily depends on the acceleration limit **a_max** and on the two clock divider parameters **pulse_div** and **ramp_div**. These two separate clock divider parameters– set to the same value for most applications –give an extremely wide dynamic range for acceleration and velocity. These two *separate* parameters allow reaching very high velocities with very low acceleration.

If the proportionality factor **p** is set too small, this results in a slow approach to the target position. If set too large, it causes overshooting and even oscillations around the target position. The calculation of the proportionality factor is simple:

The representation of the proportionality factor **p** by the two parameters **p_mul** and **p_div** is some kind of a fixed point representation. It is

p = p_mul / p_div

with

p_mul = {128, 128+1, 128+2, 128+3, ..., 128+127}

and

p_div = {2^3, 2^4, 2^5, ..., 2^{14}, 2^{15}, 2^{16}}.

Instead of direct storage of the parameters **p_mul** and **p_div**, the TMC429 stores two parameters called **pmul** and **pdiv**, with

p_mul = 128 + pmul and **p_div = 2^{3+pdiv}= 2^(3+pdiv)**

where

pmul = {0, 1, 2, 3, ..., 127} and **pdiv = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}**.

The reason why **p_mul** ranges from 128 to 255 is, that **p** is divided by **p_div** which is a power of two ranging from 8 to 65536. So, values of **p** less than 128 can be achieved by increasing **p_div**.

Note: The parameters **pmul** and **pdiv** share a single address (IDX=%1001, see Table 8-2, page 18). The MSB of **p_mul** is fixed set to '1'. So, sending **pmul** internally sets **p_mul = 128 + pmul**. In other words, %10000000 = 128 is ORed as bit vector with the content of the register **pmul**.

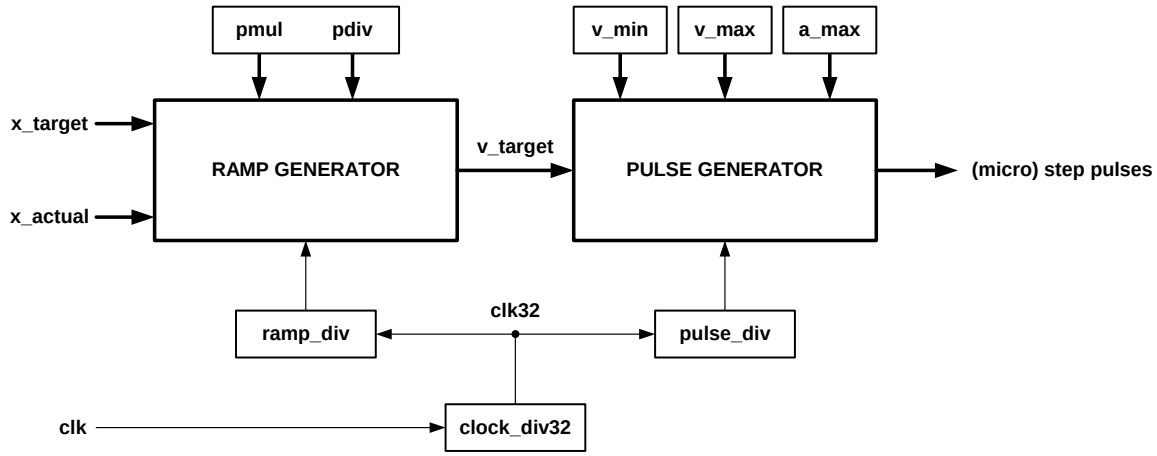

Figure 9-2: Ramp generator and pulse generator

The parameter **p** has to be calculated for a given acceleration. This calculation is not done by the TMC429 itself, because this task has to be done only once for a given acceleration limit. The acceleration limit is a stepper motor parameter, which is usually fixed in most applications. If the acceleration limit has to be changed nevertheless, the microcontroller could calculate on demand a pair of **p_mul** and **p_div** for each acceleration limit **a_max** and given **ramp_div** and **pulse_div**. Also, pre-calculated pairs of **p_mul** and **p_div** read from a table maybe sufficient.

9.11 Calculation of p_mul and p_div

The proportionality factor **p = p_mul / p_div** depends on the acceleration limit **a_max** and frequency pre-divider parameters **ramp_div** and **pulse_div**. So, a pair of **p_mul / p_div** has to be calculated once for each provided acceleration limit **a_max**. There may exist more than one valid pair of **p_mul** and **p_div** for a given **a_max**. To accelerate, the ramp generator accumulates the acceleration value to the actual velocity with each time step. Internally, the absolute value of the velocity is represented by 11+8 = 19 bits, while only the most significant 11 bits and the sign are used as input for the step pulse generator. So, there are 2^{11} = 2048 values possible to specify a velocity, ranging from 0 to 2047. The ramp generator accumulates **a_max** divided by 2^8 = 256 at each time step to the velocity during acceleration phases. So, the acceleration from velocity = 0 to maximum velocity = 2047 spans over 2048* 256 / **a_max** pulse generator clock pulses. Within that acceleration phase, the pulse generator generates S = ½ * 2048* 256 / **a_max** * T steps for the (micro) step unit. The parameter T is the clock divider ratio $T = 2^{ramp\_div} / 2^{pulse\_div} = 2^{ramp\_div - pulse\_div}$ = 2^(ramp_div-pulse_div). During acceleration, the velocity has to be increased until the velocity limit **v_max** is reached or deceleration is required to reach the target position exactly (see Figure 9-1). The TMC429 automatically decelerates, if required using the difference between current position and target position and the proportionality parameter **p**, which has to be **p** = 2048 / S. With this, one gets p = 2048 / ((½ * 2048* 256 / **a_max**) * 2^(ramp_div-pulse_div)). This expression can be simplified to

p = a_max / (128 * 2^(ramp_div-pulse_div)).

To avoid overshooting, the parameter **p_mul** should be made approximately 5% smaller than calculated. Alternatively, one can arrange **p** reduced by an amount of 5%. If the proportionality parameter **p** is too small, the target position will be reached slower, because the slow down ramp starts earlier. The target position is approached with minimal velocity **v_min**, whenever the internally calculated target velocity becomes less than **v_min**. With a good parameter **p** the minimal velocity **v_min** is reached a couple of steps before the target position. With parameter **p** set a little bit to large and small **v_min** overshooting of one step respectively one micro step may occur. Decrementation of the parameter **pmul** avoids such one-step overshooting.

Note: Changing at least one parameter out of the triple {**a_max**, **ramp_div**, **pulse_div**} requires re-calculation of the parameter pair {**pmul, pdiv**} to update the associated register if necessary.

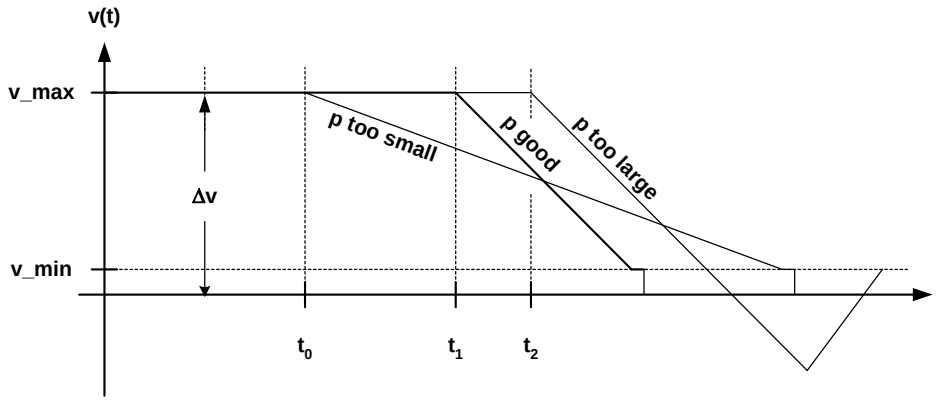

Figure 9-3: Proportionality parameter p and outline of velocity profile(s)

On first approach, to represent the parameter **p = p_mul / p_div = (128+pmul) / 2^(3+pdiv) one** chooses a pair of **pmul** and **pdiv** that approximates **p**, with **pmul** in range **0 ... 127** representing **p_mul** in range **128 ... 255** and **pdiv** one out of **{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}** representing **p_div** one out of **{8, 16 , 32 ,64, 128 , 256, 512, 1024, 2048, 4096, 8192, 16384, 32786, 65536}**. There are only 128 * 14 = 1792 pairs of **(pmul, pdiv)**. So, one can simply try all possible pairs **(pmul, pdiv)** with a program and choose a matching pair. To find a pair, one calculates

p = a_max / (128 * 2^(ramp_div-pulse_div))

and

p' = p_mul / p_div = (128+pmul) / 2^(3+pdiv)

and

q = p' / p

for each pair **(pmul , pdiv)** and select one of the pairs satisfying the condition **0.95 < q < 1.0**. So, the value **q** interpreted as a function **q(a_max, ramp_div, pulse_div, pmul, pdiv)** gives the quality criterion required. Although **q = 1.0** indicates that **(pmul , pdiv)** perfectly represents the desired **p** for a given **a_max**, this could cause overshooting because of finite numerical precision. In case of high resolution microstepping, overshooting of one micro step is negligible in most applications. To avoid overshooting, use **pmul-1** instead of the selected **pmul** or select a pair **(pmul, pdiv)** with **q = 0.95**. The first given source code example 'pmulpdiv.c' showing programming in C language based on this brute force approach. Some conversions of the base present equations help to reduce the calculation effort drastically.

9.11.1 Optimized Calculation of p_mul and p_div

With the equations above, one can simplify the calculation of the parameters **pmul** and **pdiv** using the expression

pmul = p * 2^3 * 2^pdiv – 128

with

p = a_max / (128 * 2^(ramp_div – pulse_div)).

To avoid overshooting, use

p_reduced = p * (1 – p_reduction[%])

with **p_reduction** approximately 5% instead of unreduced **p**. With this, one gets

pmul = p_reduced * 2^3 * 2^pdiv – 128 = 0.95 * p * 2^3 * 2^pdiv – 128.

With this, **pmul** becomes a function of the parameter **pdiv**. To find a valid pair {**pmul**, **pdiv**} one just has to choose that pair {**pmul**, **pdiv**} out of 14 pairs for **pdiv** = {0, 1, 2, 3, ..., 13} with **pmul** within the valid range 0 ≤ **pmul** ≤ 127. An example 'pmulpdiv.c' showing programming in C language can be found on page 60. This source code can directly be copied from the PDF datasheet file.

9.12 lp & ref_conf & ramp_mode (rm) (IDX=%1010)

The bit called **lp** (latched position) is a read only status bit. The configuration words **ref_conf** and **ramp_mode** are accessed via a common address, because these parameters normally are initialized only once. The configuration bits **ref_conf** select the behavior of the reference switches, while the two bits **ramp_mode (rm)** select one of the four possible stepping modes.

ramp_mode	mode	Function
%00	**RAMP_MODE**	default mode for positioning applications with trapezoidal ramp
%01	**SOFT_MODE**	similar to RAMP_MODE, but with soft target position approaching
%10	**VELOCITY_MODE**	mode for velocity control applications, change of velocities with linear ramps
%11	**HOLD_MODE**	velocity is controlled by the microcontroller, motion parameter limits are ignored

Table 9-3 – Outline of TMC429 motion modes

The mode called **RAMP_MODE** is provided as the default mode for positioning tasks, while the **VELOCITY_MODE** is for applications, where stepper motors have to be driven precisely with constant velocity. The **SOFT_MODE** is similar to the standard **RAMP_MODE** except that the target position is approached exponentially reduced velocity. This feature can be useful for applications where vibrations at the target position have to be minimized. The **HOLD_MODE** is provided for motion control applications, where the ramp generation is completely controlled by the microcontroller.
The TMC429 has three reference switch inputs REF1, REF2, REF3. Without additional hardware, three reference switches are available. These switches can be used as reference switches and can be used as automatic stop switches as well. Per default, one reference switch input is assigned individually to each stepper motor as a left reference switch (see Figure 10-4, page 37). The reference switch input REF3 can alternatively be assigned as the right reference switch of stepper motor number one (see Figure 10-5, page 37). In that configuration *a left and a right reference switch* is assigned to stepper motor one, a left reference switch is assigned to stepper motor two, and no reference switch is assigned to stepper motor three. The bit named **mot1r** in the stepper motor global parameter register (rrs=1 & address=%111111) selects one of these configurations. With additional hardware, up to six reference switches– a left and a right one assigned to each stepper motor –are supported. The additional hardware is just a 74HC157, where three of four 2-to-1-multiplexers are used (see Figure 10-7, page 38). The feature of multiplexing is controlled by the bit named **refmux** in the stepper motor global parameter register (rrs=1 & address=111111).
A reference switch can be used as an automatic stop switch. The reference switch indicates the reference position within a given tolerance. The automatic stop function of the switches can be enabled or disabled. Also a reference tolerance range (see register **dx_ref_tolerance**, page 30) can be programmed, to allow motion within the reference switch active range during reference point search. When a reference switch is triggered, the actual position can be stored automatically. This allows precise determination of the reference point. This is initiated by writing a dummy value to register **x_latched** (see page 30). The read-only status bit **lp** (latch position waiting) then indicates that the next change of the selected reference switch will trigger latching the position **x_actual**. The **lp** bit is automatically reset after position latching.

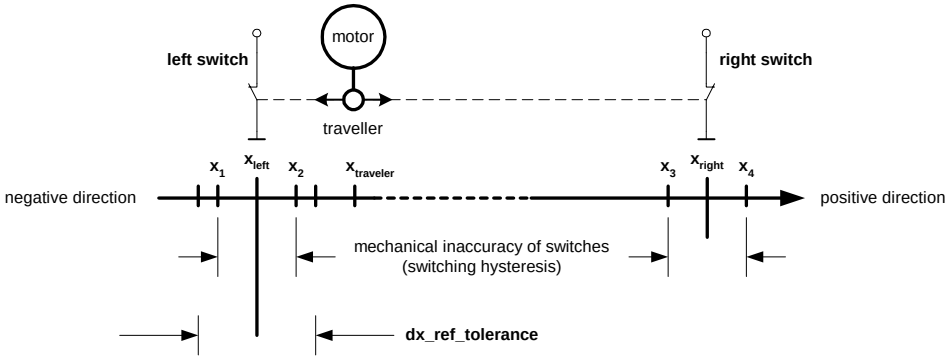

Figure 9-4: Left switch and right switch for reference search and automatic stop function

ref_conf mnemonic	Function
DISABLE_STOP_L	0 : Stops a motor if velocity is negative (**v_actual** < 0) and the left reference switch becomes active. 1 : Left reference switch is disabled as a an automatic stop switch.
DISABLE_STOP_R	0 : Stops a motor if velocity is positive (**v_actual** > 0) and the right reference switch becomes active. 1 : Right reference switch is disabled as an automatic stop switch.
SOFT_STOP	0 : Stopping takes place immediately, motion parameter limits are ignored. 1 : Stopping takes place in consideration of motion parameter limits, stops with linear ramp.
REF_RnL	0 : The left reference switch controls reference switch functions. 1 : The right (not left) reference switch controls reference switch functions.
lp	0 : This is the power-on default of the lp (latched position waiting) bit. 1 : **x_latched** has been initialized by a write access to latch the position on a change of the reference switch. It is set to '0' after a position is latched.

Table 9-4: Reference switch configuration bits (ref_conf)

Note: Definition of the reference switch by the configuration bit **REF_RnL** has no effect on the stop function of the reference switches if **DISABLE_STOP_L='0'** respectively **DISABLE_STOP_R='0'**. The bit **REF_RnL** (reference switch Right not Left) defines which switch is the reference switch: If set to '1', the right, else (set to '0') the left one is the reference switch.

The bits contained in **ref_conf** control the semantic and the actions of the reference/stop switch modes for interrupt generation as explained later. The stepper motor stops if the reference/stop switch, which corresponds to the actual driving direction, becomes active. The configuration bits named **DISABLE_STOP_L** respectively **DISABLE_STOP_R** disable these automatic stop functions. If the bit **SOFT_STOP** is set, motor stop forced by a reference switch is done within motion parameter limits while otherwise stopping is abruptly.

Hint: There is a functional difference between reference switches and stop switches. Reference switches are used to determine a reference position for a stepper motor. Stop switches are used for automatic stopping a motor when reaching a limit. The signals of switches are processed via the inputs named **REF1**, **REF2**, **REF3** might be used as automatic stop switches, reference switches, or both.

