TMC4361

Cost-effective S-ramp motion controller for stepper motors - optimized for high velocities. With SPI and Step/Dir interfaces to motor driver and encoder interface for closed loop operation

Servo Drive

Applications

Textile, Sewing Machines
Factory Automation
Lab Automation
Medical
Office Automation
Printer and Scanner
CCTV, Security
ATM, Cash recycler
POS
Pumps and Valves
Heliostat Controller
CNC Machines

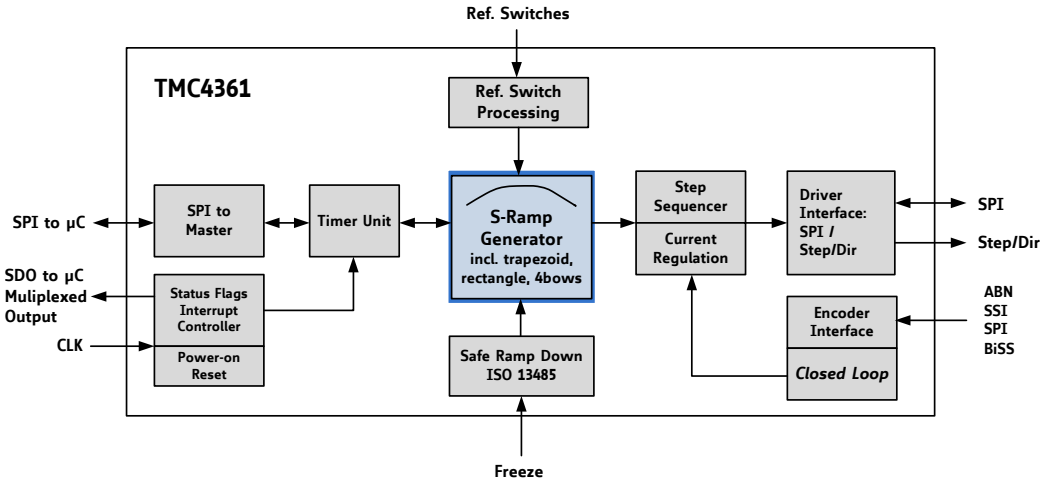
Features and Benefits

3.3V or 5V operation
SPI interface for µC with easy-to-use protocol
SPI interface for SPI motor drivers
Step/Dir interface Step/Dir motor drivers
Encoder interface: incremental ABN and serial SSI/SPI/BiSS
2x ref.-switch input
Clock frequency up to 30 MHz
Different current levels related to the motion profile status
Low power operation using clock gating technology
S-shaped velocity ramps, optimally calculated
Linear ramps with trapezoid and rectangle shapes
Programmable microstep table
On-the-fly change of target motion parameters
Read-out option for all important motion parameters
Compact Size 6x6 mm² QFN40 package
Directly controls TMC23x, TMC24x, and TMC26x motor driver

Description

The TMC4361 is intended for applications where a fast and jerk-limited motion profile is desired. This motion controller adds to any microcontroller with SPI interface. It supports S-shaped, trapezoid, and rectangle ramps. With encoder, the TMC4361 allows for an extremely quick and precise positioning. Its servo features include no step-loss, energy efficiency, and target positioning with oscillation-free algorithms. Standard SPI and STEP/DIR interfaces to the motor driver simplify communication.
High end features without software effort and small form factor enable miniaturized designs with low external component count for cost-effective and highly competitive solutions.

Block Diagram



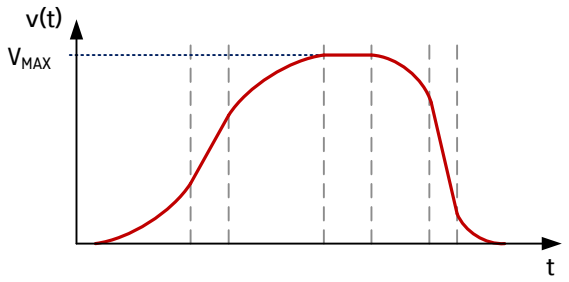
HIGH-END SOLUTION: VELOCITY MEETS PRECISION

The TMC4361 is a miniaturized high performance stepper motor controller with an outstanding cost-performance ratio. It is designed for high volume as well as for demanding industrial motion control applications. The TMC4361 motion controller is equipped with an SPI™ host interface (SPI is trademark of Motorola) with easy-to-use protocol and two driver interfaces (SPI and Step/Dir) for addressing various stepper motor driver types. The TMC4361 scores with its unique servo drive features, high integration and a versatility that covers a wide spectrum of applications, motor sizes, and encoder types.

For a comfortable handling, the chip provides the possibility to work with real world units. Extensive support at the chip, board, and software levels enables rapid design cycles and fast time-to-market with competitive products. High energy efficiency delivers further cost savings.

S-SHAPED VELOCITY PROFILE

This outstanding ramp profile is completely jerk-free. Seven segments of the ramp allow for an optimum adaption of the velocity profile to the customer specific application requirements. High torque with high velocities can be reached by calibrating the bows of the ramp in a way that the acceleration value near V_{MAX} is reduced in parallel to the available motor torque.

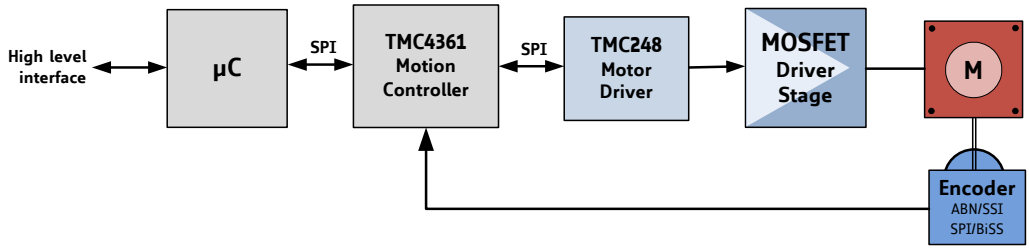


S-shaped ramp profile

COMPACT DESIGN FOR RELIABLE CLOSED LOOP OPERATION

BENEFIT FROM HIGH VELOCITIES COMBINED WITH EXTREMELY HIGH PRECISION!

Closed loop operation is an optimum choice in case a dynamic and reliable drive without step-loss and motor stall is desired. The controller IC monitors the encoder values nonstop and uses them for a sophisticated motor field control.



In case internal MOSFETs are desired, combine the TMC4361 with the TMC260, the TMC261 or the TMC2660.

ORDER CODES

Order code	Description	Size
TMC4361-LA	Motion controller with servo and dcStep features, QFN40	6 x 6 mm²

Table of Contents

1 Principles of Operation

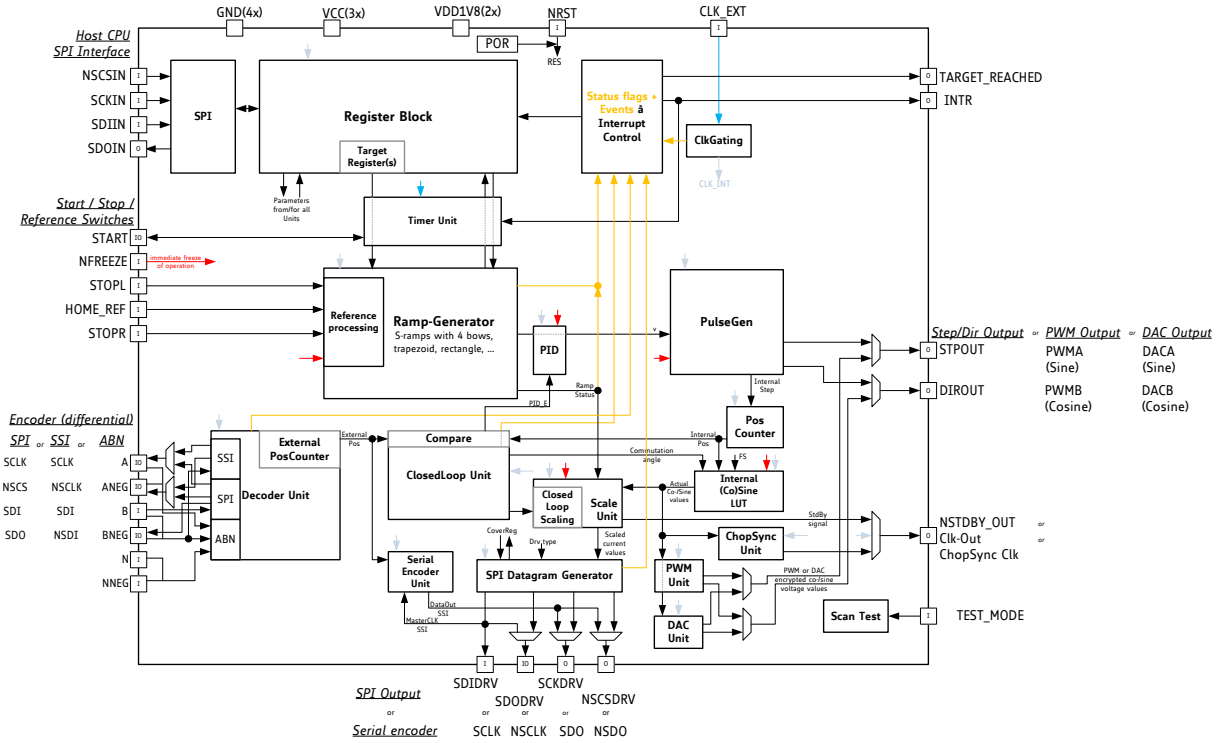


Figure 1.1 Basic application block diagram

1.1 Key Concepts

The TMC4361 realizes real time critical tasks autonomously and guarantees for a robust and reliable drive. The following features contribute toward greater precision, greater efficiency, higher reliability, higher velocity, and smoother motion in many stepper motor applications.

Interfacing The TMC4361 offers application specific interfacing via Step/Dir or SPI.

Initialization Adapt the TMC4361 to the driver type and configuration and send initial configuration data to SPI drivers. Configure microstep resolution and waveform for SPI drivers.

Positioning The TMC4361 operates motor based on user specified target positions and velocities. Modify all motion target parameters on-the-fly during motion.

Microstepping Based on internal position counters the TMC4361 performs up to $\pm 2^{31}$ (micro)steps completely independent from the microcontroller. Microstep resolutions are individually programmable. The range goes from full stepping (1 microstep = 1 full step) and half stepping (2 microsteps per full step) up to 8 bit micro stepping (256 microsteps per full step) for precise positioning and noiseless stepper motor rotation. With Step/Dir drivers any microstep resolution is possible as supported by the driver. The internal microstep table can be adapted to specific motor characteristics to further reduce torque ripple, if desired.

Servo Drive The TMC4361 provides closed loop operation for Step/Dir and SPI drivers. Using a differential or serial encoder, the closed loop unit of the TMC4361 compares the external position counter values with the internal ones and sends signals for correction.

chopSync™ The TMC4361 has an integrated chopSync chopper for very smooth motor movement.

Programming Every parameter can be changed at any time. The uniform access to any TMC4361 register simplifies application programming. A read-back option for nearly all internal registers is available.

Synchronization The TMC4361 provides synchronizing several TMC4361 motion controller chips if it is desired to move several motors simultaneously.

2 Pin Assignments

2.1 Package Outline

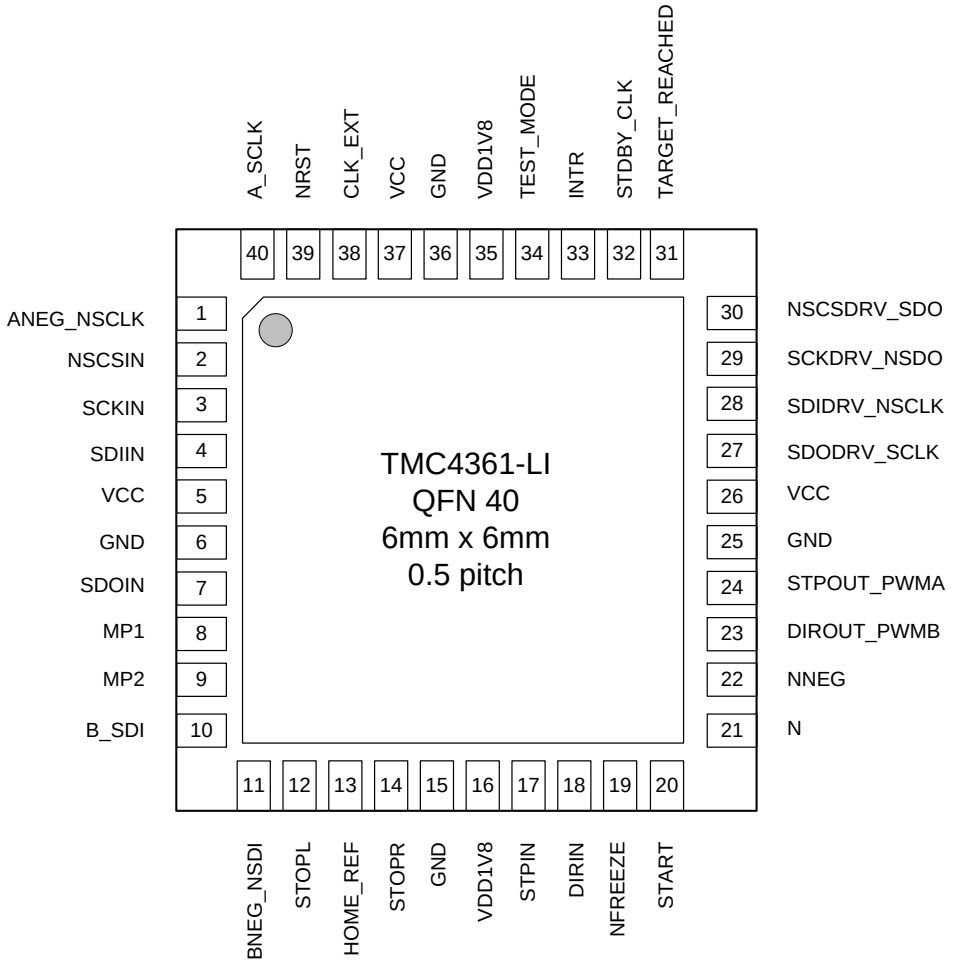


Figure 2.1 Pinning (top view)

2.2 Signal Description

Pin	Number	Type	Function
GND	6, 15, 25, 36	GND	Digital ground pin for IOs and digital circuitry
VCC	5, 26, 37	VCC	Digital power supply for IOs and digital circuitry (3.3V… 5V)
VDD1V8	16, 35	VDD	Connection of *internal generated* core voltage of 1.8V
NSCSIN	2	I	Low active chip select input of the SPI interface to the µC
SCKIN	3	I	Serial clock for the SPI interface to the µC
SDIIN	4	I	Serial data input of the SPI interface to the µC
SDOIN	7	O	Serial data output of the SPI interface to the µC (*Z* if NSCSIN=*1*)
CLK_EXT	38	I	Clock input to provide an clock with the frequency f_{CLK} for all internal operations.
NRST	39	I (PU)	Low active reset. If not connected, Power-on-Reset and internal pull-up resistor will be active.
TEST_MODE	34	I	Test mode input. VCC = 3.3V: Tie to low for normal operation. VCC = 5.0V: Tie to VDD1V8 for normal operation.
INTR	33	O	Interrupt output, programmable PD/PU for wired-and/or
TARGET_REACHED	31	O	Target reached output, programmable PD/PU for wired-and/or
STDBY_CLK	32	O	StandBy signal or internal CLK output or ChopSync output

Pin	Number	Type	Function
STOPL	12	I (PD)	Left stop switch. External signal to stop a ramp. If not connected, an internal pull-down resistor will be active.
HOME_REF	13	I (PD)	Home reference signal input. External signal for reference search. If not connected, an internal pull-down resistor will be active.
STOPR	14	I (PD)	Right stop switch. External signal to stop a ramp. If not connected, an internal pull-down resistor will be active.
STPIN	17	I (PD)	Step input for external step control
DIRIN	18	I (PD)	Direction input for external step control
START	20	IO	Start signal input/output
NFREEZE	19	I (PU)	Low active safety pin to immediately freeze output operations. If not connected, an internal pull-up resistor will be active.
N	21	I (PD)	N signal input of incremental encoder input interface If not connected, an internal pull-down resistor will be active.
NNEG	22	I (PD)	Negated N signal input of incremental encoder input interface If not connected, an internal pull-down resistor will be active.
B SDI	10	I (PD)	B signal input of incremental encoder input interface. Serial data input signal of serial encoder input interface (SSI/SPI). If not connected, an internal pull-down resistor will be active.
BNEG NSDI SDO_ENC	11	IO	Negated B signal input of incremental encoder input interface. Negated serial data input signal of SSI encoder input interface Serial data output of SPI encoder input interface.
A SCLK	40	IO	A signal input of incremental encoder interface. Serial clock output signal of serial encoder interface (SSI/SPI).
ANEG NSCLK NSCS_ENC	1	IO	Negated A signal input of incremental encoder interface. Negated serial clock output signal of serial encoder interface. Low active chip select output of SPI encoder input interface.
STPOUT PWMA DACA	24	O	Step output. First PWM signal (Sine). First DAC output signal (Sine).
DIROUT PWMB DACB	23	O	Direction output. Second PWM signal (Cosine). Second DAC output signal (Cosine).
NSCSDRV PWMB SDO	30	O	Low active chip select output of SPI interface to motor driver. Second PWM signal (Cosine) to connect with PHB (TMC23x/24x). Serial data output of serial encoder output interface.
SCKDRV MDBN NSDO	29	O	Serial clock output of SPI interface to motor driver. MDBN output signal for MDBN pin of TMC23x/24x. Negated serial data output of serial encoder output interface.
SDODRV PWMA SCLK	27	IO	Serial data output of SPI interface to motor driver. First PWM signal (Sine)) to connect with PHA (TMC23x/24x). Clock input of serial encoder output interface.
SDIDRV ERR NSCLK	28	I (PD)	Serial data input of SPI interface to motor driver. Error input signal to with ERR (TMC23x/24x). Negated clock input of serial encoder output interface
MP1	8	I (PD)	Multipurpose pin 1: DC_IN as external dcStep input control signal SYNCHRO_IN as external synchronization input control signal
MP2	9	IO	Multipurpose pin 2: DCSTEP_ENABLE as dcStep output control signal. SYNCHRO_IN as synchronization inout control signal. SPE_OUT as output signal to connect with SPE pin (TMC23x/24x)

PD: if n.c. → pull-down; PU: if n.c. → pull-up

3 Sample Circuits

The sample circuits show the connection of external components.

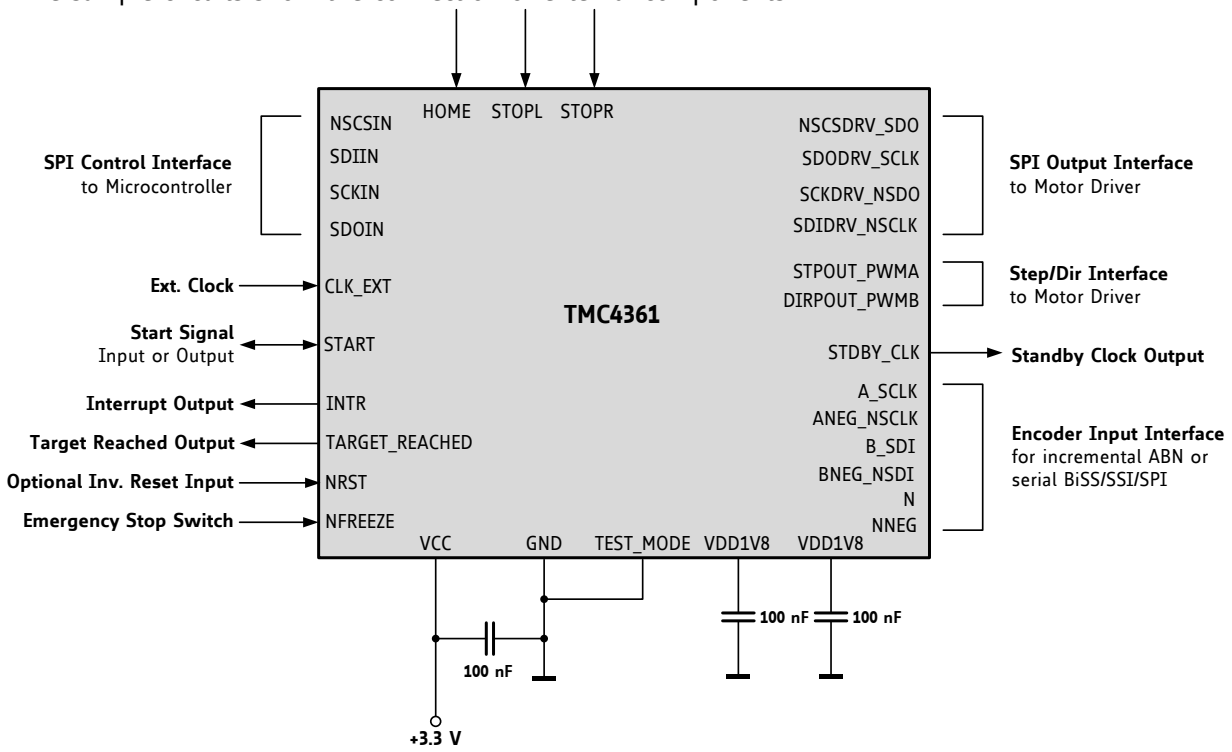


Figure 3.1 How to connect the TMC4361 (VCC = 3.3V)

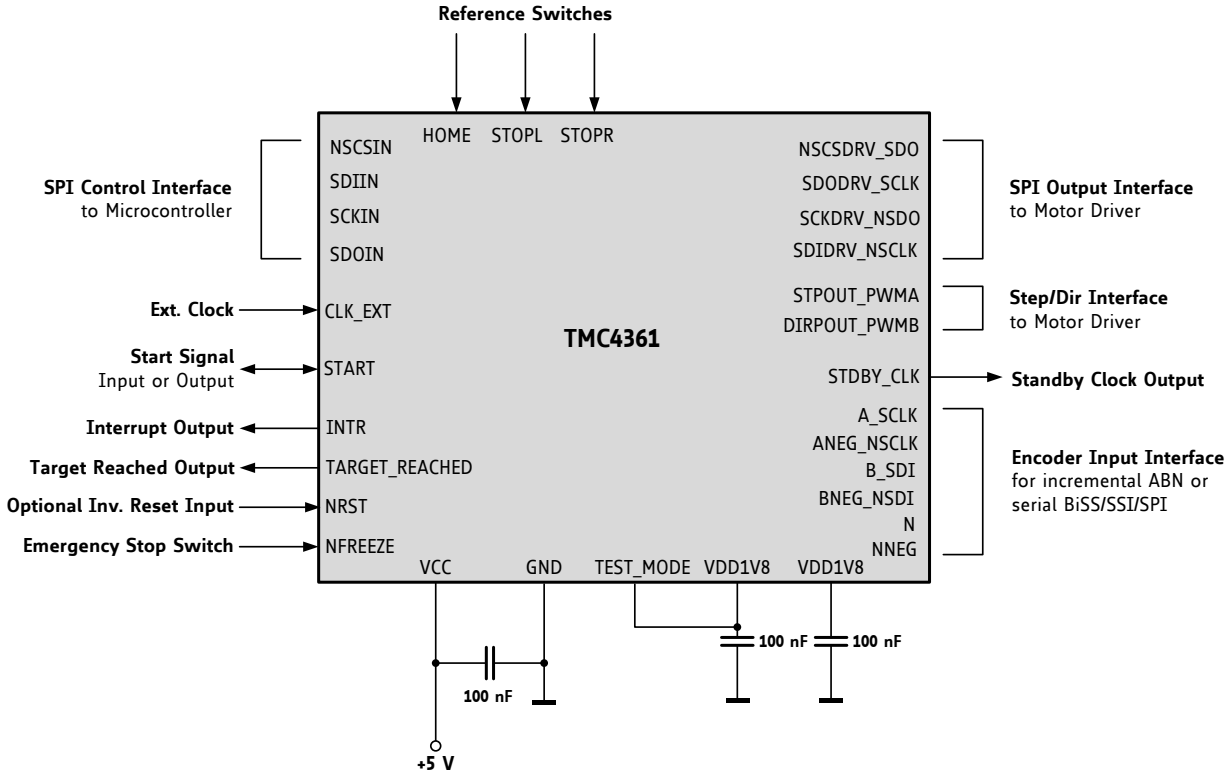


Figure 3.2 How to connect the TMC4361 (VCC = 5V)

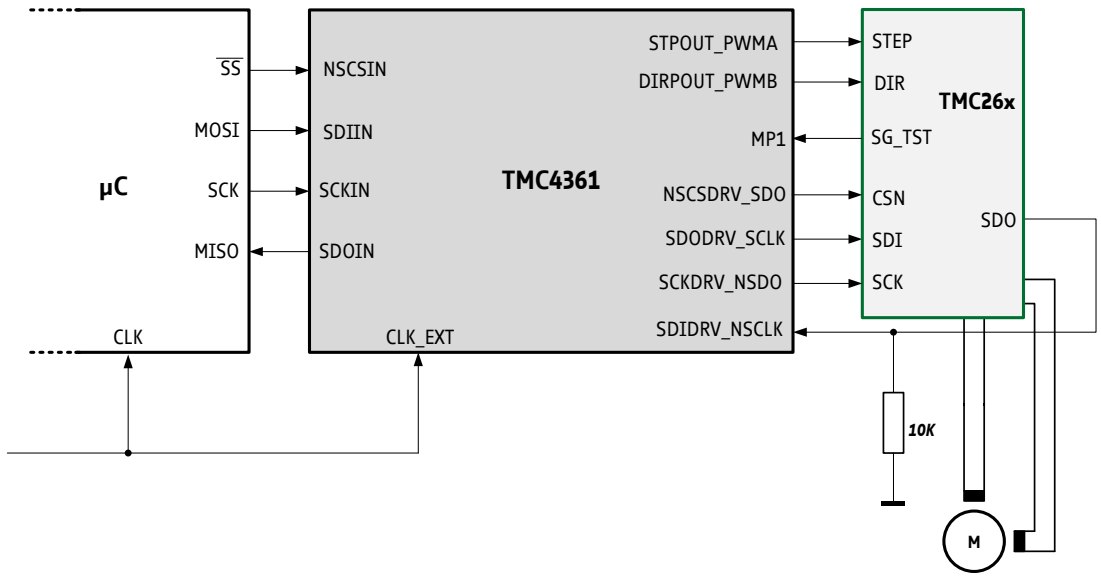


Figure 3.3 TMC4361 with TMC26x stepper driver in SPI mode or S/D mode

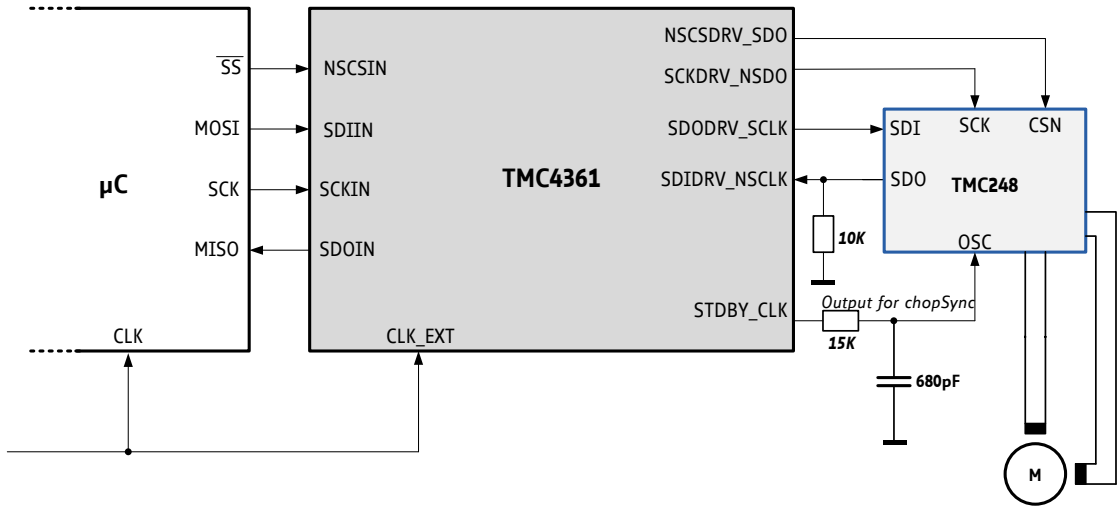


Figure 3.4 TMC4361 with TMC248 stepper driver in SPI mode

Figure 3.5 TMC4361 with TMC21xx stepper driver in SPI mode or S/D mode

4 Notes

REGISTER names are italicized with VALUE REGISTER in capital letters and switches with small letters.

PIN names are written with capital letters.

5 SPI Control Interface

The TMC4361 uses 40 bit SPI™ datagrams for communication with a microcontroller. The bit-serial interface is synchronous to a bus clock. For every bit sent from the bus master to the bus slave, another bit is sent simultaneously from the slave to the master. Communication between an SPI master and the TMC4361 slave always consists of sending one 40-bit command word and receiving one 40-bit status word. The SPI command rate typically comprises a few commands per complete motor motion.

SPI Control interface

Pin Name	Type	Remarks
NSCSIN	Input	Chip Select of the SPI-µC interface (low active)
SCKIN	Input	Clock of the SPI-µC interface
SDIIN	Input	Data input of the SPI-µC interface
SDOIN	Output	Data output of the SPI-µC interface

5.1 SPI Datagram Structure

Microcontrollers which are equipped with hardware SPI are typically able to communicate using integer multiples of 8 bit. The NSCSIN line of the TMC4361 has to be handled in a way, that it stays active (low) for the complete duration of the datagram transmission.

Each datagram sent to the TMC4361 is composed of an address byte followed by four data bytes. This allows direct 32 bit data word communication with the register set of the TMC4361. Each register is accessed via 32 data bits even if it uses less than 32 data bits.

Each register is specified by a one byte address:

- For a read access the most significant bit of the address byte is 0.
- For a write access the most significant bit of the address byte is 1.

Some registers are write only registers, most can be read additionally, and there are also some read only registers.

<table>
<tr><th colspan="40">TMC4361 SPI Datagram Structure</th></tr>
<tr><td colspan="40">MSB (transmitted first) 40 bit LSB (transmitted last)</td></tr>
<tr><td colspan="40">39 … … 0</td></tr>
<tr><td colspan="8">→ 8 bit address
← 8 bit SPI status</td><td colspan="32">← → 32 bit data</td></tr>
<tr><td colspan="8">39 … 32</td><td colspan="32">31 … 0</td></tr>
<tr><td colspan="8">→ to TMC4361: RW + 7 bit address
← from TMC4361: 8 bit SPI status</td><td colspan="8">8 bit data</td><td colspan="8">8 bit data</td><td colspan="8">8 bit data</td><td colspan="8">8 bit data</td></tr>
<tr><td colspan="8">39 / 38 … 32</td><td colspan="8">31 … 24</td><td colspan="8">23 … 16</td><td colspan="8">15 … 8</td><td colspan="8">7 … 0</td></tr>
<tr><td>W</td><td colspan="7">38…32</td><td colspan="4">31…28</td><td colspan="4">27…24</td><td colspan="4">23…20</td><td colspan="4">19…16</td><td colspan="4">15…12</td><td colspan="4">11…8</td><td colspan="4">7…4</td><td colspan="4">3…0</td></tr>
<tr><td>39</td><td>38</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

5.1.1 Selection of Write / Read (WRITE_notREAD)

The read and write selection is controlled by the MSB of the address byte (bit 39 of the SPI datagram). This bit is 0 for read access and 1 for write access. So, the bit named W is a WRITE_notREAD control bit. The active high write bit is the MSB of the address byte. Thus, 0x80 has to be added to the address for a write access. The SPI interface always delivers data back to the master, independent of the W bit. The data transferred back is the data read from the address which was transmitted with the *previous* datagram, if the previous access was a read access. If the previous access was a write access, then the data read back mirrors the previously received write data. So, the difference between a read and a write access is that the read access does not transfer data to the addressed register but it transfers the

address only and its 32 data bits are dummies. Further, the following read or write access delivers back data read from the address transmitted in the preceding read cycle.

ATTENTION A read access request datagram uses dummy write data. Read data is transferred back to the master with the subsequent read or write access. Hence, reading multiple registers can be done in a pipelined fashion. Data which will be delivered are latched immediately after the prior data transfer.

Whenever data is read from or written to the TMC4361, the MSBs delivered back contain the SPI status *SPI_STATUS*, which is a number of eight status bits. The selection of these bits will be explained in chapter 7.2.

Example:

For a read access to the register (*XACTUAL*) with the address 0x21, the address byte has to be set to 0x21 in the access preceding the read access. For a write access to the register (*VACTUAL*), the address byte has to be set to 0x80 + 0x22 = 0xA2. For read access, the data bit might have any value, e.g., 0.

action	data sent to TMC…	data received from TMC…
read *XACTUAL*	→ 0x2100000000	← 0xSS & unused data
read *XACTUAL*	→ 0x2100000000	← 0xSS & X_ACTUAL
write *VACTUAL*:= 0x00ABCDEF	→ 0xA200ABCDEF	← 0xSS & X_ACTUAL
write *VACTUAL*:= 0x00123456	→ 0xA200123456	← 0xSS00ABCDEF

*)SS: is a placeholder for the status bits *SPI_STATUS*

5.1.2 Data Alignment

All data are right aligned. Some registers represent unsigned (positive) values; some represent integer values (signed) as two's complement numbers. Single bits or groups of bits are represented as single bits respectively as integer groups.

5.2 SPI Signals

The SPI bus on the TMC4361 has four signals:

SCKIN – bus clock input
SDIIN – serial data input
SDOIN – serial data output
NSCSIN – chip select input (active low)

The slave is enabled for an SPI transaction by a transition to low level on the chip select input NSCSIN. Bit transfer is synchronous to the bus clock SCKIN, with the slave latching the data from SDIIN on the rising edge of SCKIN and driving data to SDOIN following the falling edge. The most significant bit is sent first. A minimum of 40 SCKIN clock cycles is required for a bus transaction with the TMC4361. If less than 40 clock cycles are transmitted, the transfer will not be valid, even for a read access. However, sending only eight clock cycles can be useful to obtain the SPI status because it sends the status information back first.

If more than 40 clocks are driven, the additional bits shifted into SDIIN are shifted out on SDOIN after a 40-clock delay through an internal shift register. This can be used for daisy chaining multiple chips.

NSCSIN must be low during the whole bus transaction. When NSCSIN goes high, the contents of the internal shift register are latched into the internal control register and recognized as a command from the master to the slave. If more than 40 bits are sent, only the last 40 bits received before the rising edge of NSCSIN are recognized as the command.

5.3 Timing

The SPI interface is synchronized to the internal system clock, which limits the SPI bus clock SCKIN to half of the system clock frequency. The signal processing of the SPI inputs are supported with internal Schmitt Trigger, but not with RC elements. To avoid glitches at the inputs of the SPI interface between µC and TMC4361, external RC elements have to be provided. Figure 5.1 shows the timing parameters of an SPI bus transaction and the table below specifies the parameter values.

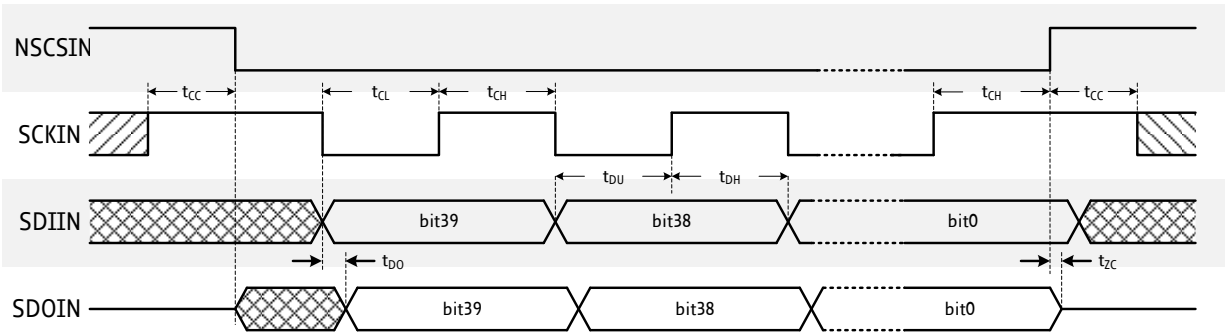


Figure 5.1 SPI timing

SPI interface timing	**AC-Characteristics** clock period: t_{CLK}					
Parameter	**Symbol**	**Conditions**	**Min**	**Typ**	**Max**	**Unit**
SCKIN valid before or after change of NSCSIN	t_{CC}		10			ns
NSCSIN high time	t_{CSH}	*) Min time is for synchronous CLK with SCKIN high one t_{CH} before SCSIN high only	t_{CLK}*)	>2t_{CLK}+10		ns
SCKIN low time	t_{CL}	*) Min time is for synchronous CLK only	t_{CLK}*)	>t_{CLK}+10		ns
SCKIN high time	t_{CH}	*) Min time is for synchronous CLK only	t_{CLK}*)	>t_{CLK}+10		ns
SCKIN frequency using external clock (Example: f_{CLK} = 16 MHz)	f_{SCK}	assumes synchronous CLK			f_{CLK} / 2 (8)	MHz
SDIIN setup time before rising edge of SCKIN	t_{DU}		10			ns
SDIIN hold time after rising edge of SCKIN	t_{DH}		10			ns
Data out valid time after falling SCKIN clock edge	t_{DO}	no capacitive load on SDOIN			t_{FILT}+5	ns

t_{CLK} = 1 / f_{CLK}

6 Input Filtering

Input signals can be noisy due to long cables and circuit paths. To prevent jamming, every input pin provides a Schmitt Trigger. Additionally, several signals are passed through a digital filter. Particular input pins are separated into four filtering groups. Each group can be programmed individually according to its filter characteristics.

PINS AND REGISTERS: INPUT FILTERING GROUPS

Pin names	Type		Remarks
A_SCLK B_SDI N ANEG_NSCLK BNEG_NSDI NNEG	Inputs		Encoder interface input pins
STOPL HOME_REF STOPR	Inputs		Reference input pins
START	Input		START input pin
SDODRV_SCLK SDIDRV_NSCLK	Inputs		Master clock input interface pins for serial encoder
STPIN DIRIN	Inputs		Step/Dir interface inputs
Pin name	Register address		Remarks
INPUT_FILT_CONF	0x03	RW	Filter configuration for all four input groups

6.1 Input Filter Configuration

Every filtering group can be configured separately with regard to input sample rate and digital filter length.

6.1.1 Input Sample Rate (SR)

Input sample rate = $f_{CLK} \cdot 1 / 2^{SR}$

where SR (extended with the particular name extension) is in [0... 7].

This means that every $(2^{SR})^{th}$ input bit will be considered for internal processing.

Sample rate configuration	
SR value	Sample rate
0	f_{CLK}
1	f_{CLK} / 2
2	f_{CLK} / 4
3	f_{CLK} / 8
4	f_{CLK} / 16
5	f_{CLK} / 32
6	f_{CLK} / 64
7	f_{CLK} / 128

6.1.2 Digital Filter Length (FILT_L)

One bit is sampled within each $(2^{SR})^{th}$ input clock cycle. The filter length FILT_L can be set within the range [0... 7]. The filter length FILT_L specifies the number of sampled bits that must have the same voltage level to set a new input bit voltage level.

Configuration of digital filter length	
FILT_L value	Filter length
0	No filtering
1	2 equal bits
2	3 equal bits
3	4 equal bits
4	5 equal bits
5	6 equal bits
6	7 equal bits
7	8 equal bits

6.1.3 StepDir input filter (changes as regards TMC4361old)

The Step/Dir input filtering setup differs slightly from the other groups as the other four groups already complete the whole *INPUT_FILT_CONF* register 0x03. Thus, it is possible to assign the Step/Dir input group to one of the existing by setting the appropriate bit in front of the setup parameters. In the following illustration, the filter settings for Step/Dir interface input pins will be taken from the Reference input pin group. If no group is selected, Step/Dir will be assigned to the encoder input interface filter group automatically.

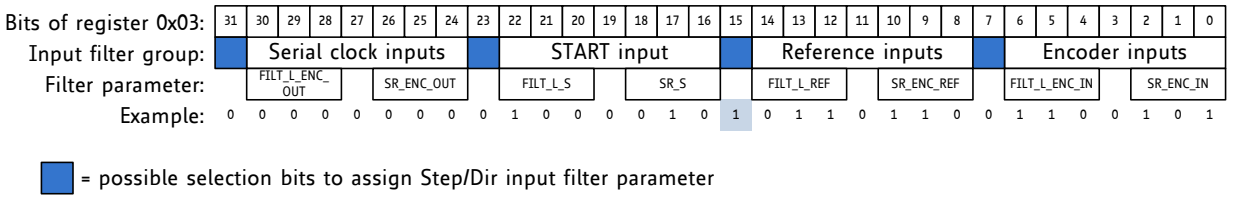


Figure 6.1 Step/Dir input pin filter settings will be derived from the Reference input filter group: SR_SDIN = 6, FILT_L_SDIN = 3 (other input filter groups: SR_ENC_IN = 5, FILT_L_ENC_IN = 6, SR_REF = 6, FILT_L_REF = 3, SR_S = 2, FILT_L_S = 4, SR_ENC_OUT = 0, FILT_L_ENC_OUT = 0)

6.1.4 Examples

The following three examples depict the input pin filtering of three different input filtering groups. The voltage levels after passing the Schmitt Trigger are compared to the internal signals which are processed by the motion controller.