32 bit DATAGRAM sent from a µC to the TMC429																															
31	30	29	28	27	26	25	**24**	23	22	21	20	19	18	17	16	15	14	13	***12***	**11**	**10**	**9**	**8**	7	6	5	4	3	2	**1**	**0**
RRS	**ADDRESS**						**RW**	**DATA**																							
0	**smda**		**1**	**0**	**1**	**0**									***lp***					**ref_conf**										**rm**	
0															***latched position (waiting)***					**REF_RnL**	**SOFT_STOP**	**DISABLE_STOP_R**	**DISABLE_STOP_L**							**%00 : RAMP, %01 : SOFT, %10 : VELOCITY, %11 : HOLD**	

Table 9-5: lp & ref_conf & ramp_mode (rm) data bit positions

9.13 interrupt_mask & interrupt_flags (IDX=%1011)

The TMC429 provides one interrupt register of eight flags for each stepper motor. Interrupt bits are named **INT_<mnemonic>**. An interrupt bit can set back to '0' by writing '1' to it. Each interrupt bit can either be enabled ('1') or disabled ('0') individually by an associated interrupt mask bit named **MASK_<mnemonic>**. The interrupt flags are forced to '0' if the corresponding mask bit is disabled ('0'). The bit mapping of the interrupt mask bits and interrupt bits itself is diagrammed in Table 9-7 on page 28. The interrupt out **SDO_C / nINT** is set active low – where the interrupt status bit INT is set active high – when at least one interrupt flag of one motor becomes set. The interrupt mask enables or disables each interrupt mask individually. So, if the interrupt status is inactive, **nINT** is high ('1') and

INT is low (‘0’). The interrupt status is mapped to the most significant bit (31) of each datagram sent back to the µC (see Table 7-4, page 15) and it is only available at the **SDO_C** / **nINT** pin of the TMC429 if the pin **nSCS_C** is high.

Demultiplexing of the multiplexed interrupt status signal at the pin **SDO_C / nINT** can be done using additional hardware. It is not necessary if the microcontroller always disables its interrupt while it sends a datagram to the TMC429.

interrupt bit mnemonic	Function
INT_POS_END	stepper motor reached target position
INT_REF_WRONG	reference switch signal was active outside the reference switch tolerance range (**dx_ref_tolerance**)
INT_REF_MISS	reference switch signal missing at null position
INT_STOP	stop forced by reference switch during motion
INT_STOP_LEFT_LOW	high to low transition of left reference switch
INT_STOP_RIGHT_LOW	high to low transition of right reference switch
INT_STOP_LEFT_HIGH	low to high transition of left reference switch
INT_STOP_RIGHT_HIGH	low to high transition of right reference switch

Table 9-6: interrupt bit mnemonics

32 bit DATAGRAM sent from a µC to the TMC429																															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RRS	**ADDRESS**						**RW**	**DATA**																								
0	**smda**		**1**	**0**	**1**	**1**										**interrupt mask**								**interrupt flags**							
																MASK_STOP_RIGHT_HIGH	**MASK_STOP_LEFT_HIGH**	**MASK_STOP_RIGHT_LOW**	**MASK_STOP_LEFT_LOW**	**MASK_STOP**	**MASK_REF_MISS**	**MASK_REF_WRONG**	**MASK_POS_END**	**INT_STOP_RIGHT_HIGH**	**INT_STOP_LEFT_HIGH**	**INT_STOP_RIGHT_LOW**	**INT_STOP_LEFT_LOW**	**INT_STOP**	**INT_REF_MISS**	**INT_REF_WRONG**	**INT_POS_END**

Table 9-7: interrupt register & interrupt mask

An interrupt flag is set to ‘1’ if its assigned interrupt condition occurs and the corresponding interrupt mask is set (‘1’). Interrupt flags are reset to ‘0’ by a write access (RW=’0’) to the interrupt register address (IDX=%1011) with a ‘1’ at the position of the bit to be cleared. Writing a ‘0’ to the corresponding position leaves the interrupt flag untouched.

If an end position is reached while the interrupt mask **MASK_POS_END** is ‘1’, the bit named **INT_POS_END** is set to one. The switches processed via the inputs REF1, REF2, REF3 can be used as stop switches for automatic motion limiting, as reference switches and for both. If a reference switch becomes active out of the reference switch tolerance range– defined by the **dx_ref_tolerance** register –the interrupt flag **INT_REF_WRONG** is set if its interrupt mask bit **MASK_REF_WRONG** is set. The interrupt flag **INT_REF_MISS** is set if the reference switch is inactive at the 0 position and the mask **MASK_REF_MISS** is enabled. The **INT_STOP** flag is set, if the reference switch has forced a stop and if the interrupt mask **MASK_STOP** is set. The **INT_STOP_LEFT_LOW** flag is set if the reference

switch changes from high to low and if the interrupt mask bit **MASK_STOP_LEFT_LOW** is set. The interrupt flag **INT_STOP_RIGHT_LOW** is similar to **INT_STOP_LEFT_LOW** but for the right reference switch. The **INT_STOP_LEFT_HIGH** indicates that the left reference switch input changes from low to high if the mask **MASK_STOP_LEFT_HIGH** is set. The **INT_STOP_RIGHT_HIGH** indicates it for the right reference switch if the mask **MASK_STOP_LEFT_HIGH** is set.

9.14 pulse_div & ramp_div & usrs (IDX=%1100)

The frequency of the external clock signal (see Figure 5-1, page 8, pin **CLK**) is divided by 32 (see Figure 9-2, page 23, block **clk_div32**). This clock drives two programmable clock dividers for the ramp generator and for the pulse generator. The pulse generator clock– defining the maximum step pulse rate –is determined by the parameter **pulse_div**. The pulse rate **R** is given by

R[Hz] = f_clk[Hz] * velocity / (2^pulse_div * 2048 * 32)

where **f_clk[Hz]** is the frequency of the external clock signal. The parameter **velocity** is in range 0 to 2047 and represents parameters **v_min**, **v_max** and absolute values of **v_target** and **v_actual**. So, the pulse generator of the TMC429 would generate one step pulse with each pulse generator clock pulse if the velocity could be set to 2048. The full step frequency $\mathbf{R_{FS}[Hz] = R[Hz] / 2\text{\^{}}usrs}$. The change $\Delta\mathbf{R}$ in the pulse rate per time unit (pulse frequency change per second – the acceleration) is given by

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

The constant 29 within the exponent is because 2^29 = 2^5 * 2^5 * 2^8 * 2^11 = 32*32*256*2048, where 32 comes from fixed clock pre-dividers, 256 comes from velocity accumulation clock pre-divider, and 2048 comes from velocity accumulation clock divider programmed by **a_max**. The parameter **a_max** is in range 0 to 2047. So, the parameter **ramp_div** scales the acceleration parameter **a_max**, where the parameter **pulse_div** scales the velocity parameters. $\Delta\mathbf{R_{FS}[Hz] = \Delta R[Hz] / 2\text{\^{}}usrs}$.

usrs			[microsteps / full step]	significant DAC bits (controlling current amplitude)	comment
0	0	0	1	-	full step (constant current amplitude)
0	0	1	2	5 (MSB)	half step
0	1	0	4	5 (MSB), 4	microstepping
0	1	1	8	5 (MSB), 4, 3	
1	0	0	16	5 (MSB), 4, 3, 2	
1	0	1	32	5 (MSB), 4, 3, 2, 1	
1	1	0	64	5 (MSB), 4, 3, 2, 1, 0 (LSB)	
1	1	1			

Table 9-8: micro step resolution selection (usrs) parameter

The angular velocity of a stepper motor can be calculated based on the full step frequency $\mathbf{R_{fs}[Hz]}$ for a given number of full steps per rotation. Similar, the angular acceleration of a stepper motor can be calculated based on the change of full step frequency per second $\mathbf{\Delta R_{FS}[Hz].}$ The three bit wide parameter **usrs** (micro (**µ**) **s**tep **r**esolution **s**election, where **u** represents **µ**) determines the micro step resolution for its associated stepper motor according to Table 9-8. There is an individual set of 6 DAC bits provided for each of the two phases (coils) for current control to provide up to 64 micro steps per full step. Depending on the micro step resolution, a subset of 6 DAC bits is significant. Using full stepping, the current amplitude is constant for both phases (coils) of a stepper motor and the polarity of one phase (coil) changes with each full step. The micro step counters are initialized to 0 during power-on reset. With each micro step an associated counter accumulates the programmed micro step resolution value **usrs**.
Generally, the number of steps **S** during linear acceleration **a** to a velocity **v** is given by **S = ½ * v^2 / a**. With **v = R[Hz]** and **a = ΔR[Hz/s]** one gets **S = ½ * velocity^2 / a_max * 2^ramp_div / 2^pulse_div / 2^3.** The number of full steps $\mathbf{S_{FS}}$ is $\mathbf{S_{FS} = S / 2\text{\^{}}usrs}$.

Important Hint: Changing **pulse_div** in VELOCITY_MODE or in HOLD_MODE might force an internal micro step (with micro step resolution defined by **usrs**) depending on the actual micro step position. This is the same for the TMC429. This behavior can be observed especially when the motor is at rest in VELOCITY_MODE or in HOLD_MODE. In RAMP_MODE this does not occur. So, **pulse_div** should only be changed in RAMP_MODE.

9.15 dx_ref_tolerance (IDX=%1101)

The switches processed via the inputs REF1, REF2, REF3 can be used as stop switches for automatic motion limiting and as reference switches defining a reference position for the stepper motor. To allow the motor to drive near the reference point, it is possible to exclude a motion range of steps from the stop switch function. The parameter **dx_ref_tolerance** disables automatic stopping by a switch around the origin (see Figure 9-4, page 26). To use the **dx_ref_tolerance** fare from the origin, the actual position has to be suitable adapted, e.g. to use it for a left side reference switch. Additionally, the parameter **dx_ref_tolerance** affects interrupt conditions as described before (section 9.13, page 27).

9.16 x_latched (IDX=%1110)

This read-only register stores the actual position read from the register **x_actual** if the reference switch state changes. The reference switch is defined by the bit **REF_RnL** of the configuration register **lp & ref_conf & ramp_mode**. Writing a dummy value to the (read-only) register **x_latched** initializes the position storage mechanism. Then the actual position is saved with the next rising edge or falling edge signal of the reference switch depending on the actual motion direction of the stepper motor. The actual position is latched when the switch defined as the reference switch by the **REF_RnL** bit changes (see Table 9-4: Reference switch configuration bits (ref_conf), page 26). The status bit **lp** signals, if latching of a position is pending. An event at the reference switch associated to the actual motion direction takes effect only during motion (when **v_actual** $\neq$ 0).

9.17 Unused Address (IDX=%1111) – TMC428 vs. TMC429

This register address (IDX=%1111) within each stepper motor register block {smda=%00, %01, %10} is unused for the TMC428. Writing to this register has no effect for the TMC428. For the TMC428 this register was mentioned to left unused, because this address space will be used for future devices.

Reading this register gives back the actual status bits and 24 data bits set to ‘0’ for the TMC428.

9.17.1 ustep_count_429 (IDX=%1111)

For the TMC429, this read write register named ustep_count_429 holds the actual micro step pointer. This register is intended for applications where the motion controller part is completely switched off for power saving modes where the motor needs to be at the same position as during power off after re-switching the power on.

10 Global Parameter Registers

The registers addressed by **RRS=0** with **SMDA=%11** are global parameter registers. To emphasize this difference, the **JDX** is used as index name instead of **IDX**.

10.1 datagram_low_word (JDX=%0000) & datagram_high_word (JDX=%0001)

The TMC429 stores datagrams sent back from the stepper motor driver chain with a total length of up to 48 bits. The register **datagram_low_word** holds the lower 24 bits of this 48 bits and the register **datagram_high_word** holds the higher 24 bits of the 48 bits. These registers together form a 48 bit shift register, where the data from pin **SDI_S** are shifted left into it with each datagram bit sent to the stepper motor driver chain via the signal **SDO_S**. A write to one of these read-only registers initializes them, to update their contents with the next datagram received from the drivers chain.

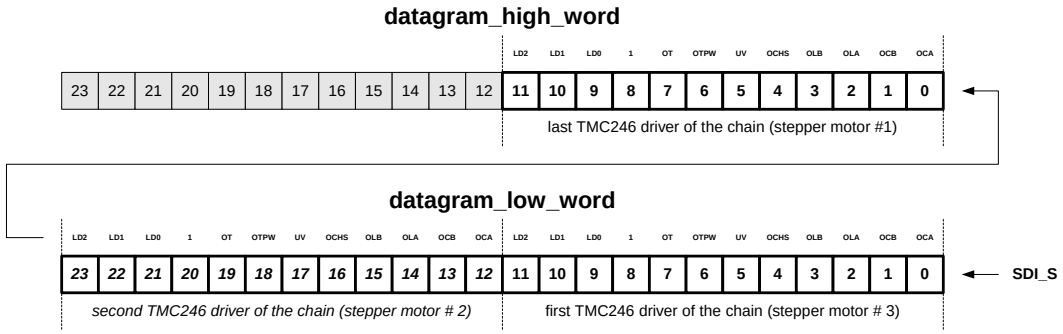

Figure 10-1: Example of status bit mapping for a chain of three TMC246 or TMC249

The **CDGW** (= **C**over **D**ata**G**ram **W**aiting, see section 10.3 on page 32) status bit is set to 1 until a datagram is received from the stepper motor driver chain. For read out of **datagram_low_word** and **datagram_high_word** the **CDGW** status bit is important to be able to detect when a datagram transfer has been completed after an initial write to one of the two registers. The fact that the **CDGW** is formed by a logical OR between the cover datagram status and the status of the **datagram_low_word** and **datagram_high_word** causes no restriction concerning its usage. This is because a write to the **cover_datagram** register forces sending a datagram which results in an update of the **datagram_low_word** and **datagram_high_word** registers. One the other side, if the **cover_datagram** mechanism is not used, the CDGW status bit is exclusively available as the status signal of **datagram_low_word** and **datagram_high_word**.

10.2 cover_pos & cover_len (JDX=%0010)

The TMC429 provides direct sending of datagrams from the microcontroller to the stepper motor drivers. This may be necessary for initialization of different driver chips and useful for reconfiguration purposes. A datagram with up to 24 bits can be transferred to the stepper motor driver by covering one datagram sent to the driver chain. The parameter **cover_pos** defines the position of the first datagram bit to be covered by the **cover_datagram (JDX=%0011)** of length **cover_len**. In contrast to the datagram numbering order of bits, the position count for the cover datagram starts with 0. The **cover_datagram** bits indexed from cover_len-1 to 0 cover the datagram sent to the drivers chain.

Important Note: *A step bit used to control stepper motor drivers must not be covered while a motor is running.*

This is because the coverage of a step bit would cause losing that associated step if the step bit is active. The TMC429 stores **cover_pos+1** instead of **cover_pos** due to internal requirements. So, one

writes **cover_pos** but reads back **cover_pos+1**. The **cw** (= cover waiting) bit available by read out of this register. The **CDGW** status bit (see section 10.3) is the result of logical OR between **cw** and an internal signal that indicates the status of the stepper motor serial driver chain send register.

10.3 cover_datagram (JDX=%0011)

This register holds up to 24 bit of a cover datagram. A cover datagram covers the next datagram sent to the stepper motor driver chain. If no datagrams are sent to the drivers chain, the cover datagram is sent immediately if a cover datagram is written into this register. The status of the cover datagram is mapped to the status bits sent back with each datagram (see Table 7-4, page 15, **CDGW** status bit). This status bit is also available for readout of cover_pos & cover_len (JDX=%0010), where **CDGW** is the most significant data bit (23).

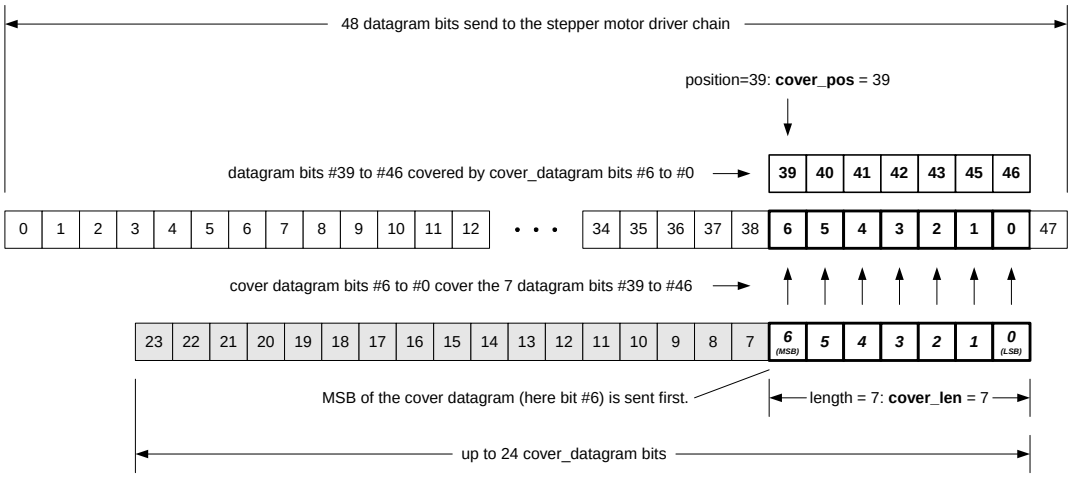

Figure 10-2: Cover datagram example

The **CDGW** (= Cover DataGram Waiting) status bit has to be checked before writing a new cover datagram into this register, to be sure that no cover datagram is waiting to be processed. The **CDGW** bit is set to 1 until the **cover_datagram** is sent. The **CDGW** status bit is also be used as a status bit for the **datagram_low_word** and **datagram_high_word** (see section 10.1 on page 31). An example for the cover datagram is given in Figure 10-2 on page 32. In that example 7 bits cover 7 bits of a 48 bit datagram from bit number 39 to bit number 46. A cover datagram with length=0 forces sending an unchanged datagram to the driver chain.

10.4 Unused Addresses (JDX={%0100, %0101, %0110, %1001}) – TMC428/429

There are unused addresses within the address range of the global parameter registers. Access to these addresses has no effect for the TMC428. However, access should be avoided, because this address space may be used for future devices. For the TMC429, the global registers ad addresses JDX={%0100, %0101, %0110, %1001}) are used for TMC429 specific function. Some bits of the clk2_div (JDX=%1111}) are used for the timing configuration in step/direction mode of the TMC429.

10.4.1 TMC429 Specific Registers

For the TMC429 some of the un-used addresses are used for additional registers of the TMC429. Additional registers of the TMC429 are named as **<name>_429** to indicate that these have TMC429 specific functions that are not available for the TMC428.

One can readout the version of the TMC429 by read access to a register called type_and_version_429. For the TMC428, a read on this register returns 0x000000 as type and version.

10.5 if_configuration_429 (JDX=%0100)

The register if_configuration_429 is the interface configuration register for the TMC429. It is for configuration of the additional reference inputs, the demultiplexed interrupt output, the step / direction interface and for association of the position compare output signal to one stepper motor.

if_configuration_429	Function
INV_REF	invert polarity of reference switches (common polarity for all reference switches
SDO_INT	map internal non-multiplexed interrupt status to nINT_SDO_C (needs SDOZ_C as SDO_C for read back information fomr the TMC429 to the micro controller); with SDO_INT='1' the nINT_SDO_C is a non-multiplexed nINT output to the micro controller
STEP_HALF	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 puls bandwith and is use full if one used e.g. low-bandwidth opto couples;
INV_STP	invert step pulse polarity; this is for adaption of the step polarity to external diver stages
INV_DIR	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, ...)
EN_SD	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 ist selected by ED_SD='1';
POS_COMP_SEL_0	select one motor out of three motors (%00, %01, %10) for the position compare function output
POS_COMP_SEL_1	of the TMC429 named **poscmp**
EN_REFR	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 an this might cause trouble it these inout are not-connected (for the SSOP16 these REFRx can not be connected;

Table 10-1: TMC429 interface configuration register control bits

32 bit DATAGRAM sent from a µC to the TMC429																															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RRS	**ADDRESS**						**RW**	**DATA**																							
	smda		**0**	**1**	**0**	**0**																	**if_configuration_429**								
0																							**EN_REFR**	**POS_COMP_SEL_1**	**POS_COMP_SEL_0**	**EN_SD**	**INV_DIR**	**INV_STP**	**STEP_HALF**	**SDO_INT**	**INV_REF**

Table 10-2: TMC429 interface configuration register

Hint: After PowerOn in SPI => '1' at SPI outputs (that are wired as STEP and DIRECTION)

Hint: Step-Dir-Mode429 : open inputs REFR1, REFR2, REFR3 have active state due to their internal pull-up resistors if REFMUX=0. If EN_REFR='0' thes additional reference switch inputs are ignored.

Hint: REFMUX does NOT work in Step-Direction-Mode because the SPI signal nSCS_S is not available because that output is signal S2 (step2) in step direction mode

Hint: Do not enable unused REFR1, REFR2, REFR3 inputs of the TMC429 as STOP switches if these inputs are open, because they have internal pull-up resistors.

10.5.1 TMC429 Step/Direction Timning

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 controllen 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[in unit MHz] of the TMC429, the length tSTEP [in unit µs] of a step pulse is

tSTEP[µs] = 16 * (1 + stpdiv_429) / fCLK[MHz].

So, for a clock frequency fCLK[MHz] of 16MHz the step pulse length can be programmed by stpdiv_429 in integrer 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.

General, the maximum step pulse frequency 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.

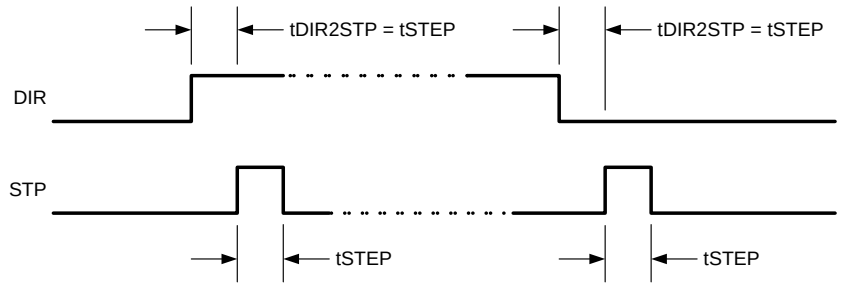

Figure 10-3: TMC429 Step Direction Timing (EN_SD='1' & STEP_HALF='0')

10.6 pos_comp_429 (JDX=%0101)

Position compare register of the TMC429 that gives a pulse on **POSCMP** output if the actual position **x_actual** of the motor selected by **pos_comp_sel** is equal to the compate position stored in register **pos_comp_429**.

10.7 pos_comp_int_429 (JDX=%0110)

The position compare interrupt mask (M) and interrupt flag (I) register hold the mask and interrupt concerning the pos_comp function of the TMC429. In principle, these bits could have been added to the existing interrupt mask and interrupt flags in the interrupt register address of the TMC428 but these TMC429 specific interrupt bits are in placed in separate register address.

10.8 power_down (JDX=%1000)

A write to the register address named **power_down** sets the TMC429 into the power down mode until it detects a falling edge at the pin **nSCS_C**. During power down, all internal clocks are stopped, all outputs remain stable, and all register contents are preserved.

10.9 type_and_version_429 (JDX=%1001)

Read only register that gives type und version of the design. For the TMC428, a read access on this register address gives back zero. For the TMC429 version 1.01 a read gives 0x429101 back.

10.10 Reference Switches l3 & r3 & l2 & r2 & l2 & r1 (JDX=%1110)

The current state of all reference switches– demultiplexed internally by the TMC429 if left and right reference switches are used –can be read from this read-only register. The bit named **continuous_update** of the **Stepper Motor Global Parameter Register (JDX=%1111)** is important for reading out of reference switches as explained below.

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

This register holds different configuration bits for the stepper motor driver chain. The absolute address (RRS & ADDRESS) of the stepper motor global parameter register is %01111110 = $7E (see Table 8-2, page 18, and Table 10-3, page 35). For the datagram configuration the number of stepper motor drivers is important. It is represented by the parameter **LSMD** (**L**ast **S**tepper **M**otor **D**river). The parameter **LSMD** has to be set to %00 for one stepper motor driver, %01 for two stepper motor drivers, and %10 for three stepper motor drivers (see Table 10-4).

32 bit DATAGRAM sent from a µC to the TMC429																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRS	ADDRESS						RW	DATA																							
0	1	1	1	1	1	1										clk2_div								polarities					LSMD	
0										mot1r	refmux				continuous_update								csCommonIndividual	polarity_DAC_AB	polarity_FD	polarity_PH_AB	polarity_sck_s	polarity_nscs_s	last stepper motor driver	

Table 10-3: Stepper motor global parameter register

LSMD	number of stepper motor drivers
%00 (=0)	1
%01 (=1)	2
%10 (=2)	3
%11 (=3)	NOT ALLOWED

Table 10-4: Global parameter LSMD (last stepper motor driver)

Five bits are used to control signal polarities. The polarity of the selection signal **nSCS_S** for the stepper motor driver chain is controlled by the polarity bit **polarity_nscs_s.** The **nSCS_S** signal is low active if this bit is set to ‘0’ and it is high active, if this bit is set to ‘1’.

The polarity of the stepper motor driver chain clock signal **SCK_S** is defined by the bit **polarity_sck_s**. If this bit is ‘0’, the clock polarity is according to Figure 7-3 on page 13. The clock signal **SCK_S** is inverted if it is set to ‘1’. The bit **polarity_PH_AB** defines the polarity of the phase bits for the stepper motor. Inverting this bit changes the rotation direction of the associated stepper motor. The bit **polarity_FD** defines the polarity of the fast decay controlling bit. If it is ‘0’ fast decay is high active and

if it is '1' fast decay is low active. The bit named **polarity_DAC_AB** defines the polarity of the DAC bit vectors. If it is '0' the DAC bits are high active and if it is '1' the DAC bits are inverted – low active.

The bit named **csCommonIndividual** defines either if a single chip select signal **nSCS_S** is used in common for all stepper motor driver chips (**TMC236, TMC239, TMC246, TMC249**) or three chip select signals **nSCS_S**, **nSCS2**, **nSCS3** are used to select the stepper motor driver chips individually. This feature is useful only for the TMC429 within the larger packages, where the two additional chip select signals **nSCS2**, **nSCS3** are available (see Figure 3-4). The one common chip select signal **nSCS_S** is used if the bit named **csCommonIndividual='0'**. The polarity control bit for the **nSCS_S** signal must be set to **polarity_nscs_s='0'** if **csCommonIndividual='1'**. The chip select polarity is always negative for three individual chips select signals.

The eight bits named **clk2_div** determine the clock frequency of the stepper motor driver chain clock signal **SCK_S**. The frequency **f_sck_s[Hz]** of the stepper motor driver chain clock signal **SCK_S** is **f_sck_s[Hz] = f_clk[Hz] / (2 * (clk2_div+1))**. A value of **255 (%11111111, $FF)** is the upper limit for the parameter **clk2_div**. With **clk2_div = 255** the clock frequency of **SCK_S** is at minimum. Due to internal processing, a value of **7 (%00000111, $07)** is the lower limit for the clock divider parameter **clk2_div**. With **clk2_div = 7** the clock frequency of **SCK_S** is at maximum. A value of **clk2_div = 7** is sufficient for the drivers TMC236 / TMC239 / TMC246 / TMC249.

Note: For most applications, a setting of **clk2_div = 7** is recommended.

For smooth motion even at high step frequencies the frequency **f_sck_s[Hz]** of the clock signal **SCK_S** should be as high as possible that is compatible with the used drivers. The frequency **f_sck_s[Hz]** of **SCK_S** does not become higher for **clk2_div < 7**, but the signal **SCK_S** becomes asymmetric with respect to its duty cycle. An asymmetric duty cycle may cause malfunction of stepper motor drivers, where stepper motor driver chips may work correctly in particular at low clock frequencies of **CLK**. So, the range of **clk2_div** is **{7, 8, 9, . . ., 253, 254, 255}**.

The default value after power-on reset is **clk2_div = 15**. The clock frequency **f_sck_s[Hz]** of **SCK_S** should be set as high as possible by choosing the parameter **clk2_div** in consideration of the data clock frequency limit defined by the slowest stepper motor driver chip of the daisy chain. If step frequencies reach the order of magnitude of the maximum datagram frequency– determined by the clock frequency of **SCK_S** and by the datagram length –the step frequencies may jitter, which is an inherent property of that serial communication. Up to which level variations of step frequencies are acceptable depends on the application. Using microstepping driver chips– as provided by TMC236 / TMC239 / TMC246 / TMC249 driver chips –avoids this problem.

The datagram frequency is **f_datagram[Hz] = f_sck_s[Hz] / (1 + datagram_length[bit] + 1).** This formula is an approximation for the upper limit. For clk2_div = 7 the processing of the NxM bit requires 1 SPI clock cycle, where the processing of the NxM bit requires 1.5 SPI clock cycles for clk2_div > 7. So, for a chain of three drivers with 12 bit datagram length each, the upper limit of the datagram frequency is **f_datagram[Hz] = f_sck_s[Hz] / (1 + 3*(12+1) + 1) = f_sck_s[Hz] / 41**.

The TMC429 sends datagrams to the stepper motor driver chain on demand only. No datagrams are send if **continuous_update** is '0' during rest periods. This reduces the communication traffic. The multiplexed reference switch inputs are processed while datagrams are sent to the stepper motor driver chain only. If reference switches are configured to stop associated stepper motors automatically, the configuration bit **continuous_update** must be set to '1' to force periodic sending of datagrams to the stepper motor driver chain and to sample the reference switches periodically, if all stepper motors are at rest. With this, a stepper motor restarts if its associated reference switch becomes inactive. Without continuous update, a stepper motor stopped by a reference switch would stay at rest until a datagram is sent to the stepper motor driver chain, if its reference switch is inactive. Then, the relevant stepper motor can be moved into the direction opposite to the reference switch or it can be moved in both directions by disabling the automatic stop function. The continuous update datagram frequency is **f_cupd_s[Hz] = f_clk[Hz] * (1 / 2^ramp_div_0 + 1 / 2^ramp_div_1 + 1 / 2^ramp_div_2) / 32768** where **ramp_div_0, ramp_div_1, ramp_div_2** are the **ramp_div** settings of the three stepper motors.

The bit **continuous_update** is also important for the automatic coil current scaling (see page 21). This bit must be set to '1' to be sure that the coil current is also scaled if all motors are at rest.

Important Hint: The TMC429-I (SSOP16 package variant) hat only one left side reference switch available when operating in step-direction mode.