The sample points are depicted as green dashed lines.

REFERENCE INPUT PINS

Here, every second clock cycle is sampled. Two sampled input bits must be equal to be a valid input voltage.

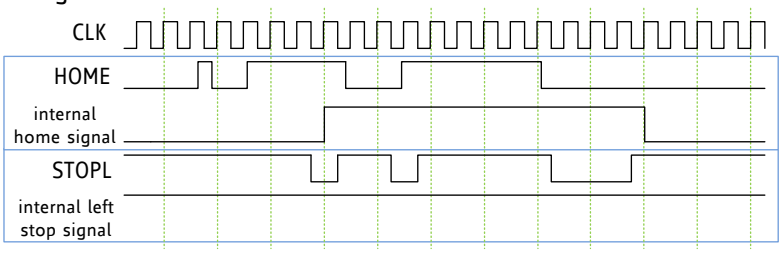


Figure 6.2 Reference input pins: SR_REF = 1, FILT_L_REF = 1

START Input Pin

Every fourth clock cycle is sampled and the sampled input bit is valid.

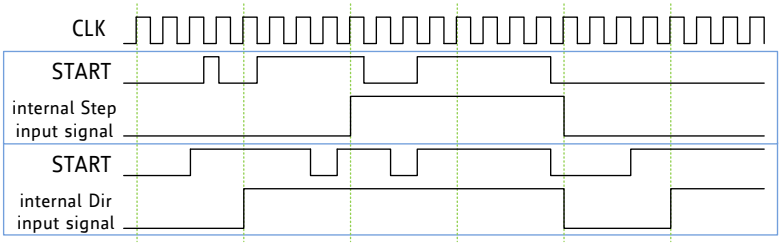


Figure 6.3 START input pin: SR_S = 2, FILT_L_S = 0

Encoder Interface Input Pins

Every clock cycle bit is sampled. Eight sampled input bits must be equal to be a valid input voltage.

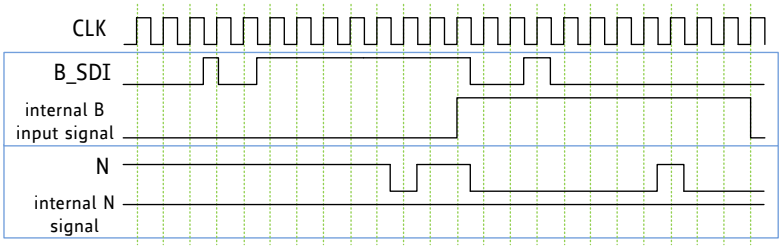


Figure 6.4 Encoder interface input pins: SR_ENC_IN = 0, FILT_L_ENC_IN = 7

7 Status Flags & Events

The TMC4361 offers several possibilities for velocity ramps. It combines target positioning and velocity ramps without interventions in between. However, the microcontroller connected to the TMC4361 normally requires status information. Therefore, TMC4361 provides 32 status flags and 32 status events. Status events can be configured customer specific and lead through using the interrupt output of the TMC4361. Further, the eight SPI status bits sent with each SPI datagram can be read out.

PINS AND REGISTERS: STATUS FLAGS AND EVENTS

Pin names	Type		Remarks
INTR	Output		Interrupt output to indicate status events
Register name	Register address		Remarks
GENERAL_CONF	0x00	RW	Bits: 15, 29, 30
STATUS_FLAGS	0x0F	R	32 status flags of the TMC4361 and the connected TMC motor driver chip
EVENTS	0x0E	R+C	32 events triggered by altered TMC4361 status bits
SPI_STATUS_SELECTION	0x0B	RW	Selection of 8 out of 32 events for SPI status bits
EVENT_CLEAR_CONF	0x0C	RW	Exceptions for cleared event bits
INTR_CONF	0x0D	RW	Selection of 32 events for INTR output

7.1 Status Flags

Status bits of the *STATUS_FLAGS* register are specified in the register chapter (see 19).

7.2 Status Events & SPI Status & Interrupts

STATUS FLAGS - STATUS EVENTS

Status events are triggered during the transition process of status bits from inactive to active level. Status bits and status events are associated in different ways:

- Several status events are associated with one status bit.
- Some status events show the status transition of one or more status bits out of a status bit group. The motor driver flags, e.g., trigger only one motor driver event *MOTOR_EV* in case one of the selected motor driver status flags becomes active.
- In case a flag consists of more than one bit, the number of associated events that can be triggered corresponds to the valid combinations. The *VEL_STATE* flag, e.g., has two bit but three associated velocity state events (b'00/b'01/b'10). Such an event is triggered if the associated combination switches from inactive to active.
- Furthermore, some events have no equivalence in the *STATUS_FLAGS* register (e.g., *COVER_DONE* which indicates new data from the motor driver chip).

ATTENTION The *EVENTS* register 0x0E is automatically cleared after reading the register subsequent to an SPI datagram request.

HOW TO AVOID A LACK OF INFORMATION

The recognition of a status event can fail in case it is triggered right before or during the *EVENTS* register 0x0E becomes cleared. To prevent events from being cleared, the *EVENT_CLEAR_CONF* register 0x0C can be assigned properly. Just set the related *EVENT_CLEAR_CONF* register bit position to 1.

Up to eight events can be selected for permanent SPI status report. Therefore, select up to eight events by writing 1 to the specific bit positions of the *SPI_STATUS_SELECTION* register 0x0B. The bit positions are sorted according to the event bit positions in the *EVENTS* register 0x0E. In case more than eight events are chosen, the first eight bits (starting from index 0 = LSB) are forwarded as *SPI_STATUS*.

7.3 Interrupts

Similar to the *EVENT_CLEAR_CONF* register and the *SPI_STATUS_SELECTION* register, events can be selected using the *INTR_CONF* register 0x0D to be forwarded to the INTR output. The active polarity of the INTR output can be set with the *intr_pol* bit of the *GENERAL_CONF* register 0x00. The selected events will be ORed to one signal. The INTR output becomes active as soon as one of the selected events triggers.

ATTENTION Due to the importance of events for interrupt generation and SPI status monitoring, it is recommended to clear the *EVENTS* register 0x0E before starting regular operation by reading it.

7.3.1 Connecting several INTR pins

The INTR pin could be configured for one interrupt signal transfer of several TMC4361 interrupt signals to the µC.
Therefore, *set intr_tr_pu_pd_en* = 1 of the *GENERAL_CONF* register 0x00. Per default then, the pin will work efficiently as **Wired-Or** due to the fact that during INTR pin is inactive, the output is pulled weakly to the inactive pin level polarity. If the pin becomes active, it will pulled strongly towards the active polarity. Connecting several pins at one signal line for the µC will result in polarity change of the whole signal line if one of the TMC4361 INTR pins will be set active.
By setting *intr_as_wired_and* = 1 of the *GENERAL_CONF* register 0x00, the signal line will act as **Wired-And**. While no interrupt is active, the INTR pin will be pulled strongly towards the inactive polarity. During the active state, the pin will be pulled weakly at active polarity. That way, the whole signal line will be pulled to active state if all pins are forwarding the active polarity.

8 Ramp Generator

Step generation is one of the main tasks of a stepper motor motion controller. The internal ramp generator of the TMC4361 provides several ways of step generation in order to form different ramp types to fit for various applications.

PINS AND REGISTERS: RAMP GENERATOR

Pin names	Type		Remarks
STPOUT_PWMA	Output		Step output signal
DIROUT_PWMB	Output		Direction output signal
Register name	Register address		Remarks
GENERAL_CONF	0x00	RW	Ramp generator affecting bits 0 → 5
STP_LENGTH_ADD DIR_SETUP_TIME	0x10	RW	Additional step length in clock cycles; 16 bits Additional time in clock cycles when no steps will occur after a direction change; 16 bits
RAMPMODE	0x20	RW	Requested ramp type and mode; 3 bits
XACTUAL	0x21	RW	Current internal microstep position; signed; 32 bits
VACTUAL	0x22	R	Current step velocity; 24 bits; signed; no decimals
AACTUAL	0x23	R	Current step acceleration; 24 bits; signed; no decimals
VMAX	0x24	RW	Maximum permitted or target velocity; signed; 32 bits= 24+8 (24 bits integer part, 8 bits decimal places)
VSTART	0x25	RW	Velocity at ramp start; unsigned; 31 bits=23+8
VSTOP	0x26	RW	Velocity at ramp end; unsigned; 31 bits=23+8
VBREAK	0x27	RW	At this velocity value, the ac-/deceleration will change during trapezoidal ramps; unsigned; 31 bits=23+8
AMAX	0x28	RW	Maximum permitted or target acceleration; unsigned; 24 bits=22+2 (22 bits integer part, 2 bits decimal places)
DMAX	0x29	RW	Maximum permitted or target deceleration; unsigned; 24 bits=22+2
ASTART	0x2A	RW	Acceleration at ramp start or below VBREAK; unsigned; 24 bits=22+2
DFINAL	0x2B	RW	Deceleration at ramp end or below VBREAK; unsigned; 24 bits=22+2
BOW1	0x2D	RW	First bow value of a complete velocity ramp; unsigned; 24 bits=24+0 (24 bits integer part, no decimal places)
BOW2	0x2E	RW	Second bow value of a complete velocity ramp; unsigned; 24 bits=24+0
BOW3	0x2F	RW	Third bow value of a complete velocity ramp; unsigned; 24 bits=24+0
BOW4	0x30	RW	Fourth bow value of a complete velocity ramp; unsigned; 24 bits=24+0
CLK_FREQ	0x31	RW	External clock frequency f_{CLK}; unsigned; 25 bits
XTARGET	0x37	RW	Target position; signed; 32 bits

8.1 Step/Dir Output Configuration

Step/Dir output signals can be configured for the driver circuit:

- For step signals that have to be longer than one clock cycle set *STP_LENGTH_ADD* (bit15:0 of register 0x10) appropriately. Then, the resulting step length is equal to *STP_LENGTH_ADD*+1 clock cycles. Thus, the step length can be chosen within the range 1...2^{16} clock cycles.
- DIROUT does not change the level during the active step pulse signal and for *STP_LENGTH_ADD*+1 clock cycles after the step signal returns to the inactive level.
- With the register *DIR_SETUP_TIME* (bit31:16 of register 0x10) the delay [clock cycles] between DIROUT and STPOUT voltage level changes can be set. Using this register, no steps are sent via STPOUT for *DIR_SETUP_TIME* clock cycles after a level change at DIROUT.

HINTS

- Per default, the step output is high active because a rising edge at STPOUT indicates a step.
- For changing the polarity, set *step_inactive_pol*=1. Now, each falling edge indicates a step.
- A step can be generated by toggeling the step output. Therefore, set *toggle_step*=1.
- Per default, a positive internal velocity VACTUAL will result in a forward motion through the internal SinLUT. Naturally, if VACTUAL < 0, the SinLUT values will be developed backwards. This relation could be altered by setting *reverse_motor_dir* = 1.
- *pol_dir_out* sets the DIROUT output level for the negative velocity direction. DIROUT is based on the internal µStep position *MSCNT* and therefore based on the SinLUT.
- *pol_dir_out*, *step_inactive_pol* (bit5), *reverse_motor_dir* (bit28), and *toggle_step* (bit4) are part of the general configuration *GENERAL_CONF* register 0x00.

8.2 Ramp Modes and Types

With proper configuration, the internal ramp generator of the TMC4361 is able to generate various ramps and the related step outputs for STPOUT. Note, that there are many possibilities to combine a *general ramp mode* (velocity mode, positioning mode) with *basic ramp types* (ramp in hold mode, trapezoidal ramp, S-shaped ramp). Therefore, select the general ramp mode and type first and proceed with further specifications, e.g., setting start and stop velocities or choosing different acceleration/deceleration values for each ramp phase.

GENERAL RAMP MODES

Two general ramp modes can be chosen with the *RAMPMODE* register. Therefore, bit2 of the *RAMPMODE* register 0x20 is used:

RAMPMODE(2)=0 *Velocity mode*. The target velocity *VMAX* will be reached using the selected ramp type.

RAMPMODE(2)=1 *Positioning mode*. *VMAX* is the maximum velocity value which will be used within the given ramp type and as long as the target position *XTARGET* will not be exceeded. Furthermore, the sign of *VMAX* is not relevant during positioning. The direction of the steps depends on *XACTUAL*, *XTARGET*, and the current ramp status.

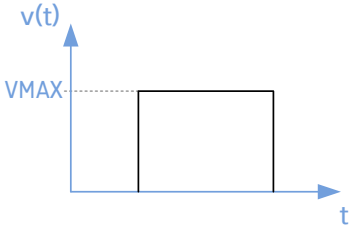
RAMP TYPES

Three basic ramp types are provided. These types differ in the velocity value development during the drive. For setting the basic ramp type, use the bits 1 and 0 of the *RAMPMODE* register 0x20:

TMC4361 RAMP TYPES		
***RAMPMODE*(1:0)**	**Ramp type**	**Function**
b'00	*Ramp in hold mode*	Follow *VMAX* only (rectangle velocity shape).
b'01	*Trapezoidal ramp*	Consideration of acceleration and deceleration values without adaption of these values.
b'10	*S-shaped ramp*	Use all ramp values (including bow values).

RAMPMODE(1:0)=b'00

Rectangle shaped ramp type in hold mode. *VACTUAL* is set immediately to *VMAX*.

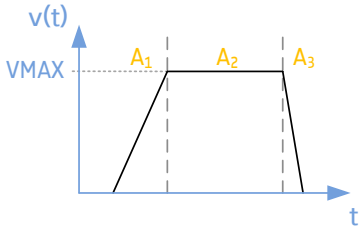


In positioning mode (*RAMPMODE*(2)=1), *VACTUAL* is set instantly to 0 if the target position is reached.
For exact positioning, it is recommended to set VMAX ≤ f_{CLK} · 1/4 pulses

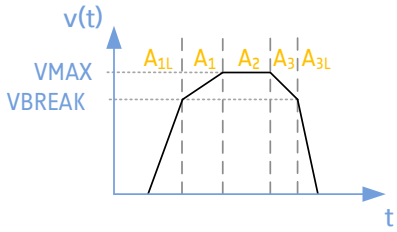
Figure 8.1 Rectangle shaped ramp type

RAMPMODE(1:0)=b'01

Trapezoidal shaped ramp type



Acceleration slope and deceleration slope have only one acceleration/deceleration value each. For this types, set *VBREAK* = 0.



The acceleration/deceleration factor alters at *VBREAK*.
In positioning mode, the ramp finishes exactly at the target position *XTARGET* by keeping *VACTUAL* = *VMAX* as long as possible.

Figure 8.2 Trapezoidal shaped ramp type

This trapezoidal ramp type reaches *VMAX* using linear ramps whereas the actual acceleration/deceleration factor *AACTUAL* register depends on the current ramp phase and the velocity which should be reached. The corresponding sign assignment for different ramp phases is depicted in the following table:

Ramp phase:	A_{1L}	A_1	A_2	A_3	A_{3L}
v>0: AACTUAL=	ASTART	AMAX	0	-DMAX	-DFINAL
v<0: AACTUAL=	-ASTART	-AMAX	0	DMAX	DFINAL

RAMPMODE(1:0)=b'10

S-shaped ramp types

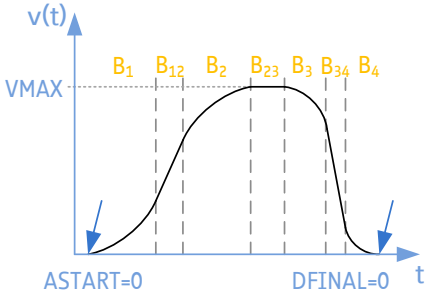
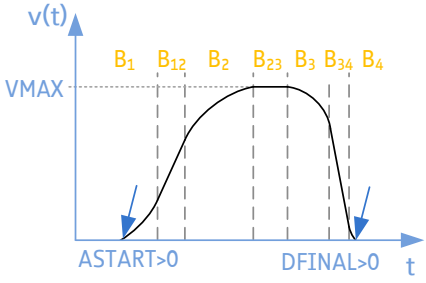


Figure 8.3 S-shaped ramp without initial and final acceleration/deceleration values



The start phase and the end phase of an S-shaped ramp can be accelerated/decelerated by *ASTART and DFINAL*. Using these parameters, the ramp starts with *ASTART* and it is ended with *DFINAL*. *DFINAL* becomes valid as soon as *AACTUAL* reaches the chosen *DFINAL* value. *ASTART* and *DFINAL* can be set separately.

Figure 8.4 S-shaped ramp type with initial acceleration and final deceleration value for B1 and B4

This ramp type reaches *VMAX* by means of S-shaped ramps whereas the acceleration/deceleration factor depends on the current ramp phase and alters every 64 clock cycles during the bow phases B_1, B_2, B_3, and B_4.

Ramp phase:	**B_1**	**B_{12}**	**B_2**	**B_{23}**	**B_3**	**B_{34}**	**B_4**
v>0: AACTUAL=	ASTART→AMAX	AMAX	AMAX→0	0	0→-DMAX	-DMAX	-DMAX→-DFINAL
BOW_{ACTUAL}=	BOW1	0	-BOW2	0	-BOW3	0	BOW4
v<0: AACTUAL=	-ASTART→-AMAX	-AMAX	-AMAX→0	0	0→DMAX	DMAX	DMAX→DFINAL
BOW_{ACTUAL}=	-BOW1	0	BOW2	0	BOW3	0	-BOW4

S-SHAPED RAMPS IN POSITIONING MODE

The ramp finishes exactly at the target position by keeping abs(*VACTUAL*) = *VMAX* as long as possible. Furthermore, the slopes to and from *VMAX* are as fast as possible without exceeding given values. It is even possible that the phases B_{12}, B_{23}, and B_{34} are left out due to given values. Nevertheless, the S-shaped ramp style is always performed in positioning mode, if *RAMP_MODE(1:0)* = b'10 is set.

ATTENTION The parameter *DFINAL* is not considered during positioning mode!

ATTENTION: Do not switch the *RAMPMODE* if VACTUAL is not constant. Also, do not switch to positioning mode if *VACTUAL* ≠ 0 and the difference between *XACTUAL* and *XTARGET* is too small for the given falling slope.
HINT: There is one exception: If circular movement (see 0) is enabled and current velocity is too high for exact positioning during one revolution, it is possible to change from velocity to positioning mode.

FASTEST POSSIBLE SLOPE IN POSITIONING MODE

The fastest possible slopes are always performed if the phases B_{12} and/or B_{34} are not reached during a rising and/or falling S-shaped slope. Thus, the ramp maintains the maximum velocity *VMAX* as long as possible in positioning mode until the falling slope finishes the ramp to reach *XTARGET* exactly. The result is the fastest possible positioning ramp in matters of time.

8.2.1 Velocity Start *VSTART* and Velocity Stop *VSTOP*

S-shaped and trapezoidal velocity ramps can be started with an initial velocity value by setting *VSTART* higher than zero (see Figure 8.5). Such an S-shaped ramp with *VSTART* > 0 is a ramp without the first ramp bow B_1. The ramp starts with *AACTUAL* = *AMAX* and *VACTUAL* = *VSTART*. Logically, the parameter *ASTART* is not considered.
It is also possible to set *VSTOP* (a final velocity value) which finishes the ramp if *VACTUAL* reaches the *VSTOP* value (see Figure 8.6). This leads to an S-shaped velocity ramp without the bow B_4. Hence, *DFINAL* is not considered.

TRAPEZOIDAL AND S-SHAPED RAMPS USING PARAMETER *VSTART*

VSTART > 0 and *VSTOP* = 0

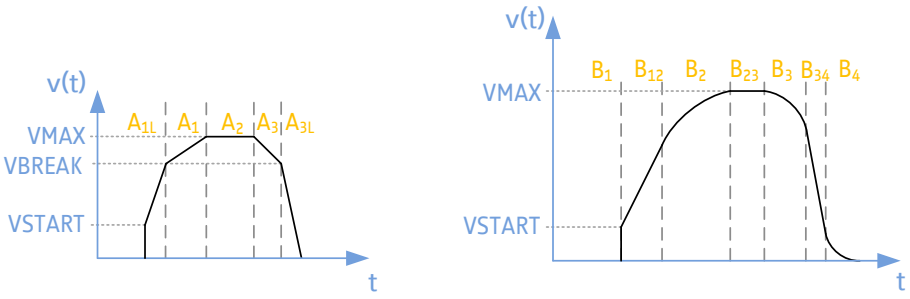


Figure 8.5 Trapezoidal and S-shaped ramps using *VSTART*

TRAPEZOIDAL AND S-SHAPED RAMPS USING PARAMETER *VSTOP*

VSTART = 0 and *VSTOP* > 0

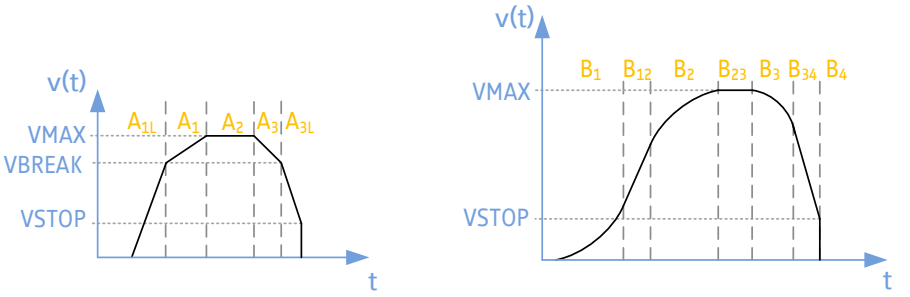


Figure 8.6 Trapezoidal and S-shaped ramps using *VSTOP*

TRAPEZOIDAL AND S-SHAPED RAMPS USING PARAMETERS *VSTART* AND *VSTOP*

VSTART > 0 and *VSTOP* > 0

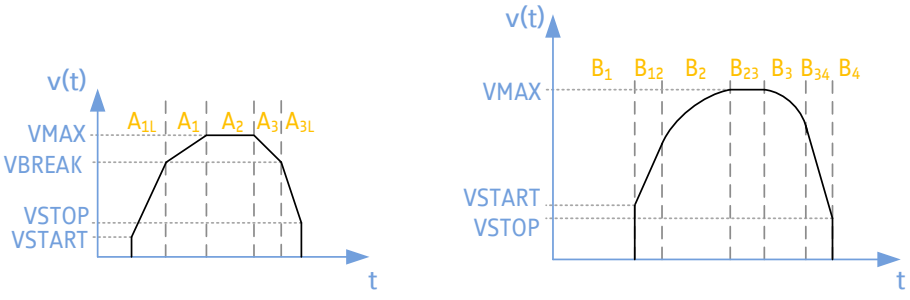


Figure 8.7 Trapezoidal and S-shaped ramps using *VSTART* and *VSTOP*

SUGGESTIONS

- *VSTART* and *VSTOP* can only be used to start or end a velocity ramp. If the velocity direction alters due to register assignments while a velocity ramp is in progress, the velocity values develop according to the current velocity ramp type without using *VSTART* or *VSTOP*.
- *VSTOP* can be used in positioning mode if the target position is reached. In velocity mode, *VSTOP* can only be used if *VACTUAL* ≠ 0 and the target velocity *VMAX* is assigned to 0.
- The unsigned values *VSTART* and *VSTOP* are valid for both velocity directions.
- Every register value change is assigned immediately.

ATTENTION *VBREAK* must set higher than *VSTOP*, except is it not used (*VBREAK*=0)!

USING VSTART AND ASTART CONCURRENTLY FOR S-SHAPED RAMPS (NOT IMPLEMENTED FOR TMC4361OLD)

In some S-shaped ramp applications, it could be useful to start with a defined velocity value (*VSTART* > 0), but not with the maximum acceleration value *AMAX*. Thus, *use_astart_and_vstart* (bit0 of the *GENERAL_CONF* register 0x00) have to be set to 1. As a result, the following special ramp types are also possible.

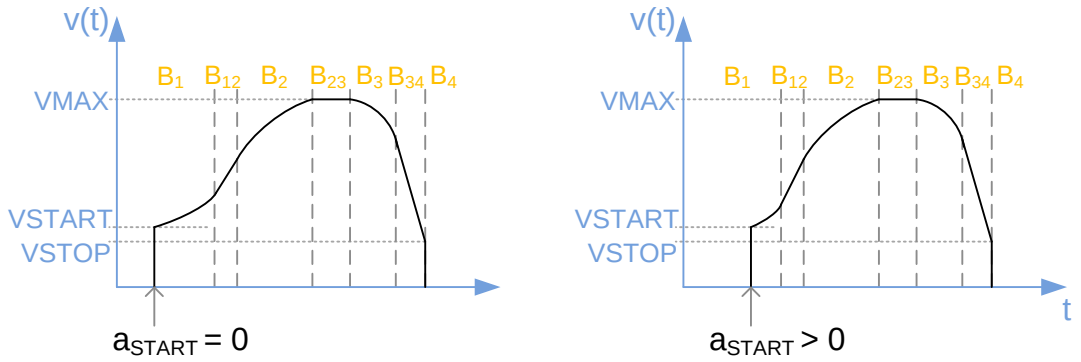


FIGURE 8.8 S-SHAPED RAMPS USING *VSTART* > 0 AND *USE_ASTART_AND_VSTART* = 1. AS A RESULT, SECTION B_1 WILL BE PASSED THROUGH, ALTHOUGH *VSTART* IS USED.

ATTENTION If the requested conditions for the acceleration slope of an S-shaped ramp (*VSTART* **and** *ASTART*, *BOW1* and *BOW2*) do not fit with *VMAX*, the starting acceleration value is altered. Note that not correctly adjusted values can lead to an acceleration slope with a final velocity after B2 greater than *VMAX*. In this case the given *VMAX* will be initiated by a falling slope to *VMAX* after finishing the acceleration slope.
In case only one of the parameters VSTART and ASTART is used, the

8.2.2 Internal Ramp Generator Units

All parameter units are real arithmetical units. Therefore, it is necessary to set the *CLK_FREQ* register to the appropriate value in [Hz] which is given by the external clock. Any value between 4.2 MHz and 32 MHz could be chosen.

VELOCITY VALUES

VACTUAL is given as a 32 bit signed value with no decimal places. The unsigned velocity values *VSTART*, *VSTOP*, and *VBREAK* consist of 23 digits and 8 decimal places. *VMAX* is a signed value with 24 digits and 8 decimal places. Velocity values are given in pulses per second [pps].

ATTENTION The maximum velocity *VMAX* is restricted by the clock frequency. Values higher than (½ puls – f_{CLK}) are prohibited because of an incorrect STPOUT output if *VACTUAL* exceeds this limit.

ACCELERATION VALUES

The unsigned values *AMAX*, *DMAX*, *ASTART*, and *DFINAL* consist of 22 digits and 2 decimal places. *AACTUAL* shows a 24 bit non decimal signed value. Acceleration and deceleration units are given in pulses per second² [pps²].

BOW PARAMETER VALUES

Bow values are unsigned 24 bit values without decimal places. They are given in pulses per second³ [pps³].

The following absolute minimum and maximum values are valid:

Value Classes	Velocity	Acceleration	Bow	Clock
Registers	VMAX, VSTART, VSTOP, VBREAK	AMAX, DMAX, ASTART, DFINAL	BOW1, BOW2, BOW3, BOW4	CLK_FREQ (f_{CLK})
Minimum	3.906250 mpps	0.250000 $mpps^2$	1 pps^3	4.194304 MHz
Maximum	8.388607 pps ½ puls * f_{CLK}	4.194303 pps^2	16.777 pps^3	30 MHz

SHORT AND STEEP RAMPS

For short and steep ramps higher acceleration/deceleration and bow values than usual are available by activating *direct_acc_val_en* and *direct_bow_val_en* (bit1 and bit2 of the *GENERAL_CONF* register 0x00). Set these parameters to 1 to change the units:

direct_acc_val_en=1 The values for *AMAX*, *DMAX*, *ASTART*, *DFINAL*, and *DSTOP* (see chapter 0) are given as velocity value change per clock cycle with 24 bit unsigned decimal places (MSB = 2^{-14}).

direct_bow_val_en=1 Bow values are given as acceleration value change per clock cycle. The values *BOW1*, *BOW2*, *BOW3*, and *BOW4* are 24 bit unsigned decimal places with the MSB defined as 2^{-29}.

EXAMPLE

With a clock frequency f_{CLK} = 16 MHz the following maximum values are valid:

Value Classes	Acceleration (*direct_acc_val_en*=1)	Bow (*direct_bow_val_en*=1)
Registers	AMAX, DMAX, ASTART, DFINAL, DSTOP	BOW1, BOW2, BOW3, BOW4
Calculation	a[pps^2] = (Δv/clk_cycle) / $2^{37} \cdot f_{CLK}^2$	bow[pps^3] = (Δa/clk_cycle) / $2^{53} \cdot f_{CLK}^3$
Minimum	-1.86 $kpps^2$	-454.75 $kpps^3$
Maximum	-31.25 $Gpps^2$	-7.63 $Tpps^3$

8.2.3 Limitations for On-the-Fly changes of ramp parameters

Every register value change is assigned immediately. Exceptions will be explained in chapter 11. Generally, following ramp parameters should only be changed if *VACTUAL* = 0 and no motion has been started so far:

- *VSTART*, *VSTOP*, *VBREAK*
- *AMAX*, *DMAX*, *ASTART*, *DFINAL*
- *BOW1*, *BOW2*, *BOW3*, *BOW4*

ATTENTION It is also possible, **but not recommended**, to change these parameters during motion (preferably if *VACTUAL* is constant). If these register value will be changed during motion, internal reevaluations could cause temporary ramp abortions in the way of the given ramp type, *VMAX* overshooting (in velocity and positioning mode) or target position overshooting (in positioning mode). For these reasons, changing the mentioned ramp parameters during motion should be carefully evaluated for every possible occurrence in the application run.

RAMPMODE CHANGES

1.) It is not recommended to switch the *RAMPMODE* register if *VACTUAL* is not constant.
2.) Furthermore, switching to positioning mode should always assigned if *VATUAL* = 0.
3.) Whereas, switching to velocity mode could also made during motion if the ramp type is not changed (*RAMPMODE*(2) = constant!).
4.) These rules are also valid for RAMPMODE changes initiated by the TIMER unit (see chapter11)
5.) If circular movement is enabled (see section 0), switching to positioning mode could also assigned during motion if ……

ASSIGNING VMAX AND XTARGET DURING MOTION IN VELOCITY MODE

Obviously, it is possible to change *XTARGET* at any time during velocity mode as the target position will be not the determining factor for the motion run.
As the velocity mode will constantly follow the currently assigned *VMAX* value, there are no restrictions as regards the point in time when *VMAX* will be changed.
Please, note that *VSTART* resp. *VSTOP* for trapezoidal and S-shaped ramps will only be used if *VACTUAL* = 0 and *VMAX* ≠ 0 resp. *VACTUAL* ≠ 0 and *VMAX* = 0.

ASSIGNING VMAX DURING MOTION IN POSITIONING MODE

VMAX could be set at any time during motion in positioning mode. TMC4361 will then use the given parameter to reach as fast as possible the newly given velocity limit VMAX, but considering always XTARGET which will be determining factor during positioning mode.

9 External step control - Electronic gearing

Steps could also generated by external steps which are manipulated internally by an electronic gearing. This feature is not available for TMC4361old.

PINS AND REGISTERS: EXTERNAL STEP CONTROL

Pin names	Type		Remarks
STPIN	Input		Step input signal
DIRIN	Input		Direction input signal
Register name	Register address		Remarks
GENERAL_CONF	0x00	RW	bits: 6→9 , 26
GEAR_RATIO	0x12	RW	Electronic gearing factor; signed; 32 bits=8+24 (8 bits integer part, 24 bits decimal places)

To synchronize with other motion controllers TMC4361 offers a step direction input interface. By setting *sdin_mode* ≠ b'00 (bit7:6 of the *GENERAL_CONF* register 0x00), the S/D input interface will be enabled which will disable concurrently the internal ramp generator. The polarity of the STPIN input could be assigned by setting *sdin_mode* = b'01 for a high active step polarity and b'10 for a low active STPIN input. If *sdin_mode* is set to b'11, every voltage level transition at STPIN will be taken as single step. The polarity of DIRIN could be set by *pol_dir_in* (bit 8 of *GENERAL_CONF* register 0x00) whereas the negative direction will be assigned directly with this switch.

If an external step is not congruent with an internal step, *GEAR_RATIO* register 0x12 have to be set accordingly. This signed parameter consists of eight bit digits and 24 bits decimal places. With every external step the chosen value of *GEAR_RATIO* will be added to an internal accumulation register. If an overflow will occur an internal step will be generated and the remainder will be kept for the next external step. Thus, any absolute gearing value between 2^-24 and 127 is possible. Gearing ratios beyond 1 are more reasonable for the SPI output which uses the internal SinLUTable because multiple steps will be generated one after another without interpolation if the accumulation register has a value above 1. In contrasts to a burst of steps at the STPOUT pin, the SPI output will only forward the new position in the inner SinLUT where only some values have been skipped if |*GEAR_RATIO*| > 1.

If the gearing factor is a negative number, the direction which are given by DIRIN and *pol_dir_in* will be inverted.

By setting *sd_indirect_control* = 1 (bit9 of the *GENERAL_CONF* register 0x00), the internal ramp generator is enabled due to the fact that external step impulses transferred via STPIN and DIRIN will not influence the internal *XACTUAL* counter directly. Instead, the *XTARGET* register will be altered by 1 with every *GEAR_RATIO* accumulation register overflow. If *XTARGET* will be increased or decreased depends (in the same way as known from the direct control) on the sign of the GEAR_RATIO, *pol_dir_in*, and DIRIN when the accumulation overflow appears. The usage of this feature allows a synchronized motion of different velocity ramps where every ramp follows its own parameters.

SWITCHING BETWEEN DIRECT EXTERNAL CONTROL AND INTERNAL RAMP GENERATION

During the direct external control (*sdin_mode* ≠ b'00 and *sd_indirect_control* = 0), the internal ramp generator will be switched off. However, if *automatic_direct_sdin_switch_off* (bit26 of the *GENERAL_CONF* register 0x00) is set to 1, a fluent transfer from direct external control to an internal ramp could be managed. If this mode is active, *VSTART* will define the actual velocity value if direct external control is switched off. The internal direction will be selected automatically. The time step of the last internal step will also taken into account to provide a smooth transition from external to internal ramp control. Thus, the only parameter which has to be set externally is *VSTART* whose value should be know by the connected µC. To support also a smooth S-shaped ramp transition, the starting acceleration value could also be set separately. By setting *ASTART* ≠0, the internal ramp will start with *AACTUAL* = *ASTART* if *a_sign_bit* = 0 or *AACTUAL* = -*ASTART* if *a_sign_bit* = 1. This *a_sign_bit* (bit31 of *ASTART* register 0x2A) have to be set also because there is no calculation of the external velocity which is a requirement for an automatic setting of the sign bit.
A detailed example will be provided in chapter… which illustrates the usage of the introduced parameters for external control and the automatic switch off.

10 Reference Switches

The reference input signals of the TMC4361 can be considered as a safety feature. The TMC4361 provides different possibilities for reference switches and allows for appropriate settings for various applications. The TMC4361 offers two switches in hardware (STOPL, STOPR) and two additional virtual stop switches (VIRT_STOP_LEFT, VIRT_STOP_RIGHT). Additionally, an home reference switch is available. Digital filter settings for the reference switches could be assigned in the filter configuration register. A limitation of *XACTUAL* will be supported also to offer an easy implementation of circular motion profiles.

PINS AND REGISTERS: REFERENCE SWITCHES

Pin names	Type		Remarks
STOPL	Input		Left reference switch
STOPR	Input		Right reference switch
HOME_REF	Input		Home switch
TARGET_REACHED	Output		Reference switch to indicate XACTUAL = XTARGET
Register name	Register address		Remarks
GENERAL_CONF	0x00	RW	Bits: 16, 27, 29, 31
REFERENCE_CONF	0x01	RW	Configuration of interaction with reference pins
HOME_SAFETY_MARGIN	0x1E	RW	Region of uncertainty around X_HOME
DSTOP	0x2C	RW	Deceleration value if stop switches STOPL/STOPR or virtual stops are used with soft stop ramps. The deceleration value allows for an automatic linear stop ramp.
POS_COMP	0x32	RW	Free configurable compare position; signed; 32 bits
VIRT_STOP_LEFT	0x33	RW	Virtual left stop that triggers a stop event at XACTUAL ≤ VIRT_STOP_LEFT; signed; 32 bits
VIRT_STOP_RIGHT	0x34	RW	Virtual left stop that triggers a stop event at XACTUAL ≥ VIRT_STOP_RIGHT; signed; 32 bits
X_HOME	0x35	RW	Home reference position; signed; 32 bits
X_LATCH REV_CNT	0x36	R	Stores XACTUAL at different conditions or Revolution Counter for circular movement; signed; 32 bits
X_RANGE		W	Limit for internal positions; unsigned; 31 bits
CIRCULAR_DEC	0x7C	W	Decimal places for circular movement

10.1 STOPL and STOPR

A left and a right stop switch are provided in hardware in order to stop the drive immediately, if one of them is triggered. Therefore, pin 12 and pin 14 of the motion controller have to be used. Both switches have to be enabled first:

- To use STOPL set *stop_left_en* = 1 (bit0 of REFERENCE_CONF register 0x01). Now, the current velocity ramp stops in case STOPL is equal to the chosen active polarity *pol_stop_left* (bit2 of REFERENCE_CONF register 0x01) and *VACTUAL* < 0.
- To use STOPR set *stop_right_en* = 1 (bit1 of REFERENCE_CONF register 0x01). Now, STOPR stops the ramp in case the STOPR voltage level matches *pol_stop_right* (bit3 of REFERENCE_CONF register 0x01) and *VACTUAL* > 0.

The deceleration slope for stopping the ramp is influenced by *soft_stop_en* (bit5 of REFERENCE_CONF register 0x01):

- Set *soft_stop_en* = 0 for a hard and quick stop.
- Set *soft_stop_en* = 1 to stop the ramp with a linear falling slope. In this case the deceleration factor is determined by *DSTOP*. *VSTOP* is not considered during the stop deceleration slope.

ATTENTION If ***DSTOP* = 0** (default value!) and soft stop is enabled, the **ramp will not stop** due to the fact that the stop deceleration slope factor is 0.
Further on, ***DSTOP* could be altered during the stop slope** to change the soft stop deceleration slope as requested.

At the same time when a stop switch becomes active, the related status flag will be set and the particular event will be released. The flag remains set as long as the stop switch remains active. After reaching *VACTUAL* = 0 due to the slope, further movement in the particular direction is not possible.
Driving on in the direction of a reference switch is possible if the following conditions are met:

- The related status event is set back. The reference switch is not active anymore or alternatively, the related enabling switch (*stop_left_en*, *stop_right_en*) is reset to 0 (switched off) to go on driving in the - prior to that - closed direction.
- Stop events are cleared by reading out the *EVENTS* register 0x0E. This is done automatically by the motion controller subsequent to an SPI datagram read request to this register. (There is only one exception to this if an event is selected for the *EVENT_CLEAR_CONF* register in order to inhibit the regular clearing.)

10.1.1 Latching configurations

Four different events can be chosen to latch the current internal position *XACTUAL* in the register *X_LATCH*. Therefore, bit13:10 of REFERENCE_CONF register 0x01 have to be set accordingly. The following events and reference configurations result in such a transfer *XLATCH* = *XACTUAL* with an event indicating the latching process:

Reference configuration	***pol_stop_left*=0**	***pol_stop_left*=1**	***pol_stop_right*=0**	***pol_stop_right*=1**
latch_x_on_inactive_l=1	STOPL = 0 → 1	STOPL = 1 → 0	---	---
latch_x_on_active_l=1	STOPL = 1 → 0	STOPL = 0 → 1	---	---
latch_x_on_inactive_r=1	---	---	STOPR = 0 → 1	STOPR = 1 → 0
latch_x_on_active_r=1	---	---	STOPR = 1 → 0	STOPR = 0 → 1

HINT Setting *invert_stop_direction*=1 (bit4 of REFERENCE_CONF register 0x01) swaps STOPL and STOPR. Thus, all configuration parameters for STOPL become valid for STOPR and vice versa.

10.2 Virtual Stop Switches

The TMC4361 provides additional virtual limits which trigger stop slopes in case the specific virtual stop switch microstep position is reached. Virtual stop positions can be set using the *VIRTUAL_STOP_LEFT* and *VIRTUAL_STOP_RIGHT* registers 0x33 resp. 0x34.

Virtual stop switches have to be enabled like non-virtual reference switches. Therefore, set *virtual_left_limit_en* (bit6 of REFERENCE_CONF register 0x01) resp. *virtual_right_limit_en* (bit7 of REFERENCE_CONF register 0x01) to 1. Hitting a virtual limit switch triggers the same process as hitting STOPL or STOPR.

At the same time when a virtual stop switch becomes active an event becomes released which has to be cleared in any case before further movement in the particular direction can be performed again.

Driving on in the direction of a virtual switch after a stop event is possible if the following conditions are met:

- For further movement in negative direction choose a new value for *VIRTUAL_STOP_LEFT* or set *virtual_left_limit_en* = 0. It is not necessary to clear the event *VSTOPL_ACTIVE* first.
- For further movement in positive direction choose a new value for *VIRTUAL_STOP_RIGHT* or set *virtual_right_limit_en* = 0. It is not necessary to clear the event *VSTOPR_ACTIVE* first.

The deceleration slope can be chosen with *virt_stop_mode* (bit9:8 of REFERENCE_CONF register 0x01):

- Set *virt_stop_mode* = b'01 for a hard and quick stop.
- Set *virt_stop_mode* = b'10 to stop the ramp with a linear falling slope. In this case the deceleration factor is determined by *DSTOP*.

ATTENTION *invert_stop_direction* has no influence on *VIRTUAL_STOP_LEFT* resp. *VIRTUAL_STOP_RIGHT*!

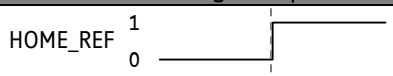
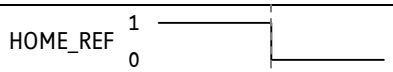
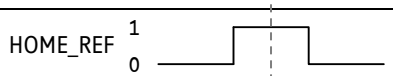
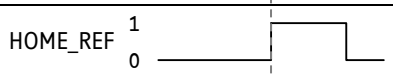
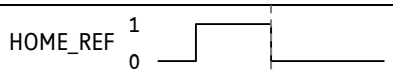
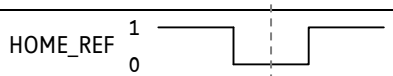
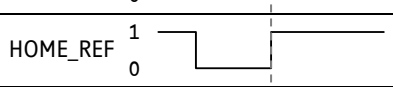
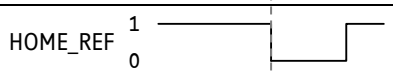
10.3 HOME Reference

For monitoring, the switch reference input HOME_REF is provided.

HOMING PROCESS

- Enable the tracking mode with *start_home_tracking* = 1 (bit20 of REFERENCE_CONF register 0x01).
- With the next home event *XACTUAL* is latched to *X_HOME*.
- The *XLATCH_DONE* event will be released (not implemented for TMC4361old) and the switch *start_home_tracking* of the REFERENCE_CONF register is automatically reset to 0.
- An error flag is permanently evaluated. This error flag indicates whether the current voltage level of the HOME_REF reference input is valid in respect to *X_HOME* and the chosen *home_event*.

Nine different home events are possible. Besides *home_event* = b'0000 which uses the N signal of an incremental ABN encoder, the following home events can be used. Therefore, configure the four *home_event* bits (bit19:16 of REFERENCE_CONF register 0x01).)

home_event	Description		X_HOME (direction: negative/positive)
b'0011	HOME_REF = 0 indicates negative direction in reference to X_HOME		
b'1100	HOME_REF = 0 indicates positive direction in reference to X_HOME		
b'0110	HOME_REF = 1 indicates home position	X_HOME in center	
b'0010		X_HOME at the left side	
b'0100		X_HOME at the right side	
b'1001	HOME_REF = 0 indicates home position	X_HOME in center	
b'1011		X_HOME at the right side	
b'1101		X_HOME at the left side	

DEFINING AN UNCERTAINTY AREA AROUND *X_HOME*

Use the register *HOME_SAFETY_MARGIN* (0x1E) for defining an uncertainty area around *X_HOME*. Then, homing uncertainties related to the special application environment are considered for the further process. There will be no error flag generated if two conditions are met:

XACTUAL ≥ X_HOME - HOME_SAFETY_MARGIN and XACTUAL ≤ X_HOME + HOME_SAFETY_MARGIN

The following examples (see Figure 10.1.) show the points at which - dependent on the chosen *home_event* - an error flag is generated. The following examples illustrate *HOME_REF* monitoring and generation of the *HOME_ERROR_Flag* for *home_event* = b'0011 (*), b'1100 (**), b'0110 (***), b'0010 (***), b'0100 (***), b'1001 (****), b'1011 (****), and b'1101 (****).

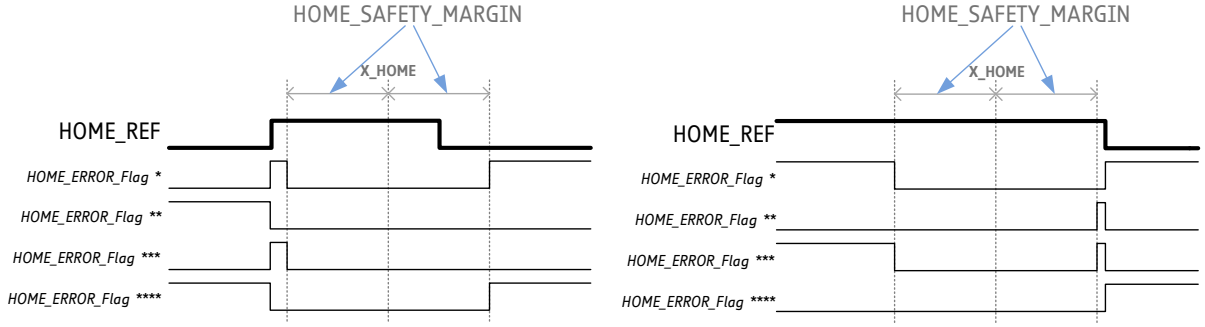


Figure 10.1 *HOME_REF* monitoring and *HOME_ERROR_FLAG*

It is recommended to set the *HOME_SAFETY_MARGIN* bigger than the period during which the *HOME_REF* level is active for the *home_event*s b'0110, b'0010, b'0100, b'1001, b'1011, and b'1101. This is necessary to avoid wrong *HOME_ERROR_Flag*s.

After homing with the N channel (*home_event* = b'0000) for a precise assignment of *X_HOME* the correct *home_event* has to be assigned in order to activate the generation of *HOME_ERROR_Flag*s. Note that *home_event* = b'0000 results in HOME_ERROR_Flag = 0 permanently.

ATTENTION In contrast to TMC4361old, it is not required to set *latch_x_on_n* and *clr_latch_cont_on_n* or *clr_latch_once_on_n* for the homing process based on the n event (*home_event* = b'0000).

HOMING WITH STOPL AND STOPR

STOPL and STOPR inputs can also be used as HOME_REF inputs. Therefore, set *stop_left_is_home* = 1 resp. *stop_right_is_home* = 1 (bit14 resp. bit15 of the *REFERENCE_CONF* register 0x01). This leads to a stop of the current ramp only after STOPL or STOPR is switching to active state and the home uncertainty region is crossed. The home uncertainty region is given by *X_HOME* and *HOME_SAFETY_MARGIN*.

ATTENTION In contrast to TMC4361old is possible to set *X_HOME* automatically with a write access to register 0x35.

10.4 Repeating Motion after reaching *XTARGET*

Usually, reaching *XTARGET* in positioning mode finishes a velocity ramp. To repeat the current ramp with its specified parameters steadily set *clr_pos_at_target* to 1 (bit21 of the *REFERENCE_CONF* register 0x01). Until velocity mode is chosen or *clr_pos_at_target* is set to 0, *XACTUAL* will be reset to 0 if *XTARGET* is reached (*XACTUAL* = *XTARGET*). Normally, the falling slope to stop the ramp is performed within each ramp cycle.

TRIGGERING FURTHER RAMPS IDENTICAL TO THE FIRST ONE (*POSITIONING MODE ONLY*)

- Set *clr_pos_at_target* = 1
- Set *XTARGET*.
- Now, *XACTUAL* is set to 0 automatically if XTARGET is reached.
- Another velocity ramp for reaching *XTARGET* becomes active now.

10.5 Circular movement

Many application are based on circular motion profiles. Therefore, TMC4361 (but not TMC4361old!) offers a limitation of the range of *XACTUAL* with an automatic overflow processing.
By setting *X_RANGE* ≠ 0 (register 0x36, only writing access!) and activating *circular_movement* = 1 (bit22 of the *REFERENCE_CONF* register 0x01), XACTUAL will be limited to:

$$-X\_RANGE \leq XACTUAL \leq X\_RANGE - 1.$$

ATTENTION By setting *circular_movement* to 1, *XACTUAL* should be located inside of the defined range!

The overflow will be processed automatically which means that if *XACTUAL* reaches the most positive position (*X_RANGE* – 1) and the motion will proceed into positive direction, the next *XACTUAL* value will be *-X_RANGE*. The same is true when moving into negative direction, where (*X_RANGE* – 1) will be the position after *-X_RANGE*.

ATTENTION During positioning mode, the movement direction will be dependent on the shortest path to the target position *XTARGET*. For example, if *XACTUAL* = 200, *X_RANGE* = 300 and *XTARGET* = -200, the positioning ramp will find its way across the overflow position (299 → -300) (see **Figure 10.2** a)).

Due to definition of the limitation range, one revolution is only able to consist of an even number of microsteps. However, some applications demand other requirements. By setting *CIRCULAR_DEC* (0x7C) properly, these demands could be fulfilled. This register allows a nearly free adjustable range of microsteps due to the fact that it represent one digit and 31 decimal places for the number of microsteps per one revolution. With every completed revolution (a revolution is completed at the overflow), the *CIRCULAR_DEC* value will be added to an internal accumulation register. If this register has an overflow at bit31, it will remain at the overflow of *XACTUAL* for one step.

EXAMPLE 1

One revolution consists of 401 microsteps.
A definition of *X_RANGE* = 200 will only provide 400 microsteps per revolution (-200 ≤ *XACTUAL* ≤ 199), whereas *X_RANGE* = 201 will result in 402 microsteps per revolution (-201 ≤ *XACTUAL* ≤ 200)
By setting *CIRCULAR_DEC* = 0x80000000 (= 2^{31} / 2^{31} = 1) every revolution an overflow will be produced at the decimals accumulation register. This leads to a microstep count of 401 per revolution.

EXAMPLE 2

One revolution consists of 400.5 microsteps.
By setting *CIRCULAR_DEC* = 0x40000000 (= 2^{30} / 2^{31} = 0.5) every second revolution an overflow will be produced at the decimals accumulation register. This leads to a microstep count of 400 every second revolution and 401 for the other half of the revolutions. On average, this leads to 400.5 microsteps per revolution.

EXAMPLE 3

One revolution consists of 401.25 microsteps.
By setting *CIRCULAR_DEC* = 0xA0000000 (= (2^{31} + 2^{29}) / 2^{31} = 1.25) every revolution an overflow will produced at the decimals accumulation register. Furthermore, every fourth revolution an additional overflow will occur which leads to another waiting step. This leads to a microstep count of 401 for three of four revolutions and 402 for the remaining fourth of all revolutions. On average, this leads to 401.25 microsteps per revolution.
By passing the overflow position the internal *REV_CNT* (0x36, only reading access) will be increased by one revolution if *XACTUAL* will changed from (*X_RANGE* – 1) to *-X_RANGE* or decreased by one revolution if *XACTUAL* will take the other direction. If *circular_cnt_as_xlatch* (bit27 of the *GENERAL_CONF* register 0x00) is set to 1, the register 0x36 will not display the *X_LATCH* value. Instead, the revolution counter *REV_CNT* could be read out at this register address.

10.5.1 Blocking zones

If circular movement is enabled (*X_RANGE* ≠ 0 and *circular_movement* = 1), virtual stops could be used to set blocking zones, if both virtual stops are enabled and set properly. Then, the blocking zones reaches from *VIRTUAL_STOP_LEFT* to *VIRTUAL_STOP_RIGHT*. During positioning, the path from *XACTUAL* to *XTARGET* will not lead through the blocking zone which could result in a longer path then the direct one. However, the chosen deceleration ramp will be initiated as soon as one of the limits is reached. This may result from the velocity mode or if the target is located in the blocking zone.

HINT To get out of the blocking zone, clear the virtual stop events and set a regular target position outside of the blocking zone. TMC4361 will initiate a ramp with the shortest way to the target.

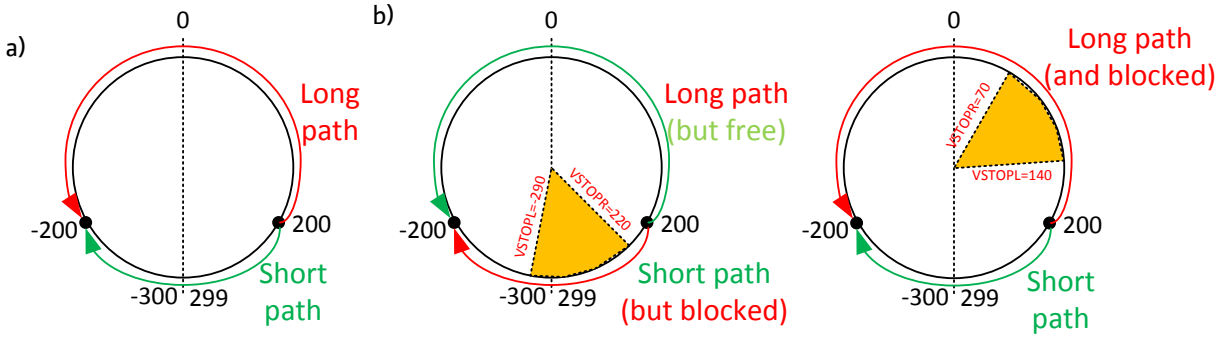


Figure 10.2 Circular movement (*X_RANGE* = 300), the green arrow depicts the path which is chosen for positioning: a) shortest path selection b) consideration of blocking zones

To match an incremental encoder in the same manner, select *circular_enc_en* = 1 (see Chapter 0)

10.6 Target Reached / Position Comparison

The TARGET_REACHED pin 31 will forward the TARGET_REACHED_Flag. Thus, if *XACTUAL* = *XTARGET*, TARGET_REACHED is active. The polarity could be configured via *invert_pol_target_reached* switch (bit16 of the *GENERAL_CONF* register 0x00).

10.6.1 Connecting several TARGET_REACHED pins (not implemented for TMC4361old)

TARGET_REACHED pin could be prepared the same way the INTR pin could be configured for one INTR signal transfer of several TMC4361 interrupt signals to the µC. Therefore, *set intr_tr_pu_pd_en* = 1 (bit 29 of the *GENERAL_CONF* register 0x00). Per default then, the pin will work efficiently as **Wired-Or** due to the fact that during TARGET_REACHED pin is inactive, the output is pulled weakly to the inactive pin level polarity. If the pin becomes active, it will pulled strongly towards the active polarity. Connecting several pins at one signal line for the µC will result in polarity change of the whole signal line if one of the TMC4361 TARGET_REACHED pins will be set active.
By setting *tr_as_wired_and* = 1 (bit 31 of the *GENERAL_CONF* register 0x00), the signal line will act as **Wired-And**. While the target position is not reached, the TARGET_REACHED pin will be pulled strongly towards the inactive polarity. During the active state, the pin will be pulled weakly at active polarity. That way, the whole signal line will be pulled to active state if all pins are forwarding the active polarity.

10.6.2 Position Comparison

Besides comparing the internal position *XACTUAL* or the external position *ENC_POS* with an arbitrary value which could be set in *POS_COMP* register 0x32, TMC4361 provides the opportunity (in contrast to TMC4361old) to compare one of the positions with other register values.

Basic Settings for position comparison (TMC4361 and TMC4361old)

- Choose a *POS_COMP* value. The position compare register provides 32 bits.
- Choose a compare parameter by setting *pos_comp_source* (bit25 of the *REFERENCE_CONF* register 0x01):
  - Set *pos_comp_source = 1 for ENC_POS.*
  - Set *pos_comp_source = 0* for *XACTUAL*.

The position compare process is permanently active. The stored *POS_COMP* position is compared with *XACTUAL* resp. *ENC_POS* automatically. If *POS_COMP = XACTUAL / ENC_POS* the status flag *POS_COMP_REACHED_F* becomes set and the *POS_COMP_REACHED* event becomes released, provided that switching to active state is done first.

- Additional, the output TARGET_REACHED can be used to report the state of position comparison instead of the target reached status. Therefore, set *pos_comp_output* = b'11 (bit24:23 of the *REFERENCE _CONF* register 0x01).

Further compare options (not implemented for TMC4361old)

Besides comparing *XACTUAL / ENC_POS* with *POS_COMP*, it is also possible to compare one of the positions with *X_HOME* or *X_LATCH / ENC_LATCH*. This will be controlled by setting *modified_pos_compare* (bit29:28 of the *REFERENCE _CONF* register 0x01). Finally, TMC4361 provides also the opportunity to compare the revolution counter *REV_CNT* with *POS_COMP*. However, only the selected compare pair will generate the *POS_COMP_REACHED_F*lag and the corresponding event. Following pairs could be chosen for this flag generation:

pos_comp_source: / *modified_pos_compare*	*'0'*	*'1'*
'00'	*XACTUAL* vs. *POS_COMP*	*ENC_POS* vs. *POS_COMP*
'01'	*XACTUAL* vs. *X_HOME*	*ENC_POS* vs. *X_HOME*
'10'	*XACTUAL* vs. *X_LATCH*	*ENC_POS* vs. *X_LATCH*
'11'	*REV_CNT* vs. *POS_COMP*	*ENC_POS* vs. *ENC_LATCH*

HINT Because it is possible that *ENC_POS* will miss a defined microstep due to a coarse encoder resolution, *POS_COMP_REACHED_F* will be also released if *ENC_POS* will only pass its compare position.

11 Ramp Timing & Synchronization

The TMC4361 provides various possibilities for ramp timing. Usually, every external register change via an SPI input is assigned immediately to the internal registers. With a proper start configuration of the TMC4361, ramp sequences without any intervening in between can be programmed. Three stages of ramp start complexity are available. First, the distinct ramp start whose triggers and consequences will be explained first. Two extensions are based on the start signal generation which could be used individually or combined. On the one hand, a complete shadow motion register set could be loaded in the current motion registers to start the next ramp with an altered motion profile. On the other hand, different target positions could be predefined which will be activated successively with opportunities of a cyclic pipeline and/or pipelining of different parameters.
Additionally, a further start state "busy" could be assigned to synchronize several motion controllers for one start event. This part differs seriously from TMC4361old. Please refer for the TMC4361old manual for the synchronization possibilities of this chip.

PINS AND REGISTERS: SYNCHRONIZATION

Pin names	Type		Remarks
START	Input, Output, Inout		External start input to get a start signal or external start output to indicate an internal start event.
Register name	Register address		Remarks
START_CONF	0x02	RW	The configuration register of the synchronization unit
SYNCHRO_SET	0x64	W	Synchronization set for closed loop considerations
START_OUT_ADD	0x11	RW	Additional active output length of external start signal
START_DELAY	0x13	RW	Delay time between start trigger and signal
X_PIPE0…7	0x38…0x3F	RW	Target positions pipeline
SH_REG0…13	0x40…0x4D	RW	Alternative Motion profile sets

11.1 Start Signal Generation

A ramp can be initiated using an internal or an external start trigger for the start signal generation. Note that a start trigger is not the start signal itself but the transition slope to the active start state. Now, for ramp start configuration consider the following steps:

1. Choose internal or external start trigger(s).
2. Adjust the timing of the start signal after a start trigger has been recognized.
3. Enable start signal processing.

11.1.1 Starting a Ramp via an Internal Start Trigger

There are different triggers available for an internal start signal. These triggers are assigned by the *trigger_events* switches (bits 5…8) of the *START_CONF* register 0x02. Every bit of *trigger_event* can be selected separately. Thus, more than one signal can trigger a start event.

trigger_events(8:5)	Description
b'0000	No start signal will be generated or processed further.
b'xxx0	Set *trigger_events*(0) = 0 for internal start trigger only. The START pin as output. (If this bit is set to 1, an external trigger is chosen and the START pin is used as input)
b'xx1x	TARGET_REACHED event is assigned as start signal for timer
b'x1xx	VELOCITY_REACHED event is assigned as start signal for timer
b'1xxx	POSCOMP_REACHED event is assigned as start signal for timer

11.1.2 Starting a Ramp via an External Start Trigger

There is one specific bit that has to be set for using an external trigger:

trigger_events(8:5)	Description
b'xxx1	Set *trigger_events*(0) = 1 for an external start trigger. Then, the START pin is assigned as input.

DEFINING THE ACTIVE VOLTAGE LEVEL AND THE LENGTH OF THE START PIN

The active voltage level of the START pin is defined by *pol_start_signal* (bit9 of START_CONF register 0x02).

EXAMPLES

1. Set *pol_start_signal* = 0 and *trigger_events*(0) = 1

Now, the voltage level transition from high to low triggers a start signal. The signal is further processed by the synchronization unit.

2. Set *pol_start_signal* = 1 and *trigger_events*(0) = 0

Now, start is used as output forwarding internal start signals with a high active level.

HINT Per default, the active start signal last for one clock cycle. To extend the length of the active start output, set the *START_OUT_ADD* register 0x11 appropriately. The value is given in clock cycles.

ATTENTION External start signals should be filtered.

11.1.3 Enabling delayed value transfer: *start_en* settings

To enable a start signal considerations for specific ramp parameters it is necessary to set *start_en* (bit4:0 of the START_CONF register 0x02).

A start signal can be used in different ways:

start_en(4:0)	Description
b'xxxx1	*XTARGET* is altered only after a start signal.
b'xxx1x	*VMAX* is altered only after a start signal.
b'xx1xx	*RAMPMODE* is altered only after a start signal.
b'x1xxx	*GEAR_RATIO* is altered only after a start signal.
b'1xxxx	Shadow register will be assigned after a start signal.

11.1.4 Adjustments Related to Start Signal Timing and Prioritizing

Every start switch can be enabled and disabled separately. In case an enable switch is set low, the particular register is changed immediately if the register is assigned by an SPI datagram. Using enable switches allows for setting specific points in time for altering register values. Thus, the assignment of SPI requests to the registers *XTARGET*, *VMAX*, *RAMP_MODE*, and *GEAR_RATIO* could be uncoupled from the SPI transfer itself. The assignment can be combined with trigger events which are related to the internal start signal generation.

START_DELAY – setting a delay time for the start signal after a trigger
For delaying an immediate ramp start, set *START_DELAY* register 0x13 to a reasonable value [clock cycles]. Then, the chosen *START_DELAY* value defines the time interval between the recognition of the chosen start trigger(s) and the internal start signal generation. For switching off an ongoing start delay countdown set *trigger_events* = b'0000.

***immediate_start_in* – prioritizing the external START signal**

For prioritizing the external START signal opposed to all other triggers set *immediate_start_in* = 1 (bit10 of *START_CONF* register 0x02). Thus, an external START is executed immediately after its recognition independently from a given *START_DELAY* time, an active timer, or other triggers.

ATTENTION If an external start trigger is not used and the START pin is also not used for communication with an external device, connect it to GND and select *pol_start_signal* = 1. Alternatively, connect START to V_{IO} supply and set *pol_start_signal* = 0.

11.1.5 Examples for Ramp Timing

The following three examples depict SPI datagrams, internal and external signal levels, corresponding velocity ramps, and additional explanations. SPI data is transferred internally at the end of each datagram.

EXAMPLE 1

Parameter	Setting	Description
RAMPMODE	b'101	The velocity value change is executed immediately. The new *XTARGET* value is assigned after *TARGET_REACHED* has been set and *START_DELAY* has been elapsed. A new ramp does not start at the end of the second ramp because there is no new *XTARGET* value assigned. START is used as output. The internal start signal is forwarded with a step length of (*START_OUT_ADD* + 1) clock cycles. This way, external devices could be synchronized.
start_en	b'00001	
trigger_events	b'0010	
START_DELAY	>0	
START_OUT_ADD	>0	
pol_start_signal	1	

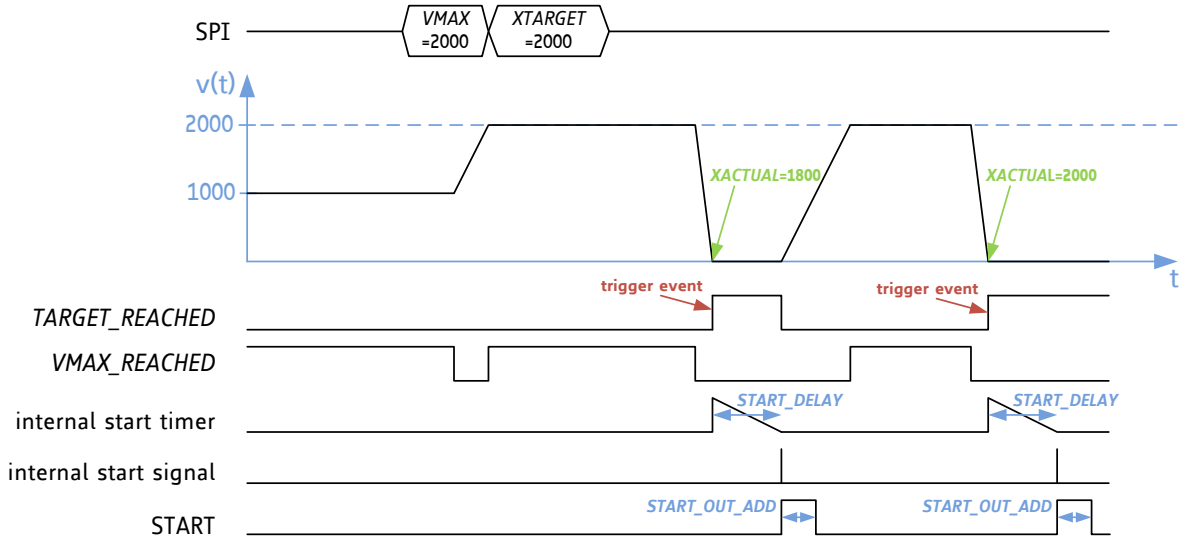


Figure 11.1 Start example 1

EXAMPLE 2

Parameter	Setting	Description
RAMPMODE	b'001	The velocity value and ramp mode value change will be executed after the first start signal. Because of the new ramp mode positioning mode and S-shaped ramps are activated and the ramp stops at target position. Due to a further target request, the ramp starts again. The active START output signal lasts only one clock cycle.
start_en	b'00111	
trigger_events	b'0110	
START_DELAY	>0	
START_OUT_ADD	0	
pol_start_signal	0	

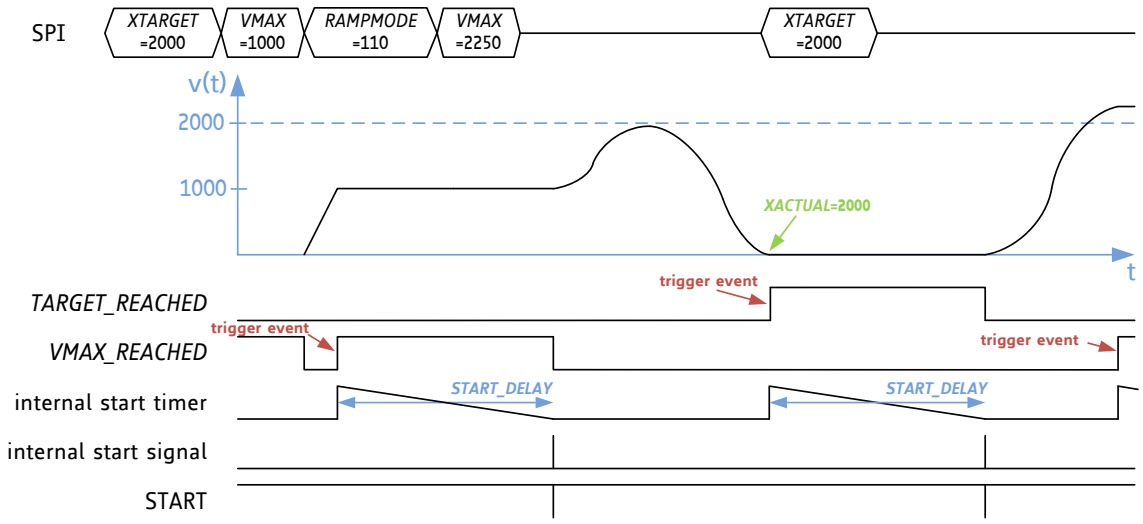


Figure 11.2 Start example 2

EXAMPLE 3

For this example start signal triggers have been prioritized due to the use of start timing via a *START_DELAY* setting and due to the setting *immediate_start_in* = 1.

Parameter	Setting	Description
RAMPMODE	b'000	When *XACTUAL* = *POSCOMP* the start timer is activated and the external start signal in between is ignored. The second start event is triggered due to the external start signal. The POSCOMP_REACHED event is ignored. The third start timer process is disrupted by the external START signal which is forced to be executed immediately due to the setting *immediate_start_in* = 1.
start_en	b'00010	
trigger_events	b'1001	
immediate_start_in	0	
START_DELAY	>0	
pol_start_signal	1	

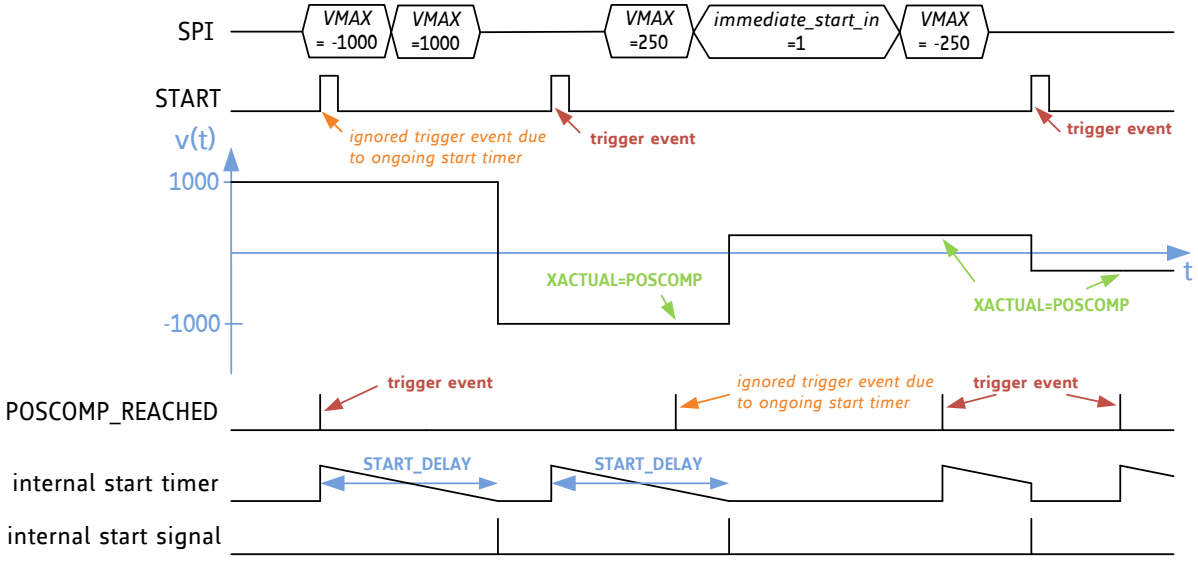


Figure 11.3 Start example 3

11.2 Shadow Register Set

Some applications requires a complete new ramp parameter set for a specific ramp situation resp. point in time. This could be achieved by setting *start_en*(4) = 1. It is also possible to write back the current motion profile into the shadow motion register set by setting *cyclic_shadow_resg* = 1 (bit18 of START_CONF register 0x02). Further on, four different options for shadow register assignment are available by setting shadow_option appropriately (bis17:16 of START_CONF register 0x02):

SHADOW OPTION 1: SET SHADOW_OPTION = B'00 FOR SINGLE-LEVEL SHADOW REGISTERS

Every relevant motion parameter will be altered at the next internal start signal.

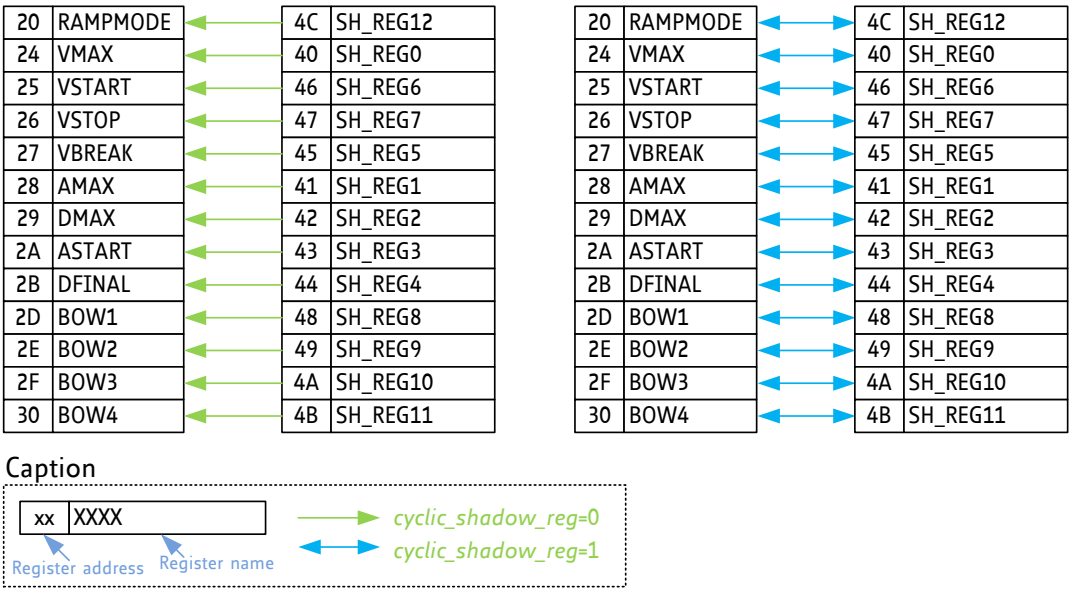


Figure 11.4 Single-level shadow register option

SHADOW OPTION 2: SET SHADOW_OPTION = B'01 FOR A DOUBLE-STAGE SHADOW REGISTER PIPELINE SUITABLE FOR S-SHAPED RAMPS

Seven relevant motion parameter for S-shaped ramps will be altered at the next internal start signal. The register are arranged to a double-stage pipeline. If cyclic shadow registers are used, the current value will be stored in the second stage with the next start signal, e.g. 0x28 (*AMAX*) will be written back to 0x48 (*SH_REG8*). The other ramp registers remain unaltered.

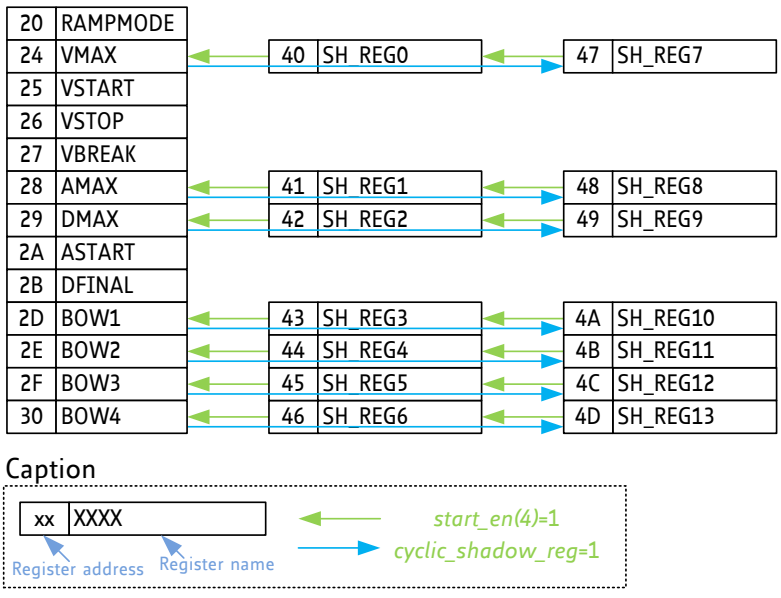


Figure 11.5 Double-stage shadow register option 1

SHADOW OPTION 3: SET SHADOW_OPTION = B'10 FOR A DOUBLE-STAGE SHADOW REGISTER PIPELINE SUITABLE FOR TRAPEZOIDAL RAMPS (VSTART INCLUDED)

Seven relevant motion parameter for trapezoidal shaped ramps will be altered at the next internal start signal. The register are arranged to a double-stage pipeline. If cyclic shadow registers are used, the current value will be stored in the second stage with the next start signal, e.g. 0x27 (*VBREAK*) will be written back to 0x4C (*SH_REG12*). The other ramp registers remain unaltered.

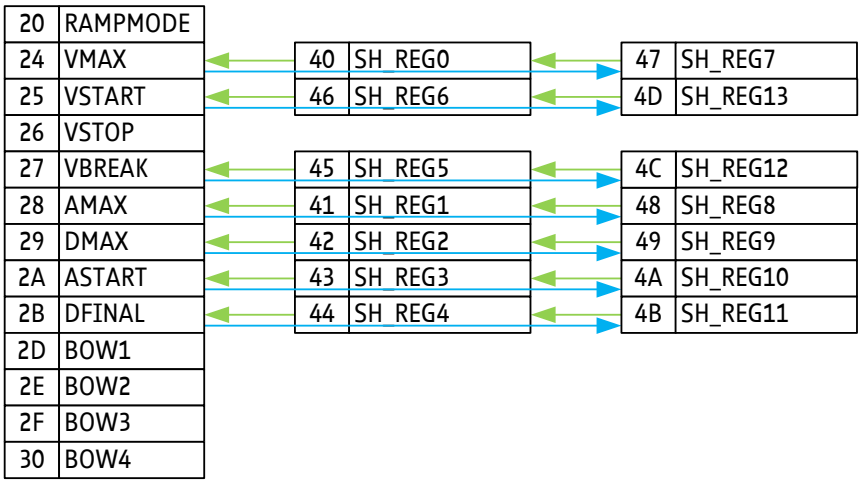


Figure 11.6 Double-stage shadow register option 2

SHADOW OPTION 4: SET SHADOW_OPTION = B'11 FOR A DOUBLE-STAGE SHADOW REGISTER PIPELINE SUITABLE FOR TRAPEZOIDAL RAMPS (VSTOP INCLUDED)

Seven relevant motion parameter for trapezoidal shaped ramps will be altered at the next internal start signal. The register are arranged to a double-stage pipeline. If cyclic shadow registers are used, the current value will be stored in the second stage with the next start signal, e.g. 0x2A (*ASTART*) will be written back to 0x4A (*SH_REG10*). The other ramp registers remain unaltered.

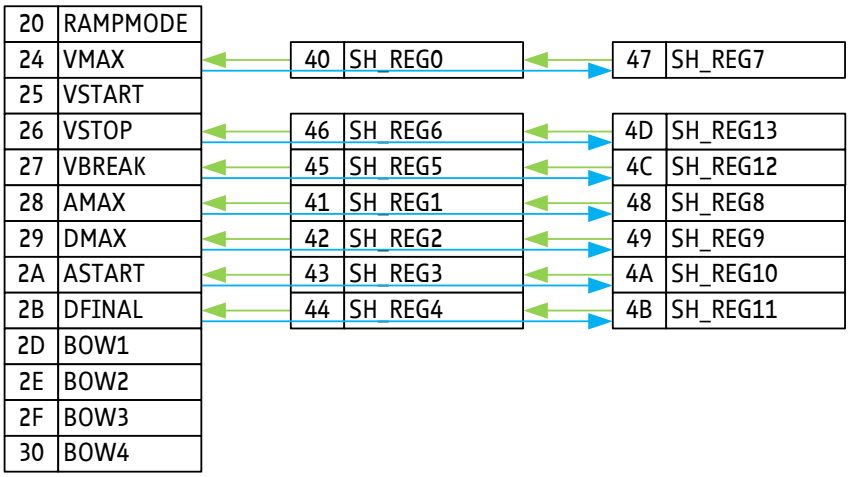


Figure 11.7 Double-stage shadow register option 3

ATTENTION The value of ramp parameters which are not affected by shadow register due to the selected shadow option will persist if the register will not be changed due to a SPI write access.