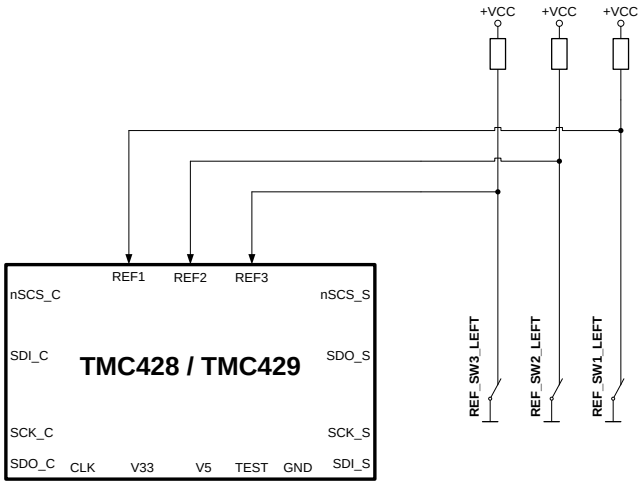

Figure 10-4: Reference switch configuration 'left-side-only' for mot1r=0 (and refmux=0)

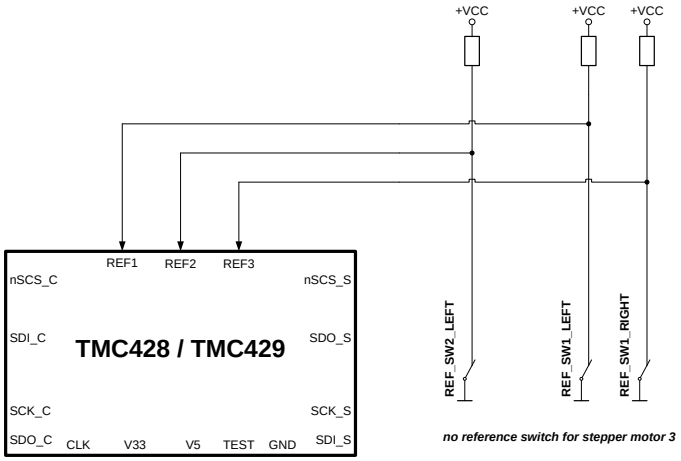

Figure 10-5: Reference switch configuration 'two-one-null' for mot1r=1 (and refmux=0)

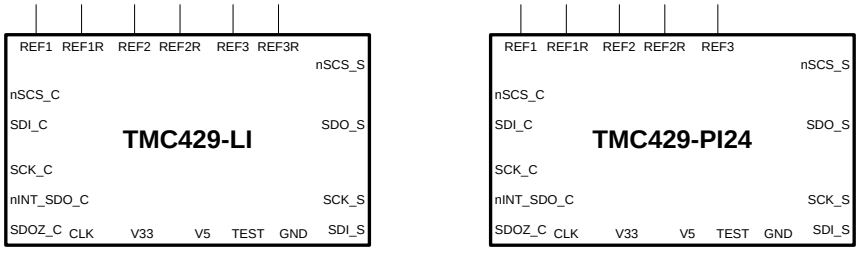

Figure 10-6: TMC429-LI has 3 right reference inputs / TMC429-PI24 has 2 right reference inputs

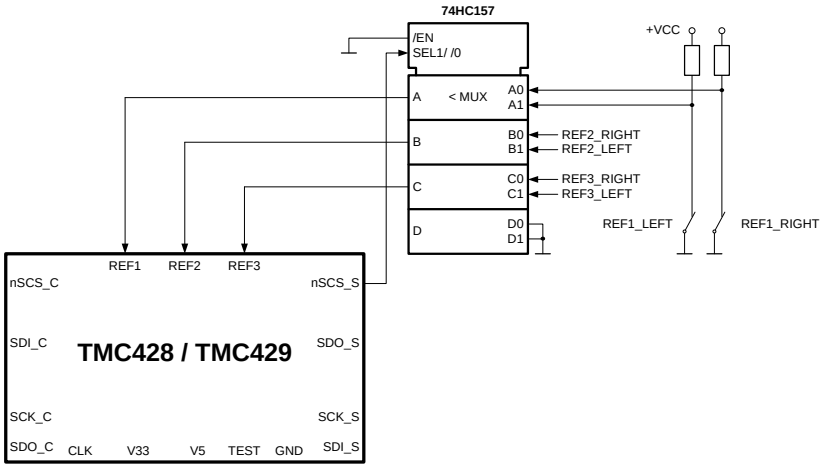

Figure 10-7: Reference switch multiplexing with 74HC157 (refmux=1)

refmux	mot1r	motor 1		motor 2		motor 3	
		left switch	right switch	left switch	right switch	left switch	right switch
0	0	REF1	%	REF2	%	REF3	%
0	1	REF1	REF3	REF2	%	%	%
1	0	REF1_LEFT	REF1_RIGHT	REF2_LEFT	REF2_RIGHT	REF3_LEFT	REF3_RIGHT
1	1	REF1_LEFT	REF1_RIGHT	REF2_LEFT	REF2_RIGHT	REF3_LEFT	REF3_RIGHT

Table 10-5: Association of reference inputs depending on configuration bits refmux & mot1r

If **continuous_update** is '1', internal reference switch bits are updated periodically, even if all stepper motors are at rest. Additionally, the chip select signal **nSCS_S** for the stepper motor driver chain is also the control signal for a multiplexer in case of using the reference switch multiplexing option (see Figure 10-7). So, the **continuous_update** must be set to '1' if automatic stop by reference switches is enabled, if six multiplexed reference switches are used, and to get the states of reference switches while all stepper motors are at rest.

The bit named **refmux** must be set to '1' to enable reference switch multiplexing (see Figure 10-7). For the two variants TMC429-PI24 and TMC429-LI, the reference switch multiplexing also works for **csCommonIndividual**='1' using three separate driver selection signals (nSCS_S, nSCS2, nSCS3) if the signal nSCS_S is connected to the multiplexer 74HC157 according to Figure 10-10.

If reference switch multiplexing is enabled, **mot1r** is ignored. With **refmux** set to '0', the association of the reference switch inputs **REF1**, **REF2**, **REF3** depends on the setting of the configuration bit **mot1r**. The power-on default value of **mot1r** is '0'. With that default value, **REF1** is associated to the left reference switch of stepper motor #1, **REF2** is associated to the left reference switch of stepper motor #2, and **REF3** is associated to the left reference switch of stepper motor #3.

If **mot1r** is set to '1' the input **REF1** is also associated with the left reference switch of stepper motor #1. **REF2** is also associated to the left reference switch of stepper motor #2. But, the input **REF3** is associated to the *right reference switch* of stepper *motor #1* and no reference switch input is associated to stepper motor number#3 (see Figure 10-5). After power-on-reset, per default **refmux=0** and **mot1r=0** selects the single reference switch configuration outlined in Figure 10-4, where each reference switch input (**REF1**, **REF2**, **REF3**) is assigned individually to one each stepper motor as the left reference switch.

10.12 Triple Switch Configuration

The programmable tolerance range around the reference switch position is useful for a triple switch configuration, as outlined in Figure 10-8. In that configuration two switches are used as automatic stop switches, and one additional switch is used as the reference switch between the left stop switch and the right stop switch. The left stop switch and the reference switch are wired or. After successful reference search, programming a tolerance range into the register **dx_ref_tolerance** allows to disable automatic stop within the range of the reference switch only.

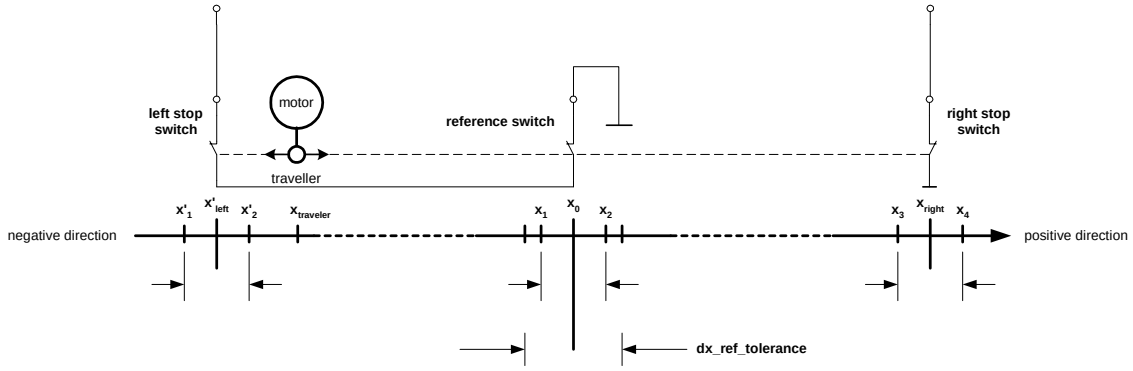

Figure 10-8: Triple switch configuration 'left stop switch – reference switch – right stop switch'

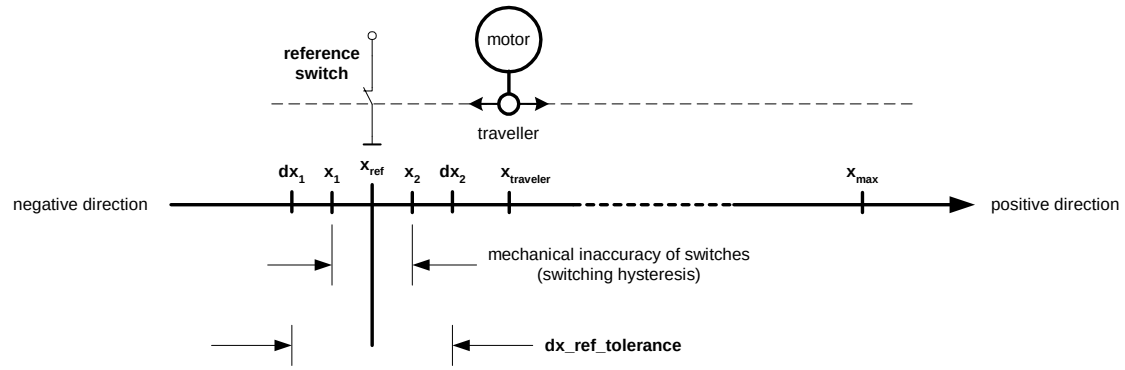

Figure 10-9: Reference search

10.13 Reference Search

The goal of the reference search is to determine the position $\mathbf{x_{ref}}$ of a reference switch (see Figure 10-9). Due to mechanical inaccuracy of switches, the reference switch is active within a range $\mathbf{x_1} < \mathbf{x_{ref}} < \mathbf{x_2}$, where $\mathbf{x_1}$ and $\mathbf{x_2}$ may vary. If the traveler is within the range $\mathbf{x_1} < \mathbf{x_{traveler}} < \mathbf{x_2}$ before reference search, it is necessary to go outside this range, because the associated reference switch is active. A dummy write access to **x_latched** initializes the position latch register. Then, with the traveler within the range $\mathbf{x_2} < \mathbf{x_{traveler}} < \mathbf{x_{max}}$ and the initialized register **x_latched**, the position $\mathbf{x_2}$ can simply be determined by motion with a target position **x_target** set to $\mathbf{-x_{max}}$. When reaching the position $\mathbf{x_2}$, this position is latched automatically. With stop switch enabled, the stepper motor automatically stops if the position $\mathbf{x_2}$ is reached. Then, the **dx_ref_tolerance** has to be set, so that motion within the active reference switch range $\mathbf{x_1} < \mathbf{x_{ref}} < \mathbf{x_2}$ is allowed and to move the traveler to a position $\mathbf{x_{traveler}} < \mathbf{x_1}$ if desired. Then the register **x_latched** has to be initialized again to latch the position $\mathbf{x_1}$ by a motion to a target position $\mathbf{x_{traveler}} < \mathbf{x_1}$. When the positions $\mathbf{x_1}$ and $\mathbf{x_2}$ are determined the reference position $\mathbf{x_{ref}} = (\mathbf{x_1} + \mathbf{x_2}) / 2$ can be set. Finally, one should move to the target position **x_target =** $\mathbf{x_{ref}}$ and set **x_target := 0** and **x_actual := 0** when reached.

10.14 Simultanous Start of up to Three Stepper Motors

Starting stepper motors simultaneously can be acheved by sending successive datagrams starting the stepper motors. If the delay between those datagrams is of the magnitude of some microseconds, the stepper motors can be considered as started simultaneously. Feeding the reference switch signals through or gates (see Figure 10-10) allows exact simultaneous start of the stepper motors under software control.

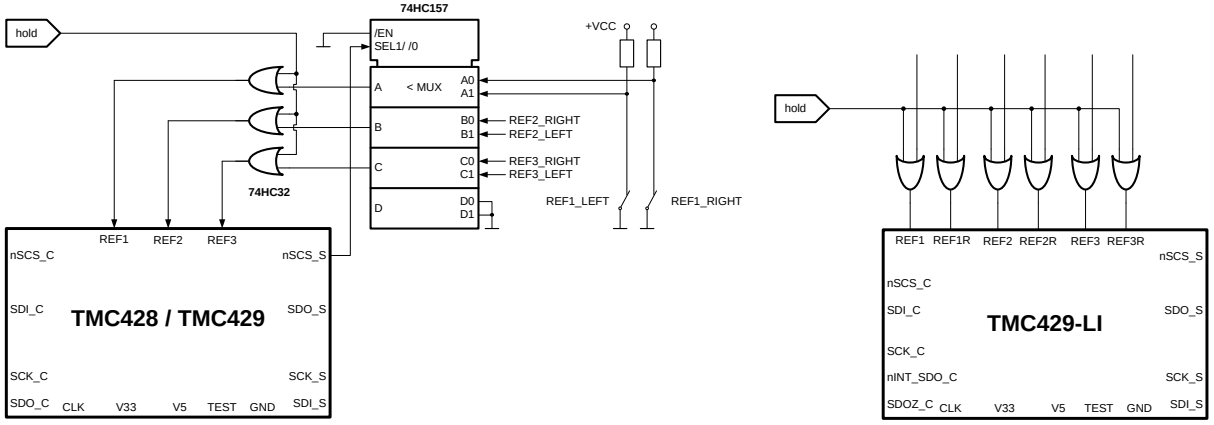

Figure 10-10: Reference switch gateing for exact simultanous stepper motor start

11 RAM Address Partitioning and Data Organization

The on-chip RAM capacity is 128 x 6 bit. These 128 on-chip RAM cells of 6 bit width are addressed via 64 addresses of 2 x 6 bit (see Table 11-1). So, from the point of view of addressing the on-chip RAM via datagrams, the address space enfolds 64 addresses of 24 bit wide data, where only 2 x 6 = 12 bits are relevant. These 64 addresses are partitioned– selected by the **RRS** (Register RAM Select, datagram bit 31)– into two address ranges of 32 addresses. The registers of the TMC429 are addressed with RRS='0'. The on-chip RAM is addressed with RRS='1'. The 64 on-chip RAM addresses are partitioned into two separate ranges by the most significant address bit of the datagram (bit 30).

The first 32 addresses are provided for the configuration of the serial stepper motor driver chain. Each of these 32 addresses stores two configuration words, composed of the so called NxM (**Ne**x**t** **M**otor) bit together with the 5 bit wide primary signal code. While sending a datagram, the primary signal code words are read internally beginning with the first address of the driver chain datagram configuration memory range. Each primary signal code word selects a signal provided by the micro step unit. If the NxM bit is '1' an internal stepper motor addressing counter is incremented. If this internal counter is equivalent to the **LSMD** (last stepper motor driver) parameter, the datagram transmission is finished and the counter is preset to %00 for the next datagram transmission to the stepper motor driver chain.

The second 32 addresses are provided to store the micro step table, which usually is a quarter sine wave period as a basic approach or the quarter period of a periodic function optimized for microstepping of a given stepper motor type. Different stepper motors may step with different micro step resolutions, but the micro step look up table (LUT) is the same for all stepper motors controlled by one TMC429. Any quarter wave period stored in the micro step table is expanded automatically to a full period wave together with its 90° phase shifted wave.

32 bit DATAGRAM sent from a µC to the TMC429 via pin SDI_C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRS	**ADDRESS**						**RW**	**DATA**																							
																		data @ odd RAM addresses									**data @ even RAM addresses**				
1	**0**	**32 x (2x6 bit) driver chain datagram configuration range**																NxM_1	signal_codes								NxM_0	signal_codes			
1	**1**	**32 x (2x6 bit) quarter period sine wave LUT range**																quarter sine wave values (amplitude)									quarter sine wave values (amplitude)				

Table 11-1: Partitioning of the on-chip RAM address space

12 Stepper Motor Driver Datagram Configuration

A number of control signals is required to drive 2-phase stepper motors. The serial driver interface forms the link between the TMC429 and the stepper motor driver chain. The stepper motor driver datagram configuration simply defines the order of the control signals serially sent from TMC429 to the stepper motor driver chain. To define the serial order of the control signals, so called primary signal codes have to be written into the stepper motor driver datagram configuration area of the on-chip configuration RAM of the TMC429.

Which control signals are required in a given application, depends on the choice of stepping mode– full step, half step, micro step –and on additional options depending on the stepper motor driver chips used. So, the TMC429 primarily provides a full set of control signals individually for each of the up to three stepper 2-phase stepper motors respectively stepper motor driver chips of the daisy chain. Mnemonics for primary signal codes are given in Table 12-1. Names of these signals may differ to the signal names of the used stepper motor drivers. Figure 12-1 on page 42 outlines the principle of connecting the control signals– internally provided by the TMC429 as parallel signals –with the signals used to control the digital part of a stepper motor driver.