Up to fifteen internal start signals could be skipped before the shadow register transfer should be executed. Therefore, *SHADOW_MISS_CNT* (bit23:20 of START_CONF register 0x02) has to be set accordingly. Figure 11.8 depicts an example for the usage of *SHADOW_MISS_CNT* where the shadow register transfer is illustrated by an internal signal sh_reg_transfer. The current miss counter could be read out at register address *START_CONF*(23:20):

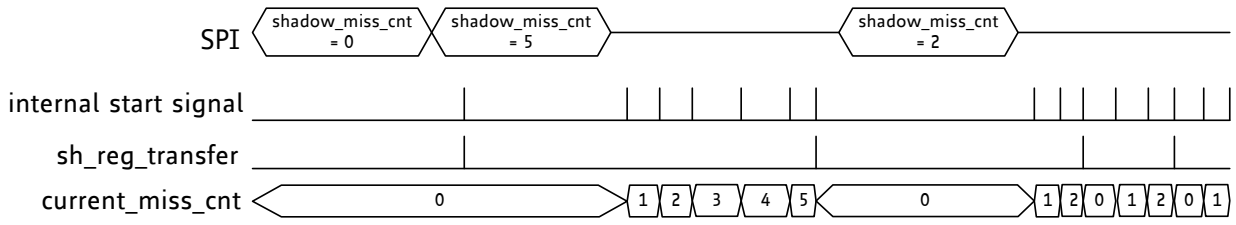


Figure 11.8 Usage of the *SHADOW_MISS_CNT* parameter to delay the shadow register transfer for several internal start signals

HINT If no cyclic values are enabled, the first stage of the pipeline will persist until its values will be changed. Thus, not altering the shadow registers will result in concurrence of the values of one pipeline if sufficient start signals have been occurred.
For example, *shadow_option* = b'01, *cyclic_shadow_reg* = 0, and *shadow_miss_cnt* = 0, will result in *BOW1* = *SH_REG3* = *SH_REG8* after two start signals and if every of these register values have not been altered so far.

ATTENTION Calculation to transfer the requested shadow BOW values into internal structures will require at most **(320 · f_{CLK}) [sec]**. Thus, it have to be assured that the point in time, when the values of the shadow registers will be transferred to valid ramp parameters, should be delayed after the last shadow bow assignment for this time span.
Following sequence could be adopted for shadow register assignment with a followed shadow register transfer (example: *shadow_option* = b'01)
1. Set *SH_REG0*, *SH_REG1*, *SH_REG2* (shadow register for *VMAX*, *AMAX*, *DMAX*)
2. Set *SH_REG3*, *SH_REG4*, *SH_REG5*, *SH_REG6* (shadow register for *BOW1…4*)
3. Wait for 320 · f_{CLK}
4. Shadow register transfer could be initiated

ATTENTION It is **strongly recommended** that the values of the **shadow ramp parameters** should be only **transferred during standstill** of the current ramp (*VACTUAL* = 0), especially if the *RAMP_MODE* will be changed. Anyhow, if the transfer should happen during motion, *VACTUAL* have to be constant for all ramp types. Further on, it have to be assured that *VACTUAL* will remain constant for a definite delay $t_{SHADOW\_TRANSFER}$ before any further *VMAX* or *XTARGET* changes will be assigned if S-Shaped ramps are performed due to recalculation reasons:

$$t_{SHADOW\_TRANSFER} > \frac{\sqrt{\max(BOW3[pps^3]; BOW4[pps^3]) \cdot VMAX[pps]}}{56 \cdot BOW3[pps^3]}$$

If positioning mode is selected, *XTARGET* have to be set accordingly to maintain a constant *VACTUAL* value.
It is also required that *VMAX* and it shadow counterpart have to be equal to avoid *VACTUAL* alteration during $t_{SHADOW\_TRANSFER}$.

11.3 Target Pipeline

The TMC4361 provides a target pipeline for sequencing subordinate targets during the drive. This way, a complex target structure can be easily arranged. Thus, *pipeline_en* = b'0001 have to be set (bit15:12 of START_CONF register 0x02). Now, the value in *X_PIPE0* becomes transferred to *XTARGET* at the next internal start signal. The complete a target pipeline *X_PIPE0…X_PIPE7* will be shifted forward step by step following the condition $X\_PIPE_n = X\_PIPE_{n+1}$.

This flexible target pipeline provides up to eight additional target positions which become transferred at the next specific start signal. The actual valid target position is written back to X_PIPE_x, where x is equal to the bit position of *XPIPE_REWRITE_REG* (bit31:24 of START_CONF register 0x02). More precisely, if *XPIPE_REWRITE_REG* = b'00010000, *X_PIPE4* = *XACTUAL* at the next internal start signal. If *XPIPE_REWRITE_REG* = b'00000000, *XTARGET* will not written back to any X_PIPE_n register. If multiple bits are set, *XTARGET* will written back to each of the selected X_PIPE_n registers.

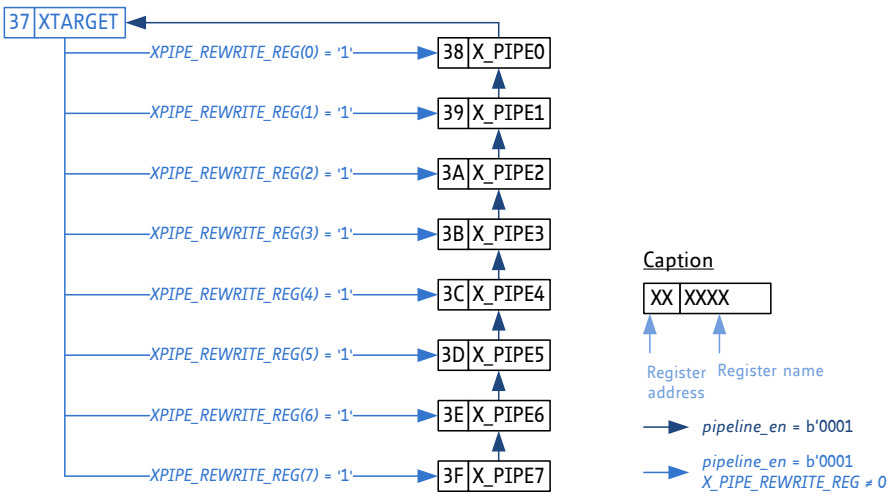


Figure 11.9 Flexible target pipeline

11.4 Partitioned Target Pipeline

The TMC4361 pipeline (registers 0x38…0x3F) could be split into up to four segments to provide a pipeline opportunity for other parameters whose defined parameter value change could be important for a continuous ramp motion and/or for reduced overhead synchronizing several motion controllers.
Besides the introduced pipelining of the target register, it is also possible to pipeline the *POS_COMP (0x32)*, the *GEAR_RATIO (0x12)*, and the *GENERAL_CONF (0x00)* registers.
The *POS_COMP* could be used to initiate a start signal generation during motion. Thus, it could be useful to pipeline this parameter to avoid the dependence on the SPI transfer speed. For instance, if the distance between two *POS_COMP* values is very close and the current velocity is high enough to miss the second value before the SPI transfer is finished, it would be better to change *POS_COMP* immediately after the start signal.
The same could be true for *GEAR_RATIO* which defines the step response on incoming step impulses. Some applications require very quick gear factor alteration of the slave controller whose immediate change at start signal appearance could be very useful in contrast to the alteration due to a SPI transfer.
Likewise, it is yet essential to change general configuration parameters at a defined point in time. A suitable application could be the defined transfer from a direct external control (*sd_in_mode* = b'01) to an internal ramp (*sd_in_mode* = b'00) or vice versa because the master/slave relationship will be interchanged.
Thus, following pipeline options are available which could be adjusted freely:

pipeline_en(3:0)	Description
b'xxx1	Pipeline for *XTARGET* is enabled.
b'xx1x	Pipeline for *POS_COMP* is enabled.
b'x1xx	Pipeline for *GEAR_RATIO* is enabled.
b'1xxx	Pipeline for *GENERAL_CONF* is enabled.

11.4.1 Pipeline sections

As stated before, the *pipeline_en* switches could be set freely. As a result the number of pipelines range from 0 to 4. This again has an impact of the pipeline depth whose variety ranges between eigth-stage, four-stage, three-stage and double-stage pipelines behind the target registers.
In the following table the arrangement and depth of the pipeline is allocated as regards pipeline setup. The transfer combination is also depicted which illustrates from which pipeline registers (X_PIPE0…7) the final target registers (*XTARGET*, *POS_COMP*, *GEAR_RATIO*, *GENERAL_CONF*) are feed.

pipeline_en(3:0)	Arrangement	Final transfer register			
		XTARGET	*POS_COMP*	*GEAR_RATIO*	*GENERAL_CONF*
b'0000	No Pipelining	-	-	-	-
b'0001	One 8-stage pipeline	*X_PIPE0*	-	-	-
b'0010		-	*X_PIPE0*	-	-
b'0100		-	-	*X_PIPE0*	-
b'1000		-	-	-	*X_PIPE0*
b'0011	Two 4-stage pipelines	*X_PIPE0*	*X_PIPE4*	-	-
b'0101		*X_PIPE0*	-	*X_PIPE4*	-
b'1001		*X_PIPE0*	-	-	*X_PIPE4*
b'0110		-	*X_PIPE0*	*X_PIPE4*	-
b'1010		-	*X_PIPE0*	-	*X_PIPE4*
b'1100		-	-	*X_PIPE0*	*X_PIPE4*
b'0111	Two 3-stage pipelines and One double-stage pipeline	*X_PIPE0*	*X_PIPE3*	*X_PIPE6*	-
b'1011		*X_PIPE0*	*X_PIPE3*	-	*X_PIPE6*
b'1101		*X_PIPE0*	-	*X_PIPE3*	*X_PIPE6*
b'1110		-	*X_PIPE0*	*X_PIPE3*	*X_PIPE6*
b'1111	Four double-stage pipelines	*X_PIPE0*	*X_PIPE2*	*X_PIPE4*	*X_PIPE6*

In the following several examples are depicted.

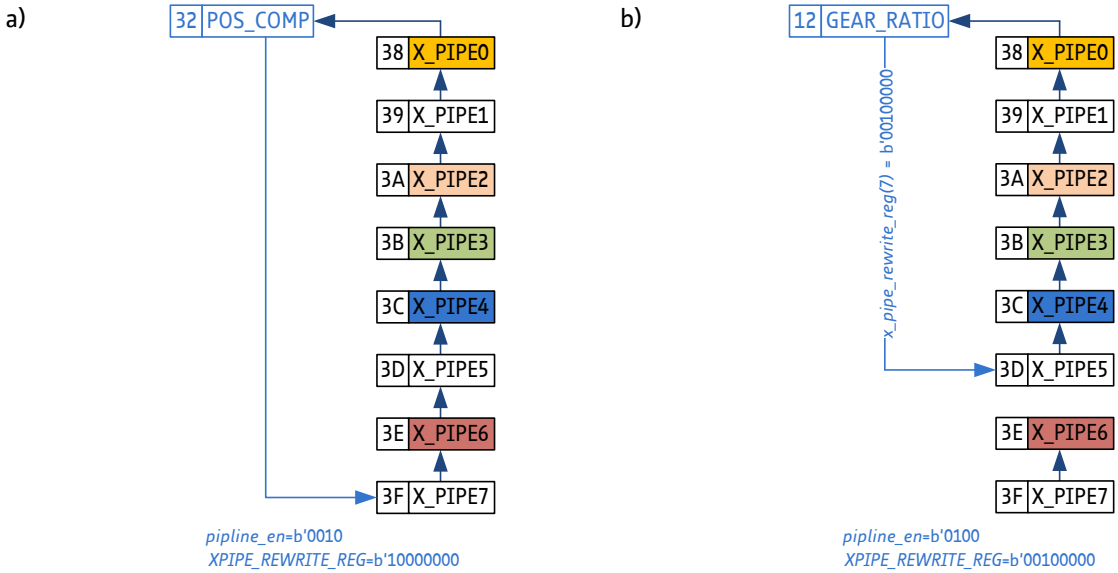


Figure 11.10 One pipeline with a) 8 stages and b) 6 stages behind the target register

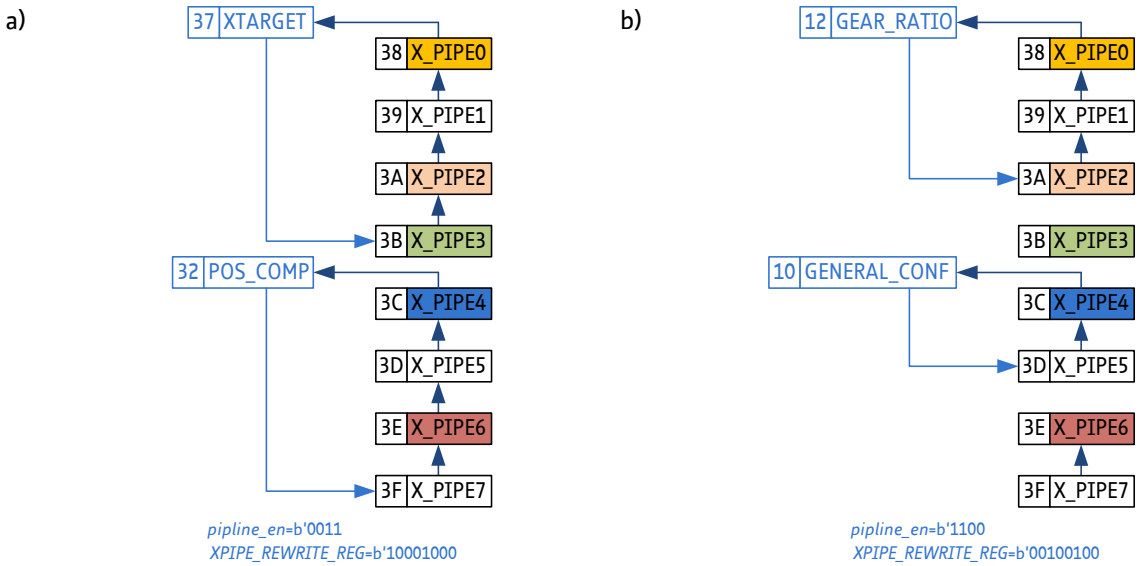


Figure 11.11 Two pipelines with a) each 4 stages and b) 3/2 stages behind the target registers

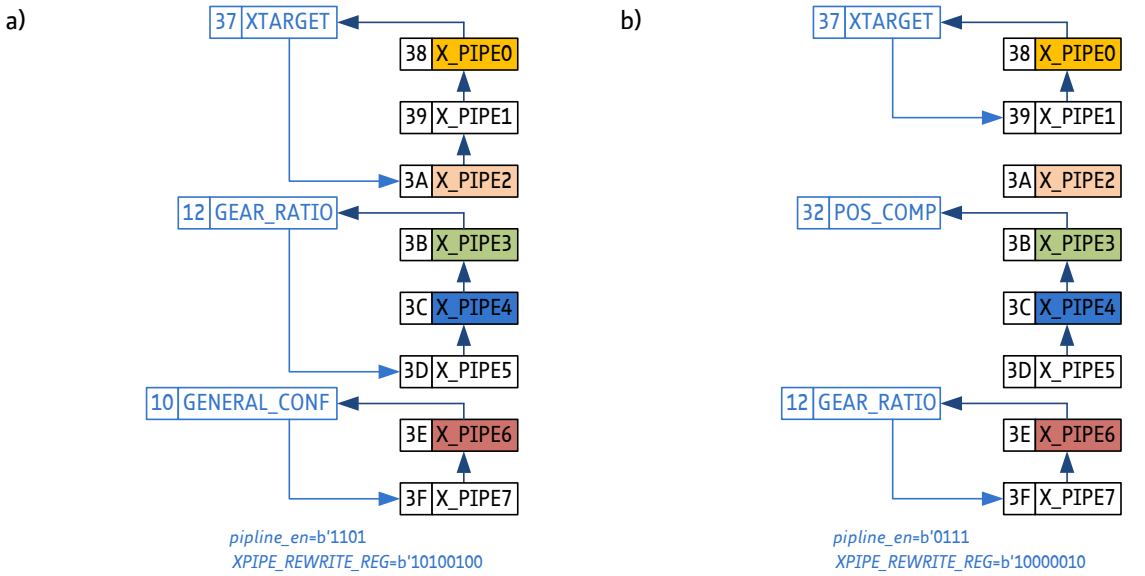


Figure 11.12 a) Two pipelines with 3 stages and one pipeline with 2 stages b) Two pipelines with 2 stages and one pipeline with 2 stages, but without writing data back

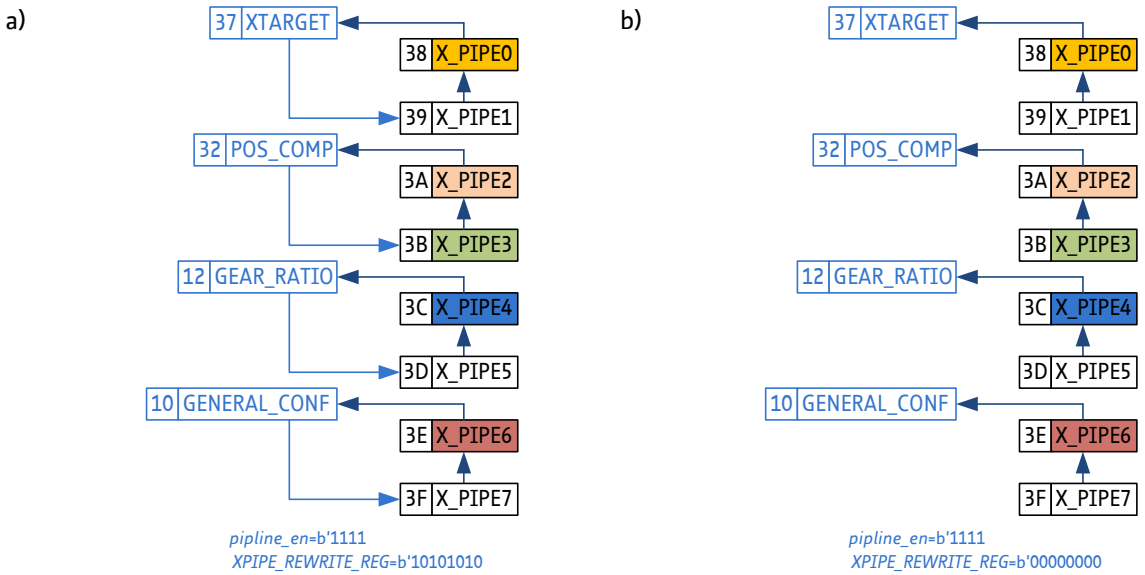


Figure 11.13 Four pipelines with 2 stages with a) and without b) writing back data

HINT As it have been depicted, the pipeline register which will get the current data of any of the target registers (*XTARGET*, *POS_COMP*, *GEAR_RATIO*, *GENERAL_CONF*) will be adjusted also by the *XPIPE_REWRITE_REG* register. By setting one of these bits to 1, the appropriate register will get the data of the particular target register. The assignment is dependent on the *pipeline_en* settings. The written back order will be always prior to the pipelining mechanism for *X_PIPE0…X_PIPE7*. Therefore, shorter cyclic pipelines than default ones are always possible.

11.5 Synchronizing Several Motion Controllers

Coming soon

ATTENTION If the START pin is connected with START pins of other TMC4361 devices, a **series resistor (e.g. 220 Ω)** should be connected **between the devices to limit the short circuit current** flowing if the configuration of the START signals will result in different voltage levels at the START pins of the different devices. A possible short circuit must last only for a **short time**.

12 Serial Data Output

The TMC4361 provides an SPI interface for initialization and configuration of the motor driver (additional to the Step/Dir output) before and during motor motion. Furthermore, it is possible to control TRINIAMIC stepper drivers during SPI motor drive. The SPI interface is used for principal tasks:

- Two current values of the integrated sine wave look-up table can be transferred at a time to the driver chip in order to energize the motor coils. This is done within each SPI datagram. A series of current values is transferred to move the motor. Values of the MSLUT (microstep sine wave look-up table) are adjusted using velocity ramp dependent scale values. This way, maximum amplitude current values are aligned to the requirements of certain velocity slopes.
- The TMC4361 integrates an adjustable cover register for configuration purposes. This way, TRINIAMIC motor driver chips and third parties chips can be adjusted with only little effort.

PINS AND REGISTERS: SPI TO MOTOR DRIVER

Pin names	Type		Remarks
NSCSDRV_*SDO*	Output		Chip select output to motor driver, low active
SCKDRV_*NSDO*	Output		Serial clock output to motor driver
SDODRV_*SCLK*	InOut as Output		Serial data output to motor driver
SDIDRV_*NSCLK*	Input		Serial data input from motor driver
STDBY_CLK	Output		Clock output, standby output, or ChopSync clock output
Register name	Register address		Remarks
GENERAL_CONF	0x00	RW	Bit 14→13, bit 19, bit 20, bit28
REFERENCE_CONF	0x01	RW	Bit 26, bit 27, bit 30
SPIOUT_CONF	0x04	RW	Configuration register for SPI output communication
STEP_CONF	0x0A	RW	Microsteps/fullstep, fullstep/revolution, and motor status bit event selection
DAC_ADDR	0x1D	RW	SPI addresses/commands which are put in front of the DAC values: CoilA: DAC_ADDR(15:0); CoilB: DAC_ADDR(31:16)
SPI_SWITCH_VEL	0x1F	RW	Velocity at which automatic cover datagram will be sent
CHOPSYNC_DIV			Chopper clock divider (bit 11→0)
FS_VEL	0x60	W	Velocity at which fullstep drive will be enabled
COVER_LOW	0x6C	W	Lower 32 bit of the cover register (µC to motor driver)
COVER_HIGH	0x6D	W	Upper 32 bit of the cover register (µC to motor driver)
COVER_DRV_LOW	0x6E	R	Lower 32 bit of the cover register (motor driver to µC)
COVER_DRV_HIGH	0x6F	R	Upper 32 bit of the cover register (motor driver to µC)
Register name	Register address		Remarks
CURRENT_CONF	0x05	RW	Current scaling configuration
SCALE_VALUES	0x06	RW	Current scaling values
STDBY_DELAY	0x15	RW	Delay time after standby mode is valid
FREEWHEEL_DELAY	0x16	RW	Delay time after freewheeling is valid
VDRV_SCALE_LIMIT	0x17	RW	Velocity setting for changing the drive scale value
UP_SCALE_DELAY	0x18	RW	Increment delay to a higher scaling value; 24 bit
HOLD_SCALE_DELAY	0x19	RW	Decrement delay to the hold scaling value; 24 bit
DRV_SCALE_DELAY	0x1A	RW	Decrement delay to the drive scaling value
BOOST_TIME	0x1B	RW	Delay time after ramp start when boost scaling is valid
SCALE_PARAM	0x7C	R	Actual scaling parameter; 8 bit
CURRENTA CURRENTB	0x7A	R	Actual current values of the MSLUT: SIN (coil A) and SIN90_120 (coil B); each 9 bit
CURRENTA_SPI CURRENTB_SPI	0x7B	R	Actual scaled current values of the MSLUT: SIN (coil A) and SIN90_120 (coil B); each 9 bit
Register name	Register address		Remarks
MSLUT registers	0x70…78	W	MSLUT values definitions
MSCNT	0x79	R	Current microstep position of the MSLUT
START_SIN START_SIN90_120 DAC_OFFSET	0x7E	RW	Sine start value of the MSLUT (bit 7→0) Cosine start value of the MSLUT (bit 23→16) Offset value for DAC output values (bit 31→24)

HINT For a good start with a TRINAMIC motor driver, setup *SPIOUT_CONF* register 0x04 properly. Thus, the TMC4361 offers presets for current transfer and automatic configuration routines if the correct driver is selected.
Status bits of TMC motor drivers are also transmitted to the status register of the motion controller.

12.1 Sine Wave Look-up Table

TMC4361 provides a programmable look-up table for storing the microstep current wave. As a default, the tables are pre-programmed with a sine wave, which is a good starting point for most stepper motors. Reprogramming the table to a motor specific wave allows drastically improved microstepping especially with low-cost motors. In order to minimize required memory and the amount of data to be programmed, only a quarter of the wave becomes stored. The internal microstep table maps the microstep wave from 0° to 90°. It becomes symmetrically extended to 360°. When reading out the table the 10-bit microstep counter *MSCNT* addresses the fully extended wave table. The table is stored in an incremental fashion, using each one bit per entry. Therefore only 256 bits (*ofs00* to *ofs255*) are required to store the quarter wave. These bits are mapped to eight 32 bit registers.
Each *ofs* bit controls the addition of an inclination *Wx* or *Wx*+1 when advancing one step in the table. As the wave can have a higher inclination than 1, the base inclinations *Wx* can be programmed to -1, 0, 1, or 2 using up to four flexible programmable segments within the quarter wave. This way, even a negative inclination can be realized. The four inclination segments are controlled by the position registers *X1* to *X3*.
When modifying the wave, care must be taken to ensure a smooth and symmetrical zero transition when the quarter wave becomes expanded to a full wave. The maximum resulting swing of the wave should be adjusted to a range of -248 to 248, in order to give the best possible resolution while leaving headroom for the hysteresis based chopper to add an offset.

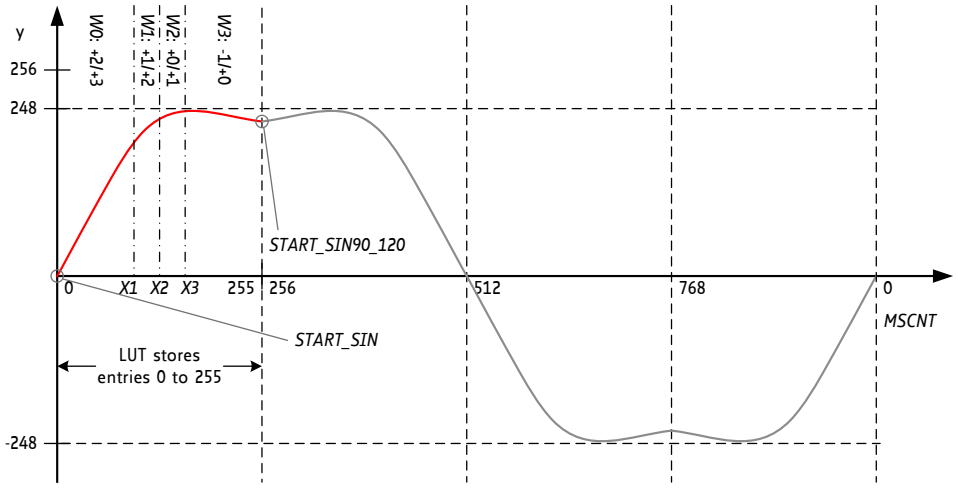


Figure 12.1 LUT programming example

When the microstep sequencer advances within the table, it calculates the actual current values for the motor coils with each microstep and stores them to the registers *CURRENTA* and *CURRENTB*. However the incremental coding requires an absolute initialization, especially when the microstep table becomes modified. Therefore, *CURRENTA* and *CURRENTB* become initialized whenever *MSCNT* passes zero.

TWO REGISTERS CONTROL THE STARTING VALUES OF THE TABLES:

- As the starting value at zero is not necessarily 0 (it might be 1 or 2), it can be programmed into the starting point register *START_SIN*.
- In the same way, the start of the second wave for the second motor coil needs to be stored in *START_SIN90_120*. This register stores the resulting table entry for a phase shift of 90° for 2-phase stepper motors.

12.1.1 Programming the Incremental Microstep Table

For understanding the background of the incremental coding of the microstep table, it is good to have an idea of the characteristics of the microstep wave.

A MICROSTEP TABLE FOR A TWO PHASE MOTOR HAS CERTAIN CHARACTERISTICS:

1. It is in principle a reverse characteristic of the motor pole behavior.
2. It is a smoothened wave to provide a smooth motor behavior. There are no jumps within the wave.
3. The phase shift between both phases is exactly 90°, because this is the optimum angle of the poles within the motor.
4. The zero transition is at 0°. The curve is symmetrical within each quadrant (like a sine wave).
5. The slope of the wave is normally positive, but due to torque variations it can also be (slightly) negative.
6. But it must not be strictly monotonic as the example in the previous chapter shows.

Considering these facts, it becomes clear that the wave table can be compressed. The incremental coding used in the TMC4361 uses a format which reduces the required information per entry of the 8 bit by 256 entry wave table to slightly more than a single bit.

INCREMENTAL ENCODING

The principle of incremental encoding just stores the difference between the actual and the next table entry. To have an absolute start value, the first entry is directly stored (*START_SIN*). For the ease of use, also the first entry of the shifted table for the second motor phase is stored (*START_SIN_90_120*).
The TMC4361 provides four inclination segments (0, 1, 2, and 3) with the base inclinations (W0, W1, W2, and W3) and the segment borders (0, X1, X2, X3, and 255).

Inclination segment	Base inclination	Segments
0	W0	0… X1
1	W1	X1… X2
2	W2	X2… X3
3	W3	X3… 255

Table 12.1 Inclination segments of TMC4361

EXPLANATORY NOTES AND EXAMPLES

Using a single bit per table entry allows any inclination between 0 and 1. E.g., a *0-bit* can mean *do not add anything* and a *1-bit* can mean *add one*. This allows describing a digital slope of 0° (all bits zero) to 45° (all bits one).
It becomes clear, that higher inclinations are necessary. However, the inclination will not drastically change from point to point. Therefore, the wave can be divided into up to four segments with different base inclinations.
Using a base inclination of one, a *0-bit* means *add one* and a *1-bit* means *add two*. This way, a slope between 45° (all bits zero) and 77.5° is yielded (all bits one).

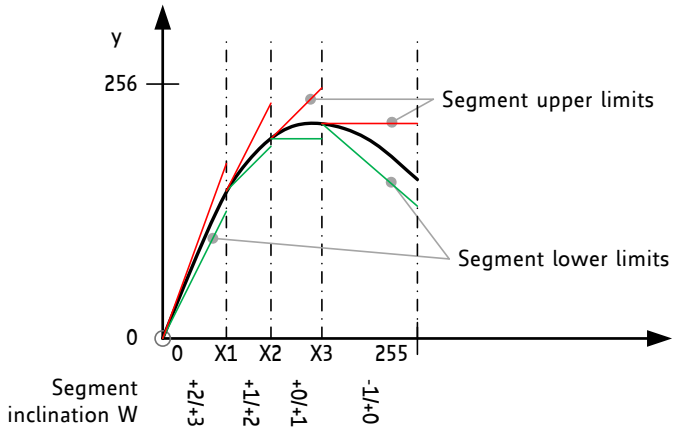


Figure 12.2 Wave showing segments with all possible base inclinations (highest inclination first)

HINT The base inclinations can be set between -1 (falling slope) and +2. This way, slopes between -45° and 78.75° can be described.
The default sine wave table in TRINAMIC drivers uses one segment with a base inclination of 1 and one segment with a base inclination of 0.

EXAMPLE

CONSIDER THE GIVEN CONDITIONS:

The microstep table for the standard sine wave begins with the eight entries (0 to 7) {0, 1, 3, 4, 6, 7, 9, 10 ...} etc.
The maximum inclination in this area is 2 (1+2=3).
The minimum inclination in these eight entries is 1.
The start value is 0.
Advancing in the table, the first time the inclination becomes lower than +1 is from position 153 to position 154. Both entries are identical.
The calculated value for position 256 (start of cosine wave) is 247.

THEREFORE, THE FOLLOWING SETTINGS NEED TO BE MADE:

- Set a starting value *START_SIN*=0 matching sine wave entry 0.
- Set a base inclination range of *W0*: +1 / +2 (*W0*=%10), valid from 0 to *X1*.
- Calculate the differences between each two entries: {+1, +2, +1, +2, +1, +2, +1,...}
- Set the microstep table entries *ofsxx* to 0 for the lower value (+1), 1 for the higher value (+2). Thus, the first seven microstep table entries *ofs00* to *ofs06* are: {0, 1, 0, 1, 0, 1, 0 ...}
- Latest at position 153, the inclination must be lowered. Use the next inclination range 1 with *W1*: +0 / +1 (*W1*=%01). Therefore, *X1* becomes set to 153 in order to switch to the next inclination range. Thus, starting from position 153, an offset *ofsxx* of 0 means add nothing, 1 means add +1.
- *START_SIN90_120* becomes equal to the value at position 256, i.e. 247.
- As the wave does not more have segments with different inclinations, the remaining inclination ranges *W2* and *W3* shall be set to the same value as W1, and *X2* and *X3* can be set to 255. This way, only two inclination segments are effective.

OVERVIEW OF EXAMPLE												
Microstep number	0	1	2	3	4	5	6	7	...	153	154	...
Desired table entry	0	1	3	4	6	7	9	10	...	200	200	...
Difference to next entry	1	2	1	2	1	2	1	...	...	0	...	...
Required segment inclination	+1	+1	+1	+1	+1	+1	+1	...	...	+0	...	...
Offs bit entry	0	1	0	1	0	1	0	...	...	0	...	...

12.2 SPI Output Parameters

The TMC4361 provides SPI output parameters to adjust a proper communication with the motor driver. Set *serial_enc_out_enable* = 0 (bit24 of *GENERAL_CONF* register 0x00) to enable the SPI output communication. The TMC4361 generates the necessary SPI output clock frequency and forwards it to the SCKDRV_NSDO output pin. The low phase of the serial clock is set with *SPI_OUT_LOW_TIME* (bit23:20 of *SPIOUT_CONF* register 0x04), whereas *SPI_OUT_HIGH_TIME* (bit27:24 of *SPIOUT_CONF* register 0x04) sets the high phase. Additionally, an *SPI_OUT_BLOCK_TIME* (bit31:28 of *SPIOUT_CONF* register 0x04) can be set for a minimum time period where no new datagram will be sent after the last SPI output datagram. During this inactive phase SCKDRV_NSDO stays high. All three SPI output parameters consist of 4 bit and represent a number of clock cycles.

Pins which are affected by SPI output communication

NSCSDRV_SDO	low active chip select signal
SCKDRV_NSDO	SPI output clock
SDODRV_SCLK	used as output to transfer the datagram to the motor driver
SDIDRV_NSCLK	receives the response from the motor driver. The response is sampled during the data transfer to the motor driver.

Minimum and Maximum Time Period

The minimum time period for all three parameters is $1/f_{CLK}$. If an SPI output parameter is set to 0 it becomes altered to 2 clock cycles internally. A maximum time period of $15/f_{CLK}$ can be set for all three parameters.
Thus, SPI clock frequency f_{SPI_CLK} covers the following range: $f_{CLK}/30 \le f_{SPI\_CLK} \le f_{CLK}/2$. The timing of the SPI output communication is illustrated in Figure 12.3.

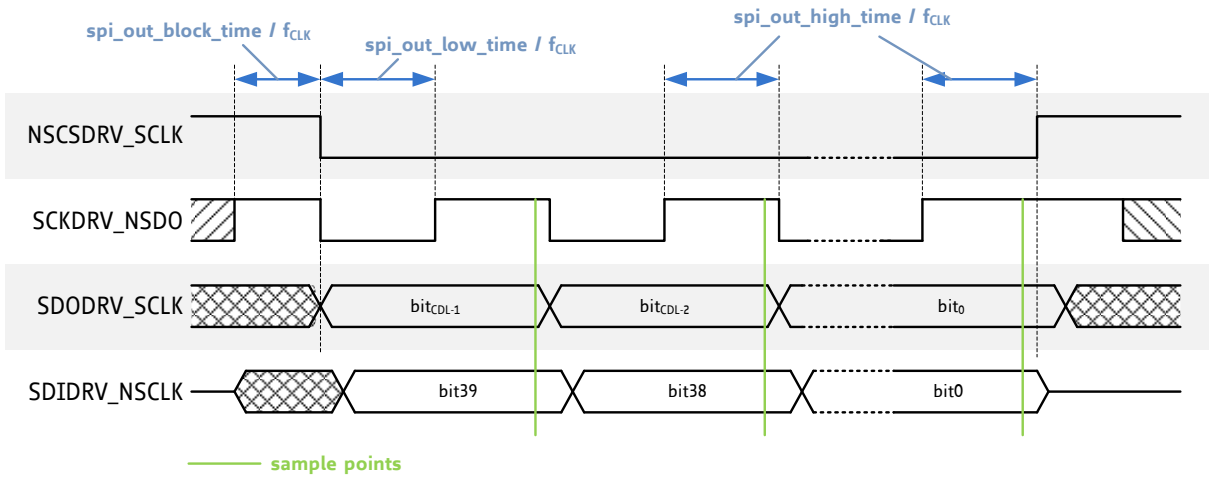


Figure 12.3 SPI output datagram timing (CDL – cover_data_length)

COVER_DONE

At the end of a successful data transmission, the event *COVER_DONE* becomes set. This indicates that the cover register data have been sent to the motor driver and that received responses have been stored in the registers *COVER_DRV_HIGH* (0x6E) and *COVER_DRV_LOW* (0x6F). *COVER_DRV_HIGH* and *COVER_DRV_LOW* form the cover response register.
The event *COVER_DONE* becomes also set after a successful current datagram transmission.

64 BIT SPI COVER REGISTERS FOR COMMUNICATION BETWEEN µC AND DRIVER

The 64 bit SPI cover register is separated into two 32 bit registers - *COVER_HIGH* (0x6C) and *COVER_LOW* (0x6D). Using the cover registers, an additional SPI communication channel between microcontroller and motor driver is not needed. The total length of the cover register can be set by *COVER_DATA_LENGTH*. If this parameter is set higher than 64, the cover register data length is still 64 bits at its maximum. The LSB (last significant bit) of the whole cover register is located at *COVER_LOW*(0). Thus, if less than 33 bits are required for SPI communication, only *COVER_LOW* respectively a part of it will be transmitted (in accordance to *COVER_DATA_LENGTH*). The cover register and the datagram structure are illustrated in Figure 12.4.

Every SPI communication starts with the most significant bit (MSB):

- MSB is *COVER_LOW*(*COVER_DATA_LENGTH* - 1) if *COVER_DATA_LENGTH* < 33.
- MSB is *COVER_HIGH*(*COVER_DATA_LENGTH* - 33) if *COVER_DATA_LENGTH* ≥ 33.

HINT Similar to *COVER_LOW* and *COVER_HIGH*, the motor driver response is divided in the registers *COVER_DRV_LOW* and *COVER_DRV_HIGH*. The composition of the response cover register and the positioning of the MSB follow the same structure.

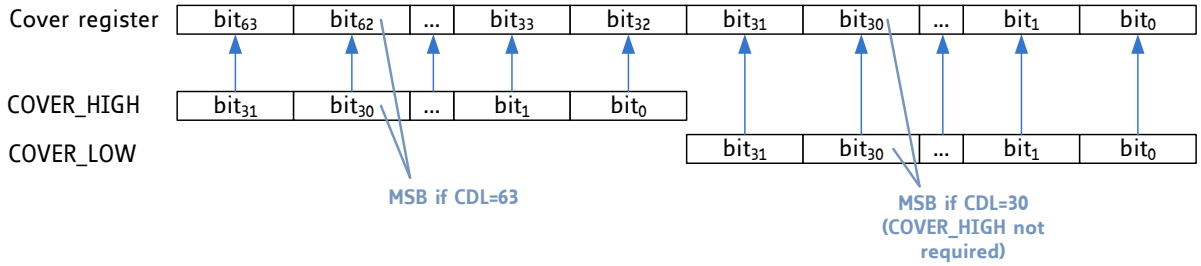


Figure 12.4 Cover data register composition (CDL – cover_data_length)

12.3 Automatic cover datagrams

TMC4361 provides the opportunity to send automatically cover datagrams if a certain ramp velocity is reached. If *automatic_cover* is set to 1 the data in *COVER_HIGH* will be transferred if *VACTUAL* passes *SPI_SWITCH_VEL* to an absolute higher velocity. Whereas, *COVER_LOW* will be sent if *VACTUAL* passes *SPI_SWITCH_VEL* to absolute lower values. Obviously, the SPI slave should not have expect datagrams with more than 32 bits.

12.4 Current Datagrams

TMC4361 uses the introduced internal microstep look-up table (MSLUT) for providing current data for the motor driver. With every step initialized by the ramp generator the *MSCNT* value becomes increased or decreased, dependent on the ramp direction. The *MSCNT* register 0x79 (readable value) contains the current microstep position of the sine value. Accordingly, the current values *CURRENTA* (0x7A) and *CURRENTB* (0x7B) are altered.

In case the output configuration of the TMC4361 allows for automatic current transfer an updated current value leads to a new datagram transfer. This way, the motor driver always receives the latest data. The length for current datagrams could be set automatically and the TMC4361 converts new values into the selected datagram format, usually divided in amplitude and polarity bit for TMC motor drivers.

Note that the TMC23x and TMC24x only forward new current data if the upper five bits of one of the two 9 bit current values have changed. This is because TMC23x and TMC24x current data consist of four bit current values and one polarity bit for each coil. Further on, TMC23x and TMC24x current datagrams forward mixed decay bits. These bits can be set with *mixed_decay* (bit5:4 of *SPIOUT_CONF* register 0x04). Please refer to the TMC23x/TMC24x datasheets to get more information about setting mixed decay bits correctly.

12.5 TMC Motor Driver

For connecting a TMC stepper motor driver proceed as follows:

The TMC4361 is able to set the cover register length automatically. Therefore, set *COVER_DATA_LENGTH* = 0 (bit19:13 of *SPIOUT_CONF* register 0x04). Now, the cover register length is set according to the chosen *spi_output_format* setting (bit3:0 of *SPIOUT_CONF* register 0x04). *spi_output_format* is the essential parameter for choosing predefined SPI default settings for the particular TMC motor driver.

TMC Stepper Motor Driver and Settings

TMC motor driver	*spi_output_format* 3→0	Automatic current datagram transfer	Cover register length *COVER_DATA_LENGH*=0
TMC23x	b'1000	✓	12
TMC24x	b'1001	✓	12
TMC26x/389 *SPI output for conf. only*	b'1010 b'1011	✓ -	20 20
TMC21xx/51xx *SPI output for conf. only*	b'1101 b'1100	✓ -	40 40

12.5.1 Switching from µSteps to Fullsteps

TMC4361 provides switching to fullstep mode if the absolute velocity value *VACTUAL* exceeds the parameter *FS_VEL*, which is the minimum fullstep velocity. In case, e.g., the Step/Dir output is used, switching from microsteps to fullsteps can lead to a step rate which is 256 times lower than before, assumed that the highest microstep resolution is set. To indicate this microstep resolution change to the microcontroller, the event *FS_ACTIVE* becomes released and thus the microcontroller can adapt the motor driver configuration properly.

Hint For enabling fullstep drive set *fs_en*=1 (bit19 of *GENERAL_CONF* register 0x00).

TMC260, TMC261, TMC262, TMC2660, TMC389, TMC21xx/51xx: Automatic Fullstep Switchover

These advanced motor driver chips offer two interfaces for communication with the motion controller: SPI and Step/Dir. The TMC4361 provides related data for both interfaces concurrently. Decreasing the microstep resolution during a velocity ramp has to be done very carefully. For the ease of use, the TMC4361 provides configuring TMC motor drivers automatically.

SPI output used for configuration and current datagrams

For this configuration set *spi_output_format* = b'1010 (TMC26x/389) resp. b'1101 (TMC21xx/51xx).
Now, current values become switched to fullstep values if |*VACTUAL*| > *FS_VEL*, the internal microstep position of the TMC4361 suits, and *fs_en* = 1 has been set before. Consistently, a switchback from fullsteps to microsteps becomes executed if |*VACTUAL*| < *FS_VEL*.

Step/Dir interface used for moving the motor and SPI output only used for configuration

For this configuration set *spi_output_format* = b'1011 (TMC26x) resp. b'1100 (TMC21xx/51xx), *fs_en* = 1, and *fs_sdout* = 0.

Attention Note that *fs_sdout* (bit20 of *GENERAL_CONF* register 0x00) is only to be used if a motor driver does not provide switching between fullsteps and microsteps automatically. Setting this bit to active state, automatic fullstep interchange will be disturbed.

A continuous polling for SPI datagrams is necessary to get status data from the drivers. Therefore, set *disable_polling* = 0 (bit6 of *SPIOUT_CONF* register 0x04). By setting *POLL_BLOCK_MULT* (bit12:7 of *SPIOUT_CONF* register 0x04) properly, the time between two consecutive polling datagrams becomes

extended to (*POLL_BLOCK_TIME* + 1) · *SPI_OUT_BLOCK_TIME* / f_{CLK} during polling. A high fullstep frequency requires a short SPI datagram polling time.

ATTENTION The fullstep switch for the TMC26x and TMC389 requires a correct assignment of the read selection bits in the driver registers. If these bits are not set to b'00 the transition to fullsteps cannot not be executed due to the fact that the TMC4361 does not receive any microstep data from the driver.

If fullstep drive is requested and |*VACTUAL*| > *FS_VEL*, the motor driver (TMC26x or TMC21xx/51xx) is polled to recognize the correct point in time to switch to full steps. This moment becomes reached when the microstep position of the motor driver equals a fullstep position. The same operation is carried out if fullstep drive has to be switched back to microstep drive.

TMC23x and TMC24x: Automatic Switchover to Fullsteps

Set *spi_output_format* = b'1000 for TMC23x or *spi_output_format* = b'1001 for TMC24x motor drivers. Now, current values become switched to fullstep values if |*VACTUAL*| > *FS_VEL*, the internal microstep position of the TMC4361 suits, and *fs_en* = 1 has been set before. Consistently, a switchback from fullsteps to microsteps becomes executed in case |*VACTUAL*| < *FS_VEL*.

Changing the Microstep Resolution

By altering the microstep resolution from 256 (*MSTEP_PER_FS* = b'0000) to a lower value, an internal step results in more than one MSLUT step. If, e.g., the microstep resolution is set to 64 (*MSTEP_PER_FS* = b'0010), the MSCNT becomes in-/decreased by 4 for one internal step. Accordingly, the passage through the MSLUT skips three current values for each internal step to match the new microstep resolution.

12.5.2 How to Use the Current Scale Parameter via SPI Output

Further automatic driver configuration for *spi_output_format* = b'1100 and *spi_output_format* = b'1011 can be used by setting *scale_val_transfer_en* = 1 (bit5 of *SPIOUT_CONF* register 0x04). Using this feature, the current scale parameter *SCALE_PARAM* is sent via SPI output to the motor driver. Pre-settings (made before via cover datagrams) become considered if the particular registers become overwritten with the new scaling value or with the new microstep resolution. The configuration of automatic scaling will be explained in chapter 12.6.

12.5.3 Configuration for the TMC389 3-Phase Stepper Driver

If a TMC389 is connected to the SPI output and a microstep resolution of 256 is set, a three phase stepper output for coil B can be generated. Therefore, set *three_phase_stepper_en* = 1 (bit4 of *SPIOUT_CONF* register 0x04). Now, the *CURRENTB* and *CURRENTB_SPI* values are shifted for 120° (instead of 90° for 2-phase stepper motors).

12.5.4 ChopSync™ Configuration for TMC23x/TMC24x Stepper Drivers

- Connect the clock output signal STDBY_CLK of TMC4361 to the OSC input of the TMC23x/24x stepper driver. (This input is used as PWM clock input.) Now, the chopSync feature can be used for a fast and smooth drive.
- Set *stdby_clk_pin_assignment* = b'10 (bit14:13 of *GENERAL_CONF* register 0x00) to forward the internal ChopSync™ clock to the STDBY_CLK output pin.
- The clock frequency of the PWM is assigned by setting *CHOPSYNC_DIV* (0x1F). The internal clock of TMC4361 is divided by this parameter to assign the PWM frequency f_{OSC} = f_{CLK} / *CHOPSYNC_DIV* with 96 ≤ *CHOPSYNC_DIV* ≤ 818.
- If *stdby_clk_pin_assignment* = b'11 is set the internal clock is forwarded via STDBY_CLK and the chopSync™ feature is not available.

12.5.5 Motor Driver Status Bits and Stall Detection

When a TMC motor driver receives a current datagram (transmitted via the SPI output of the TMC4361) status data is sent back to the TMC4361 controller immediately. These responses from the driver are stored in the *cover response register* which consists of *COVER_DRV_LOW* (0x6E) and if necessary *COVER_DRV_HIGH* (0x6F). Additionally, motor driver status bits are forwarded to the *STATUS* register. Refer to chapter 19 for detailed information about status bits of TMC motor driver chips.

Events and Interrupts Based on Motor Driver Status Bits

- The MSTATUS_SELECTION (bit23:16 of *STEP_CONF* register 0x0A) can be set in a way that selected motor driver status bits release an *MOTOR* event if a status bit becomes active.
- For generating an interrupt the motor driver event *EVENTS*(30) can be configured as interrupt source.

Stall Detection Handling

- TMC motor driver chips always return the stall detection status to the TMC4361 motion controller in response to every received SPI datagram. In most cases, one bit indicates that a motor stall occurred.
- If
  - *stop_on_stall* = 1 (bit26 of *REFERENCE_CONF* register 0x01) is set and
  - |*VACTUAL*| > *VSTALL_LIMIT* (0x67)

  an active stall status is handled as a stop event with a hard stop.
- The subsequently released *stop_on_stall* event immediately stops the currently valid velocity ramp.
- For starting a new velocity ramp set *drv_after_stall* = 1 (bit27 of *REFERENCE_CONF* register 0x01). Now, the *stop_on_stall* event becomes reset.
- The *drv_after_stall* switch has to be set back manually.

ATTENTION As long as the current absolute velocity is below *VSTALL_LIMIT*, only the ACTIVE_STALL flag will be released and no hard stop will be executed.

TMC26xx, TMC21xx/51xx, and TMC389

Motor driver status bits as response from current datagrams are received automatically. One stall detection status bit is returned to the microcontroller in response to every received SPI datagram.

TMC24x stallGuard Characteristics

The TMC24x forwards stallGuard values (=LD2&LD1&LD0) instead of one stallGuard status bit. These bits represent an unsigned value between 0 and 7. The lower the value the higher is the mechanical load. By setting *STALL_LOAD_LIMIT* (bit10:8 of *SPIOUT_CONF* register 0x04) properly, a stall is indicated when (LD2&LD1&LD0) ≤ *STALL_LOAD_LIMIT* which results in a hard stop if *stop_on_stall* = 1.
Set *stall_flag_instead_of_uv_en* = 1 (bit7 of *SPIOUT_CONF* register 0x04) to replace the undervoltage status bit in the *STATUS* register with the stall status of TMC24x drivers.
A standby datagram is sent to the TMC24x stepper driver if *stdby_on_stall_for_24x* = 1 (bit6 of *SPIOUT_CONF* register 0x04) and a *stop_on_stall* event occurs. This datagram sets current values to 0 which results in a power down of the TMC24x motor driver.

12.6 Connecting Driver Chips from Other Parties

The TMC4361 provides also configuration data for driver chips of other companies via the cover registers. Please note that the *COVER_DATA_LENGTH* has to be set properly. Furthermore, it is possible to support automatic current data transfer. The following format settings can be chosen:

Output formats	*spi_output_format*	Automatic current datagram transfer	Automatic cover register length (if *COVER_DATA_LENGH*=0)
SPI output off	b'0000	-	1
Signed current data	b'0101	✓	1
Unsigned scaling factor	b'0100	-	1
DAC scaling factor	b'0110	-	1
DAC absolute values	b'0010 / b'0011	✓	1
DAC adapted values	b'0001	✓	1

Comments on the Table

- *spi_output_format* = b'0000 switches off the SPI output.
- *spi_output_format* = b'0101 leads to a transfer of both signed current values one after the other in an 18 bit datagram.
- With *spi_output_format* = b'0100, the 8 bit scaling factor is transmitted if it has been altered. This scaling data could also be transmitted for a DAC by setting *spi_output_format* = b'0110, assumed that the SPI capabilities of the DAC fit.
- *spi_output_format* = b'0010 converts the current values for the SPI capable DAC into absolute values. The current phases of both coils are forwarded via the STPOUT (coilA) and DIROUT (coilB) outputs. A phase bit polarity of 0 indicates a positive value.
- *spi_output_format* = b'0011 converts the current values for the SPI capable DAC into absolute values. The current phases of both coils are forwarded via the STPOUT (coilA) and DIROUT (coilB) outputs. A phase bit polarity of 0 indicates a negative value.
- With *spi_output_format* = b'0001 the currents are mapped to an unsigned value. Therefore, a value of 256 is added to the signed current values. Thus, the current value 0 results in a 9 bit value of b'10000000 whereas the minimum value of -256 is exported as b'000000000 and the maximum value of 255 as b'111111111.
- Additionally *fs_sdout* can be set to 1 in case switching from microsteps to fullsteps and back is desired.

DAC Value Offset and Length of Datagram

- An offset can be added for the values of both coils by setting *DAC_OFFSET* (bit31:24 of register 0x7E) to compensate for a shifted base line.
- Usually, SPI transfers require an address or a command in front of a transmitted value. The length of the prefixed command or address can be assigned by setting *DAC_CMD_LENGTH* (bit11:7 of *SPIOUT_CONF* register 0x04).
- The bit stream which constitutes the command or address can be stored in the *DAC_ADDR* register 0x1D with 16 bits for both coils separately. Due to the transfer of only one value per datagram, two datagrams are sent in a row: first the coilA command and value are sent and afterwards the coilB command and value. If the cover register length comprises more bits than the combination of command and value, zeros are added at the end. This is because the cover register length determines the length of the datagram for DAC values. Note that the command bits consist of the least significant bits of *DAC_ADDR* if the command length is less than 16 bit.

Changing SPI Output Transfer Conditions

Sometimes, other SPI output transfer conditions are required. Therefore, further configuration is possible:

- By setting *sck_low_before_csn* = 1 (bit4 of *SPIOUT_CONF* register 0x04), SCKDRV_NSDO is tied low before NSCSDRV_SDO. (Per default setting, SCKDRV_NSDO is tied high)
- Further on, TMC drivers sample the master data with the rising edge of the master clock. Thus, TMC4361 shifts the output data at SDODRV_SCLK with the falling edge of SCKDRV_NSDO. In case the data is sampled with the falling edge of the master clock at the driver's side, the data at SDODRV_SCLK has to be shifted with the rising edge of SCKDRV_NSDO. Therefore, set *new_out_bit_at_rise* = 1 (bit5 of *SPIOUT_CONF* register 0x04).

12.7 Current Scaling & Ramp Status

The current values of the microstep look-up table MSLUT represent the maximum 9bit signed values which could be sent via the SPIOUT output interface (values could be read out at register 0x7A - *CURRENTA* at bit8:0 and *CURRENTB* at bit24:16).
In most cases of the velocity ramp, it is not required to drive the motor with the full amplitude. Various possibilities have been implemented to adapt the actual current values of the internal microstep look-up table MSLUT to the current ramp status whose signed values).

Scale parameters are available for boost current, hold current, and drive current. These parameters could be assigned independently in the *SCALE_VALUES* register 0x06 and will be used automatically for different states of the velocity ramp if enabled:

- Boost scale value: **BOOST_SCALE_VAL** = SCALE_VALUES(7:0)
- Drive scale value: **DRV1_SCALE_VAL** = SCALE_VALUES(15:8)
- Alternative drive scale value: **DRV2_SCALE_VAL** = SCALE_VALUES(23:16)
- Hold scale value: **HOLD_SCALE_VAL** = SCALE_VALUES(31:24)

The feasible scaling situations will be introduced after a brief explanation of the scaling calculation.

If scaling is enabled for the current ramp state, the actual current values of the MSLUT will be multiplied with the *MULT_SCALE* parameter, which is deduced from one of the four *SCALE_PARAM* values:

MULT_SCALE* = (**actual_SCALE_VAL* **+ 1) / 256** with actual = {*HOLD*, *BOOST*, *DRV1*, *DRV2*}.

Thus, a MULT_SCALE parameter will be generated which ranges from 0 to 1: **0 < *MULT_SCALE* ≤ 1**.
Combining this parameter with the maximum current values will result in the actual transferred current values which could be read out at register 0x7B:

CURRENTA_SPI* = *CURRENTA* · *MULT_SCALE = bit8:0 of 0x7B
CURRENTB_SPI* = *CURRENTB* · *MULT_SCALE = bit24:16 of 0x7B

As it could be seen, any value between 0 and the maximum current value available by the MSLUT entry could be sent using the scaling capability

ATTENTION If TMC drivers are used in ***StepDir output mode*** using SPIOUT only for configuration (*spi_output_format* = b'1011/b'1100 and *scale_val_transfer_en* = 1), scaling values comprises only 5bit due to the fact that driver maximum scale value is 31. Thus, only the last five bits of the eight bit scaling registers are transferred in Step/Dir output mode. Furthermore, *MULT_SCALE* is calculated at the driver devices using the following equation: ***MULT_SCALE* = (actual_*SCALE_VAL* + 1) / 32.**

For scaling the current values during standstill two settings are available:

STANDBY SCALING

- Set *HOLD_CURRENT_SCALE_EN* = 1.
- The *STDBY_DELAY* timer is started as soon as *VACTUAL* reaches 0.
- In case the standby timer expires and *VACTUAL* is still 0, standby mode is valid and currents are scaled down using *HOLD_SCALE_VAL* now.
- In case *STDBY_DELAY* is set to 0 standby mode is valid immediately after reaching *VACTUAL*=0.

Note: if *stdby_clk_pin_assignment*(1) = 0, the STDBY_CLK output pin forwards the standby signal with active polarity which is equal to the setting *stdby_clk_pin_assignment*(0).

SCALING FOR FREEWHEELING

- For freewheeling set *freewheling_en* = 1.
- As soon as standby mode is reached, the *FREEWHEEL_DELAY* timer is started. It expires while standby mode remains active.
- When *FREEWHEEL_DELAY* is elapsed, *freewheeling mode* becomes enabled and thus all current values are altered to 0.
- In case *FREEWHEEL_DELAY* is set to 0, *freewheeling mode* becomes valid immediately after reaching standby mode.

It is also possible to manipulate standard current values during the ramp:

BOOST SCALING AT RAMP START

- Set *boost_current_after_start_en* = 1 for scaling current values with *BOOST_SCALE_VAL*.
- *Boost scaling at ramp start* begins with the onset of a velocity ramp, assumed that *VACTUAL* has been set to 0 before.
- At the ramp start the *BOOST_TIME* (value represents a number of clock cycles) becomes initialized. When this timer expires, boost scaling after start is finished.

BOOST SCALING ON ACCELERATION RAMPS

- If *RAMP_STATE* = b'01 and *boost_current_on_acc_en* = 1 are set, actual current values are scaled with *BOOST_SCALE_VAL*.
- *RAMP_STATE* = b'01 is always valid when the absolute velocity value increases.

BOOST SCALING ON DECELERATION RAMPS

- If *RAMP_STATE* = b'10 and *boost_current_on_dec_en* = 1 are set, the actual current values are scaled with *BOOST_SCALE_VAL*.
- *RAMP_STATE* = b'10 is always valid when the absolute velocity value decreases.

DRIVE SCALING

- If *drive_current_scale_en* is set to 1, current values are scaled with *DRV1_SCALE_VAL*, assumed that no other scaling mode is active at that moment.
- In case *sec_drive_current_scale_en* = 1 is chosen additionally, *DRV1_SCALE_VAL* is only used if the condition *VACTUAL* ≤ *VDRV_SCALE_LIMIT* is met.
- If *sec_drive_current_scale_en* = 1, *drive_current_scale_en* = 1, and *VACTUAL* > *VDRV_SCALE_LIMIT* are valid, current values are scaled with *DRV2_SCALE_VAL*, assumed that no other scaling mode is active.

Setup of scaling values for Step/Dir operation with TMC21xx/51xx, TMX26xx, or TMC389

Scaling values are transmitted directly to the driver in case *Step/Dir output mode* and *scale_val_transfer_en* = 1 is valid. Please note that the maximum scale value is 31 due to the fact that scale values are stored as 5 bit numbers. Thus, only the last 5 bits of the eight bit scaling registers are transferred in *Step/Dir output mode*. Furthermore, *MULT_SCALE* is calculated at the driver devices using the following equation: *MULT_SCALE* = (actual_*SCALE_VAL* + 1) / 32

Controlling the transition process from one scale mode to another

The transition from one scale value to the next can be configured and has not to be abruptly. Three parameters are available for controlling the progression:

UP_SCALE_DELAY
Set the period of clock cycles during which a current scale value is increased by one step towards the higher target scale value with *UP_SCALE_DELAY*.

HOLD_SCALE_DELAY
Set the period of clock cycles during which a current scale value is decreased by one step towards the lower target scale value *HOLD_SCALE_VAL* with *HOLD_SCALE_DELAY*.

DRV_SCALE_DELAY
DRV_SCALE_DELAY is the time period that is required to decrease the actual scale value towards a scale value which is smaller than the current one.

Setting any of these parameters to 0 will result in an immediate transition to the next scale value for the introduced conditions.

The following two examples illustrate how scaling modes are to be used.

EXAMPLE 1

Standby scaling, *freewheeling*, *boost scaling at start*, *boost scaling on deceleration ramps*, and *drive scaling I* are enabled. Current scale parameters (*SCALE_PARAM*) are shown as well as their related scale timers in clock cycles. The timers are used to finish boost scaling after start and to start standby scaling and freewheeling. The three depicted delay values are calculated as follow:

t_{DN_SCALE} = (*BOOST_SCALE_VAL - DRV1_SCALE_VAL*) · *DRV_SCALE_DELAY*
t_{UP_SCALE} = (*BOOST_SCALE_VAL - DRV1_SCALE_VAL*) · *UP_SCALE_DELAY*
t_{HOLD_SCALE} = (*DRV1_SCALE_VAL - HOLD_SCALE_VAL*) · *HOLD_SCALE_DELAY*

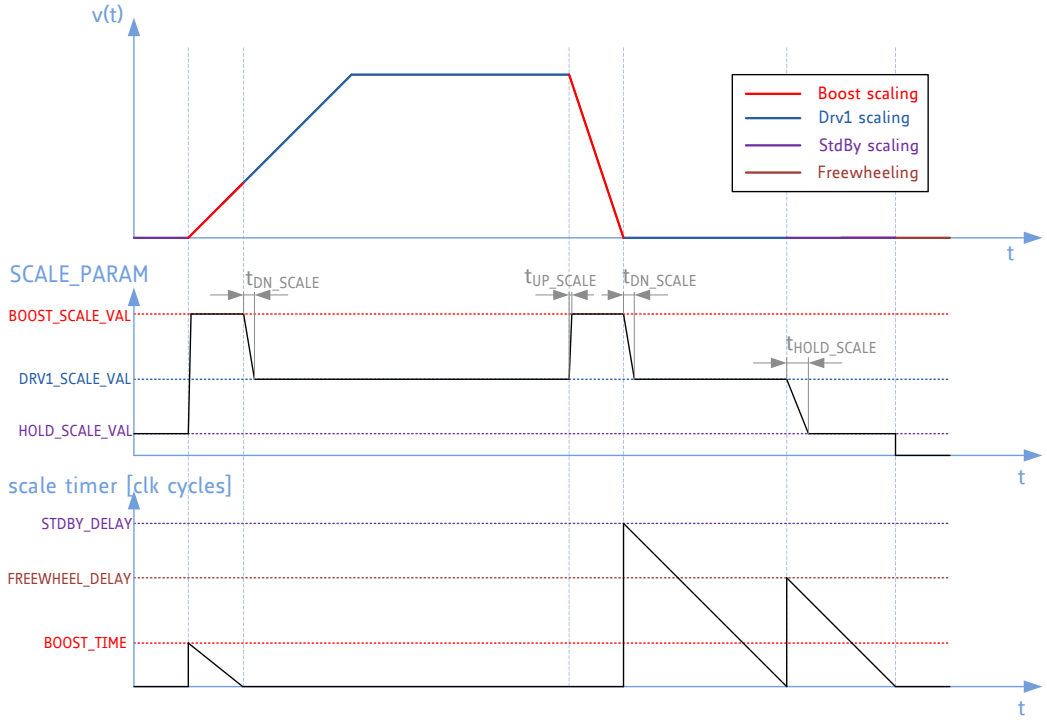


Figure 12.5 Scaling: example 1

EXAMPLE 2

Boost scaling on acceleration ramps and *both drive scaling modes* are enabled. As long as *VACTUAL* < *VDRV_SCALE_LIMIT*, drive scaling I is active. Both drive scaling modes are used for the deceleration ramp due to *boost_current_on_dec* = 0. When *VACTUAL* reaches 0, the *RAMP_STATUS* switches to acceleration ramp and boost scaling becomes enabled a second time.

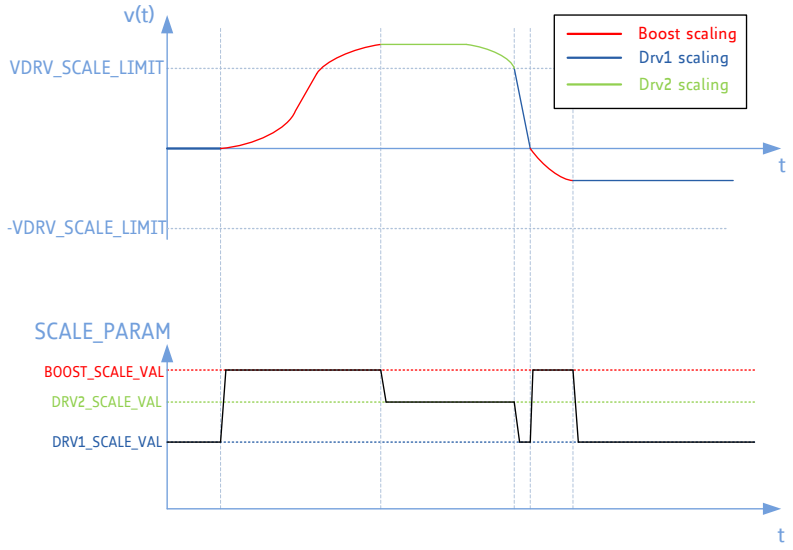


Figure 12.6 Scaling: example 2

13 NFREEZE: Emergency-Stop

In case of dysfunctions at board level, some applications require an additional strategy to end current operations without any delay. Therefore, the TMC4361 provides the low active safety pin NFREEZE.

PINS AND REGISTERS: FREEZE FUNCTIONALITY

Pin names	Type		Remarks
NFREEZE	Input		External enable pin; low active
Register name	Register address		Remarks
DFREEZE	0x4E (23→0)	RW	Deceleration value in the case of an active FREEZE event
IFREEZE	0x4E (31→24)	RW	Current scaling value in the case of an active FREEZE event

NFREEZE is low active. An active NFREEZE input transition from high to low level stops the current ramp immediately in a user configured way. At the moment when NFREEZE switches to low, an event (*FROZEN*) is triggered at *EVENTS*(10). *FROZEN* remains active until the reset of the TMC4361.

Due to an input filter of three consecutive sample points it is necessary to tie NFREEZE low for at least three clock cycles.

13.1 Freeze Function Configuration

Two parameters (*DFREEZE* and *IFREEZE*) are necessary for using the TMC4361 freeze function. They are integrated in the freeze register which can be written only once after an active reset, assumed that there has been no ramp started before. Thus, the freeze parameters should be set directly in the beginning of operation. Note that the chosen values cannot be altered until the next active reset.
These restrictions are necessary to protect the TMC4361 freeze configuration from incorrect SPI data sent from the microcontroller in case of error.

Note
The polarity of the NFREEZE input cannot be assigned.
The freeze register can always be read out.
During freeze state ramp register values can be read out.

CONFIGURING *DFREEZE* FOR AN AUTOMATIC RAMP STOP

- Set *DFREEZE* = 0 for a hard stop.
- Set *DFREEZE* ≠ 0 for a linear deceleration ramp.

Due to the independence of *DFREEZE* from internal register values like *direct_acc_val_en* or the given clock frequency *CLK_FREQ* (which can be altered by erroneous SPI signals) the deceleration value *DFREEZE* is always given as velocity value change per clock cycle. Therefore, the *DFREEZE* value is calculated as follows:

$$d\_freeze[pps^2] = DFREEZE / 2^{37} \cdot f_{CLK}^2$$

This leads to the same behavior of the motor as a direct_acc_val_en = 1 setting during normal operation.

CONFIGURING THE *IFREEZE* CURRENT SCALING VALUE

IFREEZE is a current scaling value which becomes valid in case *NFREEZE* has been tied to low and the related event (*FROZEN*) has been released. In case *IFREEZE* is set to 0, the last scaling value before the emergency event is assigned permanently. The scale value *IFREEZE* then manipulates the current value in the same way as explained in chapter 12.6.

14 Controlled PWM Output

The TMC4361 allows for using PWM output values instead of Step/Dir outputs.

PINS AND REGISTERS: PWM OUTPUT

Pin names	Type		Remarks
STPOUT_PWMA	Output		PWM output for coil A
DIROUT_PWMB	Output		PWM output for coil B
Register name	**Register address**		**Remarks**
GENERAL_CONF	0x00	RW	Bit 21: *pwm_out_en*
CURRENT_CONF	0x05	RW	PWM_AMPL(31:16) if bit8 (pwm_scale_reg_chn)= '1'
PWM_AMPL	0x06	RW	Second assignment to *SCALE_VALUES*(15→0): PWM amplitude at *VACTUAL* = 0 if bit8 (pwm_scale_reg_chn)= '1'
PWM_VMAX	0x17	RW	Second assignment to *VDRV_SCALE_LIMIT*: velocity at which the PWM scale parameter reaches 1 (max)
PWM_FREQ	0x1F	RW	# of clock cycles which forms one PWM period

14.1 PWM Output Generation

For generating a PWM output, set *pwm_out_en* = 1. Now, the Step/Dir output is disabled and PWM signals are forwarded via STPOUT_PWMA and DIROUT_PWMB. The PWM frequency is calculated as follows: $f_{PWM} = f_{CLK} / PWM\_FREQ$.
The duty cycle for both coils is indicated by a high output level. For higher velocity a higher duty cycle is required. Therefore, the TMC4361 alters a PWM scale parameter (*PWM_SCALE*) as a function of the current velocity:

- If *VACTUAL* = 0, *PWM_SCALE* = (*PWM_AMPL* + 1) / 2^{17}.
- With increasing velocity, the scale parameter raises linear to a maximum of *PWM_SCALE* = 0.5 at *VACTUAL* = *PWM_VMAX*.
- The minimum duty cycle is calculated with *DUTY_MIN* = (0.5 – PWM_SCALE).
- The maximum duty cycle is calculated with DUTY_MAX = (0.5 + PWM_SCALE).

The current duty cycle for both coils is calculated using the microstep loop-up table MSLUT. In this case the MSLUT describes a voltage (co-)sine curve whose amplitudes become transferred to the PWM phases. The values are scaled related to minimum and maximum duty cycles.

In the following illustration, the calculation of minimum/maximum PWM duty cycles with *PWM_AMPL* = 32767 is pointed out at the left side. Resulting duty cycles for different positions in the sine voltage curve are depicted at the right side. Calculated delays of minimum/maximum duty cycles are also shown.

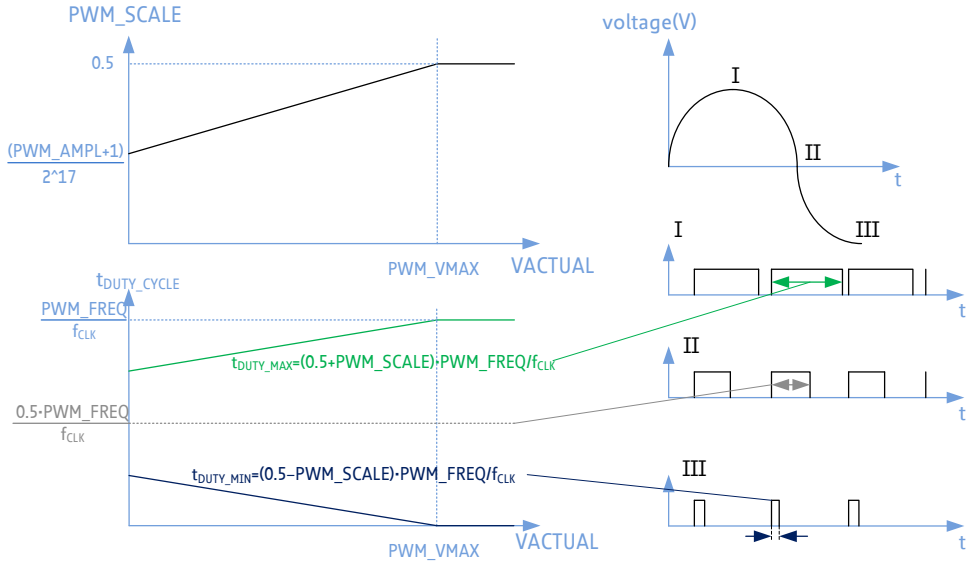


Figure 14.1 Calculation of PWM duty cycles

HINT If a 2nd assignment for SCALE_VALUES should be avoided, pwm_scale_reg_chn have to be set to 1, then the PWM_AMPL register is changed from register 0x06 SCALE_VALUES(15:0) to register 0x05 CURRENT_CONF(31:16). Thus, switching between PWM and SPI mode could be executed without any need to alter the scale settings.

15 Support of the dcStep feature of TMC motor drivers

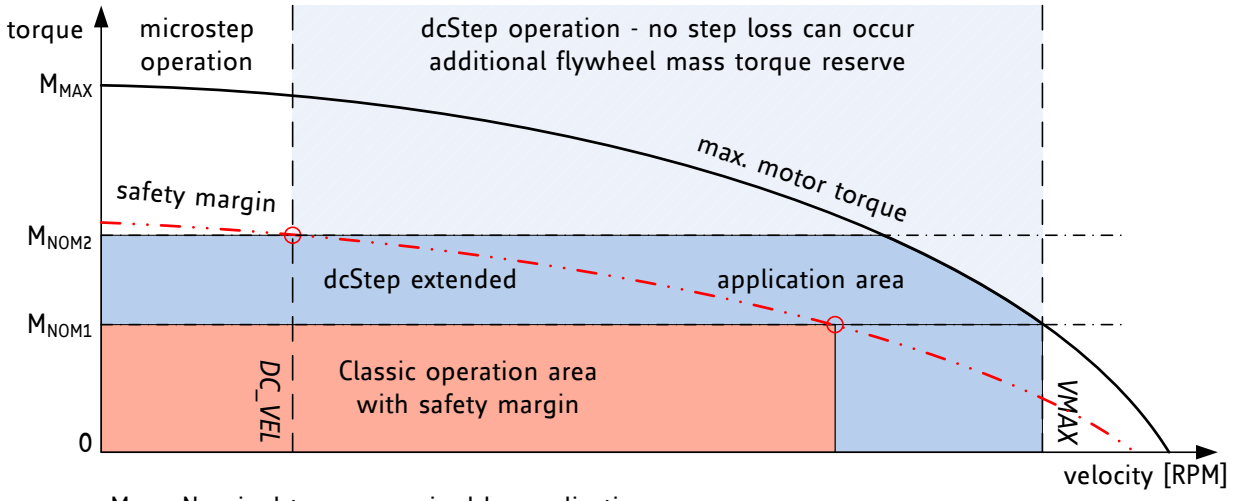
dcStep is an automatic commutation mode for the stepper motor drivers. It allows the stepper to run with its nominal velocity taken from the ramp generator as long as it can cope with the motor load. In case the motor becomes overloaded, it slows down to a velocity, where the motor can still drive the load. This way, the stepper motor never stalls and can drive heavy loads as fast as possible. Its higher torque available at lower velocity, plus dynamic torque from its flywheel mass allow compensating for mechanical torque peaks. In case the motor becomes completely blocked, the stall flag will be set.

15.1 Essential pins and registers

Pin name	Pin type		Remarks
MP1	Input		dcStep input signal
MP2	Inout as Output		dcStep output signal
Register name	**Register address**		**Remarks**
GENERAL_CONF	0x00	RW	Bit22→21: dc_step_mode
DC_VEL	0x60	W	Velocity at which dcStep starts (fullstep); 24 bit
DC_TIME	0x61(7:0)	W	Upper PWM on time limit for internal dcStep calculation
DC_SG	0x61(15:8)	W	Maximum PWM on time for step loss detection (multiplied by 16!)
DC_BLKTIME	0x61(31:16)	W	dcStep blank time after fullstep release
DC_LSPTM	0x62	W	dcStep low speed timer; 32 bit

15.2 Designing-In dcStep into an Application

In a classical application, the operation area is limited by the maximum torque required at maximum application velocity. A safety margin of up to 50% torque is required, in order to compensate for unforeseen load peaks, torque loss due to resonance and aging of mechanical components. dcStep allows using up to the full available motor torque. Even higher short time dynamic loads can be overcome using motor and application flywheel mass without the danger of a motor stall. With dcStep the nominal application load can be extended to a higher torque only limited by the safety margin near the holding torque area (which is the highest torque the motor can provide). Additionally, maximum application velocity can be increased up to the actually reachable motor velocity.



M_{NOM}: Nominal torque required by application

M_{MAX}: Motor pull-out torque at v=0

Safety margin: Classical application operation area is limited by a certain percentage of motor pull-out torque

Figure 15.1: dcStep extended application operation area

15.3 Enabling dcStep

Please refer to the corresponding manuals for the correct motor driver settings. Despite particular motor settings which could be tunneled via SPI through TMC4361, dcStep requires only a few settings. It directly feeds back motor motion to the ramp generator, so that it becomes seamlessly integrated into the motion ramp, even if the motor becomes overloaded with respect to the target velocity. dcStep operates the motor in fullstep mode at the ramp generator target velocity *VACTUAL* or at reduced velocity if the motor becomes overloaded. By setting *dc_step_mode*(1) = '1', dc Step will be enabled for TMC26x drivers (*dc_step_mode*(0) = '1') or for TMC21xx drivers (*dc_step_mode*(0) = '0'). If one of both already chosen for *spi_output_format*, *dc_step_mode* could be set to "01" which will automatically select the correct dcStep module.
dcStep requires setting the minimum operation velocity *DC_VEL*. *DC_VEL* shall be set to the lowest operating velocity where dcStep gives a reliable detection of motor operation. If an overload appears, an internal dcStep signal will be generated which paused the internal step generation. After *DC_LSPTM* clock cycles expires without lifting the internal dcStep signal, a step will be enforced to remain a minimum step frequency of f_{CLK} / *DC_LSPTM*.

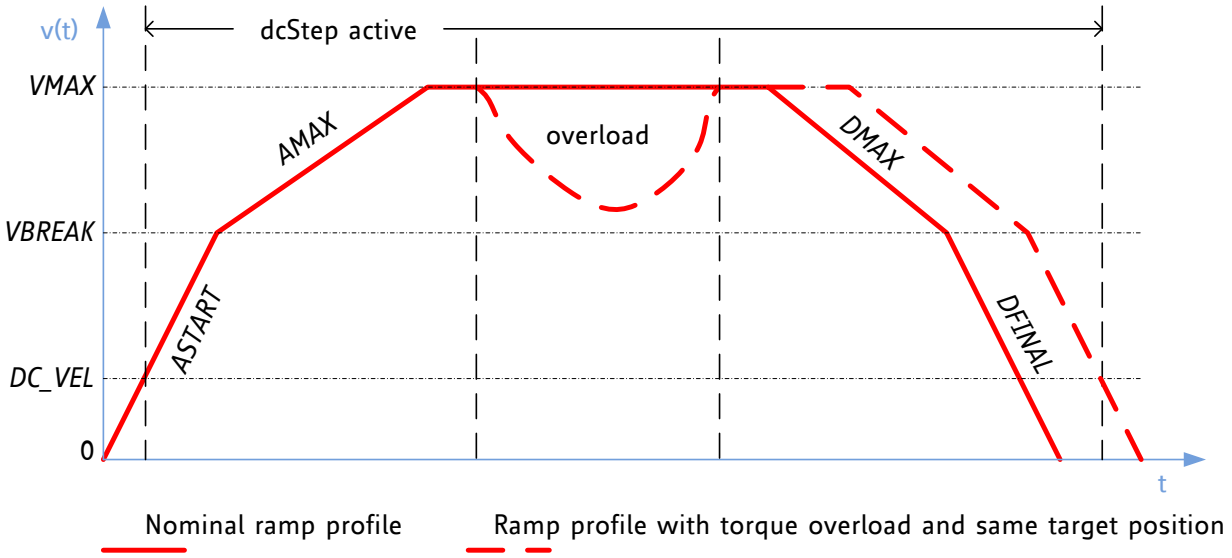


Figure 15.2: Velocity profile with impact by overload situation

15.3.1 dcStep with TMC21xx

dcStep operation with TMC21xx is similar to an handshake procedure. If *VACTUAL* ≥ *DC_VEL*, MP2 output will be set to '1' to indicate that dcStep is possible. TMC21xx will wait for the next fullstep position to switch to dcStep operation. The dcStep signal will be provided by the TMC21xx itself. This signal will be supplied by the MP1 input pin. If MP1 is low, an overload is occurred. This will pause the internal step generator. If MP1 will stay low until *DC_LSPTM* clock cycles have been expired, the TMC4361 will generate an step with a step length of 4096 clock cycles to enforce TMC21xx generating a step.

15.3.2 dcStep with TMC26x

Connected to TMC26x drivers, TMC4361 have to generate the dcStep signal internally. Therefore, the DCSTEP_STALL input has to be connected to SG_TST output pin of the TMC26x driver. Furthermore, the TST_MODE pin of TMC26x have to be tied to VDD and following TMC26x settings have to be made: CHM = '1', HSTRT = '0', TST = '1' and SGT0 = SGT1 = '1'.
The internal dcStep signal will approve further step generation if the input step signals of the DCSTEP_STALL input pin will be smaller than the *DC_TIME* step length in clock cycles which should be slightly higher than chopper blank time TBL.
While dcStep is able to decelerate the motor upon overload, it cannot avoid a stall in every operation situation. Once the motor is blocked, or it becomes decelerated below a motor dependent minimum velocity where the motor operation cannot safely be detected any more, the motor may stall and loose steps. In order to safely detect a step loss and avoid restarting of the motor, the stop on stall can be enabled (see *stop_on_stall*). In the case of dcStep operation with TMC26x the stall bit from the driver

status will be substituted by the dcStep stall detection bit. Therefore, the first step of DCSTEP_STALL after an step release will be checked against the *DC_SG* value which is the maximum PWM on time. If the signal step length is smaller than *DC_SG* a stall has occurred. *DC_BLKTIME* specifies the number of clock cycles after a fullstep release when nothing will be compared due to possible fragmented steps which will be sent over MP1. The first step after release which will checked is the first step after the blank time. The switch to fullstep drive will be done automatically as explained in chapter 12.5.