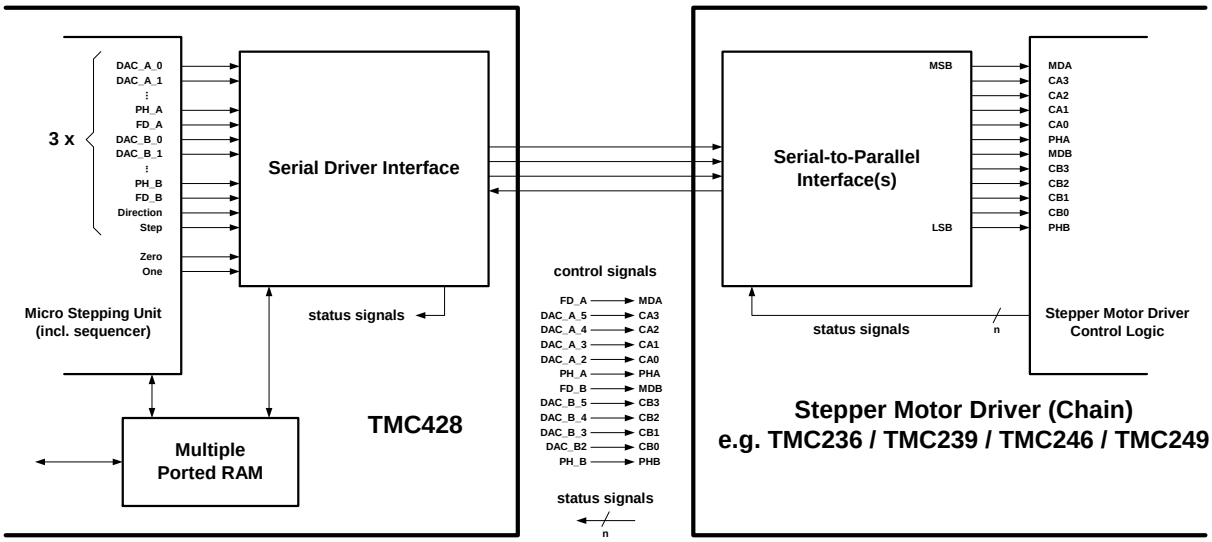

Figure 12-1: Serially transmitted control and status signals between TMC429 and driver chain

MNEMONIC	PRIMARY SIGNAL CODE		FUNCTION	
	hex	bin		
DAC_A_0	$00	%00000	DAC A, bit 0 (LSB)	COIL A
DAC_A_1	$01	%00001	DAC A, bit 1	
DAC_A_2	$02	%00010	DAC A, bit 2	
DAC_A_3	$03	%00011	DAC A, bit 3	
DAC_A_4	$04	%00100	DAC A, bit 4	
DAC_A_5	$05	%00101	DAC A, bit 5 (MSB)	
PH_A	$06	%00110	phase polarity bit A	
FD_A	$07	%00111	fast decay bit A	
DAC_B_0	$08	%01000	DAC B, bit 0 (LSB)	COIL B
DAC_B_1	$09	%01001	DAC B, bit 1	
DAC_B_2	$0A	%01010	DAC B, bit 2	
DAC_B_3	$0B	%01011	DAC B, bit 3	
DAC_B_4	$0C	%01100	DAC B, bit 4	
DAC_B_5	$0D	%01101	DAC B, bit 5 (MSB)	
PH_B	$0E	%01110	phase polarity bit B	
FD_B	$0F	%01111	fast decay bit B	
Zero	$10	%10000	constant '0'	
One	$11	%10001	constant '1'	
Direction	$12	%10010	0 : up / 1 : down resp. counter clockwise / clockwise	
Step	$13	%10011	step bit for step/direction control of drivers	
UNUSED (these codes might be used for future devices)	$14	%10100	'1' for TMC429-I, TMC429-PI24, TMC429-LI	
	$15	%10101		
	$16	%10110		
	$17	%10111		
	$18	%11000		
	$19	%11001		
	$1A	%11010		
	$1B	%11011		
	$1C	%11100		
	$1D	%11101		
	$1E	%11110		
	$1F	%11111		

Table 12-1: Primary signal codes

The control signals for each of the two coils of a 2-phase stepper motor are 6 bits for the DAC controlling the current of a coil, a phase polarity bit, and a fast decay bit for those stepper motor driver chips with a fast decay feature for the coil current. These signals are available individually for each coil (COIL A and COIL B). Constant configuration bits named Zero and One are provided. Additionally, step and direction bits are available. One unique 5 bit code word– named primary signal code –is assigned to each primary control signal (see Table 12-1).

The micro step unit (including sequencer) provides the full set of control signals for three stepper motor driver chips. A subset of these control signals is selected by the stepper motor driver datagram configuration, which is stored within the first 32 addresses representing 64 values of the on-chip RAM (see Table 11-1, page 41). The stepper motor drivers are organized in a daisy chain. So the addressing of the stepper motor driver chips within the daisy chain is by its position.
As mentioned before, the TMC429 sends datagrams to the stepper motor driver chain on demand. To guarantee the integrity of each datagram sent to the stepper motor driver chain, the status of all primary control signals is buffered internally before sending. Afterwards, the transmission starts with selection of the buffered primary control signals of the first motor (**smda**=%00) by reading the first primary signal code word (even data word at on-chip RAM address %00000) from on-chip configuration RAM area. The primary signal codes select the primary signals provided for the first stepper motor. The first stepper motor is addressed until the **NxM** (next motor) bit is read from on-chip configuration RAM. The stepper motor driver address is incremented with each **NxM='1'** as long as the current stepper motor driver address is below the value set by the parameter **LSMD** (last stepper motor driver). If the stepper motor driver address is equivalent to the **LSMD** parameter, a **NxM='1'** indicates the completion of the transmission. With that, the stepper motor driver address counter of the serial interface is reinitialized to %00 and the unit waits for the next transmission request.
So, the order of primary signal codes in the on-chip RAM configuration area determines the order of datagram bits for the stepper motor driver chain, whereas the prefixed NxM bit determines the stepper motor driver positions. If no NxM bit with a value of '1' is stored within the on-chip RAM, the TMC429 will send endlessly. So, the on-chip RAM has to be configured first. After power-on reset, the registers of the TMC429 are initialized in a way, that no transmission of datagrams to the motor driver chain is required. Access to on-chip RAM is always possible, also during transmission of datagrams to the driver chain.

12.1 Initialization of on-chip-RAM by µC after power-on

All registers are initialized by the automatic power-on reset. The registers are initialized, and the stepper motors are at rest. The on-chip RAM of the TMC429 is initialized by the power-on reset with a default configuration for a TMC236/TMC239/TMC246/TM249 stepper motor driver chain. Writing to registers may involve action of the stepper motor units initiated by the TMC429 resulting in sending datagrams to the stepper motor driver chain. For the TMC428, datagrams have a random power-on configuration of the on-chip-RAM. For the TMC429, the power-on default configuration is for a chain of TMC236/TMC239/TMC246/TMC249. So, before trying to move a motor when using other stepper motor drivers (TMC260, TMC261, or TMC262) in a SPI chain as TMC236/TMC239/TMC246/TMC249, the on-chip RAM must be initialized first.

12.2 An Example of a Stepper Motor Driver Datagram Configuration

The following example demonstrates how to configure the datagram and shows what has to be stored within the on-chip RAM to represent the desired configuration. That example refers to a driver chain of three TRINAMIC stepper motor drivers of type TMC236, TMC239, TMC246, TMC249. From the TMC429 datagram configuration point of view, there is no difference between these drivers. All these drivers have a serial interface of 12 bits length. The configuration is as follows. For the first and the second stepper motor driver of the chain the fast decay control bit (FD_A, FD_B) is fixed to '0'. For the third driver the fast decay control bit are used. The corresponding content of the configuration on-chip RAM is outlined in Table 12-2. The sequence to be sent to the TMC429 for this configuration is outlined in Table 12-3.

Hint: The stepper motor driver datagram configuration can be accessed at any time without conflict, e.g. to changed between a configuration using fast decay versus a configuration where fast decay is disabled.

position within datagram	driver	NxM bit	TMC429 signal code	RAM address	RAM data {POR}	TMC429 mnemonic of primary signal {POR default}		TMC23x / TMC24x Bit Name
0	driver#1 (SMDA=%00)	0	$10	$00	$10 {$11}	Zero {One}	→	MDA
1		0	$05	$01	$05	DAC_A_5	→	CA3
2		0	$04	$02	$04	DAC_A_4	→	CA2
3		0	$03	$03	$03	DAC_A_3	→	CA1
4		0	$02	$04	$02	DAC_A_2	→	CA0
5		0	$06	$05	$06	PH_A	→	PHA
6		0	$10	$06	$10 {$11}	Zero {One}	→	MDB
7		0	$0D	$07	$0D	DAC_B_5	→	CB3
8		0	$0C	$08	$0C	DAC_B_4	→	CB2
9		0	$0B	$09	$0B	DAC_B_3	→	CB1
10		0	$0A	$0A	$0A	DAC_B_2	→	CB0
11		**1**	$0E	$0B	**$2E**	PH_B	→	PHB
12	*driver#2 (SMDA=%01)*	*0*	*$10*	*$0C*	*$10 {$11}*	*Zero {One}*	→	*MDA*
13		*0*	*$05*	*$0D*	*$05*	*DAC_A_5*	→	*CA3*
14		*0*	*$04*	*$0E*	*$04*	*DAC_A_4*	→	*CA2*
15		*0*	*$03*	*$0F*	*$03*	*DAC_A_3*	→	*CA1*
16		*0*	*$02*	*$10*	*$02*	*DAC_A_2*	→	*CA0*
17		*0*	*$06*	*$11*	*$06*	*PH_A*	→	*PHA*
18		*0*	*$10*	*$12*	*$10 {$11}*	*Zero {One}*	→	*MDB*
19		*0*	*$0D*	*$13*	*$0D*	*DAC_B_5*	→	*CB3*
20		*0*	*$0C*	*$14*	*$0C*	*DAC_B_4*	→	*CB2*
21		*0*	*$0B*	*$15*	*$0B*	*DAC_B_3*	→	*CB1*
22		*0*	*$0A*	*$16*	*$0A*	*DAC_B_2*	→	*CB0*
23		***1***	*$0E*	*$17*	***$2E***	*PH_B*	→	*PHB*
24	driver#3 (SMDA=%10)	0	$07	$18	$07 {$11}	FD_A {One}	→	MDA
25		0	$05	$19	$05	DAC_A_5	→	CA3
26		0	$04	$1°	$04	DAC_A_4	→	CA2
27		0	$03	$1B	$03	DAC_A_3	→	CA1
28		0	$02	$1C	$02	DAC_A_2	→	CA0
29		0	$06	$1D	$06	PH_A	→	PHA
30		0	$0F	$1E	$0F {$11}	FD_B {One}	→	MDB
31		0	$0D	$1F	$0D	DAC_B_5	→	CB3
32		0	$0C	$20	$0C	DAC_B_4	→	CB2
33		0	$0B	$21	$0B	DAC_B_3	→	CB1
34		0	$0A	$22	$0A	DAC_B_2	→	CB0
35		**1**	$0E	**$23**	**$2E**	PH_B	→	PHB
With **LSMD** = %10 the (third) NxM bit at address $**23** (position 35) finishes the datagram transmission								

Table 12-2: Datagram example and RAM contents for three stepper motor driver chain

Hint: Power-On default initialization values of the TMC429 are marked as {One} within theTable 12-2.

```
 binary datagram specification    :  hexadecimal datagram
%10000000----------000101-010000 : $80000510
%10000010----------000011-000100 : $82000304
%10000100----------000110-000010 : $84000602
%10000110----------001101-010000 : $86000D10
%10001000----------001011-001100 : $88000B0C
%10001010----------101110-001010 : $8°002E0A
%10001100----------000101-010000 : $8c000510
%10001110----------000011-000100 : $8e000304
%10010000----------000110-000010 : $90000602
%10010010----------001101-010000 : $92000D10
%10010100----------001011-001100 : $94000B0C
%10010110----------101110-001010 : $96002E0A
%10011000----------000101-000111 : $98000507
%10011010----------000011-000100 : $9°000304
%10011100----------000110-000010 : $9c000602
%10011110----------001101-001111 : $9e000D0F
%10100000----------001011-001100 : $a0000B0C
%10100010----------101110-001010 : $a2002E0A
```

Table 12-3: Configuration datagram sequence for the example (with '-' (don't cares))

13 Initialization of the Micro Step Look-Up-Table

The TMC429 provides a look-up-table (LUT) of 64 values of 6 bit for micro stepping. The micro step LUT can be adapted by storing an arbitrary quarter period of a periodic function to match individual stepper motor characteristics. It is common to use one period of a sine wave function for micro stepping. With that function, the current of one phase is driven with the sine function whereas the other phase is driven with the cosine function.

To initialize the LUT for micro stepping one simply has to load a quarter sine wave period into the micro step LUT. Two successive values of the sine wave function are included in one datagram similar to the primary signal code words for the stepper motor driver chain configuration. The TMC429 automatically expands the quarter sine wave period to a full sine and cosine function. The necessary data values y(i) to represent a ¼ sine wave period for the micro step LUT are defined by

$$y(i) = \text{int}[\ ½ + 64 * \sin(¼ * 2 * \pi * i / 64)\ ] \quad \text{with} \quad i = \{ 0, 1, 2, 3, ..., 60, 61, 62, 63 \},$$

where the conditional replacement y(i) := *63* for y(i) > 63 has to be done. The last five values (which are calculated to be 64) have to be replaced by *63*. With this replacement one finally gets y(i) = { 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* }.

Hint: The power-on reset default initialization of the TMC429 sine wave look-up table is with offset for TMC236/TMC239/TMC246/TMC249 with mixed decay control = One (ON).
Pleses refer TMC260/TMC261/TMC262 datasheet in case of SPI motion control. For step/direction control the TMC260/TMC261/TMC262 have their own optimized build-in sine wave look-up table.

32 bit DATAGRAM sent from a µC to the TMC429 via pin SDI_C																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRS	ADDRESS						RW	DATA																							
									$(x)_{10}$	data @ odd RAM addresses						$(x)_{10}$	data @ even RAM addresses												
1	1	0	0	0	0	0	0		2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	1			5	0	0	0	1	0	1	3	0	0	0	0	1	1									
		0	0	0	1	0			8	0	0	1	0	0	0	6	0	0	0	1	1	0									
		0	0	0	1	1			11	0	0	1	0	1	1	9	0	0	1	0	0	1									
		0	0	1	0	0			14	0	0	1	1	1	0	12	0	0	1	1	0	0									
		⋮	⋮	⋮	⋮	⋮				⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮	⋮	⋮	⋮	⋮									
		1	1	0	1	1			62	1	1	1	1	1	0	62	1	1	1	1	1	0									
		1	1	1	0	0			63	1	1	1	1	1	1	63	1	1	1	1	1	1									
		1	1	1	0	1			*63*	1	1	1	1	1	1	63	1	1	1	1	1	1									
		1	1	1	1	0			*63*	1	1	1	1	1	1	*63*	1	1	1	1	1	1									
		1	1	1	1	1			*63*	1	1	1	1	1	1	*63*	1	1	1	1	1	1									

Table 13-1: Scheme of ¼ sine wave period with 6 bit resolution and 64 (32 x 2) values

These 64 values represent a quarter sine period in the interval [0 ... π/4[which is expanded automatically by the TMC429 to a full sine cosine period (see section 13.1, page 47). The table is sent to the on-chip RAM of the TMC429 by 32 datagrams:

```
% binary representation of the datagram          : decimal represented pair of values : $ hexadecimal
                                                      (separated by & character)          representation

% 11 00000 0 00000000 00 000010 00 000000 :   2 & 0 : $C0000200
% 11 00001 0 00000000 00 000101 00 000011 :   5 & 3 : $C2000503
% 11 00010 0 00000000 00 001000 00 000110 :   8 & 6 : $C4000806
% 11 00011 0 00000000 00 001011 00 001001 :  11 & 9 : $C6000B09
% 11 00100 0 00000000 00 001110 00 001100 :  14 & 12 : $C8000E0C
% 11 00101 0 00000000 00 010001 00 010000 :  17 & 16 : $CA001110
% 11 00110 0 00000000 00 010100 00 010011 :  20 & 19 : $CC001413
% 11 00111 0 00000000 00 010111 00 010110 :  23 & 22 : $CE001716

% 11 01000 0 00000000 00 011010 00 011000 :  26 & 24 : $D0001A18
% 11 01001 0 00000000 00 011101 00 011011 :  29 & 27 : $D2001D1B
% 11 01010 0 00000000 00 100000 00 011110 :  32 & 30 : $D400201E
% 11 01011 0 00000000 00 100010 00 100001 :  34 & 33 : $D6002221
% 11 01100 0 00000000 00 100101 00 100100 :  37 & 36 : $D8002524
% 11 01101 0 00000000 00 100111 00 100110 :  39 & 38 : $DA002726
% 11 01110 0 00000000 00 101010 00 101001 :  42 & 41 : $DC002A29
% 11 01111 0 00000000 00 101100 00 101011 :  44 & 43 : $DE002C2B

% 11 10000 0 00000000 00 101110 00 101101 :  46 & 45 : $E0002E2D
% 11 10001 0 00000000 00 110000 00 101111 :  48 & 47 : $E200302F
```

```
% 11 10010 0 00000000 00 110010 00 110001 : 50 & 49 : $E4003231
% 11 10011 0 00000000 00 110100 00 110011 : 52 & 51 : $E6003433
% 11 10100 0 00000000 00 110110 00 110101 : 54 & 53 : $E8003635
% 11 10101 0 00000000 00 111000 00 110111 : 56 & 55 : $EA003837
% 11 10110 0 00000000 00 111001 00 111000 : 57 & 56 : $EC003938
% 11 10111 0 00000000 00 111011 00 111010 : 59 & 58 : $EE003B3A

% 11 11000 0 00000000 00 111100 00 111011 : 60 & 59 : $F0003C3B
% 11 11001 0 00000000 00 111101 00 111100 : 61 & 60 : $F2003D3C
% 11 11010 0 00000000 00 111110 00 111101 : 62 & 61 : $F4003E3D
% 11 11011 0 00000000 00 111110 00 111110 : 62 & 62 : $F6003E3E
% 11 11100 0 00000000 00 111111 00 111111 : 63 & 63 : $F8003F3F
% 11 11101 0 00000000 00 111111 00 111111 : 63 & 63 : $FA003F3F
% 11 11110 0 00000000 00 111111 00 111111 : 63 & 63 : $FC003F3F
% 11 11111 0 00000000 00 111111 00 111111 : 63 & 63 : $FE003F3F
```

Table 13-2 – Datagrams for initialization of a quarter sine wave period microstep look-up-table

These 32 datagrams (Table 13-2) are sufficient for all programmable micro step resolutions. If micro stepping is used for at least one stepper motor, these 32 datagrams have to be sent once to the TMC429 for initialization of the micro step table after power-on reset. The initialization of the micro step look-up-table is not necessary, if full stepping is used for *all* stepper motors. The on-chip RAM is not initialized during power-on reset. So, the full initialization of the whole micro step look-up-table is recommended to avoid trouble caused by missing look-up table entries. Additionally, a fully initialized micro step look-up-table allows the selection of individual micro step resolutions for different steppes.

Hint: The previous section describes how to calculate values for the TMC429 sine wave look-up table, and how to write those values into the look-up table RAM of the TMC429 via SPI datagramms. Nevertheless, other formulas might generate sine wave look-up tables that better fit with respect to equidistance of micro steps or concerning torque ripple. Alternatively, a sine wave look-up table based on

$$y(i) = int[63 * sin(¼ * 2 * \pi * i / 64)]$$

can be used. Even, adding some offset o within theoretical range from 0 to 63

$$y(i) = int[o + (63-o) * sin(¼ * 2 * \pi * i / 64)]$$

might enhance the micro step behavior together with mixed decay. Finally, a sine wave look-up table has to be tested within the application and might need some fine adjustments for optimization.

13.1 Stepping through the Wave Look-Up-Table

The 64 values of the wave look-up table (LUT) hold a quarter period of a sine wave, indexed from 0 to 63. This quarter sine wave is expanded to full sine wave and full cosine wave by indexing the 64 values of the LUT. The LUT index is mapped from a wave index of a full period from 0 to 255. The stepping through the LUT is done with fixed increment width. The increment width is a power of two and it depends on the micro step resolution selection **usrs**. The Table 13-4 gives the indices for the LUT for the different micro step resolutions. For motion in positive direction, the full period index is incremented from micro step to micro step. For motion into negative direction, the full period index is decremented from micro step to micro step.

The sine is associated to phase A (PH_A) and the cosine is associated to phase B (PH_B). The phase bits represent the sign of the sine resp. cosine function. The polarity of the phase bit (PH_A, PH_B) and the polarity of the fast decay control bits (FD_A, FD_B) can be changed by the polarity bits within the global stepper motor parameter register (see section 10.11, page 35).

MSBs of full wave index (range)	Quadrant	PH_A (sin)	PH_B (cos)	FD_A (sin)	FD_B (cos)
%00 (0...63)	1st	1	1	0	1
%01 (64...127)	2nd	1	0	1	0
%10 (128...191)	3rd	0	0	0	1
%11 (192...255)	4th	0	1	1	0

Table 13-3: Phase Bits and Fast Decay Control Bits

Hints: One should always initialize the whole LUT to be sure to read valid wave values, even if one changes the micro step resolution after some micro steps have been done on higher resolution. Even if one uses e.g. 16 times microstepping, one gets a smoother move for the stepper motor if one initializes the full sine wave LUT according to Table 13-2 – Datagrams for initialization of a quarter sine wave period microstep look-up-table on page 47 and using the MSB DAC bits (DAC_A_5, DAC_A_4, DAC_A_3, DAC_A_2 and DAC_B_5, DAC_B_4, DAC_B_3, DAC_B_2). So, one should always completely initialize the quarter sine wave LUT, no matter what micro step resolution is used. The addressing starts at 0 after power-on reset only. Changing the micro step resolution after some micro steps have been made causes an offset for the addressing that has to be taken into account for positioning a stepper motor.

usrs	increment width		1st quadrant	2nd quadrant	3rd quadrant	4th quadrant
%110	1	sin	0,1,2, 3, …, 61, 62,63,	63, 63, 62, 61,…, 3, 2,1,	0,1,2, 3, …, 61, 62,63,	63, 63,62,61,…, 3, 2,1
		cos	63, 63, 62, 61,…,2,1,	0,1,2, 3, …, 61, 62,63,	63, 63,62,61,…,3, 2,1,	0,1,2, 3, …, 61, 62,63
%101	2	sin	0,2, 4, 6, …, 58, 60,62,	63, 60, 58, 56,…, 4, 2,	0,2, 4, 6, …, 58, 60,62,	63, 60, 58, 56,…, 4, 2
		cos	63, 60, 58, 56,…, 4, 2,	0,2, 4, 6, …, 58, 60,62,	63, 60, 58, 56,…, 4, 2,	0,2, 4, 6, …, 58, 60,62
%100	4	sin	0, 4, 8, 12, …, 56, 60,	63, 60, 56, …, 12, 8, 4,	0, 4, 8, 12, …, 56, 60,	63, 60, 56, …, 12, 8, 4
		cos	63, 60, 56, …, 12, 8, 4,	0, 4, 8, 12, …, 56, 60,	63, 60, 56, …, 12, 8, 4,	0, 4, 8, 12, …, 56, 60
%011	8	sin	0, 8, 16, 24, … , 48, 56	63, 56, 48, 40,…,16, 8,	0, 8, 16, 24, … , 48, 56	63, 56, 48, 40,…,16, 8
		cos	63, 56, 48, 40,…,16, 8,	0, 8, 16, 24, … , 48, 56	63, 56, 48, 40,…,16, 8,	0, 8, 16, 24, … , 48, 56
%010	16	sin	0, 16, 32, 48,	63, 48, 32, 16,	0, 16, 32, 48,	63, 48, 32, 16
		cos	63, 48, 32, 16,	0, 16, 32, 48,	63, 48, 32, 16,	0, 16, 32, 48
%001 (HS)	32	sin	0, 32,	63, 32,	0, 32,	63, 32
		cos	63, 32,	0, 32,	63, 32	0, 32
%000 (FS)	64	sin	0,	63,	0,	63
		cos	63,	0,	63	0

Table 13-4: Wave look-up table (LUT) indices for different microstep resolutions

13.2 Partial look-up table initialization option

A partially initialized micro step table may be sufficient, if all stepper motors– except those driven in full step mode –are programmed to use the same micro step resolution constantly before a single micro step is processed. But with a partial initialized micro step look-up table, the micro step resolution *must not be changed* after any step is made after power-on reset. So, a partially initialized look-up table should be taken into account only, if it is a must because of too small memory of the host microcontroller. Instead of partial initialization of the look-up table of the TMC429, initialization with a triangular function $f_{rhomb}(\varphi)$ would be a better choice.

13.3 Micro Step Enhancement

Even for stepper motors optimized for sine-cosine control, it is possible to improve micro step behavior by adapting the micro step look-up table (LUT). For different types of stepper motors, a periodic trapezoidal or triangular function similar to a sine function or a superposition of these function as a replacement of the pure sine wave function (Figure 13-1) might be a better choice. Taking the physics of stepper motors into account, the choice of the function for microstepping can be determined by a single shape parameter σ as explained below. The programmability of the micro step look-up table provides a simple and effective facility to enhance microstepping for a given type of two-phase stepper motor. Enhanced microstepping requires accurate current control. So, stepper motor driver chips with enabled and well tuned fast decay (resp. mixed decay) operational mode are need to be used, e.g. TRINAMICs smart power TMC236 / TMC239 / TMC246 / TMC249 drivers.

Non-linearity resulting from magnetic field configuration determined by shapes of pole shoes, ferromagnetic characteristics, and other stepper motor characteristics effect non-linearity in micro step behavior of real stepper motors. The non-linearity of microstepping causes micro step positioning displacements, vibrations and noise, which can be reduced dramatically with an adapted micro step table. The best fitting micro step table can be determined by measuring the micro step motor behavior, e.g. using a laser pointer based on the sine-cosine microstepping table.

Nevertheless sine-cosine microstepping is a good first order approach for microstepping. The micro step enhancement possible with the TMC429 is based on replacement of the look-up table initialization function $\sin(\varphi)$ used for sine-cosine microstepping by a function with the shape parameter σ. A quarter sine wave period is the basic approach for initialization of the micro step look-up-table. A quarter of a trapezoidal function or a quarter of a triangular function is chosen depending on the shape parameter σ for a given stepper motor type.

$$f_\sigma(\varphi) = \begin{cases} f_{box\_circle}(\varphi) & for \quad \sigma > 0 \\ f_{circle}(\varphi) & for \quad \sigma = 0 \\ f_{circle\_rhomb}(\varphi) & for \quad \sigma < 0 \end{cases} \qquad with \qquad -1.0 \le \sigma \le +1.0 \qquad and \qquad 0 \le \varphi < \frac{\pi}{2}.$$

The look-up table ($f(\varphi)$) of the TMC429 enfolds a quarter period ($0 \le \varphi < \pi/2$) only. This quarter period is expanded to a full period ($0 \le \varphi < 2\pi$) and the phase shifted companion function value ($f(\varphi - \pi/2)$) is added automatically by the TMC429 during operation. So, to reach function value ($f(\varphi)$), one automatically gets a pair of function values {$f(\varphi)$; $f(\varphi - \pi/2)$} respectively {$\sin(\varphi)$; $\cos(\varphi)$}. This automatic expansion of the TMC429– primary provided for sine cosine microstepping ($f(\varphi) = \sin(\varphi)$) – also works fine with other micro step wave forms f_σ.

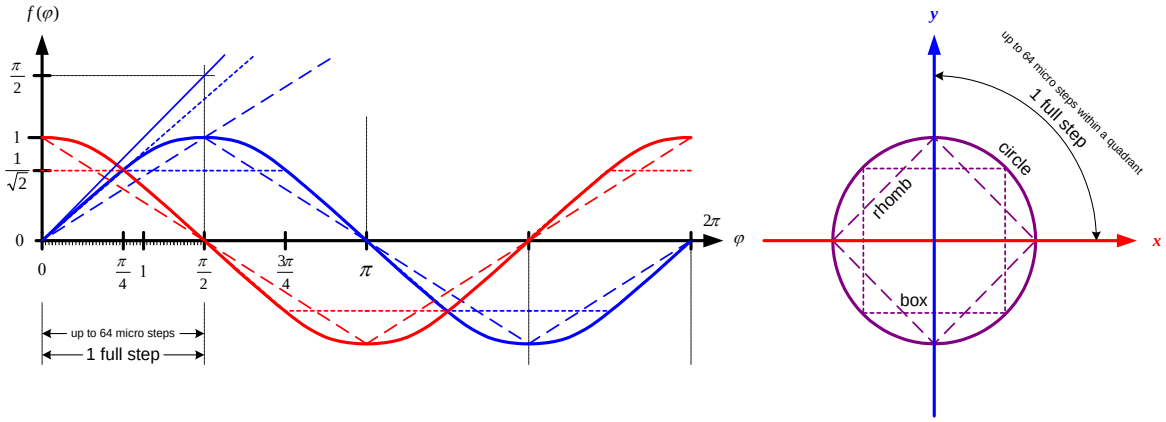

Figure 13-1: Microstep enhancemant by introduction of a shape function $f_\sigma(\varphi)$

The shape parameter σ selects one of three functions $f_{box}(\varphi)$, $f_{circle}(\varphi)$, $f_{rhomb}(\varphi)$, respectively a superposition of two of them. The shape parameter $\sigma = 0$ selects the function $f_{circle}(\varphi)$ which is the sine function $\sin(\varphi)$ as used for sine cosine microstepping. With this, one gets the unit circle (r=1.0) by transformation to Cartesian coordinates {$y = \sin(\varphi)$; $x = \cos(\varphi)$} as outlined in Figure 13-1, a shape parameter σ = +1.0 results in a box, and a shape parameter σ = -1.0 results in a rhomb. Other values except those, result in something between box and circle respectively something between circle an rhomb.

The data values y(i) of the look-up table range from 0 to 63 and the argument I ranges also from 0 to 63. In the following, natural angles (radians) ranging from ($0 \le \varphi < 2\pi$) are used for the description. The three functions for superposition controlled by the shape parameter σ are

$$f_{box}(\varphi) = \begin{cases} \frac{4}{\pi \cdot \sqrt{2}} \cdot \varphi & if \quad 0 \le \varphi < \frac{\pi}{4} \\ \frac{1}{\sqrt{2}} & if \quad \varphi \ge \frac{\pi}{4} \end{cases}$$

$$f_{circle}(\varphi) = \sin(\varphi)$$

$$f_{rhomb}(\varphi) = \frac{2}{\pi} \cdot \varphi$$

All together, these three functions are combined to form the function

$$f_{\sigma}(\varphi) = \begin{cases} f_{circle}(\varphi) + \sigma \cdot [f_{box}(\varphi) - f_{circle}(\varphi)] & for \quad \sigma > 0 \\ f_{circle}(\varphi) & for \quad \sigma = 0 \\ f_{circle}(\varphi) + \sigma \cdot [f_{circle}(\varphi) - f_{rhomb}(\varphi)] & for \quad \sigma < 0 \end{cases}$$

So, the shape parameter σ selects the type of function and it also provides a continuous transition between circle and box respectively circle and rhomb. To estimate, what function would be best for a given type of stepper motor, one can try microstepping based on different shape parameters σ by downloading different micro step tables on-the-fly into the TMC429 during motion of a stepper motor. For calculation of data for the micro step look-up table of the TMC429, one has to replace $\varphi \rightarrow \varphi_i$ *ranging from 0 to* $\pi/2$ for the quarter period by

$$\varphi_i = \frac{\pi}{2} \cdot \frac{i}{64} \quad with \quad i = \{0,1,2,3,...,63\}.$$

The amplitude of the shape function $f_{\sigma}(\varphi_i)$ has to be limited to the range of 0.0 to 1.0 respectively to the range of 0 to 63 for the on-chip RAM as described in the beginning of the microstepping section.