ATTENTION All settings as regards dcStep should be made before beginning dcStep operation!

HINT It is possible to transfer automatically cover datagrams to the TMC26x (see 12.3). Thereby, it is possible to switch rapidly the chopper settings of the TMC26x shortly before reaching the dcStep velocity. It is recommended to use this feature as dcStep requires const toff chopper settings whereas driving with µSteps and a spreadCycle chopper provides better driving characteristics. Therefore, set *SPI_SWITCH_VEL* a little bit smaller than *DC_VEL* and turn on automatic_cover. After transferring all required cover datagrams, fill in *COVER_LOW* the chopper settings for spreadCycle below the *DC_VEL*. In *COVER_HIGH* the constToff settings for the chopper during dcStep (fullsteps) should be filled in.

16 Decoder Unit & Closed Loop

The TMC4361 is equipped with an encoder input interface for incremental ABN encoders or absolute encoders like SSI, SPI, or BiSS encoders. Motor feedback can be analyzed and even closed loop behavior can be reached by setting the registers appropriately.

PINS AND REGISTERS: DECODER UNIT

Pin names	Type		Remarks
A_SCLK	Input or Output		A signal of ABN encoder or serial clock output for SSI, SPI or BiSS encoders
ANEG_NSCLK	Input or Output		Negated A signal of ABN encoder or negated Serial Clock output for SSI / BiSS encoder or low active chip select signal for SPI encoders
B_SDI	Input		B signal of ABN encoder or serial data input of SSI, SPI or BiSS encoders
BNEG_NSDI	Input or Output		Negated B signal of ABN encoder or negated Serial Data input of SSI / BiSS encoders or serial data output of SPI encoder
N	Input		N signal of ABN encoder
NNEG	Input		Negated N signal of ABN encoder
Register name	**Register address**		**Remarks**
GENERAL_CONF	0x00	RW	Bit11→10 *serial_enc_in_mode* Bit12 *diff_enc_in_disable*
INPUT_FILT_CONF	0x03	RW	Input filter configuration (SR_ENC_IN, FILT_L_ENC_IN)
ENC_IN_CONF	0x07	RW	Encoder configuration register
ENC_IN_DATA	0x08	RW	Serial encoder input data structure
STEP_CONF	0x0A	RW	Motor configurations
ENC_POS	0x50	RW	Current absolute encoder position in microsteps
ENC_LATCH	0x51	R	Latched absolute encoder position
ENC_POS_DEV	0x52	R	Deviation between *XACTUAL* and *ENC_POS*
ENC_CONST	0x54	R	Internally calculated encoder constant
Encoder Register Set	0x51…58 0x62…63	W	Encoder configuration parameter
Encoder velocity	0x65 0x66	R	Current encoder velocity (unsigned) Current filtered encoder velocity (signed)
ADDR_TO_ENC DATA_TO_ENC	0x68 0x69	W	Serial encoder request data
ADDR_FROM_ENC DATA_FROM_ENC	0x6A 0x6B	R	Serial encoder request data response
Encoder compensation	0x7D	W	Encoder compensation register set
Register name	**Register address**		**Remarks**
Closed loop settings	0x1C 0x59	RW	Closed loop / PID configuration parameter
	0x5A…5F 0x60…61	W	
	0x5A..5D	R	PID control parameter
Closed loop scaling settings	0x18…1A	RW	Closed loop configuration parameter for current scaling
CURRENT_CONF	0x05	RW	Bit7 *closed_loop_scale_en*
SCALE_VALUES	0x06	RW	Current scaling values and limits for closed loop operation

16.1 General Encoder Interface

The encoder interface consists of six pins (A_SCLK, ANEG_NSCLK, B_SDI, BNEG_NSDI, N, NNEG). For configuration set *serial_enc_in_mode*. N and NNEG are only required for incremental encoders. All encoder input signals become filtered using the digital filter introduced in chapter 6. Thus, *SR_ENC_IN* and *FILT_L_ENC_IN* have to be set properly.

Choosing the Serial Encoder_IN Mode

serial_enc_in_mode = b'00: ABN incremental encoder setting. All six interface pins are inputs. Signals are interpreted as ABN signals of an incremental encoder.

serial_enc_in_mode = b'01: Absolute SSI encoder setting. A_SCLK and ANEG_NSCLK are outputs which forward the master clock signals to the motor encoder interface. Only B_SDI and BNEG_NSDI are required inputs to receive data from the encoder.

serial_enc_in_mode = b'10: Absolute BiSS encoder setting. A_SCLK and ANEG_NSCLK are outputs which provide the master clock signal to the motor encoder interface. Only B_SDI and BNEG_NSDI are required as inputs to receive data from the encoder.

serial_enc_in_mode = b'11: Absolute SPI encoder setting. A_SCLK is the serial clock output, ANEG_NSCLK is the low active chip select output, B_SDI functions as serial data input from the SPI encoder, and BNEG_NSDI is the serial data output.. *diff_enc_in_disable* is set automatically to 1.

How to Enable or Disable Differential Encoder Signals

diff_enc_in_disable=0 Differential encoder interface inputs are enabled. Differential inputs are treated as digital differential inputs. For advertizing a valid level the levels have to be inversed.

diff_enc_in_disable=1 Differential encoder interface inputs are disabled. For SPI encoders this is done automatically. Signals are handled as single signals and every negated pin becomes ignored.

16.2 Incremental ABN Encoder

The incremental ABN encoder increments or decrements the internal *ENC_POS* counter. This is based on A and B signal level transitions.

ABN encoder configuration

1. Choose the number of microsteps per AB transition:
   - Set the fullstep resolution of the motor with *FS_PER_REV* and the microstep resolution *MSTEP_PER_FS* for the driver in the *STEP_CONF* register 0x0A.
   - Then, set the encoder resolution with *ENC_IN_RES*.
2. Choose the direction of the encoder with *invert_enc_in*. Set 1 for inverting if desired.
3. Verify the encoder constant. *ENC_CONST* is calculated automatically, if the settings for *FS_PER_REV*, *MSTEP_PER_FS*, and *ENC_IN_RES* fit properly. *ENC_CONST* gives the number of microsteps which are added or subtracted from *ENC_POS* (dependent on the direction) with every AB transition. It is calculated with

$$ENC\_CONST = MSTEP\_PER\_FS \cdot FS\_PER\_REV / ENC\_IN\_RES$$

ENC_CONST can be read out. It consists of 15 digits and 16 decimal places. If 16 bit are not sufficient for a binary representation of the decimal places, TMC4361 tries to match them to a multiply of 10000 within these 16 decimal places. This way, a perfect match can be achieved in case decimal representation is preferred to a binary one. If also the decimal representation does not fit completely, the type of the decimal places of *ENC_CONST* can be chosen by hand with *ENC_IN_CONF*(0). Set *ENC_IN_CONF*(0) to 0 for the binary representation or set it to 1 for the decimal one. But beware, this way *ENC_POS* could be slightly different to the real position.

Hint *ENC_CONST* could also be set manually. It is recommended to use the equation above for the manual calculation. Therefore, bit31 of the register 0x54 have to be set to '1'. This way, writing to 0x54 do not represent *ENC_IN_RES*, instead it is the direct assignment of *ENC_CONST* in this register 0x54.

16.2.1 N Signal resp. Z Channel

The N signal (or Z channel) is either used to clear the position counter or to take a snapshot. The following configuration parameters are provided:

pol_n — Set the active polarity with this parameter.

n_chan_sensitivity — Set *n_chan_sensivity* for special requests:

- 00 N event is active if the voltage level at N fits *pol_n*
- 01 N event is triggered at the positive edge when N becomes active.
- 10 N event is triggered at the negative edge when N becomes active.
- 11 N event is triggered at both edges when N becomes active and/or inactive.

pol_a_for_n, *pol_b_for_n* — Some encoders require a validation of the N signal at a certain configuration of A and B polarities. This can be controlled by *pol_a_for_n* and *pol_b_for_n* switches in the *ENC_IN_CONF* register. When, e.g., *pol_a_for_n* and *pol_b_for_n* are set, an active N event is only accepted if the polarity of the A channel and the B channel is high.

ignore_ab — Set *ignore_ab* = 1 to disable the validation of the N signal via A and B channel polarity. Then, these channel polarities have no influence on the N signal event.

latch_enc_on_n — Set *latch_enc_on_n* = 1 to monitor the encoder position *ENC_POS* on an active N event. Still, additional switches are required to monitor the encoder position. Refer to *clr_latch_cont_on_n* and *clr_latch_once_on_n.*

clear_on_n — To clear the encoder position *ENC_POS* on the next N event set *clear_on_n* = 1. Again, additional switches are required to clear *ENC_POS*. Refer to *clr_latch_cont_on_n* and *clr_latch_once_on_n.*

clr_latch_cont_on_n — Set *clr_latch_cont_on_n* = 1 to clear or monitor continuously on an active N event.

clr_latch_once_on_n — Set *clr_latch_once_on_n* = 1 to clear or monitor on an active N event only once. After latching and/or clearing the encoder position *clr_latch_once_on_n* is disabled automatically. This is necessary if only the next of periodic N events (e.g. once for every revolution) should be considered.

latch_x_on_n — Set *latch_x_on_n* = 1 to monitor *XACTUAL* on *X_LATCH*. The tasks of encoder latching are adopted for *X_LATCH*. Please note that *latch_enc_on_n* has to be set just as *clr_latch_cont_on_n* or *clr_latch_once_on_n*.

HINT For clearing the encoder position *ENC_POS* with the next active N event set *clear_on_n* = 1 and *clr_latch_once_on_n* = 1 or *clr_latch_cont_on_n* = 1.

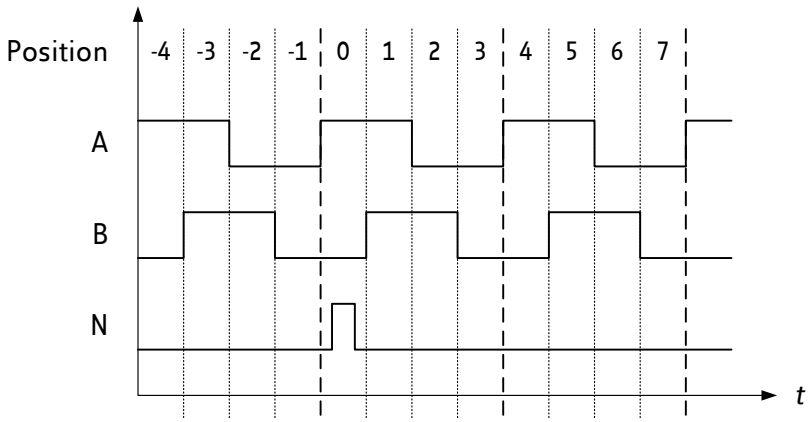


Figure 16.1 Outline of ABN signals of an incremental encoder

16.3 Absolute Encoder

In this chapter, common settings for SSI, SPI, and BiSS encoders are explained. Specific information about respective serial encoders is given in subsections.

Serial encoders provide absolute encoder data instead of step transition, whose information is delivered from incremental encoders. Due to the serial data input the TMC4361 provides an external clock for the encoder.

The TMC4361 provides different possibilities for the serial data stream. Single turn data or multi turn data can be used. In case single turn data is transmitted the TMC4361 is able to calculate the number of revolutions permanently.

Choosing Encoder Data and Data Type for Transmission

multi_turn_in_en — In case *multi_turn_en* = 1 the serial encoder transmits the actual motor angle as well as data about the number of revolutions. If set to 0, the serial encoder transmits only the actual motor angle (single turn).

multi_turn_in_signed — In case it is desired to interpret the data about the number of revolutions as a signed value, set *multi_turn_in_signed* = 1. Otherwise it is assigned as unsigned value.

calc_multi_turn_behav — For calculating the number of revolutions out of single turn data set *calc_multi_turn_behav* = 1 and *multi_turn_in_en* = 0. In case two sequenced values differ in more than half a revolution, switch *calc_multi_turn_behav* off because the calculation will provide false data.

The encoder constant *ENC_CONST* is calculated the same way as for incremental ABN encoders. But in contrast to ABN encoders it always represents a binary value. The value is multiplied with the transferred angle value for calculating the microstep position, which is stored in *ENC_POS* afterwards. *ENC_CONST* is generated automatically, if the settings for *FS_PER_REV*, *MSTEP_PER_FS*, and *ENC_IN_RES* fit properly. It is calculated as follows:

$$ENC\_CONST = MSTEP\_PER\_FS \cdot FS\_PER\_REV / ENC\_IN\_RES$$

Hint *ENC_CONST* could be read out. It consists of 15 digits and 16 decimal places in case a serial encoder is selected.

Settings Related to Data Transmission

SINGLE_TURN_RES — Set the number of bits for single turn data here. *SINGLE_TURN_RES* defines the most significant bit (MSB). The number of angle data bits within one revolution is *SINGLE_TURN_RES* + 1.

MULTI_TURN_RES — Set the number of bits for multi turn data here. *MULTI_TURN_RES* defines the most significant bit (MSB). The number of data bits for revolution count is *MULTI_TURN_RES* + 1. Information about the number of rotations is always expected to be sent before actual motor angle data!

STATUS_BIT_CNT — Set the number of status bits which are transmitted from the encoder here. Status bits consist of three bits at maximum.

left_aligned_data — This parameter is used to choose a sequential arrangement for status and serial input data. Set *left_aligned_data*=0 for receiving the flags first and then the input data. Set *left_aligned_data*=1 for serial input data first and flags afterwards.

serial_enc_variation_limit — If erroneous data transmission from the encoder is expected, set *serial_enc_variation_limit* = 1. Now, the differences of sequenced encoder values become calculated. If a difference between two values exceeds one eighth of *ENC_IN_RES*, the last encoder data is skipped. The *MULTI_CYCLE_FAIL_F* status flag becomes set and the *SER_ENC_DATA_FAIL* event becomes triggered. As a result, *calc_multi_turn_behav* can be used with no doubt due to the fact that the values never differ in more than half a revolution.

ATTENTION

Generating a clock which requires more than the given serial bit range

If more than three status bits or additional fill bits are sent, clock errors can occur because the number of transferred clock bits is calculated by

#serial_clock_cycles = (*SINGLE_TURN_RES* + 1) + (*MULTI_TURN_RES* + 1) + *STATUS_BIT_CNT.*

In order to prevent clock failures *MULTI_TURN_RES* can be set to a higher value than required, even if no multi turn data is provided.

Note that this setting may result in erroneous multi turn data. This can be corrected by setting *multi_turn_in_en* = 0 for skipping multi turn data automatically. Further, *calc_multi_turn_behav* can be set to 1 for compensating unavailable multi turn data.

16.3.1 SSI Encoder Serial Clock

After indicating data transfer by switching the clock output to low level, the transfer starts with the next rising edge of the serial clock output. The periods for low level and for high level have to be set separately using *SER_CLK_IN_LOW* and *SER_CLK_IN_HIGH* which are given in internal clock cycles.

START/END DELAY TIME

Sample points of serial data are set at the falling edges of the serial clock. Some encoders need more clock cycles than the low clock phase to prepare data for transfer. Further, due to long wires data transfer can take more time. Considering these points, the delay time for compensation *SSI_IN_CLK_DELAY* can be specified in clock cycles and the parameter delays the sampling start. Due to the delay, more clock cycles of the serial clock can be required which is automatically taken into account. Per default, *SSI_IN_CLK_DELAY* is set to 0 and therefore *SER_CLK_IN_HIGH* is valid instead.

DELAY TIME BETWEEN SEQUENCED DATA REQUESTS

After a data request the next one is sent past *SER_PTIME* clock cycles. Choose the value of this parameter higher than 21µs.

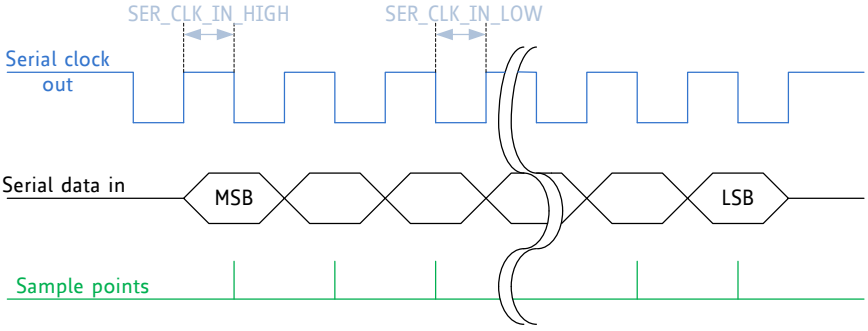


Figure 16.2 SSI signals with *SSI_IN_CLK_DELAY=SER_CLK_IN_HIGH* when *SSI_IN_CLK_DELAY=0*

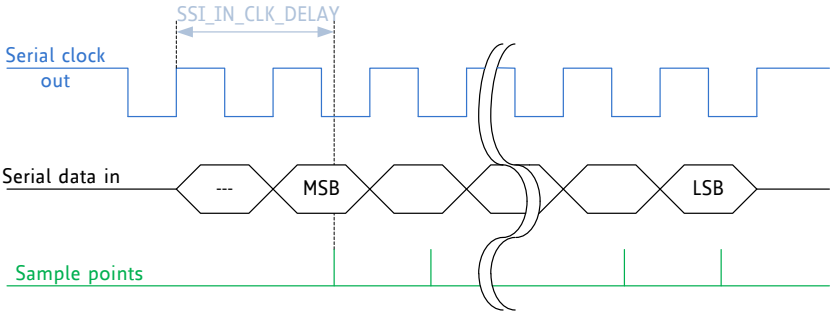


Figure 16.3 SSI signals with *SSI_IN_CLK_DELAY* for compensating processing time and long wires

REPEATED DATA TRANSFER (MULTI CYCLE REQUEST)

If a repeated data transfer is requested, set *ssi_multi_cycle_data* = 1. Further, adjust *SSI_IN_WTIME*. This parameter configures the waiting time between two equal data requests (the same value becomes transferred more than once). Set the *SSI_IN_WTIME* smaller than 19µs.

If two data values that normally must be equal (due to the multi cycle data request) are not equal, the *MULTI_CYCLE_FAIL_F* flag and the *SER_ENC_DATA_FAIL* event are generated, assumed that *serial_enc_variation_limit* is 0.

GRAY ENCODED DATA STREAMS

Serial data streams can be gray encoded. Set *ssi_gray_code_en* to 1 in order to decode them properly.

16.3.2 SPI Encoder

The number of bits per transfer is calculated automatically. Therefore, set *SINGLE_TURN_RES*, *MULTI_TURN_RES*, and *STATUS_BIT_CNT* properly.

COMMUNICATION PROCESS

Typically, the answer of SPI data transfer requests is sent with the next transmission. When the TMC4361 receives the answer from the encoder, it calculates *ENC_POS* immediately. The encoder slave does not send any data without receiving a request first. Therefore, the TMC4361 always sends the *ADDR_TO_ENC* value to request encoder data from the SPI encoder slave device. The LSB of the serial data output is *ADDR_TO_ENC*(0). The clock is generated by the values given from *SER_CLK_IN_HIGH* and *SER_CLK_IN_LOW*.

The MSB results from the number of bits that are required for the whole transmission:

$$MSB_{SPI\_ENC} = (SINGLE\_TURN\_RES + 1) + (MULTI\_TURN\_RES + 1) + STATUS\_BIT\_CNT - 1.$$

With the *SER_PTIME* register a time period after the last request can be set. During this period no further data transfer becomes initiated.

STORING ENCODER VALUES

Received encoder data is stored in *ADDR_FROM_ENC*. Thus, encoder values can be verified and compared to microcontroller data later on.

SPI Data Transfer within the Current Transmission

For applications providing a response to SPI data transfer requests within the current transmission, suggestions explained in ***Generating a clock which requires more than the given serial bit range*** (see chapter 0) are valid. Therefore, *MULTI_TURN_RES* has to be expanded for reaching the required data length. Further, the first bits of *B_SDI* must be zero if the multi turn data of the data transfer should also be evaluated. This way, proper evaluation of incoming data is guaranteed.

FURTHER SETTINGS RELATED TO ENCODER TYPES

Per default, SPI encoder data transfer is managed just as the communication between microcontroller and TMC4361. For supporting all SPI encoder types, the TMC4361 provides further settings:

spi_low_before_cs	Set this parameter to 1 if the SPI clock has to be low before chip select switches to low level.
spi_data_on_cs	Set this parameter to 1 for sending the MSB of the data output with the transition of the chip select signal to active level. Thus, BNEG_NSDI provides output data immediately at ANEG_NSCLK transition. If set to 0, the data is sent after the first transition of the clock signal. In both cases data from B_SDI becomes sampled at the next clock transition.

To achieve a correct synchronization, it is possible to choose *SSI_IN_CLK_DELAY* > 0. Thus, a delay time for *SER_CLK_IN_HIGH* and *SER_CLK_IN_LOW* can be defined. This delay setting can be considered in case the clock scheme provided by *SER_CLK_IN_HIGH* and *SER_CLK_IN_LOW* does not fit perfectly to the delay between the first and the last transition of signals of the chip select (ANEG_NSCLK) and the clock output (A_SCLK). *SSI_IN_CLK_DELAY* is shown in Figure 16.4 as red lined distances.

The following diagrams are examples for different *spi_data_on_cs* and *spi_low_before_cs* settings.

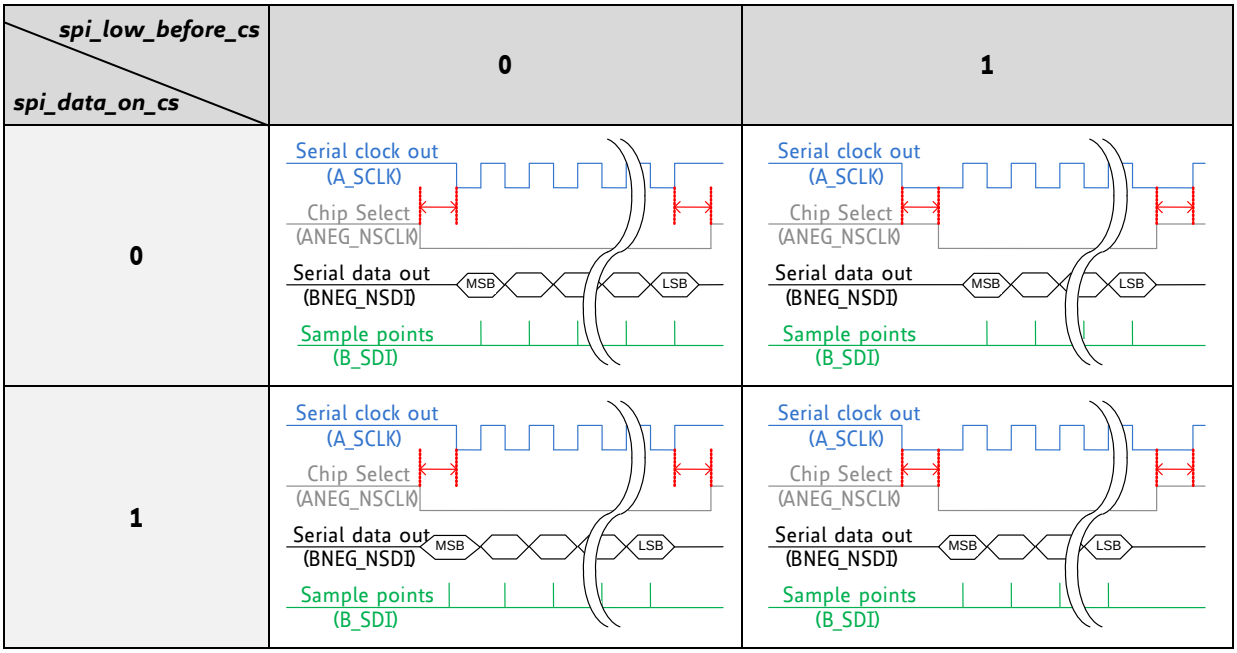


Figure 16.4 Supported SPI encoder data transfer modes

SPI Encoder Configuration via TMC4361

The SPI encoder can be configured by the TMC4361. Thus, a connection between microcontroller and encoder is not necessary any more. A configuration request is sent using the settings of the *SERIAL_ADDR_BITS* and *SERIAL_DATA_BITS* which define the transferring bit numbers. Due to repeated encoder data requests a call for configuration has to be done in the meantime. Therefore, *DATA_TO_ENC* can be used for configuration while the continuous requests for encoder data are interrupted during configuration process.

Proceed as follows

1. Write access to *DATA_TO_ENC* to notify a configuration request.
2. Write the requested address to *ADDR_TO_ENC*.
3. Write the requested data to *DATA_TO_ENC*.
4. Now, three datagrams are sent to the SPI encoder:
   - The address data from *ADDR_TO_ENC* (data from encoder is not stored).
   - The register data from *DATA_TO_ENC* (the received data from the encoder is stored in *ADDR_FROM_ENC*).
   - A no-operation datagram (*NOP*) to get the answer from the encoder (which becomes stored in *DATA_FROM_ENC*).
5. Now, returned and stored data from the encoder can be read out and checked by the microcontroller. Read out *ADDR_FROM_ENC* first.
6. Write the required address for the continuous encoder value requests to *ADDR_TO_ENC*.
7. Read out *DATA_FROM_ENC* to finish configuration process. Afterwards, repeated data requests become initiated immediately.

Hint No further encoder value request becomes sent before *DATA_FROM_ENC* has been read out!

16.4 Regulation Possibilities with Encoder Feedback

The encoder feedback can be used for controlling *the motion controller outputs* in a way that the internal actual position matches or rather follows the real position *ENC_POS*. Generally, two possibilities for position control are provided: PID control and closed loop operation. Closed loop operation is a very good choice in case the encoder is mounted directly on the back of the motor and position data is evaluated precisely, whereas PID control is well suited where the encoder is located at the drive side and no fixed connection between motor and drive side is given, e.g. belt drives.

The following *regulation_modus* settings are provided:

regulation_modus = b'10 Use this setting for a PID control unit (with pulse generator base = 0).

regulation_modus = b'11 Use this setting for a PID control unit (with pulse generator base = *VACTUAL*.)

regulation_modus = b'01 Use this setting for closed loop operation.

If PID regulation is not selected (*regulation_modus*(1) = 0), the internal velocity which delivers the input for the pulse generator is equal to the ramp velocity (*VEL_ACT_PID* = *VACTUAL*).

Target Reached during Regulated Behavior

If one of the regulation modes is selected, *TARGET_REACHED* event and flag will only be released if *XACTUAL* = *XTARGET* and if the encoder position *ENC_POS* and the internal *XACTUAL* are only differ at the most for *TR_TOLERANCE* microsteps. Reaching *XACTUAL* = *XTARGET* and the difference exceeds the limit, the *TARGET_REACHED* event or flag will occur not before the difference is decreased below the limit which could be done by the regulation itself (recommended!) or by setting *ENC_POS* manually (not recommended!).

16.4.1 PID Based Control of *XACTUAL*

Based on a position difference error *PID_E* = *XACTUAL* – *ENC_POS* the PID (proportional integral differential) controller calculates a signed velocity value (v_{PID}) which is used for minimizing the position error. During this process, the TMC4361 moves with v_{PID} until |*PID_E*| – *PID_TOLERANCE* ≤ 0 is reached and the position error is removed.

v_{PID} is calculated by:

$$v_{PID} = \frac{PID\_P}{256} \cdot PID\_E \cdot \left[\frac{1}{s}\right] + \frac{PID\_I}{256} \cdot \int_0^t PID\_E \cdot dt + PID\_D \cdot PID\_E \cdot \frac{d}{dt}$$

$$v_{PID} = \frac{PID\_P}{256} \cdot PID\_E \cdot \left[\frac{1}{s}\right] + \frac{PID\_I}{256} \cdot PID\_ISUM + PID\_D \cdot PID\_E \cdot \frac{d}{dt}$$

$$v_{PID} = \frac{PID\_P}{256} \cdot PID\_E \cdot \left[\frac{1}{s}\right] + \frac{PID\_I}{256} \cdot PID\_E \cdot \frac{f_{CLK}}{128} + PID\_D \cdot PID\_E \cdot \frac{d}{dt}$$

with

PID_P = proportional term
PID_I = integral term
PID_D = derivate term

Control Parameters and Clipping Values

PID_DV_CLIP To avoid large velocity variations, the v_{PID} value can be limited with *PID_DV_CLIP*. This clipping parameter limits v_{PID} as well as *PID_VEL* (current PID velocity output).

PID_I_CLIP The error sum *PID_ISUM* is generated by the integral term. For limiting *PID_ISUM* set *PID_I_CLIP*. Note that the maximum value of *PID_I_CLIP* should meet the condition *PID_I_CLIP* ≤ *PID_DV_CLIP* / *PID_I*. If the error sum *PID_ISUM* is not clipped, it is increased with each time step by *PID_I* · *PID_E*. This continues as long as the motor does not follow.

PID_E Use this register for reading out the actual position deviation between *XACTUAL* and *ENC_POS*.

PID_D_CLKDIV Time scaling for deviation (with respect to error correction periods) is controlled by the *PID_D_CLKDIV* register. Note that during error correction the fixed clock frequency $f_{PID\_INTEGRAL}$[Hz] = f_{CLK}[Hz] / 128 is valid.

VEL_ACT_PID The internal velocity *VEL_ACT_PID* alters the current ramp velocity *VACTUAL*. Two settings are provided:
If *regulation_modus* = b'11, *VACTUAL* is assigned as pulse generator base value and *VEL_ACT_PID* is calculated by *VEL_ACT_PID* = *VACTUAL* + v_{PID}.
If *regulation_modus* = b'10, zero is assigned as pulse generator base value. In this case *VEL_ACT_PID* = v_{PID} is valid.

PID_TOLERANCE The TMC4361 provides the programmable hysteresis *PID_TOLERANCE* for target position stabilization. Oscillations due to error correction can be avoided if *XACTUAL* is close to the real mechanical position. The PID controller of the TMC4361 is programmable up to approximate 100kHz update rate at f_{CLK} = 16 MHz. This high speed update rate qualifies it for motion stabilization.

Note that detailed knowledge of a particular application (including dynamics of mechanics) is necessary for PID controller parameterization which is done in a direct way!

16.4.2 Closed Loop Behavior

The closed loop unit of the TMC4361 modifies output currents resp. Step/Dir outputs directly. For closed loop, set *regulation_modus* = b'01. After starting this mode of operation, the closed loop calibration becomes executed and the *CL_OFFSET* value becomes set.

With closed loop, current values are not controlled using the internal step generator. The currents values at the SPI output resp. the Step/Dir outputs are verified using the closed loop offset value *CL_OFFSET* which correlates to the evaluated difference between *XACTUAL* and *ENC_POS*. Nevertheless, the internal ramp generator still generates steps for *XACTUAL*.

Basic Parameters for Calibration and Position Deviation Compensation

CL_OFFSET The *CL_OFFSET* register contains the offset value which is necessary for closed loop operation and offers read-write access. Use the write access in case it is desired to define a fixed offset value which has been tested first.

cl_calibration_en Set *cl_calibration_en* = 1 to update/reset *CL_OFFSET*. For evaluation of a proper value, be sure to meet the following conditions:

- Maximum current without scaling is used.
- *VACTUAL* is set to 0.
- *XACTUAL* refers to a motor fullstep position.

ENC_POS_DEV The deviation parameter *ENC_POS_DEV* contains the deviation between *XACTUAL* and *ENC_POS*.

CL_BETA *CL_BETA* is the maximum commutation angle which can be used to compensate an evaluated deviation *ENC_POS_DEV*. If the *CL_BETA* value becomes reached due to a large difference in microsteps, the *CL_MAX* event becomes triggered.

CL_TOLERANCE Set this parameter to choose a tolerance range for position deviation. In case the *CL_TOLERANCE* value is not exceeded and |*ENC_POS_DEV*| < *CL_TOLERANCE* the *CL_FIT_F*lag becomes set. If there has been a mismatch between *XACTUAL* and *ENC_POS* before, the *CL_FIT* event becomes triggered in order to indicate that everything fits now.

Note: *CL_TOLERANCE* is part of *PID_TOLERANCE*. A *PID_TOLERANCE* value is automatically chosen if the eight lower *CL_TOLERANCE* bits are set. The *PID_TOLERANCE* parameter contains 20 bits. The upper 12 bits can be used for further *PID_TOLERANCE* adjustment for the PI controller which sets the upper limit for the error correction velocity during closed loop operation.

CL_DELTA_P *CL_DELTA_P* is a proportional controller for compensating a detected position deviation. As soon as |*ENC_POS_DEV*| > *CL_TOLERANCE* the closed loop unit of the TMC4361 multiplies *ENC_POS_DEV* with *CL_DELTA_P* and adds the resulting value to the current *ENC_POS*. Thus, a current commutation angle for higher stiffness for position maintenance clipped at CL_BETA becomes calculated.

CL_DELTA_P consists of 24 bits. The last 16 bits represent decimal places. Figure 16.5 depicts how register values affect the final output angle at Step/Dir and/or SPI.

The final proportional term is calculated by

$$p_{PID} = \frac{CL\_DELTA\_P}{65536}$$

Note: the higher the p_{PID} term the faster the reaction on position deviations!

A high p_{PID} *term* can lead to oscillations which should be avoided. As long as *ENC_POS_DEV* is in the range of *XACTUAL* ± *CL_TOLERANCE*, the proportional term is automatically set to 1.

If |*ENC_POS_DEV*| > *CL_BETA*, the *CL_MAX* event triggers assumed that |*ENC_POS_DEV*| < *CL_BETA* has been valid before.

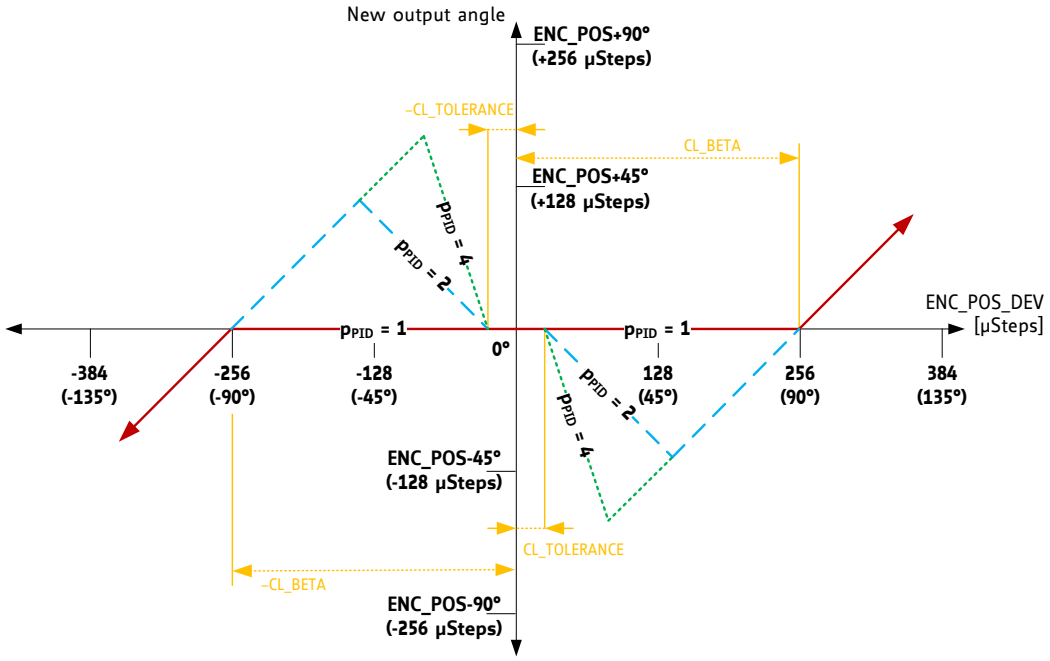


Figure 16.5 Calculation of the output angle by setting *CL_DELTA_P* properly.

Error Compensation in Case of Larger Deviations

In case a deviation *ENC_POS_DEV* between *X_ACTUAL* and *ENC_POS* exceeds the certain limit *ENC_POS_DEV_TOL*, *XACTUAL* becomes set to the *ENC_POS* value, assumed that *cl_clr_xact* is set to 1. Exceeding *ENC_POS_DEV_TOL* always sets a status flag *ENC_FAIL_F* and triggers the *ENC_FAIL* event.

Velocity Regulation Parameters

To limit catch-up velocities in case a disturbance of regular motor motion has to be compensated the following parameters can be adjusted.

cl_vlimit_en	Set *cl_vlimit_en* = 1 for limiting the maximum step velocity during closed loop operation. By setting *CL_VMAX_CALC_P* and/or *CL_VMAX_CALC_I* accordingly, a PI controller becomes used for velocity regulation of the current ramp. The PI controller calculates the maximum step velocity which is subsequently generated by the closed loop unit.
CL_VMAX_CALC_P	P parameter of the PI regulator which controls the maximum velocity.
CL_VMAX_CALC_I	I parameter of the PI regulator which controls the maximum velocity.
PID_DV_CLIP	To avoid large velocity variations and to limit the maximum velocity deviation above the maximum velocity *VMAX*, *PID_DV_CLIP* can be set. *PID_DV_CLIP* is used with closed loop and for PID controlled operation.
PID_I_CLIP	Together with *PID_DV_CLIP* this parameter is used for limiting the velocity for error compensation. The error sum *PID_ISUM* is generated by the

integral term. For limiting, set *PID_I_CLIP*. The maximum value of *PID_I_CLIP* should meet the condition *PID_I_CLIP* ≤ *PID_DV_CLIP* / *PID_I*. If the error sum *PID_ISUM* is not clipped, it is increased with each time step by *PID_I* · *PID_E*. This continues as long as the motor does not follow. The parameter *PID_DV_CLIP* is used with closed loop and for PID controlled operation.

In case position deviation at the end of an internal ramp calculation is still left, the SPI and/or Step/Dir output ramp for correction is a linear deceleration ramp, independently from the preset ramp type. This final ramp for error compensation is a function of *ENC_POS_DEV* and PI control parameters (*CL_VMAX_CALC_P*, *CL_VMAX_CALC_I*, *PID_I_CLIP*, and *PID_DV_CLIP*).
Due to the usage of a PI controller for the maximum velocity, the velocity offset depends on *ENC_POS_DEV*. It is recommended to set a limit for the error correction velocity using *PID_DV_CLIP* and *PID_I_CLIP*.

HINT Note that a higher velocity than *VMAX* resp. *-VMAX* is possible if the following conditions are met:

- The PI controller is enabled and *PID_DV_CLIP* > *0*.
- *CL_VMAX_CALC_P* and *CL_VMAX_CALC_I* are higher than 0.
- *ENC_POS_DEV* > *CL_TOLERANCE* resp. *ENC_POS_DEV* < *CL_TOLERANCE*.

Closed Loop Velocity mode

For some application it is only required to hold the maximum velocity no matter of the position deviation. This could be reached by setting *cl_velocity_mode_en* = 1. If this mode is switched on, the current position *XACTUAL* will be reset to (*CL_BETA* – 1) as soon as |*XACTUAL* – *ENC_POS*| ≥ *CL_BETA*. By activating the velocity regulation parameters (*cl_vlimit_en* = 1) and setting *PID_DV_CLIP* slightly higher than 0, the position deviation will recover as soon as the cause of the position mismatch will disappear. Setting *PID_DV_CLIP* > 0 will lead to slightly higher velocity than *VMAX* during recovering to *VACTUAL* = *VMAX* which could be unwanted. But setting *PID_DV_CLIP* = 0 will not overhaul the position deviation and a small value of *PID_DV_CLIP* will almost not be recognized.

How to Benefit from the Scaling Unit

The TMC4361 provides a scaling unit which can be used during closed loop operation. Therefore, set *closed_loop_scale_en* to 1.

How the scaling unit works

1. To decrease current consumption if *XACTUAL* and *ENC_POS* match, set *CL_IMIN* to an appropriate value which is the valid scaling value as long as |*ENC_POS_DEV*| ≤ *CL_START_UP*.
2. In case the threshold value *CL_START_UP* is exceeded, the current scaling value becomes increased linearly until |*ENC_POS_DEV*| = *CL_BETA*.
3. If |*CL_BETA*| is overshot also, *CL_IMAX* is the valid current scaling value.

HINT Note that any other scaling becomes disabled if closed loop scaling is enabled!