14 How to get Started in Running a Motor

First of all, the Stepper Motor Driver Datagram Configuration has to be written into its RAM area. Additionally, the Microstep Look-Up-Table has to be initialized when using microstepping. The parameter **LSMD**, that is part of the global parameter register, has to be initialized. After that, the parameters **v_min**, **v_max**, and the clock pre-dividers **pulse_div** and **ramp_div** and the micro step resolution **usrs** has to be set. Then, **a_max** together with a valid pair of **pmul** and **pdiv** has to be set. The switch configuration **ref_conf** together with the ramp mode **rm** has to be chosen. The reference switch inputs **REF1**, **REF2**, **REF3** should be pulled down to ground or disabled by setting **ref_conf**. With those settings, the TMC429 runs a motor if one writes either **x_target** or **v_target**, depending on the choice of the ramp mode **rm**. An application note named “TMC428 – Getting Started” together with C source code examples are available on www.trinamic.com

15 How to Run a Motor with Start-Stop-Speed in RAMP_MODE

Either the TMC428 or the TMC429 has an integrated automatic start-stop-speed mechanism because this can easily realized by a simple command sequence. To start and stop with with a speed V_START_STOP different from zero, one has to do the follwoing:

1. set VMIN := V_START_STOP
2. set RAMP_MODE
3. set X_TARGET to desired target position
4. set V_ACTUAL immedeately with correct sign for V_ACTUAL to +V_MIN resp. –V_MIN, depending on the direction of positioning

16 Package Outlines and Dimensions

16.1 Shrink Small Outline Package with 16 Pins (SSOP16, 150 MIL, JEDEC drawing MO-137 (150 mils)) of TMC429

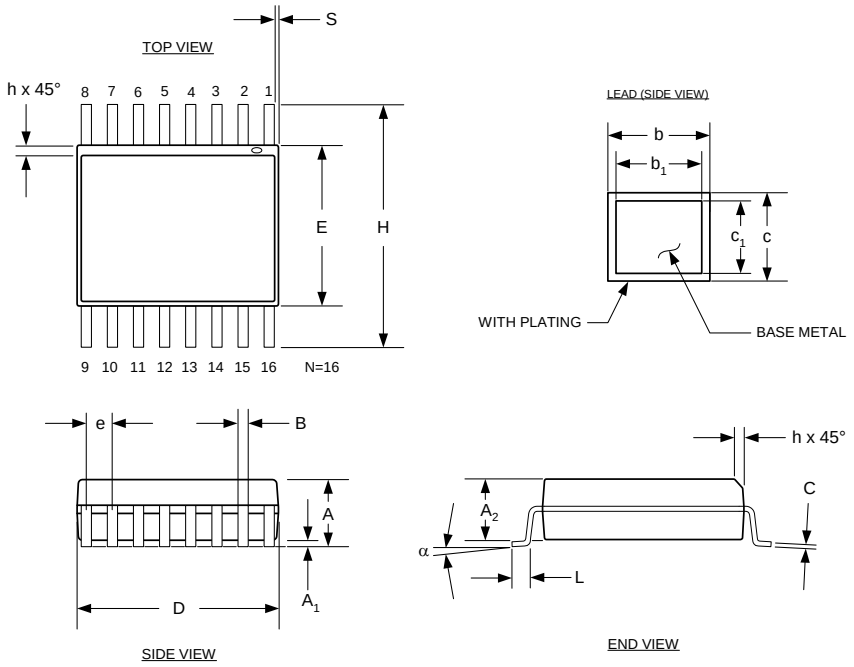

Figure 16-1: Package Outline Drawing SSOP16, 150 MILS, 0.635mm (*0.025 inch*) pitch

Symbol	Dimensions in MILLIMETERS			*Dimensions in INCHES*		
	Min	**Typ**	**Max**	***Min***	***Typ***	***Max***
A	1.55	1.63	1.73	*0.061*	*0.064*	*0.068*
A1	0.10	0.15	0.25	*0.004*	*0.006*	*0.0098*
A2	1.40	1.47	1.55	*0.055*	*0.058*	*0.061*
b	0.20		0.30	*0.008*		*0.012*
b1	0.20	0.25	0.28	*0.008*	*0.010*	*0.011*
c	0.18		0.25	*0.007*		*0.010*
c1	0.18	0.20	0.23	*0.007*	*0.008*	*0.009*
B	0.20	0.25	0.31	*0.008*	*0.010*	*0.012*
C	0.19	0.20	0.25	*0.0075*	*0.008*	*0.0098*
D	4.80	4.93	4.98	*0.189*	*0.194*	*0.196*
E	3.91 BSC			*0.154 BSC*		
e	0.635 BSC			*0.025 BSC*		
H	6.02 BSC			*0.237 BSC*		
h	0.25	0.33	0.41	*0.010*	*0.013*	*0.016*
L	0.41	0.635	0.89	*0.016*	*0.025*	*0.035*
N	**16**			**16**		
S	0.051	0.114	0.178	*0.0020*	*0.0045*	*0.0070*
α	0°	5°	8°	0°	5°	8°

Table 16-1: Dimensions of Package SSOP16, 150 MILS (Note: BSC ≈ Best Case)

16.2 Small Outline Package with 24 Pins (SOP24) of TMC429-PI24, JEDEC drawing MS-013 (300 mils)

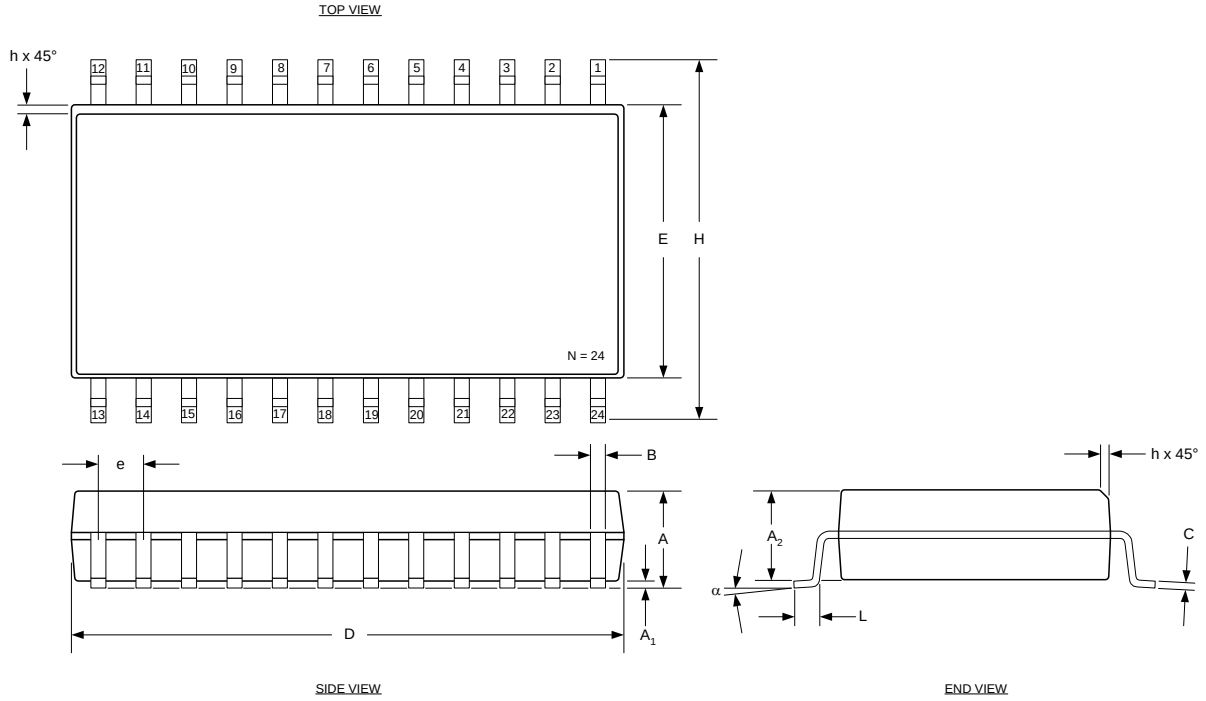

Figure 16-2: Package Outline Drawing SOP24, 300 MILS

Symbol	Dimensions in MILLIMETERS			*Dimensions in INCHES*		
	Min	**Typ**	**Max**	***Min***	***Typ***	***Max***
A	2.35		2.65	*0.0926*		*0.1043*
A1	0.1		0.3	*0.004*		*0.0118*
A2						
B	0.33		0.51	*0.013*		*0.02*
C	0.23		0.32	*0.0091*		*0.0125*
D	15.2		15.6	*0.5985*		*0.6141*
E	7.4		7.6	*0.2914*		*0.2992*
e	1.27 BSC			*0.05 BSC*		
H	10		10.65	*0.394*		*0.419*
h	0.25		0.75	*0.01*		*0.029*
L	0.4		1.27	*0.016*		*0.05*
N	**24**			**24**		
α	0°		8°	0°		8°

Table 16-2: Dimensions of Package SOP24, 300 MILS (Note: BSC ≈ Best Case)

16.3 QFN Package with 32 Pins (QFN32) of TMC429-LI

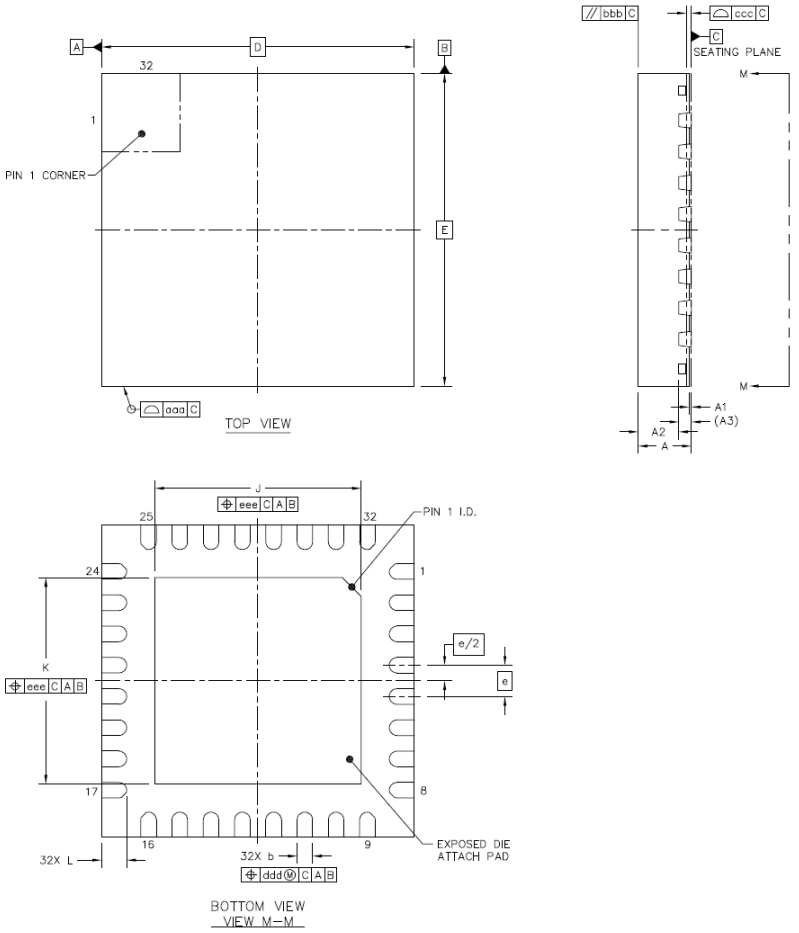

Figure 16-3: Package Outline Drawing QFN32

Parameter	Symbol	Dimensions in MILLIMETERS		
		Min	Typ	Max
total thickness	**A**	0.80	0.85	0.90
stand off	**A1**	0.00	0.035	0.05
mold thickness	**A2**	-	0.65	0.67
lead frame thickness	**A3**		0.203	
lead width	**b**	0.2	0.25	0.3
body size X	**D**		5.0	
body size Y	**E**		5.0	
lead pitch	**e**		0.5	
exposed die pad size X	**J**	3.2	3.3	3.4
exposed die pad size Y	**K**	3.2	3.3	3.4
lead length	**L**	0.35	0.4	0.45
package edge tolerance	**aaa**			0.1
mold flatness	**bbb**			0.1
co-planarity	**ccc**			0.08
lead offset	**ddd**			0.1
exposed pad offset	**eee**			0.1

Table 16-3: Dimensions of 5x5mm Package QFN32

17 Marking

Product Name	**TMC429-I**
Package	SSOP16 – 150 MILS
Date Code	WWYY (week WW and year YY)
Lot Number Identifier	LLLL
Logo	No
Zoomed Size	

Product Name	**TMC429-LI**
Package	QFN32 5mm * 5mm
Date Code	WWYY (week WW and year YY)
Lot Number Identifier	LLLL
Logo	Yes
Zoomed Size	

Product Name	**TMC429-PI24**
Package	SOP24 – 300 MILS
Date Code	WWYY (week WW and year YY)
Lot Number Identifier	LLLL
Logo	Yes
Zoomed Size	

18 On-Chip Voltage Regulator

The on-chip voltage regulator delivers a 3.3V supply for the chip core. An external 470nF ceramic capacitor has to be connected between the V33 pin (see Figure 18-1, page 56) and ground, with connections as short as possible. Additionally, an external 100nF ceramic capacitor (CBLOCK) has to be connected between pin V5 and ground– with connections as short as possible –in 5V operational mode. In 3.3V operational mode an external 100nF ceramic capacitor (see Figure 18-1, page 56) is necessary only between pin V33 and ground, with connections as short as possible.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
TRANGEREG	Temperature range	Industrial	-40		85	°C
VDD5REG	Supply voltage vdd5	5 V Operational Mode	4.5	5	5.5	V
CBLOCK	Block capacitor	5 V Operational Mode, x7r ceramic capacitor		100		nF
VDD3REG	Supply voltage vdd3	3.3 V Operational Mode	2.9	3.3	3.6	V
ICCNLREG	Current consumption	no load		50	100	µA
tSREG	Startup time	no external capacitor connected			20	µs
tSREGC	Startup time	C_load = 470 nF			150	µs
TDRFT	Temperature drift				300	ppm / °C
VRIPPLE	Ripple on vdd3	With ripple over 50 mV the input thresholds may differ from that specified in the data sheet			100	mV
CREG	External capacitor	On V33 pin, x7r ceramic, necessary capacity depending on ripple requirements. Using external capacitor with capacity other than typical, the ripple should be measured on pin v33 to be sure that requirements are satisfied.	33	470		nF
COPT	Optional capacitor	Optional parallel capacitor for additional reduction of high frequency ripple, c0g ceramic, unnecessary in most cases		470		pF
PSRRDC	power supply ripple rejection	DC		50		dB

Table 18-1: Characteristics of the on-chip voltage regulator

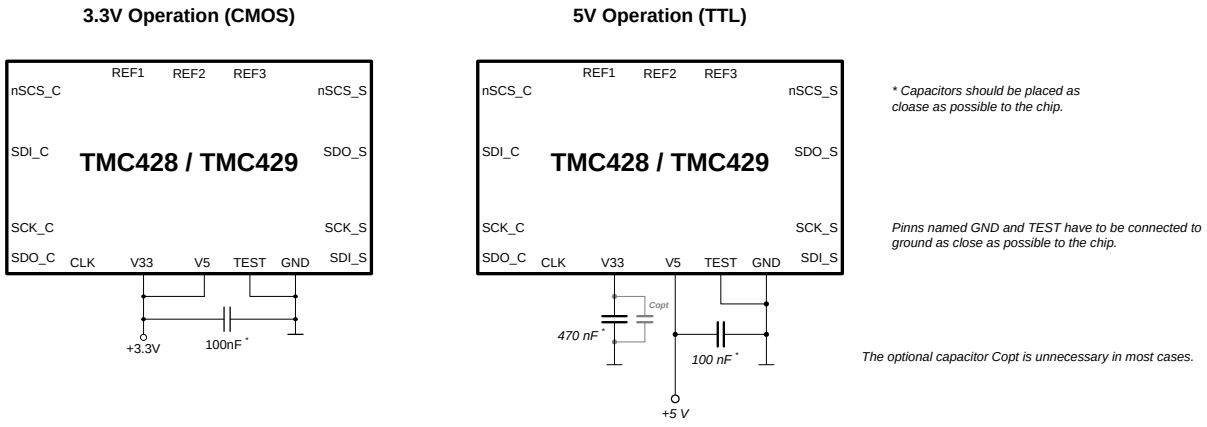

Figure 18-1: 3.3V operation (CMOS) vs. 5V operation (TTL)

19 Power-On-Reset

The TMC429 is equipped with a static and dynamic reset with internal hysteresis (see Figure 19-1). So, it performs an automatic reset during power-on. If the power supply voltage goes below a threshold, an automatic power on reset is performed also. The power on reset time tRESPOR also depends on the power up time of the on-chip voltage regulator (see Table 18-1).

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDD	Power supply range		3.0	3.3	3.6	V
Temp	Temperature		-55	25	125	°C
Vop	Reset on/off				0.80	V
Voff	Reset off		1.58	2.13	2.85	V
Von	Reset on		1.49	1.98	2.70	V
tRESPOR	Reset time of on-chip power-on-reset		2.14	3.31	5.52	µs

Table 19-1: Characteristics of the on-chip power-on-reset

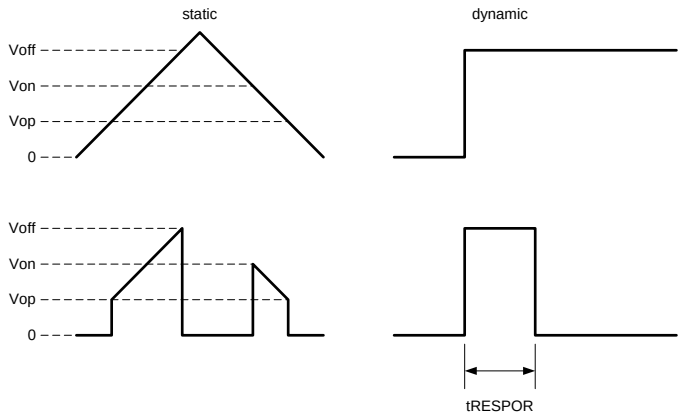

Figure 19-1: Operating principle of the power-on-reset

20 Characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
VDD3	DC Supply Voltage	Voltage at Pin V33 in 3.3V mode	-0.3	3.6	V
VI3	DC Input Voltage, 3.3 V I/Os		-0.3	VDD3 + 0.3	V
VO3	DC Output Voltage, 3.3 V I/Os		-0.3	VDD3 + 0.3	V
VDD5	DC Supply Voltage	Voltage at Pin V5	-0.3	5.5	V
VI5	DC Input Voltage, 5V I/Os	Continuous DC Voltage	-0.3	VDD5 + 0.3, 5.5 max	V
VO5	DC Output Voltage, 5V I/Os	Continuous DC Voltage	-0.3	VDD5 + 0.3, 5.5 max	V
VESD	ESD Voltage	PAD cells are designed to resist ESD voltages according to Human Body Model according to MIL-STD-883, with R_C = 1 – 10 MΩ, R_D = 1.5 KΩ, and C_S = 100 pF, but it cannot be guaranteed.		±2000	V
TEMP_D2	Ambient Air Temperature Range	Industrial / Consumer type	-40	+85	°C
TEMP_D3	Ambient Air Temperature Range	Automotive type	-55	+125	°C
TEMP_D4	Ambient Air Temperature Range	Industrial type	-40	+105	°C
TSG	Storage Temperature		-60	+150	°C

Table 20-1 : Absolute maximum ratings

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ILC	Input Leakage Current				1	µA
CIN	Input Capacitance			7		pF

Table 20-2: DC characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDD3	DC Supply Voltage		3.0	3.3	3.6	V
VI3	DC Input Voltage		0		VDD3	V
VIL3	Low Level Input Voltage	Pin TEST only	0		0.3 x VDD3	V
VIH3	High Level Input Voltage	Pin TEST only	0.7 x VDD3		VDD3 + 0.3	V
VLTH3	Low Level Input Voltage Threshold	All Inputs except TEST	0.9		1.2	V
VHTH3	High Level Input Voltage Threshold	All Inputs except TEST	1.5		1.9	V
VHYS3	Schmitt-Trigger Hysteresis		0.4		0.7	V
VOL3	Low Level Output Voltage	IOL = 0.3 mA			0.1	V
VOH3	High Level Output Voltage	IOH = 0.3 mA	VDD3 – 0.1			V
VOL3	Low Level Output Voltage	IOL = 2 mA			0.4	
VOH3	High Level Output Voltage	IOH = 2 mA	VDD3 – 0.4			V

Table 20-3: DC characteristics – 3.3V supply mode

Note: Ripple on VDD3 has to be taken into account concerning measurement of thresholds and hysteresis.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDD5	DC Supply Voltage		4.5	5	5.5	V
VI5	DC Input Voltage		0		VDD5	V
VIL5	Low Level Input Voltage	Pin TEST only	0		0.3 x VDD5	V
VIH5	High Level Input Voltage	Pin TEST only	0.7 x VDD5		VDD5 + 0.3	V
VLTH5	Low Level Input Voltage Threshold	All Inputs except TEST, VDD5=5V	0.9		1.2	V
VHTH5	High Level Input Voltage Threshold	All Inputs except TEST, VDD5=5V	1.5		1.9	V
VHYS5	Schmitt-Trigger Hysteresis		0.4		0.7	V
VOL5	Low Level Output Voltage	IOL = 0.3 mA			0.1	V
VOH5	High Level Output Voltage	IOH = 0.3 mA	VDD5 – 0.1			V
VOL5	Low Level Output Voltage	IOL = 4 mA			0.4	
VOH5	High Level Output Voltage	IOH = 4 mA	VDD5 – 0.4			V

Table 20-4: DC characteristics – 5V supply mode

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ISC32MHZ	Supply Current	f = 32 MHz at Tc=25°C		15		mA
ISC16MHZ	Supply Current	f = 16 MHz at Tc=25°C		5	10	mA
ISC8MHZ429	Supply Current	f = 8 MHz at Tc=25°C (IOs driven)		5		mA
ISC4MHZ	Supply Current	f = 4 MHz at Tc=25°C		1.25	2.5	mA
IPDN25C	Power Down Current	Power Down Mode at Tc=25°C, 5V Supply		70	150	µA

Table 20-5: Power dissipation

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
fCLK	Operation Frequency	fCLK = 1 / tCLK	0	16	32	MHz
tCLK	Clock Period	Raising Edge to Raising Edge of CLK	31.25		∞	ns
tCLK_L	Clock Time Low		12.5		∞	ns
tCLK_H	Clock Time High		12.5		∞	ns
tRISE_I	Input Signal Rise Time	10% to 90% except TEST pin	0.5		∞	ns
tFALL_I	Input Signal Fall Time	90% to 10% except TEST pin	0.5		∞	ns
tRISE_O_428	Output Signal Rise Time	10% to 90%		3		
tFALL_O_428	Output Signal Fall Time	90% to 10%		3		
tRISE_O_429	Output Signal Rise Time	10% to 90%		7		ns
tFALL_O_429	Output Signal Fall Time	90% to 10%		7		ns
tSU	Setup Time	relative to falling clock edge at CLK	1			ns
tHD	Hold Time	relative to falling clock edge at CLK	1			ns
tPD_428	Propagation Delay Time	50% of rising edge of the clock CLK to the 50% of the output	1	5		ns
tPD_429	Propagation Delay Time	50% of rising edge of the clock CLK to the 50% of the output	1	10		ns

Table 20-6: General timing parameters (TMC429 with EMI optimized output drivers)

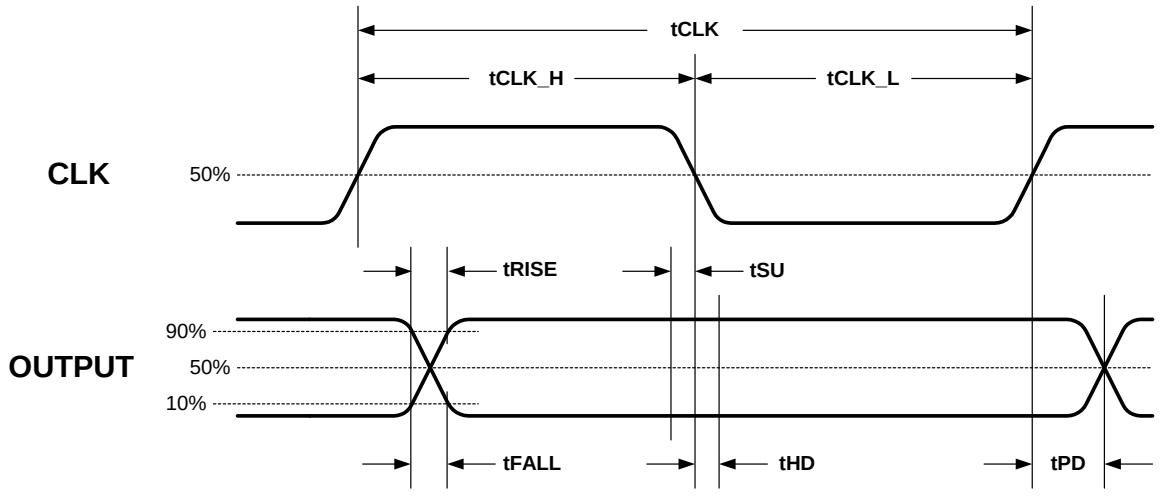

Figure 20-1: General timing parameters

21 Example for Calculation of p_mul and p_div for the TMC429

```
/* PROGRAM EXAMPLE 'pmulpdiv.c' : How to Calculate p_mul & p_div for the TMC429 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void CalcPMulPDiv(int a_max, int ramp_div, int pulse_div, float p_reduction,
                  int *p_mul, int *p_div, double *PIdeal, double *PBest, double *PRedu )
{
  int    pdiv, pmul, pm, pd ;
  double p_ideal, p_best, p, p_reduced;

  pm=-1; pd=-1; // -1 indicates : no valid pair found
  p_ideal   = a_max /  (pow(2, ramp_div-pulse_div)*128.0);
  p         = a_max / ( 128.0 * pow(2, ramp_div-pulse_div) );
  p_reduced = p * ( 1.0 - p_reduction );

  for (pdiv=0; pdiv<=13; pdiv++)
  {
       pmul = (int)(p_reduced * 8.0 * pow(2, pdiv)) - 128;

            if ( (0 <= pmul) && (pmul <= 127) )
            {
                pm = pmul + 128;
                pd = pdiv;
            }
  }

  *p_mul = pm;
  *p_div = pd;

  p_best = ((double)(pm)) / ((double)pow(2,pd+3));

  *PIdeal = p_ideal;
  *PBest  = p_best;
  *PRedu  = p_reduced;
}

int main(int argc, char **argv)
{
    int  a_max=0, ramp_div=0, pulse_div=0, p_mul, p_div,
                  a_max_lower_limit=0, a_max_upper_limit=0;
         double pideal, pbest, predu;
         float  p_reduction=0.0;

    char **argp;

    if (argc>1)
    {
      while (argv++, argc--)
      {
        argp = argv + 1;   if (*argp==NULL) break;

        if (      (!strcmp(*argv,"-a")) ) sscanf(*argp,"%d",&a_max);
        else if ( (!strcmp(*argv,"-r")) ) sscanf(*argp,"%d",&ramp_div);
        else if ( (!strcmp(*argv,"-p")) ) sscanf(*argp,"%d",&pulse_div);
        else if ( (!strcmp(*argv,"-pr"))) sscanf(*argp,"%f",&p_reduction);
      }
    }
    else
    {
      fprintf(stderr,"\n  USAGE : pmulpdiv -a <a_max> -r <ramp_div> -p <pulse_div> -pr <0.00 .. 0.10>\n"
                     "  EXAMPLE : pmulpdiv -a 10 -r 3 -p 3 -pr 0.05\n");
      return 1;
    }

    printf("\n\n  a_max=%d\tramp_div=%d\tpulse_div=%d\tp_reduction=%f\n\n",
                          a_max, ramp_div, pulse_div, p_reduction);

    CalcPMulPDiv(a_max, ramp_div, pulse_div, p_reduction, &p_mul, &p_div, &pideal, &pbest, &predu );

         printf("  p_mul = %3.3d\n  p_div = %3d\n\n p_ideal = %f\n  p_best = %f\n  p_redu = %f\n\n",
               p_mul, p_div, pideal, pbest, predu);

    a_max_lower_limit = (int)pow(2,(ramp_div-pulse_div-1));
    printf("\n a_max_lower_limit = %d",a_max_lower_limit);
    if (a_max < a_max_lower_limit) printf("   [WARNING: a_max < a_max_lower_limit]");
    a_max_upper_limit = ((int)pow(2,(12+(ramp_div-pulse_div)))) -1;
    printf("\n a_max_upper_limit = %d",a_max_upper_limit);
    if (a_max > a_max_upper_limit) printf("   [WARNING: a_max > a_max_upper_limit]");
    printf("\n\n");

    return 0;
}
/* ---------------------------------------------------------------------------- */
```

22 Revision History

Version	Date (Author)	Comment
1.00	December 08, 2009 (LL)	TMC429 Datasheet v. 1.00 based on TMC428 Datasheet v. 2.03 / December 18, 2009; section 9.11.1 Optimized Calculation of p_mul and p_div corrected (p. 25) ~~p = a_max / (128 * 2^(pulse_div – ramp_div))~~ p = a_max / (128 * 2^(ramp_div - pulse_div));
	May 27, 2010 (LL)	QFN32 package pinning added (table Table 5-1: TMC429 , page 9); Hint added concerning changing of pluse_div while VELOCITY_MODE or HOLD_MODE that might force am internal step pulse depending on the current position (section 9.14 pulse_div & ramp_div & usrs (IDX=%1100), page 29); DIL20 package of TMC428 removed;
	August 5, 2010 (LL)	pin out GND @ 25 and n.c. @ 32 added for QFN32 package; pinning table updated with TMC428 SOP24 package variant; hints added for new functions of the TMC429; section 9.17 Unused Address (IDX=%1111) – TMC428 vs. TMC429, p. 30, section 9.17.1ustep_count_429 (IDX=%1111) modified / added because this register was unused for the TMC428 but is a read / write register for the TMC429;
	August 26, 2010 (LL)	SDO_C renamed to iINT_SDC_C at package drawings; SPI contrograration outline added;
	September 10, 2010 (LL)	Figure 7-2 : TMC428 SDO_C output high impedance with single gate 74HCT1G125 vs. The TMC429 has a dedicated high impedance pin SDOZ_C available and the nINT_SDO_C can be nINT.12 updated concernit SDOZ_C and nINT_SDO_C of TMC429; description of if_Configuration_429 finalized; step/direction timing figure and description added (section 10.5.1 TMC429 Step/Direction Timning, page 34); short description of differences between TMC428 vs. TMC429; section 15 How to Run a Motor with Start-Stop-Speed in RAMP_MODE, page , 51;
1.01	October 10, 2010 (LL)	Marking (section 17, p. 55) updated; Timing parameter updated (Table 20-6, p. 59)
1.02	November 1st, 2010 (LL)	Pin name Figure 5-1: TMC429 pin out, page 8 of SSOP16 (pin 10) package variant corrected according to Table 5-1: TMC429 pinning (TMC428 SOP24: pin 1 ⇔ n.c. / TMC429 SOP24: pin 1 ⇔ output) page 9
1.03	November 5, 2010 (LL)	TMC428 drawings (Figure 7-2, Figure 10-4, Figure 10-5, Figure 10-6, Figure 10-7, Figure 10-10, Figure 18-1) updated concerning TMC428 / TMC429; feature list updated concerning step direction interface;
1.04	November 17, 2010 (LL)	section 10.5.1 TMC429 Step/Direction Timning, page 34 equation tSTEP[µs] corrected
1.05	December 6, 2010 (LL) April 6th, 2011 (LL) May 17, 2011 (LL) August 2nd, 2011 (LL) December 15, 2011 (LL) January 19, 2012 (LL)	Figure 3-3 & Figure 3-4 updated concerning TMC429; position compare control register (pos_comp) and associated output poscmp labelling unified; TMC428 changed to TMC429 in text if necessary; preliminary removed and version date updated; no changes within the document itself; hint concerning micro step frequency added in section 10.5.1, p. 34; section 9.6 v_actual (IDX=%0101) statement removed to avoid miss understanding: ~~For stop detection there is a dedicated bit within the interrupt register, which can simply be read out by the micro processor or generate an interrupt.~~ There is not an interrupt bit that indicates that the internal velocity register is equal to zero – there are different interrupt bits that indicate different stop conditions triggered by stop switches; Post address updated; hint concerning current scaling at rest added in section 9.9 is_agtat & is_aleat & is_v0 & a_threshold (IDX=%1000), page 21; Table 20-5: Power dissipation typical current ISC32MHZ added, and symbol ISC4MHZ429 corrected/changed to ISC8MHZ429; sections 6, 8.3, 12.1, 12.2 updated concerning power-on default initialization of the SPI driver chain for TMC236/TMC239/TMC246/TMC249 SPI stepper motor chain; TMC429 power-on reset (POR) default values added to Table 12-2: Datagram example and RAM contents for three stepper motor driver chain; hint concerning power-on defaults of sine wave table added in section 13;
1.06	January 27, 2012 (LL) March 28, 2012 (LL) May 22, 2012 (LL)	section 15 How to Run a Motor with Start-Stop-Speed in RAMP_MODE item set X_TARGET ~~:= V_START_STOP corrected~~ to set X_TARGET to desired position; section 12.1 Initialization of on-chip-RAM by µC after power-on updated; Hint concerning frequency of SPI clock SCK relative to TMC429 clock fCLK added in section 7.3, page 13; Figure 10-3: TMC429 Step Direction Timing (EN_SD='1' & STEP_HALF='0'), p. 34 updated concerning dimensioning drawing at rising edge of direction signal (signal shape itself was correct);
1.07	August 01, 2012 (SD)	Section 10.5 Table 10-2 corrected: smda 0100.

23 Table of Figures

24 Table of Tables

25 Table of Contents

Please refer to www.trinamic.com for updated data sheets and application notes on this product and on other products.