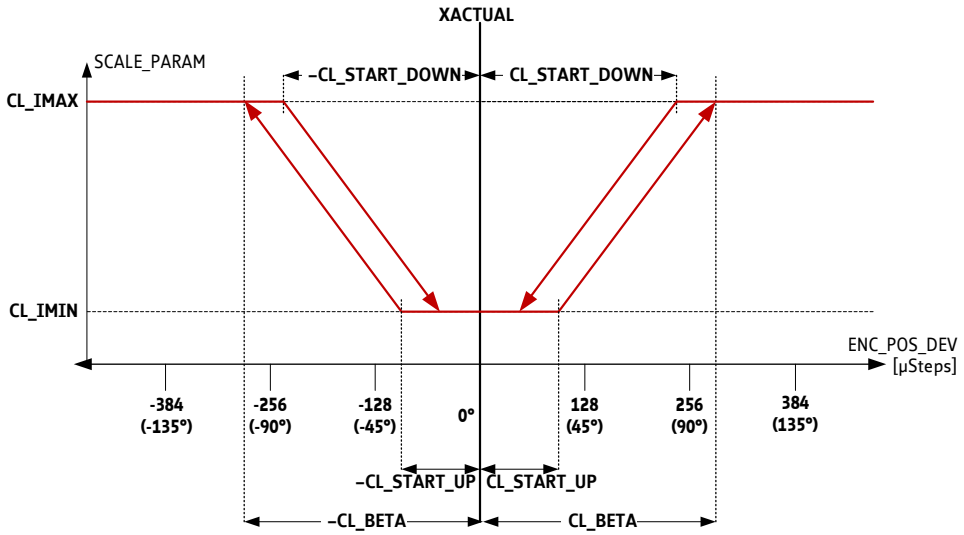


Figure 16.6 Current scaling in with closed loop

In case *CL_START_DOWN* is set to 0, *CL_BETA* is the starting point for downscaling. In contrast to the upscaling process the start of downscaling can differ from *CL_BETA*. Beware of oscillations due to the resulting hysteresis if *CL_START_DOWN* ≠ *CL_BETA*. The current scale parameter which regards to the position deviation *ENC_POS_DEV* is depicted in Figure 16.6. For evaluating different upscaling and/or downscaling ramps the delay parameters *CL_UPSCALE_DELAY* and *CL_DNSCALE_DELAY* can be used, too. These values define the clock cycles which are used to alter the current scale value for one step towards *CL_IMAX* resp. *CL_IMIN*. Figure 16.7 depicts the current scaling timing behavior as a function of *CL_UPSACLE_DELAY* and *CL_DNSCALE_DELAY.*

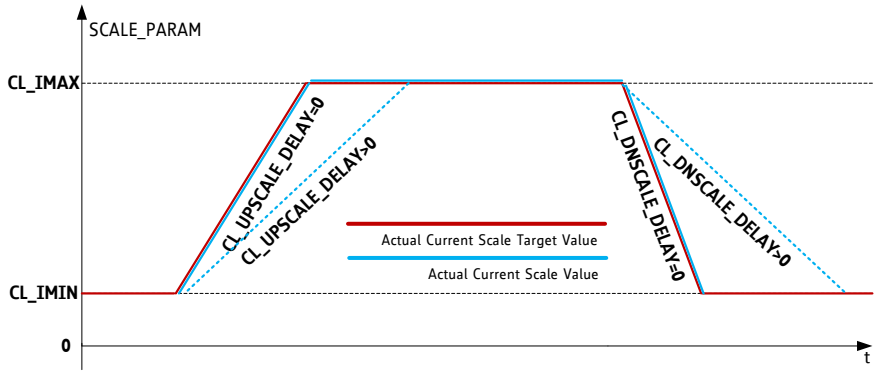


Figure 16.7 Current scaling timing behavior

16.4.3 Special Parameters for ABN Encoders Using Closed Loop

Consideration of Back EMF

For higher motor velocities and ABN encoders the load angle due to back EMF can be compensated. Therefore, set *cl_emf_en* = 1. The compensation angle normally does not exceed *CL_BETA*, but for back EMF error compensation another angle called *GAMMA* becomes added. Thereby, the direction of movement is considered.

The GAMMA value is greater than 0 if the encoder velocity *V_ENC_MEAN* exceeds *CL_VMIN_EMF* and GAMMA reaches its maximum value *CL_GAMMA* at *V_ENC_MEAN* = *CL_VMIN_EMF* + *CL_VADD_EMF* as depicted in Figure 16.8.

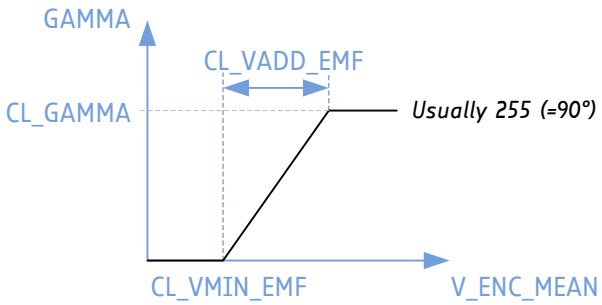


Figure 16.8 Calculation of the current load angle *CL_GAMMA*

Settings for Velocity Read Out

V_ENC The current encoder velocity *V_ENC* is calculated with every AB transition. If no AB transitions have been recognized for *ENC_VEL_ZERO* clock cycles, *V_ENC* is assigned to 0 and the *ENC_VEL0* event becomes triggered, assumed that the *V_ENC* value has not been zero before.

Note: V_ENC is always set to zero with absolute encoders.

V_ENC_MEAN Current filtered encoder velocity. This parameter is calculated every *ENC_VMEAN_WAIT* clock cycles with the filter exponent *ENC_VMEAN_FILTER*. The parameter ***V_ENC_MEAN* is crucial for *CL_GAMMA***. It is calculated as follows:

$$V_{ENC_{MEAN}} = V_{ENC_{MEAN}} - \frac{V_{ENC_{MEAN}}}{2^{ENC\_VMEAN\_FILTER}} + \frac{V_{ENC}}{2^{ENC\_VMEAN\_FILTER}}$$

ENC_VEL_ZERO Delay time after the last incremental encoder value change. After *ENC_VEL_ZERO* clock cycles *V_ENC_MEAN* is set to zero.

ENC_VMEAN_WAIT Set this time period [clock cycles] to choose a delay before the next current encoder velocity value becomes considered for *V_ENC_MEAN* calculation. It is recommend to set *ENC_VMEAN_WAIT* to a higher value than 16!

ENC_VELO Event becomes triggered: encoder velocity has reached zero.

HINT A good starting value for **ENC_VMEAN_WAIT** is 128 and for **ENC_VMEAN_FILTER** is 7. Both should be adapted in conjunction if ABN encoder are used. Further on, the lower both values are the faster the V_ENC_MEAN is adapted to the current velocity. But this also results in higher gradients of the mean velocity which could lead to regulation jumps if the GAMMA correction is enabled.
To prevent this, check the V_ENC_MEAN velocity values during motion transferring the velocity limits CL_VMIN_EMF and (CL_VMIN_EMF + CL_VADD_EMF). If the mean encoder velocity is adapted smoothly during motion GAMMA correction will be also executed properly.

16.5 Compensation of Encoder Misalignments

A deficiently installed encoder can send values which do not result in a circle. Often, the deviation from the real position results in a new function which is similar to a sine function. Adding offset that follows a triangular shape can improve the encoder value evaluation significantly (refer to Figure 16.9).

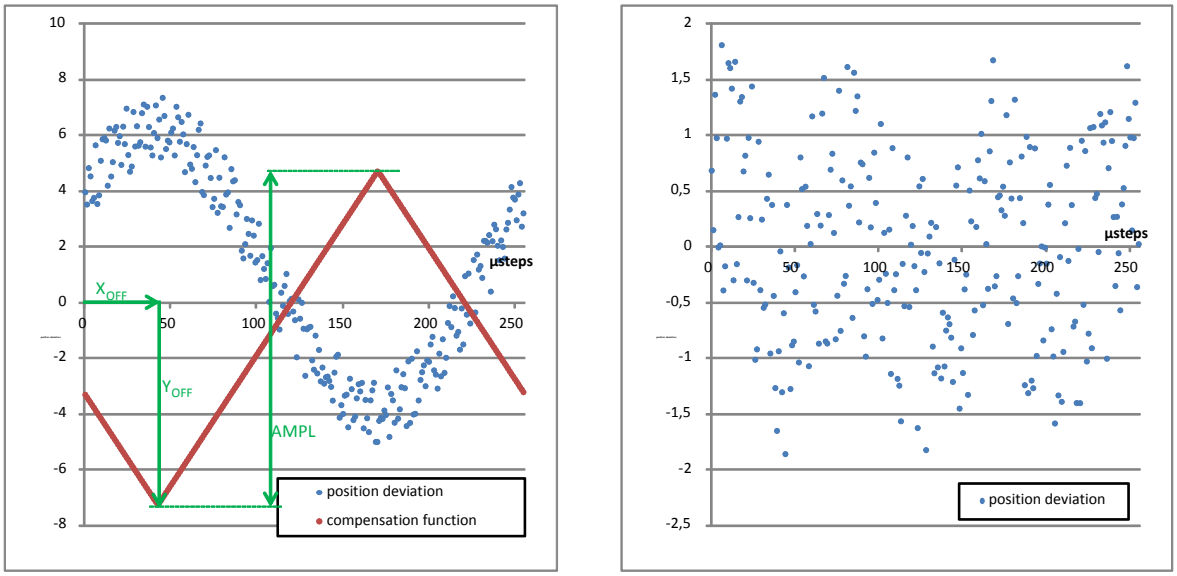


Figure 16.9 Implemented triangular function to compensate for encoder misalignments

The left graph illustrates the difference between encoder position and real position as a µstep function within the encoder resolution. After error compensation the position differences are minimized significantly as shown on the right side.

For error compensation, the following triangular function has to be calculated and mapped to the deviation function as offset:

$$AMPL - ENC\_COMP\_AMPL;\ X_{OFF} - ENC\_COMP\_XOFFSET;\ Y_{OFF} - ENC\_COMP\_YOFFSET$$

Parameter	Description
ENC_COMP_AMPL	Defines the maximum amplitude of the offset function.
ENC_COMP_XOFFSET	Define starting points of the offset function.
ENC_COMP_YOFFSET	

17 Serial Encoder Output Unit

The TMC4361 provides the possibility to render any regular encoder data into absolute SSI encoder data. The absolute SSI data is forwarded using the output pins which are used as normal SPI output otherwise.

PINS AND REGISTERS: SERIAL ENCODER OUTPUT UNIT

Pin names	Type		Remarks
NSCSDRV_SDO	Output		Serial data output
SCKDRV_NSDO	Output		Negated serial data output
SDODRV_SCLK	In/Out as Input		Serial clock input
SDIDRV_NSCLK	Input		Negated serial clock input
Register name	**Register address**		**Remarks**
GENERAL_CONF	0x00	RW	Bit25→24
SSI_OUT_MTIME	0x04	RW	Bit24→4 Monoflop time of serial encoder output
ENC_IN_CONF	0x07	RW	Bit15→14, bit30
ENC_OUT_DATA	0x09	RW	Number of data bits for encoder output structure
ENC_OUT_RES	0x55	W	Resolution of the singleturn data for encoder output

17.1 Providing SSI Output Data

For providing SSI output data the following steps and considerations have to be made:

- Generally, the internal *ENC_POS* is transferred into SSI output data. The structure of the output data can be altered freely to match master requirements.
- For switching from SPI output to SSI output, set *serial_enc_out_enable* to 1. Now, the master clock input SDO_DRV_SCLK switches to SSI protocol requirements. Thereafter, NSCSDRV_SDO acts as serial data output and sends data.
- Due to the regular differential SSI protocol the particular negated ports are SCKDRV_NSDO for data output and SDI_DRV_NSCLK for clock input. The evaluation of the differential clock input is based on the digital input levels. If *serial_enc_out_diff_disable* is set to 1, SSI is supported without differential pairs of data input and clock output.
- If *multi_turn_out_en* is set to 1, the output data consists of multiturn and singleturn data. Multiturn data is expressed as a signed number.
- If *multi_turn_out_en* is set to 0, only singleturn data (the angle as microsteps within one revolution) is transferred. The resolution for singleturn data can be set with *ENC_OUT_RES*. The internal *ENC_POS* parameter becomes matched to this resolution.
- Set the number of the single- and multiturn data with *SINGLE_TURN_RES_OUT* resp. *MULTI_TURN_RES_OUT*. The real number of data bits is the given parameter setting + 1 (see Figure 17.1).
- Additionally, the complete data can be gray coded by setting *enc_out_gray* to 1.
- After the last master request transferred data remains unchanged until *SSI_OUT_MTIME* clock cycles are expired to support multi cycle data transfers.

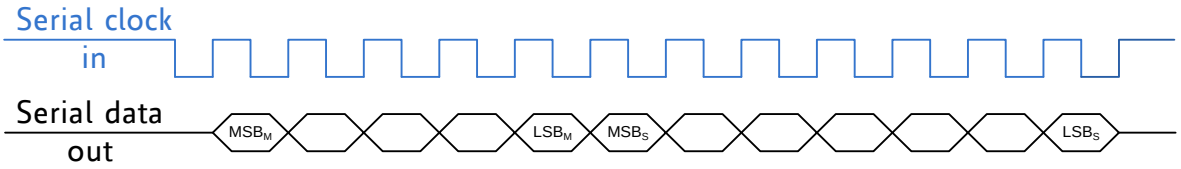


Figure 17.1 Example for SSI output configuration

M: MULIT_TURN_RES_OUT = 4 → transfer of 5 multiturn data bits
S: SINGLE_TURN_RES_OUT = 6→ transfer of 7 singleturn data bits

18 CLK Gating

If TMC4361 is not used for a while, clock gating can be used to reduce power consumption.

PINS AND REGISTERS: CLK GATING

Pin names	Type		Remarks
NSCS_IN	Input		Wakeup signal
CLK_EXT	Input		Input pin of external clock generator
Register name	**Register address**		**Remarks**
GENERAL_CONF	0x00	RW	Bit18→17
CLK_GATING_DELAY	0x14	RW	Delay time before clock gating is enabled
CLK_GATING_REG	0x4F	RW	Enable trigger for clock gating by setting bit2→0 to 111.

18.1 Clock Gating and Wake-up

CONFIGURATION FOR CLOCK GATING

1. To enable clock gating set *clk_gating_en* to 1.
2. Now, clock gating becomes active as soon as *CLK_GATING_REG* is set to 111.
3. For a delay time between setting the register and starting the clock gating itself, *CLK_GATING_DELAY* can be set accordingly. The delay is given in clock cycles.
4. Immediately after the start of the *CLK_GATING_DELAY* timer a *SLEEP_TIMER* event is triggered to indicate a soon clock gating phase.

HINT Clock gating affects every unit except the SPI input unit, the timing unit, and essential registers for freeze processes, which are fed directly by the external clock.

AUTOMATIC CLOCK GATING

It is possible to use automatic clock gating. Therefore, set *clk_gating_stdby_en* to 1. After the TMC4361 has reached the standby state clock gating becomes enabled assumed that *CLK_GATING_DELAY* is set to 0. Else, clock gating becomes started immediately after the clock gating timer expires.

WAKE-UP PROCEDURE

There are three possibilities for wake-up:

- To wake up from clock gating NSCS_IN has to switch to the active low level.
- The wake-up process can be automated by using the internal start signal for ramp initialization. So, shortly before the ramp starts, clock gating is cancelled for loading the essential registers for starting the ramp properly.
- External start signals can also be used to finish clock gating.

Figure 18.1 shows the process of manual clock gating (due to setting the appropriate register) and manual wake up (by using the SPI input lines) including different clock gating delays. First the clock gating timer has to expire and then the second phase of clock gating starts.

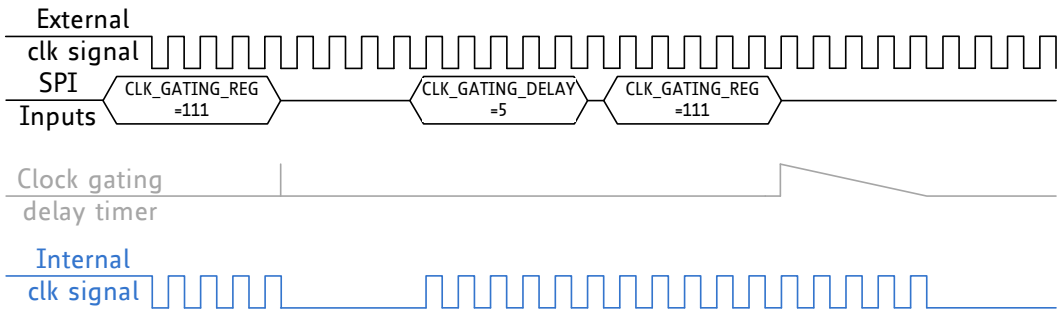


Figure 18.1 Manual clock gating and manual wake up.

Figure 18.2 depicts a complete automatic clock gating transition from inactive to active and back. Here, automatic clock gating is realized using the internal standby status (active high level at *STDBY_CLK* output) and the internal start timer. Clock gating ends slightly before the start signal becomes active. Thus, essential registers can be loaded before ramp start.

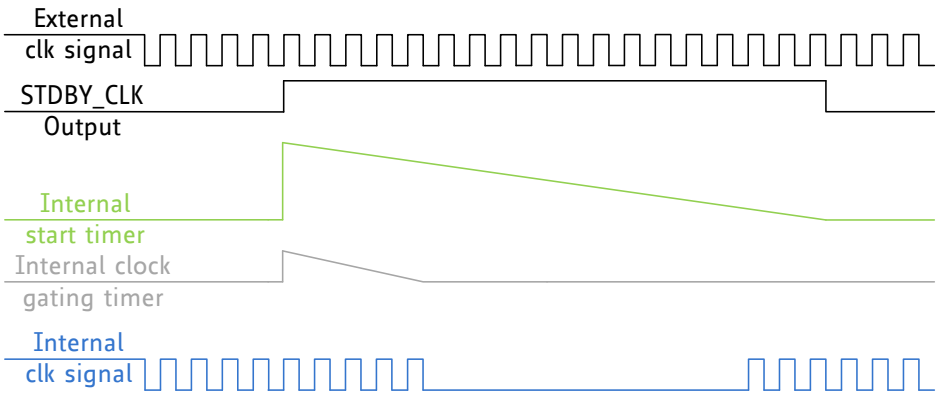


Figure 18.2 Automatic clock gating

HINT Note that manual and automated clock gating transition can be used together.

19 Registers and Switches

Changes as regards TMC4361old are displayed in green letters.

19.1 General Configuration

GENERAL CONFIGURATION (0X00)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x00	*GENERAL_CONF* *Default:* *0x00006020*	0	*use_astart_and_vstart* (only valid for S-shaped ramps) 0 if *VSTART* ≠ 0, *AACTUAL* will be set to *AMAX* or *-AMAX* 1 if *VSTART* ≠ 0, *AACTUAL* will be set to *ASTART* or *-ASTART*
			1	*direct_acc_val_en* 0 acceleration values are divided by CLK_FREQ 1 acceleration values are set directly as steps per clock cycle
			2	*direct_bow_val_en* 0 bow values are calculated due to division by CLK_FREQ 1 bow values are set directly as steps per clock cycle
			3	*step_inactive_pol* 0 STPOUT=1 indicates an active step 1 STPOUT=0 indicates an active step
			4	*toggle_step* 0 only STPOUT transitions from inactive to active polarity indicate steps 1 every level change of STPOUT indicates a step
			5	*pol_dir_out* 0 DIROUT = 0 indicates negative direction 1 DIROUT = 1 indicates negative direction
			7:6	*sdin_mode* 00 internal step control (internal ramp generator will be used) 01 external step control (STPIN/DIRIN) with high active steps 10 external step control (STPIN/DIRIN) with low active steps 11 external step control (STPIN/DIRIN) with toggling STPIN input signals
			8	*pol_dir_in* 0 DIRIN = 0 indicates negative direction 1 DIRIN = 1 indicates negative direction
			9	*sd_indirect_control* 0 STPIN/DIRIN input signals will manipulate steps directly 1 STPIN/DIRIN input signals will manipulate XTARGET register value, internal ramp generator usage despite *sdin_mode* ≠ 0
			11:10	*serial_enc_in_mode* 00 incremental encoder connected to encoder interface 01 absolute SSI encoder connected to encoder interface 10 reserved (prohibited to use!) 11 absolute SPI encoder connected to encoder interface
			12	*diff_enc_in_disable* 0 differential encoder interface inputs enabled 1 differential encoder interface inputs disabled (automatically for SPI encoder)
			14:13	*stdby_clk_pin_assignment* 00 Standby signal becomes forwarded with an active low level 01 Standby signal becomes forwarded with an active high level 10 *STDBY_CLK* passes ChopSync clock (TMC23x, TMC24x only) 11 Internal clock is forwarded to STDBY_CLK output pin
			15	*intr_pol* 0 *INTR*=0 indicates an active interrupt 1 *INTR*=1 indicates an active interrupt

GENERAL CONFIGURATION (0X00)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
			16	*invert_pol_target_reached* 0 TARGET_REACHED signal set to 1 indicates target reached event 1 TARGET_REACHED signal set to 0 indicates target reached event
			17	*clk_gating_en* 0 No clock gating. 1 Internal clock is gated due to clock gating register. A delay until sleep is possible (see clock gating timer register). Wakeup is possible due to active NSCSIN or if a start signal (internal or external) is generated.
			18	*clk_gating_stdby_en* 0 No clock gating due to standby phase. 1 Internal clock is gated due to standby of motion controller.
			19	*fs_en* 0 No fullsteps. 1 SPI output forwards fullsteps, if |*VACTUAL*| > *FS_VEL* (TMC4361 calculates internally with the µStep resolution)
			20	*fs_sdout* 0 No fullsteps for Step/Dir output. 1 Fullsteps are forwarded via Step/Dir output also.
			22:21	*dcstep_mode* 00 dcStep is disabled 01 dcStep signal generation will be selected automatically 10 dcStep with external STEP_READY signal generation (TMC21xx/51xx) 11 dcStep with internal STEP_READY signal generation (TMC26x) TMC26x config: use const_toff-Chopper (CHM = 1), slow decay only (HSTRRT = 0) and TST = 1 and SGT0=SGT1=1 (on_state_xy)
			23	*pwm_out_en* 0 PWM output disabled (Step/Dir out). 1 STPOUT/DIROUT used as PWM output (PWMA/PWMB).
			24	*serial_enc_out_enable* 0 No encoder connected to SPI output. 1 SPI output used as SSI encoder interface to pass out absolute SSI encoder data.
			25	*serial_enc_out_diff_disable* 0 Differential serial encoder output enabled or disabled (= 1)
			26	*automatic_direct_sdin_switch_off* 0 *VACTUAL*=0 & *AACTUAL*=0 after switching off Direct_SDin_mode *1 VACTUAL = VSTART and AACTUAL = ASTART after switching off direct_SDin_mode*
			27	*circular_cnt_as_xlatch* 0 X_LATCH will be forwarded at 0x36 1 REV_CNT (# of internal revolutions) will be forwarded at 0x36
			28	*reverse_motor_dir* 0 Direction of internal SinLUT is regular 1 Direction of internal SinLUT will be reversed
			29	intr_tr_pu_pd_en 0 INTR and TARGET_REACHED are only outputs 1 INTR and TARGET_REACHED will be used as outputs and with gated pull-up and/or pull-down functionality
			30	*intr_as_wired_and* 0 INTR output function could be used as wired-or 1 INTR output function could be used as wired-and
			31	*tr_as_wired_and* 0 TARGET_REACHED output function could be used as wired-or 1 TARGET_REACHED output function could be used as wired-and

19.2 Reference Switch Configuration

REFERENCE SWITCH CONFIGURATION (0X01)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x01	*REFERENCE_CONF* *Default:* *0x00000000*	0	*stop_left_en* 0 STOPL signal processing disabled. 1 STOPL signal processing enabled.
			1	*stop_right_en* 0 STOPR signal processing disabled. 1 STOPR signal processing enabled.
			2	*pol_stop_left* 0 Motor stops if STOPL signal is 0. 1 Motor stops if STOPL signal is 1.
			3	*pol_stop_right* 0 Motor stops if STOPR signal is 0. 1 Motor stops if STOPR signal is 1.
			4	*invert_stop_direction* 0 STOPL/STOPR stop motor in negative/positive direction. 1 STOPL/STOPR stop motor in positive/negative direction.
			5	*soft_stop_en* 0 Hard stop enabled. *VACTUAL* is immediately set to 0 on any external stop event. 1 Soft stop enabled. A linear velocity ramp is used for decreasing VACTUAL to v = 0.
			6	*virtual_left_limit_en* 0 Position limit *VIRT_STOP_LEFT* disabled. 1 Position limit *VIRT_STOP_LEFT* enabled.
			7	*virtual_right_limit_en* 0 Position limit *VIRT_STOP_RIGHT* disabled. 1 Position limit *VIRT_STOP_RIGHT* enabled.
			9:8	*virt_stop_mode* 00 The current ramp type defines the deceleration ramp triggered by a virtual stop event **(Prohibited for S-Shaped ramps!)**. 01 *VACTUAL* is set to 0 on a virtual stop event (hard stop) 10 Soft stop is enabled with linear velocity ramp (from *VACTUAL* to v = 0).
			10	*latch_x_on_inactive_l* 0 No latch of *XACTUAL* if *STOPL* becomes inactive. 1 *X_LATCH* = *XACTUAL* will be triggered if *STOPL* becomes inactive.
			11	*latch_x_on_active_l* 0 No latch of *XACTUAL* if *STOPL* becomes active. 1 *X_LATCH* = *XACTUAL* will be triggered if *STOPL* becomes active.
			12	*latch_x_on_inactive_r* 0 No latch of *XACTUAL* if *STOPR* becomes inactive. 1 *X_LATCH* = *XACTUAL* will be triggered if *STOPR* becomes inactive.
			13	*latch_x_on_active_r* 0 No latch of XACTUAL if STOPR becomes active. 1 *X_LATCH* = *XACTUAL* will be triggered if STOPR becomes active.
			14	*stop_left_is_home* 0 STOPL input signal is not HOME position. 1 STOPL input signal is also HOME position.
			15	*stop_right_is_home* 0 STOPR input signal is not HOME position. 1 STOPR input signal is also HOME position.

REFERENCE SWITCH CONFIGURATION (0X01)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
			19:16	*home_event* 0000 Next active N signal of ABN encoder signal indicates HOME position. 0011 HOME = 0 indicates negative region/position from home. 1100 HOME = 1 indicates negative region/position from home. 0110 HOME = 1 indicates an active home event – X_HOME is located in the middle of the active range. 0010 HOME = 1 indicates an active home event – X_HOME is located at the rising edge of the active range. 0100 HOME = 1 indicates an active home event – X_HOME is located at the falling edge of the active range. 1001 HOME = 0 indicates an active home event – X_HOME is located in the middle of the active range. 1011 HOME = 0 indicates an active home event – X_HOME is located at the rising edge of the active range. 1101 HOME = 0 indicates an active home event – X_HOME is located at the falling edge of the active range.
			20	*start_home_tracking* 0 No storage of *X_HOME* = *XACTUAL* by passing home position. 1 Storage of *XACTUAL* as *X_HOME* at next regular home event. The switch will be reset after an executed homing. An XLATCH_DONE event will be also released then.
			21	*clr_pos_at_target* 0 Ramp stops at *XTARGET* if positioning mode is active. 1 Set *XACTUAL* = 0 after *XTARGET* has been reached. The next ramp starts immediately.
			22	*circular_movement_en* 0 range of *XACTUAL* will not be limited 1 range of *XACTUAL* will be limited to *-X_RANGE ≤ XACTUAL ≤ (X_RANGE - 1)*
			24:23	*pos_comp_output* 00 *TARGET_REACHED* is set active on *TARGET_REACHED* event. 11 *TARGET_REACHED* triggers on *POSCOMP_REACHED* event.
			25	*pos_comp_source* 0 *POS_COMP* is compared to internal position *XACTUAL*. 1 *POS_COMP* is compared with external position ENC_POS.
			26	*stop_on_stall* 0 Motor will not be stopped in case of stall. 1 Motor will be stopped with a hard stop in case of stall.
			27	*drv_after_stall* Set to 1 and reset *stop_on_stall* flag. Moving the motor is not possible as long as this flag is not set after a *stop_on_stall* event.
			29:28	*modified_pos_compare: POS_COMP_REACHED_F I* event is based on comparison between *XACTUAL* resp. *ENC_POS* and 00 POS_COMP 01 X_HOME 10 X_LATCH / ENC_LATCH 11 REV_CNT
			30	*automatic_cover* 0 SPI output will not transfer automatically any cover datagram 1 SPI output interface will send automatically cover datagrams when *VACTUAL* crosses *SPI_SWITCH_VEL.*
			31	*circular_enc_en* 0 range of *ENC_POS* will not be limited 1 range of *ENC_POS* will be limited to *-X_RANGE ≤ ENC_POS ≤ (X_RANGE - 1)*

19.3 Start Switch Configuration

START SWITCH CONFIGURATION (0X02)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x02	*START_CONF* *Default:* *0x00000000*	4:0	*start_en* xxxx1 Change of *XTARGET* requires a distinct start signal. xxx1x Change of *VMAX* requires a distinct start signal. xx1xx Change of *RAMPMODE* requires a distinct start signal. x1xxx Change of *GEAR_RATIO* requires a distinct start signal. 1xxxx Usage of shadow registers enabled.
			8:5	*trigger_events* 0000 Start signal generation is disabled. xxx0 START pin is assigned as output. xxx1 External START signal is assigned as start signal for timer. xx1x *TARGET_REACHED* event is assigned as start signal for timer. x1xx *VELOCITY_REACHED* event is assigned as start signal for timer. 1xxx *POSCOMP_REACHED* event is assigned as start signal for timer.
			9	*pol_start_signal* (same polarity for input or output) 0 START = 0 is active start polarity 1 START = 0 is inactive start polarity
			10	*immediate_start_in* 0 Active START input starts internal start timer 1 Active START will be executed immediately
			11	*busy_state_en* 0 START pin is used as input or output only 1 Busy start state enabled: Before the internal start is valid, start output will be set busy (strong inactive polarity). Then, if the internal start signal is generated (after start timer is elapsed), the start signal is set to the active polarity (weak output driving strength). If the signal at the START input is set to the active polarity, e.g. because all members at the signal line are ready, the START output remains active (strong driving strength) for *START_OUT_ADD* clock cycles. Then, busy state is active again until the next start signal.
			15:12	*pipeline_en* 0000 No pipeline is active. xxx1 *X_TARGET* will be considered for pipelining. xx1x *POS_COMP* will be considered for pipelining. x1xx *GEAR_RATIO* will be considered for pipelining. 1xxx *GENERAL_CONF* will be considered for pipelining.
			17:16	*shadow_option* 00 Single-level shadow registers for 13 relevant ramp parameters 01 Double-stage shadow register (appropriate for S-shaped ramps). 10 Double-stage shadow register (appropriate for trapezoidal ramps without *VSTOP* consideration). 11 Double-stage shadow register (appropriate for trapezoidal ramps without *VSTART* consideration).
			18	*cyclic_shadow_regs* 0 Current ramp parameters will not written back to the shadow regs 1 Current ramp parameters will be written back to the appropriate shadow registers at the next internal start signal
			19	*synchro_via_mp_pins* 1 Closed loop state of different TMC4361 could be coordinated by the usage of the multipurpose pins MP1 and MP2
			23:20	*SHADOW_MISS_CNT:* Number of unused start signals between two consecutive shadow register transfers
			31:24	*XPIPE_REWRITE_REG* Indicates which *X_PIPEx* register becomes changed to *X_TARGET* value on next internal start event and if *x_pipeline_en* ≠ 0.

19.4 Input Filter Configuration

INPUT FILTER CONFIGURATION (0x03)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x03	*INPUT_FILT_CONF* *Default:* *0x00000000*	2:0	*SR_ENC_IN* Input sample rate = $f_{CLK}/2^{SR\_ENC\_IN}$ for A_SCLK, ANEG_NSCLK, B_SDI, BNEG_NSDI, N, and NNEG
			3	Reserved. Set to 0.
			6:4	*FILT_L_ENC_IN* # additionally sampled input bits whose voltage level has to be equal to the recently sampled bit to provide a valid input bit level for A_SCLK, ANEG_NSCLK, B_SDI, BNEG_NSDI, N, and NNEG.
			7	SD_FILT0: Selection bit to assign SD input interface pins (STPIN, DIRIN) to the _ENC_IN input filter group.
			10:8	*SR_REF* Input sample rate = $f_{CLK} / 2^{SR\_REF}$ for STOPR, HOME_REF, and STOPL.
			11	Reserved. Set to 0.
			14:12	*FILT_L_REF* # additionally sampled input bits whose voltage level has to be equal to the recently sampled bit to provide a valid input bit level for STOPR, HOME_REF, and STOPL.
			15	SD_FILT1: Selection bit to assign SD input interface pins (STPIN, DIRIN) to the _REF input filter group.
			18:16	*SR_S* Input sample rate = $f_{CLK} / 2^{SR\_S}$ for START.
			19	Reserved. Set to 0.
			22:20	*FILT_L_S* # additionally sampled input bits whose voltage level has to be equal to the recently sampled bit to provide a valid input bit level for START.
			23	SD_FILT2: Selection bit to assign SD input interface pins (STPIN, DIRIN) to the _S input filter group.
			26:24	*SR_ENC_OUT* Input sample rate = $f_{CLK} / 2^{SR\_ENC\_OUT}$ for SDODRV_SCLK, and SDIDRV_NSCLK.
			27	Reserved. Set to 0.
			30:28	*FILT_L_ENC_OUT* # additionally sampled input bits whose voltage level has to be equal with the recently sampled bit to provide a valid input bit level for SDODRV_SCLK, and SDIDRV_NSCLK
			31	SD_FILT3: Selection bit to assign SD input interface pins (STPIN, DIRIN) to the _ENC_OUT input filter group.

19.5 SPI-Out Configuration

SPI-OUT CONFIGURATION (0X04)

R/W	Addr	*Reg name*	Bit	Description
RW	0x04	*SPIOUT_CONF* *Default:* *0x00000000*	**BASIC SPI-OUT SETTINGS**	
			3:0	*spi_output_format* 0000 SPI-Out off 1000 SPI-Out connected to TMC23x driver 1001 SPI-Out connected to TMC24x driver 1010 SPI-Out connected to TMC26x/389 1011 SPI-Out connected to TMC26x/389. Steps only via STPOUT. 1100 SPI-Out connected to TMC21xx/51xx. Steps only via STPOUT. 1101 SPI-Out connected to TMC21xx/51xx. 0100 The actual unsigned scaling factor is assigned to SPI-Out. 0101 Both actual signed SinLUT values are assigned to SPI-Out. 0110 The actual unsigned scaling factor is merged with DAC_ADDR_A as output for a SPI-DAC. 0010 SPI-Out is connected with a DAC. Absolute values. Phase of coilA via STPOUT. Phase of coilB via DIROUT. Phase bit = 0 : positive values. 0011 SPI-Out is connected with a DAC. Absolute values. Phase of coilA via STPOUT. Phase of coilB via DIROUT. Phase bit = 1 : positive values. 0001 SPI-Out is connected with a DAC. Values are mapped. Current = 0 : VDD/2 Current = -(max_value) : 0 Current = max_value : VDD 1111 Only SPI cover datagrams will be transferred
			19:13	*COVER_DATA_LENGTH* (0…64) Set to 0 for automatic assign, if a TMC driver is connected. Otherwise it will set to 1 if *COVER_DATA_LENGTH* = 0 and no TMC driver is selected.
			23:20	*SPI_OUT_LOW_TIME* SPI-Output clock low phase [clock cycles]
			27:24	*SPI_OUT_HIGH_TIME* SPI-Output clock high phase [clock cycles]
			31:28	*SPI_OUT_BLOCK_TIME* SPI-Output blockage time [clock cycles]
			SETTINGS USED WITH SERIAL ENCODER OUTPUT ONLY: *serial_enc_out_enable* = 1	
			23:4	*SSI_OUT_MTIME* SSI output monoflop time: delay time during which the absolute data remain stable after the last master request. [clock cycles]

SPI-OUT CONFIGURATION (0X04)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
			SETTINGS USED WITH TMC23X/24X MOTOR DRIVERS ONLY: *spi_output_format*(3:1) = b'100	
			5:4	*mixed_decay* (coilA and coilB for a TMC23x or TMC24x driver) 00 Both mixed decay bits are always off. 01 During falling ramps until reaching the value of 0 mixed decay bits are on. 10 Both mixed decay bits are always on, except standby mode is active. 11 Both mixed decay bits are always on.
			6	*stdby_on_stall_for_24x* (TMC24x only) 0 No standby datagram. 1 In case of a *stop_on_stall* event a standby datagram is sent to the TMC24x driver.
			7	*stall_flag_instead_of_uv_en* (TMC24x only) 0 Undervoltage flag is forwarded as STATUS(24). 1 Calculated stall status of TMC24x is forwarded as STATUS(24).
			10:8	*STALL_LOAD_LIMIT* (TMC24x only) A stall is detected if *STALL_LOAD_LIMIT* ≥ (LD2&LD1&LD0) of the driver status.
			11	*pwm_phase_shft_enable* 0 No phase shift during PWM mode. 1 During PWM mode the internal SinLUT microstep position will be shifted for *MS_OFFSET* microsteps to shift the sine/cosine values for a phase shift of (*MS_OFFSET* / 1024 · 360°)
			SETTINGS USED WITH TMC26XX/21XX MOTOR DRIVER ONLY: *spi_output_format* = b'101x or b'110x	
			4	*three_phase_stepper_en* (TMC26x/389 only) 0 A 2-phase stepper driver is connected (TMC26x) 1 A 3-phase stepper driver is connected (TMC389)
			5	*scale_val_transfer_en* 0 No transfer of scale values to the TMC driver. 1 Transfer of current scale values to the correct driver registers.
			6	*disable_polling* 0 Permanent transfer of datagrams to check driver status (Step/Dir output only) 1 No transfer of polling datagrams. (recommended for *spi_output_format* = b'1101)
			12:7	*POLL_BLOCK_MULT* Multiplier for calculating the time interval between consecutive polling datagrams. t_{POLL} = (*POLL_BLOCK_MULT*+1) · *SPI_OUT_BLOCK_TIME* / f_{CLK}
			SETTINGS USED WITH MOTOR DRIVERS FROM THIRD PARTIES: *spi_output_format*(3) = 0	
			4	*sck_low_before_csn* 0 NSCSDRV_SDO tied low before SCKDRV_NSDO 1 SCKDRV_NSDO tied low before NSCSDRV_SDO
			5	*new_out_bit_at_rise* 0 SDODRV_SCLK is shifted at falling edge of SCKDRV_NSDO. 1 SDODRV_SCLK is shifted at rising edge of SCKDRV_NSDO.
			11:7	*DAC_CMD_LENGTH* # of bits for command address if *spi_output_format* = b'0001 or b'0010 or b'0011 or b'0110.

19.6 Current Configuration

CURRENT CONFIGURATION (0X05)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x05	*CURRENT_CONF* *Default:* *0x00000000*	0	*hold_current_scale_en* 0 No hold current scaling during standby phase. 1 Hold current scaling during standby phase.
			1	*drive_current_scale_en* 0 No drive current scaling during motion. 1 Drive current scaling during motion.
			2	*boost_current_on_acc_en* 0 No boost current scaling for acceleration ramps. 1 Boost current scaling if *RAMP_STATE* = b'01.
			3	*boost_current_on_dec_en* 0 No boost current scaling for deceleration ramps. 1 Boost current scaling if *RAMP_STATE* = b'10.
			4	*boost_current_after_start_en* 0 No boost current at ramp start. 1 Temporary boost current if *VACTUAL* = 0 and new ramp starts.
			5	*sec_drive_current_scale_en* 0 One drive current value for the whole motion ramp. 1 Second drive current scaling for *VACTUAL* > *VDRV_SCALE_LIMIT*.
			6	*freewheeling_en* 0 No freewheeling. 1 Freewheeling after standby phase.
			7	*closed_loop_scale_en* 0 No closed loop current scaling. 1 Closed loop current scaling: Sets *CURRENT_CONF*(6:0) = 0 *Turn off for closed loop calibration with maximum current!*
			8	pwm_scale_reg_chn 0 *PWM_AMPL* value will be taken from register 0x06(15:0). 1 *PWM_AMPL* value will be taken from register 0x05(31:16).
			15:9	Reserved. Set to b'0000000.
			31:16	*PWM_AMPL* if pwm_scale_reg_chn = 1

19.7 Current Scale Values

CURRENT SCALE VALUES (0X06)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x06	*SCALE_VALUES* *Default:* *0xFFFFFFFF*	7:0	*BOOST_SCALE_VAL:* Open loop boost scaling value.
				CL_IMIN: closed loop minimum scaling value.
			15:8	*DRV1_SCALE_VAL:* Open loop first drive scaling value.
				CL_IMAX: closed loop maximum scaling value.
			23:16	*DRV2_SCALE_VAL:* Open loop second drive scaling value.
				CL_START_UP: |*ENC_POS_DEV*| value at which closed loop scaling increases the current scaling value above *CL_IMIN*.
			31:24	*HOLD_SCALE_VAL:* Open loop standby scaling value.
				CL_START_DOWN: |*ENC_POS_DEV*| value at which closed loop scaling decreases the current scaling value below *CL_IMAX*. If set to 0 it is automatically equal to *CL_BETA*.
			15:0	*PWM_AMPL:* PWM amplitude at *VACTUAL* = 0. Maximum duty cycle = (0.5 + (*PWM_AMPL* + 1) / 2^{17}) Minimum duty cycle = (0.5 – (*PWM_AMPL* + 1) / 2^{17}) PWM_AMPL = 2^{16} – 1 at *VACTUAL* = *PWM_VMAX*

Use following scale values: Real scaling value = (x+1) / 32 if *spi_output_format* = b'1011 or b'1100
= (x+1) / 256 any other *spi_output_format* setting

19.8 Encoder Signal Configuration

ENCODER SIGNAL CONFIGURATION (0X07)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x07	*ENC_IN_CONF* *Default:* *0x00000400*	0	*enc_sel_decimal* 0 Encoder constant represents a binary number. 1 Encoder constant represents a decimal number (for ABN only).
			1	*clear_on_n* 0 *ENC_POS* is not reset at active N events. 1 *ENC_POS* is set to 0 or *ENC_RESET_VAL* on every N event. *Do not use for closed loop operation!* *ENC_IN_CONF(2) or ENC_IN_CONF(3) have to be set!*
			2	*clr_latch_cont_on_n* 1 Value of *ENC_POS* is cleared and/or latched to *ENC_LATCH* register on every N event.
			3	*clr_latch_once_on_n* 1 Value of *ENC_POS* is cleared and/or latched to *ENC_LATCH* register on the next N event. *This bit is set to 0 after latching/clearing once.*
			4	*pol_n* 0 Active polarity for N event is low level. 1 Active polarity for N event is high level.
			6:5	*n_chan_sensitivity* 00 N event is active as long as N is active. 01 N event triggers when N becomes active (positive edge). 10 N event triggers when N becomes inactive (negative edge) 11 N event triggers when N becomes active or inactive (both edges)
			7	*pol_a_for_n* 0 A polarity has to be low for a valid N event. 1 A polarity has to be high for a valid N event.
			8	pol_b_for_n 0 B polarity has to be low for a valid N event 1 B polarity has to be high for a valid N event
			9	*ignore_ab* 0 Consider A and B polarities for a valid N event. 1 Ignore polarities of A and B signals for a valid N event.
			10	latch_enc_on_n 0 no latch of ENC_POS on an active N event 1 ENC_LATCH = ENC_POS triggered on an active N event *ENC_IN_CONF(2) or ENC_IN_CONF(3) have to be set!*
			11	*latch_x_on_n* 0 Do not latch *XACTUAL* on active N event. 1 *X_LATCH = XACTUAL* triggered on an active N event.
			12	*multi_turn_in_en* 0 Serial encoder input transmits singleturn values. 1 Serial encoder input transmits singleturn and multiturn values.
			13	*multi_turn_in_signed* 0 Multiturn values from serial encoder input are unsigned numbers. 1 Multiturn values from serial encoder input are signed numbers.
			14	*multi_turn_out_en* 0 Serial encoder output transmits singleturn values. 1 Serial encoder output transmits singleturn and multiturn values.
			15	*use_usteps_instead_of_xrange* 0 *X_RANGE* is also valid if circular movement is used for encoders 1 *USTEPS_PER_REV* is valid if circular movement is used for encoders
			16	*calc_multi_turn_behav* 0 No multiturn calculation. 1 Multiturn calculation for singleturn encoder data.

ENCODER SIGNAL CONFIGURATION (0X07)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
			17	*ssi_multi_cycle_data* 0 Every absolute SSI value request is executed once. 1 Every absolute SSI value request is executed twice.
			18	*ssi_gray_code_en:* SSI input data is binary coded (=0) or gray coded (=1).
			19	*left_aligned_data* 0 Serial input data is aligned right (first flags, then data). 1 Serial input data is aligned left (first data, then flags).
			20	*spi_data_on_cs* (SPI encoder only (serial_enc_in_mode = b'11)) 0 BNEG_NSDI will provide output data at next A_SCLK transition 1 BNEG_NSDI will provide output data immediately if ANEG_NSCLK input signal level is tied low transition
			21	*spi_low_before_cs* (SPI encoder only (serial_enc_in_mode = b'11)) 0 A_SCLK will tied low after ANEG_NSCLK switches to low level 1 A_SCLK will tied low before ANEG_NSCLK switches to low level
			23:22	*regulation_modus* 00 No feedback consideration. 01 Closed loop operation. 10 PID regulation. Pulse generator base is zero. 11 PID regulation. Pulse generator base is *VACTUAL*.
			24	*cl_calibration_en* 0 No closed loop calibration. 1 Closed loop calibration is active. *Use maximum current without scaling during calibration!* *The motor driver should be positioned at a fullstep position and VACTUAL has to be set to 0 during the calibration process!*
			25	*cl_emf_en* 0 Do not consider back EMF during closed loop operation. 1 Closed loop operation considers back EMF if *VACTUAL* > *CL_VMIN*.
			26	*cl_clr_xact* 0 *ENC_POS_DEV* will not evaluated to manipulate *X_ACTUAL* 1 *XACTUAL* = *ENC_POS*, if *ENC_POS_DEV* > *ENC_POS_DEV_TOL*
			27	*cl_vlimit_en* 0 No maximum velocity limit for closed loop regulation. 1 PI maximum velocity regulation during closed loop operation.
			28	*cl_velocity_mode_en* 0 No limit for difference between *XACTUAL* and *ENC_POS* 1 Difference between *XACTUAL* and *ENC_POS* is limited to CL_OFFSET + 768 µsteps (+3 fullsteps @ 256 µSteps/FS). If the limit is exceeded *XACTUAL* will be set accordingly.
			29	*invert_enc_dir:* Set this parameter to 1 for inverting the *ENC_POS* value.
			30	*enc_out_gray:* SSI output data is binary coded (=0) or gray coded (=1).
			31	*no_enc_vel_preproc* = 0 AB signal is preprocessed for encoder velocity (could only be used with *serial_enc_in_mode* = b'00). Set *no_enc_vel_preproc* to 1 to end AB signal preprocessing. *serial_enc_variation_limit* = 1 Two consecutive serial encoder values must no vary more than one eighth of the encoder resolution *ENC_IN_RES* to be valid. This setting can only be used with *serial_enc_in_mode* ≠ 0.

19.9 Serial Encoder Data IN

SERIAL ENCODER DATA IN (0X08)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x08	*ENC_IN_DATA* *Default:* *0x00000000*	4:0	*SINGLE_TURN_RES* # data bits for angle within one revolution = SINGLE_TURN_RES + 1.
			9:5	*MULTI_TURN_RES* # data bits for revolution count = MULTI _TURN_RES + 1. If *MULTI _TURN_RES* = 0, no data bits will be expected.
			11:10	*STATUS_BIT_CNT:* # bits of status data
			15:12	*CRC_BIT_CNT:* Length of CRC polynomial (#bits)
			23:16	*SERIAL_ADDR_BITS* # bits for addresses within SPI datagram / BiSS frame.
			31:24	*SERIAL_DATA_BITS* # bits for data within SPI datagram / BiSS frame.

19.10 Serial Encoder Data OUT

SERIAL ENCODER DATA OUT (0X09)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x09	*ENC_OUT_DATA* *Default:* *0x00000000*	4:0	*SINGLE_TURN_RES_OUT:* # data bits for angle within one revolution = SINGLE_TURN_RES_OUT + 1.
			9:5	*MULTI_TURN_RES_OUT* # data bits for revolution count = MULTI_TURN_RES_OUT + 1.
			31:10	Reserved. Set all bits to 0.

19.11 Motor Driver Settings

MOTOR DRIVER SETTINGS (0X0A)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x0A	*STEP_CONF* *Default:* *0x 00FB0C80*	3:0	*MSTEP_PER_FS* b'0000 Highest µstep resolution: 256 µsteps per fullstep. b'0001… b'0111 128 µsteps … half steps b'1000 Full steps (maximum possible setting) Note: - *Set to 256 for closed loop operation.* - When using a Step/Dir driver, it must be capable of a 256 resolution via Step/Dir input for best performance (but lower resolution Step/Dir drivers can be used as well).
			15:4	*FS_PER_REV:* Fullsteps per revolution
			23:16	*MSTATUS_SELECTION*: ORed with Motor Driver Status Register Set (7→0) → if set here & particular flag is set, an event will be generated at EVENTS(30)
			31:24	Reserved: Set to b'00000000

19.12 Event Selection Registers

EVENT SELECTION				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
RW	0x0B	*SPI_STATUS_SELECTION* *Default: 0x82029805*	31:0	Events which bits are selected (=1) in this register are forwarded to the eight status bits that are transferred with every SPI datagram (first eight bits from LSB are significant!).
	0x0C	*EVENT_CLEAR_CONF* *Default: 0x00000000*	31:0	Events which bits are selected (=1) in this register are not cleared by reading out the EVENTS register 0x0E.
	0x0D	*INTR_CONF* *Default: 0x00000000*	31:0	ORed with interrupt event register set : if set here and the particular flag is set an interrupt becomes generated.

19.13 Status Event Register

STATUS EVENTS (0X0E)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
R+C	0x0E	*EVENTS* *Default:* *0x00000000*	0	*TARGET_REACHED* has been triggered.
			1	*POS_COMP_REACHED* has been triggered.
			2	*VEL_REACHED* has been triggered.
			3	*VEL_STATE* = b'00 has been triggered (*VACTUAL* = 0).
			4	*VEL_STATE* = b'01 has been triggered (*VACTUAL* > 0).
			5	*VEL_STATE* = b'10 has been triggered (*VACTUAL* < 0).
			6	*RAMP_STATE* = b'00 has been triggered (*AACTUAL* = 0).
			7	*RAMP_STATE* = b'01 has been triggered (|*AACTUAL*| increases).
			8	*RAMP_STATE* = b'10 has been triggered (|*AACTUAL*| decreases).
			9	*MAX_PHASE_TRAP*: Trapezoidal ramp has reached its limit speed using maximum values for *AMAX* and *DMAX* for acceleration/deceleration (*VACTUAL* > *VBREAK*; *VBREAK* ≠ 0).
			10	*FROZEN:* NFREEZE has been tied low. *Reset TMC4361 for further motion!*
			11	*STOPL* has been triggered. Movement in negative direction is not executed until this event is cleared and (STOPL is not active any more or *stop_left_en* is set to 0).
			12	*STOPR* has been triggered. Movement in positive direction is not executed until this event is cleared and (STOPR is not active any more or *stop_right_en* is set to 0).
			13	*VSTOPL_ACTIVE: VSTOPL* has been activated. No further movement in negative direction until this event is cleared and (a new value is chosen for *VSTOPL or X_ACTUAL* or set *virtual_left_limit_en* = 0).
			14	*VSTOPR_ACTIVE: VSTOPR* has been activated. No further movement in positive direction until this event is cleared and (a new value is chosen for *VSTOPR or X_ACTUAL* or set *virtual_right_limit_en* = 0).
			15	*HOME_ERROR: HOME_REF has wrong polarity and* is outside of safety margin around *X_HOME.*
			16	*XLATCH_DONE*: indicates if *X_LATCH* or *X_HOME* has been newly written.
			17	*FS_ACTIVE:* Fullstep has been activated.
			18	*ENC_FAIL:* Mismatch between *XACTUAL* and *ENC_POS* has been triggered.
			19	*N_ACTIVE:* Active N event has been triggered.
			20	*ENC_DONE* indicates if *ENC_LATCH* has been newly written.
			21	*SER_ENC_DATA_FAIL:* Failure during multi cycle data evaluation or between two consecutive data requests.
			22	*reserved*
			23	*SER_DATA_DONE:* Data has been received from serial encoder (SPI, BiSS).
			24	One of the *SERIAL_ENC_FLAGS* has been set.
			25	*COVER_DONE:* SPI datagram have been sent to the motor driver.
			26	*ENC_VEL0:* Encoder velocity has been reached 0.
			27	*CL_MAX:* Closed loop commutation angle has reached maximum value.
			28	*CL_FIT: CL_FIT_F* has been triggered.
			29	*STOP_ON_STALL:* Motor stall detected. Motor has been stopped.
			30	*MOTOR_EV:* One of the chosen TMC motor driver flags has been triggered.
			31	*RST* has been activated. (divergence to TMC4361old where *SLEEP_TIMER* event has been reported here).

19.14 Status Flag Register

STATUS FLAGS (0X0F)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
R	0x0F	*STATUS* *Default:* *0x00000000*	0	*TARGET_REACHED_F* is set high if *XACTUAL* = *XTARGET*
			1	*POS_COMP_REACHED_F* is set high if *XACTUAL* = *POS_COMP*
			2	*VEL_REACHED_F* is set high if *VACTUAL* = abs(*VMAX*)
			4:3	*VEL_STATE_F:* Current velocity state: 00 *VACTUAL* = 0 01 *VACTUAL* > 0 10 *VACTUAL* < 0
			6:5	*RAMP_STATE_F:* Current ramp state. 00 *AACTUAL* = 0 01 Acceleration phase: *AACTUAL* > 0 if *VACTUAL* > 0 *or* *AACTUAL* < 0 if *VACTUAL* < 0 10 Deceleration phase: *AACTUAL* < 0 if *VACTUAL* > 0 *or* *AACTUAL* > 0 if *VACTUAL* < 0
			7	*STOPL_ACTIVE_F:* Left stop switch is active.
			8	*STOPR_ACTIVE_F:* Right stop switch is active.
			9	*VSTOPL_ACTIVE_F:* Left virtual stop switch is active.
			10	*VSTOPR_ACTIVE_F:* Right virtual stop switch is active.
			11	*ACTIVE_STALL_F:* Motor stall is detected and *VACTUAL* > *VSTALL_LIMIT*.
			12	*HOME_ERROR_F* HOME_REF input signal level is not equal to the expected home level
			13	*FS_ACTIVE_F:* Fullstep operation is active.
			14	*ENC_FAIL_F:* Mismatch between *XACTUAL* and *ENC_POS* is out of tolerated range *ENC_POS_DEV_TOL*.
			15	*N_ACTIVE_F:* N event is active.
			16	*ENC_LATCH_F: ENC_LATCH* is newly written.
			17	*MULTI_CYCLE_FAIL_F (serial_enc_in_mode* ≠ 00): indicates a failure during last multi cycle data evaluation. *SER_ENC_VAR_F (serial_enc_in_mode* ≠ 00): indicates a failure during last serial data evaluation due to a substantial deviation between two consecutive serial data values (*serial_enc_variation_limit = '1'*). The variation is bigger than one eighth of *ENC_IN_RES*.
			18	*reserved*
			19	*CL_FIT_F:* Active if *ENC_POS_DEV* < *CL_TOLERANCE*. The current mismatch between *XACTUAL* and *ENC_POS* is within tolerated range.
			23:20	*SERIAL_ENC_FLAGS:* Flags received from serial encoder. These flags are reset with a new encoder transfer request.
			24	*SG:* StallGuard2 status (rcvd from TMC26x / TMC21xx/51xx motor driver) or stallGuard status (calculated for TMC24x). *UV_SF:* Undervoltage flag (rcvd from TMC23x / TMC24x motor driver).
			25	*OT*: Overtemperature shutdown (rcvd from any TMC motor driver).
			26	*OTPW*: Overtemperature warning (rcvd from any TMC motor driver).
			27	*S2GA:* Short to ground detection bit for high side MOSFET of coil A (rcvd from TMC26x / TMC21xx/51xx motor driver). *OCA*: Overcurrent bridge A (rcvd from TMC23x / TMC24x motor driver).
			28	*S2GB:* Short to ground detection bit for high side MOSFET of coil B (rcvd from TMC26x / TMC21xx/51xx motor driver). *OCB:* Overcurrent bridge B (rcvd from TMC23x / TMC24x motor driver).
			29	*OLA:* Open load indicator of coil A (rcvd from any TMC motor driver).
			30	*OLB:* Open load indicator of coil B (rcvd from any TMC motor driver).
			31	*STST:* Standstill indicator (rcvd from TMC26x / TMC21xx/51xx motor driver). *OCHS:* Overcurrent high side (rcvd from TMC23x / TMC24x motor driver).

19.15 Various Configuration Registers

VARIOUS CONFIGURATION REGISTERS: CLOSED LOOP, SWITCHES...					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x10	*STP_LENGTH_ADD (0x0000)*	U	15:0	Additional length [# clock cycles] for active step polarity to indicate an active output step at STPOUT
		DIR_SETUP_TIME (0x0000)		31:16	Delay [# clock cycles] between DIROUT and STPOUT voltage level changes.
	0x11	*START_OUT_ADD (0x00000000)*	U	31:0	Additional length [# clock cycles] for active start signal. Active start signal length = 1+START_OUT_ADD
	0x12	*GEAR_RATIO (0x01000000)*	S	31:0	Constant value which will be added to internal position counter with every active step at STPIN Value representation: 8 digits and 24 decimal places.
	0x13	START_DELAY *(0x00000000)*	U	31:0	Delay time [# clock cycles] between start trigger and internal start signal release.
	0x14	*CLK_GATING_DELAY (0x00000000)*	U	31:0	Delay time [# clock cycles] between trigger and initialization of an active clock gating.
	0x15	*STDBY_DELAY (0x00000000)*	U	31:0	Delay time [# clock cycles] after a ramp end before activating the standby phase.
	0x16	*FREEWHEEL_DELAY (0x00000000)*	U	31:0	Delay time [# clock cycles] between initialization of an active standby phase and freewheeling initialization.
	0x17	*VDRV_SCALE_LIMIT (0x00000000)*	U	23:0	Drive scaling limit: *DRV2_SCALE_VAL* is active if *VACTUAL* > *VDRV_SCALE_LIMIT*, else *DRV1_SCALE*
		PWM_VMAX (0x00000000)			PWM: velocity value at which the scaled PWM value reaches the maximum scale parameter 1.
	0x18	*UP_SCALE_DELAY (0x000000)*	U	23:0	Increment delay [# clock cycles]. The value defines the clock cycles which are used to increase the current scale value for one step towards higher values.
		CL_UPSCALE_DELAY (0x000000)			Increment delay [# clock cycles]. The value defines the clock cycles which are used to increase the current scale value for one step towards higher current values during closed loop operation
	0x19	*HOLD_SCALE_DELAY (0x000000)*	U	23:0	Decrement delay [# clock cycles] to decrease the actual scale value by one step towards hold current.
		CL_DOWNSCALE_DELAY (0x000000)			Decrement delay [# clock cycles] to decrease the current scale value by one step towards lower current values during closed loop operation.
	0x1A	*DRV_SCALE_DELAY (0x000000)*	U	23:0	Decrement delay [# clock cycles] to decrease the current scale value by one step towards lower value.
	0x1B	*BOOST_TIME (0x00000000)*	U	31:0	Time [# clk cycles] after a ramp start when boost scaling is active.
	0x1C	*CL_BETA (0x0FF)*		8:0	Maximum commutation angle for closed loop regulation. Set CL_BETA > 255 carefully (esp. if *cl_vlimit_en* = 1). Exactly 255 is recommended for best performance.
		CL_GAMMA (0xFF)		23:16	Maximum balancing angle to compensate back EMF at higher velocities during closed loop regulation.
	0x1D	*DAC_ADDR_A (0x0000)*	U	15:0	Fixed command/address which is sent via SPI output before sending CURRENTA_SPI values.
		DAC_ADDR_B (0x0000)		31:16	Fixed command/address which is sent via SPI output before sending current CURRENTB_SPI values.
		SPI_SWITCH_VEL (0x0000)		23:0	Velocity at which automatic cover datagrams could be sent or at which PWM-SPI mode switching (for TMC23x/24x) will be done automatically.
	0x1E	*HOME_SAFETY_MARGIN (0x0000)*	U	15:0	HOME_REF polarity could be invalid within *X_HOME* ± *HOME_SAFETY_MARGIN*.

VARIOUS CONFIGURATION REGISTERS: CLOSED LOOP, SWITCHES…					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
	0x1F	*PWM_FREQ (0x0280)*	U	15:0	Number of clock cycles for one PWM period.
		CHOPSYNC_DIV (0x0280)		11:0	Chopper clock divider the chopper frequency f_{OSC}: $f_{OSC} = f_{CLK}$/CHOPSYNC_DIV with 96≤CHOPSYNC_DIV≤818

19.16 Ramp Generator Registers

RAMP GENERATOR					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x20	*RAMPMODE (0x0)*		2	*Ramp_mode* 1 *Positioning*: *XTARGET* is superior objective for velocity ramp. 0 *Velocity mode*: *VMAX* is superior objective for velocity ramp.
				1:0	*Ramp_type* 00 *Hold mode*: always *VACTUAL* = *VMAX* (rectangle velocity shape). 01 *Trapezoidal ramp*: consideration of ac- and deceleration values for generating *VACTUAL*, but no adaption of these values. 10 *S-shaped ramp*: consideration of all ramp values (incl. bow values) for generating *VACTUAL*.
RW	0x21	*XACTUAL (0x00000000)*	S	31:0	Current internal motor position [pulses]: $-2^{31} \le XACTUAL \le 2^{31} - 1$
R	0x22	*VACTUAL (0x00000000)*	S	31:0	Current ramp generator velocity [pulses per second]: 1 pps ≤ |*VACTUAL*| ≤ *CLK_FREQ* · ½ pulses (f_{CLK} = 16 MHz → 8 Mpps)
R	0x23	*AACTUAL (0x00000000)*	S	31:0	Current acceleration/deceleration value [pulses per sec^2]: 1 pps^2 ≤ |*AACTUAL*| and -2^{31} pps^2 ≤ *AACTUAL* ≤ 2^{31} - 1
RW	0x24	*VMAX (0x00000000)*	S	31:0	Maximum ramp generator velocity in *positioning mode*. Target ramp generator velocity in *velocity mode* and *hold mode*. Value representation: 23 digits and 8 decimal places. 4 mpps ≤ |*VMAX* [pps]| ≤ *CLK_FREQ* · ½ pulses
RW	0x25	*VSTART (0x00000000)*	U	30:0	Absolute start velocity in *positioning mode* and *velocity mode* Value representation: 23 digits and 8 decimal places. *Positioning mode*: If *VACTUAL* = 0 and *XTARGET* ≠ *XACTUAL*: no acceleration phase for *VACTUAL* = 0→*VSTART*. *Velocity mode*: If *VACTUAL* = 0 and *VACTUAL* ≠ *VMAX*: no acceleration phase for *VACTUAL* = 0→*VSTART*. After switching off Direct_SDin_mode: |*VACTUAL*| = *VSTART* if direct SDin mode is switched off & *automatic_direct_sdin_switch_off* = 1. The direction is dependent on the last STPIN/DIRIN configuration and the defined SD input parameter settings.
RW	0x26	*VSTOP (0x00000000)*	U	30:0	Absolute stop velocity in *positioning mode* and in *velocity mode*. Value representation: 23 digits and 8 decimal places. *Positioning mode*: If *VACTUAL* ≤ *VSTOP* and *XTARGET* = *XACTUAL*: *VACTUAL* will be set to 0 immediately. *Velocity mode*: If *VACTUAL* ≤ *VSTOP* and *VMAX* = 0: *VACTUAL* will be set to 0 immediately. If *VSTOP* ≠ 0 → no last bow phase B_4 for S-shaped ramps If *VSTOP* is very small and *positioning mode* is used → eventually long period at ramp end with *VACTUAL* = *VSTOP* to reach *XTARGET*.
RW	0x27	*VBREAK (0x00000000)*	U	30:0	Absolute break velocity in *positioning mode* and in *velocity mode*, but only for trapezoidal ramp types. Value representation: 23 digits and 8 decimal places. If |*VACTUAL*| < *VBREAK* → |*AACTUAL*| = *ASTART* / *DFINAL* If |*VACTUAL*| ≥ *VBREAK* → |*AACTUAL*| = *AMAX* / *DMAX* If *VBREAK* = 0 → pure linear ramps (*AMAX* / *DMAX* only). **Set always VBREAK > VSTOP!**

RAMP GENERATOR					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x28	*AMAX (0x000000)*	U	23:0	S-shaped ramp types: Maximum acceleration value. Trapezoidal ramps types: Acceleration value if |*VACTUAL*| ≥ *VBREAK* or if *VBREAK* = 0. *frequency mode*: [pulses per sec²] Value representation: 22 digits and 2 decimal places 250 mpps² ≤ *AMAX* ≤ 4 Mpps² *direct mode:* Δv per clock cycle: a[Δv per clock cycle]= *AMAX* / 2^{37} *AMAX* [pps²] = *AMAX* / 2^{37} · f_{CLK}^2 (≤ 31.25 Gpps² at f_{CLK} = 16 MHz)
RW	0x29	*DMAX (0x000000)*	U	23:0	S-shaped ramp types: Maximum deceleration value. Trapezoidal ramps types: Deceleration value if |*VACTUAL*| ≥ *VBREAK* or if *VBREAK* = 0. *frequency mode*: [pulses per sec²] Value representation: 22 digits and 2 decimal places 250 mpps² ≤ *DMAX* ≤ 4 Mpps² *direct mode:* Δv per clk cycle: a[Δv per clk_cycle]= *DMAX* / 2^{37} *DMAX* [pps²] = *DMAX* / 2^{37} · f_{CLK}^2 (≤ 31.25 Gpps² at f_{CLK} = 16 MHz)
RW	0x2A	*ASTART (0x000000)*	U	23:0	S-shaped ramp types: Start acceleration value. Trapezoidal ramps types: Acceleration value if |*VACTUAL*| < *VBREAK* *frequency mode*: [pulses per sec²] Value representation: 22 digits and 2 decimal places 250 mpps² ≤ *ASTART* ≤ 4 Mpps² *direct mode:* Δv per clk cycle: a[Δv per clk_cycle]= *ASTART* / 2^{37} *ASTART* [pps²] = *ASTART* / 2^{37} · f_{CLK}^2 (≤ 31.25 Gpps² at f_{CLK} = 16 MHz) After switching off Direct SDin mode: |*AACTUAL*| = *ASTART* if direct SDin mode is switched off & *automatic_direct_sdin_switch_off* = 1. The direction is dependent on the astart_sign_bit.
		a_sign_bit (0)		31	Sign of *AACTUAL* after switching off direct SDinput mode.
RW	0x2B	*DFINAL (0x000000)*	U	23:0	S-shaped ramp types: Stop deceleration value. (only velocity mode) Trapezoidal ramps types: Deceleration value if |*VACTUAL*| < *VBREAK* *frequency mode*: [pulses per sec²] Value representation: 22 digits and 2 decimal places 250 mpps² ≤ *DFINAL* ≤ 4 Mpps² *direct mode:* Δv per clk cycle: a[Δv per clk_cycle]= *DFINAL* / 2^{37} *DFINAL* [pps²] = *DFINAL* / 2^{37} · f_{CLK}^2 (≤ 31.25 Gpps² at f_{CLK} = 16 MHz)
RW	0x2C	*DSTOP (0x000000)*	U	23:0	Deceleration value for an automatic linear stop ramp to *VACTUAL* = 0. *DSTOP* will be used with activated external stop switches (STOPL or STOPR) if *soft_stop_enable* is set to 1 or with activated virtual stop switches and *virt_stop_mode* is set to b'10.
RW	0x2D	*BOW1 (0x000000)*	U	23:0	Bow value 1 (first bow B_1 of the acceleration ramp) *frequency mode*: [pulses per sec³] Value representation: 24 digits and 0 decimal places 1 mpps³ ≤ *BOW1* ≤ 16 Mpps³ *direct mode:* Δa per clk cycle: bow[Δa per clk_cycle] = *BOW1* / 2^{53} *BOW1* [pps³] = *BOW1* / 2^{53} · f_{CLK}^3 (≤ 7.63 Tpps³ at f_{CLK} = 16 MHz)
RW	0x2E	*BOW2 (0x000000)*	U	23:0	Bow value 2 (second bow B_2 of the acceleration ramp) Vide BOW1 for value representation and conversion calculations.
RW	0x2F	*BOW3 (0x000000)*	U	23:0	Bow value 3 (first bow B_3 of the deceleration ramp) Vide BOW1 for value representation and conversion calculations.
RW	0x30	*BOW4 (0x000000)*	U	23:0	Bow value 4 (second bow B_4 of the deceleration ramp) Vide BOW1 for value representation and conversion calculations.

RAMP GENERATOR					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x31	*CLK_FREQ (0x0F42400)*	U	24:0	External clock frequency value f_{CLK} [Hz] with 4.2 MHz ≤ f_{CLK} ≤ 30 MHz

19.17 Target and Compare Registers

TARGET AND COMPARE REGISTERS					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x32	*POS_COMP (0x00000000)*	S	31:0	Compare position.
RW	0x33	*VIRT_STOP_LEFT (0x00000000)*	S	31:0	Virtual left stop.
RW	0x34	*VIRT_STOP_RIGHT (0x00000000)*	S	31:0	Virtual right stop.
RW	0x35	*X_HOME (0x00000000)*	S	31:0	Current home position.
R	0x36	*X_LATCH (0x00000000)*	S	31:0	Storage position for certain triggers (*circular_cnt_as_xlatch = 0)*
		REV_CNT (0x00000000)			Number of revolution during circular mode (*circular_cnt_as_xlatch = 1)*
W		*X_RANGE (0x00000000)*	U	30:0	Circular movement: Limitation for X_ACTUAL *-X_RANGE ≤ XACTUAL ≤ X_RANGE - 1*
RW	0x37	*X_TARGET (0x00000000)*	S	31:0	Target motor position in positioning mode. *Set all other motion profile parameters before!*

19.18 Pipeline Registers

PIPELINE REGISTERS					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x38	*X_PIPE0 (0x00000000)*	S	31:0	1st pipeline register
RW	0x39	*X_PIPE1 (0x00000000)*	S	31:0	2nd pipeline register
RW	0x3A	*X_PIPE2 (0x00000000)*	S	31:0	3rd pipeline register
RW	0x3B	*X_PIPE3 (0x00000000)*	S	31:0	4th pipeline register
RW	0x3C	*X_PIPE4 (0x00000000)*	S	31:0	5th pipeline register
RW	0x3D	*X_PIPE5 (0x00000000)*	S	31:0	6th pipeline register
RW	0x3E	*X_PIPE6 (0x00000000)*	S	31:0	7th pipeline register
RW	0x3F	*X_PIPE7 (0x00000000)*	S	31:0	8th pipeline register

19.18.1 Possible pipeline partitioning

Coming soon.

19.19 Shadow Registers

TARGET AND COMPARE REGISTERS						
R/W	**Addr**	***Reg name***	***(default)***	**S/U**	**Bit**	**Description**
RW	0x40	*SH_REG0*	*(0x00000000)*	S	31:0	1st shadow register
RW	0x41	*SH_REG1*	*(0x00000000)*	U	31:0	2nd shadow register
RW	0x42	*SH_REG2*	*(0x00000000)*	U	31:0	3rd shadow register
RW	0x43	*SH_REG3*	*(0x00000000)*	U	31:0	4th shadow register
RW	0x44	*SH_REG4*	*(0x00000000)*	U	31:0	5th shadow register
RW	0x45	*SH_REG5*	*(0x00000000)*	U	31:0	6th shadow register
RW	0x46	*SH_REG6*	*(0x00000000)*	U	31:0	7th shadow register
RW	0x47	*SH_REG7*	*(0x00000000)*	S/U	31:0	8th shadow register
RW	0x48	*SH_REG8*	*(0x00000000)*	U	31:0	9th shadow register
RW	0x49	*SH_REG9*	*(0x00000000)*	U	31:0	10th shadow register
RW	0x4A	*SH_REG10*	*(0x00000000)*	U	31:0	11th shadow register
RW	0x4B	*SH_REG11*	*(0x00000000)*	U	31:0	12th shadow register
RW	0x4C	*SH_REG12*	*(0x00000000)*	U	31:0	13th shadow register
RW	0x4D	*SH_REG13*	*(0x00000000)*	U	31:0	14th shadow register

19.19.1 Possible shadow register partitioning

Coming soon.

19.20 Freeze Register

The whole freeze register can only be written once after an active reset and before motion starts.

It is always readable.

FREEZE					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x4E	*DFREEZE (0x000000)*	U	23:0	Deceleration value. If *NFREEZE* switches to low the parameter is used for an automatic linear ramp stop. Setting DFREEZE to 0 leads to an hard stop. Value representation: [Δv per clk_cycle] a[Δv per clk_cycle]= *DFREEZE* / 2^{37} *DFREEZE* [pps^2] = *DFREEZE* / 2^{37} · f_{CLK}^2 (≤ 31.25 Gpps^2 at f_{CLK} = 16 MHz)
		IFREEZE (0x00)	U	31:24	Scaling value if NFREEZE is tied low. If IFREEZE=0, current active scaling value will be valid at *FROZEN* event.

19.21 Clock Gating Enable Register

CLOCK GATING				
R/W	**Addr**	***Reg name (default)***	**Bit**	**Description**
RW	0x4F	*CLK_GATING_REG (0x0)*	2:0	Setting all bits to 1 and *VACTUAL* = 0 initializes the clock gating countdown. If sleep state is active, this register is set to 0 and also the internal clock remains at low level (sleep state). Enable clock gating by setting *clk_gating_en = '1'*

19.22 Encoder Registers

ENCODER REGISTERS					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x50	*ENC_POS (0x00000000)*	S	31:0	Current encoder position [µsteps].
R	0x51	*ENC_LATCH (0x00000000)*	S	31:0	Latched encoder position.
W		*ENC_RESET_VAL (0x00000000)*			Defined reset value for *ENC_POS* if the encoder position should be cleared to another value than 0 and an N event is recognized.
R	0x52	*ENC_POS_DEV (0x00000000)*	S	31:0	Current deviation between *XACTUAL* and *ENC_POS*.
W		*CL_TR_TOLERANCE (0x00000000)*	U		Tolerated absolute tolerance between *XACTUAL* and *ENC_POS* to trigger TARGET_REACHED_F and event.
W	0x53	*ENC_POS_DEV_TOL (0xFFFFFFFF)*	U	31:0	Tolerated value of |(*X_ACTUAL* – *ENC_POS*)|.
W	0x54	*ENC_IN_RES (0x00000000)*	U	30:0	Resolution [encoder steps per revolution] of the encoder connected to the encoder inputs.
R		*ENC_CONST (0x00000000)*			Encoder constant. Value representation: 15 digits and 16 decimal places
W		*manual_enc_const (0)*		31	0 *ENC_CONST* will be calculated automatically. 1 manual definition of *ENC_CONST* = *ENC_IN_RES*
W	0x55	*ENC_OUT_RES (0x00000000)*	U	31:0	Resolution [encoder steps per revolution] of the serial encoder output interface.
W	0x56	*SER_CLK_IN_HIGH (0x00A0)*	U	15:0	High voltage level time of serial clock output [# clock cycles]
		SER_CLK_IN_LOW (0x00A0)		31:16	High voltage level time of serial clock output [# clock cycles]
W	0x57	*SSI_IN_CLK_DELAY (0x0000)*	U	15:0	SSI encoder: Delay time [# clock cycles] between next data transfer after a rising edge of serial clock output (if set to 0 → *SSI_IN_CLK_DELAY* = *SER_CLK_IN_HIGH)* SPI encoder: Delay [# clock cycles] at start and end of data transfer between serial clock output & negated chip select. (if set to 0 → *SSI_IN_CLK_DELAY* = *SER_CLK_IN_HIGH)*
		BISS_TIMEOUT (0x0000)			BiSS encoder: *BiSS* timeout parameter at the end of a BiSS data transfer.
		SSI_IN_WTIME (0x0F0)		25:16	Delay parameter tw [# clock cycles] between two clock sequences for a multiple data transfer (of the same data). SSI recommendation: tw < 19 µs.
		BISS_IN_BUSYS (0x0F0)			Maximum evaluation time [# clock cycles] of slave device during sensor modus.
W	0x58	*SER_PTIME (0x00190)*	U	19:0	SSI and SPI encoder: Delay time period tp [# clock cycles] between two consecutive clock sequences for new data request. SSI recommendation: tp > 21 µs.
		BISS_IN_BUSYR (0x00190)			Maximum evaluation time [# clock cycles] of the slave device during register modus.
		CRC_GEN_POLYNOM (0x00)		31:24	CRC generator polynomial.

19.24 PID and Closed Loop Registers

PID / CLOSED LOOP					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
RW	0x59	*CL_OFFSET (0x00000000)*	S	31:0	Offset between *ENC_POS* and *XACTUAL* during closed loop calibration.
W	0x5A	*PID_P (0x000000)*	U	23:0	Proportional term of PID regulator = PID_P / 256 · PID_E PID mode: Parameter P of PID regulator for direct velocity control (Enable PID regulation; *regulation_modus*(1) = '1') ClosedLoop mode: Parameter P of PI regulator which controls maximum velocity during closed loop regulation. (Set cl_vlimit_en = '1' if velocity limit during closed loop mode - *regulation_modus* = "01" - should be used)
		CL_VMAX_CALC_P (0x000000)			
R		*PID_VEL (0x00000000)*	S	31:0	Current PID output velocity.
W	0x5B	*PID_I (0x000000)*	U	23:0	Integral term = PID_I / 256 · PID_ISUM = PID_I / 256 · PID_E · f_{CLK}/128 PID mode: Parameter I of PID regulator for direct velocity control (Enable PID regulation; *regulation_modus*(1) = '1') ClosedLoop mode: Parameter I of PI regulator which controls maximum velocity during closed loop regulation. (Set cl_vlimit_en = '1' if velocity limit during closed loop mode - *regulation_modus* = "01" - should be used)
		CL_VMAX_CALC_I (0x000000)			
R		PID_ISUM_RD *(0x00000000)*	S	31:0	Current PID integrator sum. PID_ISUM = PID_E · f_{CLK}/128 Update frequency = f_{CLK}/128
W	0x5C	*PID_D (0x000000)*	U	23:0	PID mode: Parameter D of PID regulator. *PID_E* is sampled with f_{CLK} / 128 / *PID_D_CLKDIV*. Derivative term = (PID_E_{LAST} – PID_E_{ACTUAL}) · PID_D (Enable PID regulation; *regulation_modus*(1) = '1')
		CL_DELTA_P (0x000000)			ClosedLoop mode: Gain parameter which is multiplied with the current position difference to calculate the current commutation angle for higher stiffness for position maintenance. Clipped at *CL_BETA*. Value representation: 8 digits and 16 decimal places Real value = CL_DELTA_P / 2^{16} Example: 65536 = factor of 1 (no gain)
W	0x5D	*PID_I_CLIP (0x0000)*	U	14:0	PID and PI velocity regulator for ClosedLoop mode: Clipping parameter for *PID_ISUM*. Real value = PID_ISUM · 2^{16} · PID_ICLIP (Enable PID regulation; *regulation_modus*(1) = '1')
		PID_D_CLKDIV (0x00)		23:16	PID and PI velocity regulator for ClosedLoop mode: Clock divider for D part calculation. (Enable PID regulation; *regulation_modus*(1) = '1')
R		*PID_E (0x00000000)*	S	31:0	Current position deviation.
W	0x5E	*PID_DV_CLIP (0x00000000)*	U	30:0	PID and PI velocity regulator for ClosedLoop mode: Clipping parameter for *PID_VEL*. (Set cl_vlimit_en = '1' if velocity limit during closed loop mode - *regulation_modus* = "01" - should be used or enable PID regulation; *regulation_modus*(1) = '1')
W	0x5F	*PID_TOLERANCE (0x00000)*	U	19:0	PID mode: Tolerated position deviation: *PID_E* = 0 if |*PID_E*| < *PID_TOLERANCE*
		CL_TOLERANCE (0x00)		7:0	ClosedLoop mode: Tolerated position deviation: *CL_DELTA_P* = 65536 (Gain=1) if |*ENC_POS_DEV*| < *CL_TOLERANCE*

19.25 Misc Registers

Closed loop operation is internally processed using a 256 microstep resolution. It is possible to use any encoder resolution since it is scaled to 256 microsteps.

Closed loop it is not possible with a Step/Dir input resolution of less than 256 microsteps!

MISC					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
W	0x60	*FS_VEL (0x000000)*	U	23:0	Minimum fullstep velocity [pps]. If |*VACTUAL*| > *FS_VEL* fullstep operation is possible.
		DC_VEL (0x000000)			Minimum dcStep velocity [pps]. If |*VACTUAL*| > *FS_VEL* dcStep will be enabled
		CL_VMIN_EMF (0x000000)			Encoder velocity at which back EMF consideration starts during closed loop operation.
W	0x61	*DC_TIME (0x00)*	U	7:0	Upper PWM on-time limit for commutation (only TMC26x) Set slightly above effective blank time TBL of the driver.
		DC_SG (0x0000)		15:8	Maximum PWM on-time [# clock cycles · 16] for step loss detection. If a loss is detected (step length of first regular step after blank time of the dcStep input signal is below DC_SG), a stall event will be released. (only valid for dcStep with TMC26x)
		DC_BLKTIME (0x0000)		31:16	Blank time [# clock cycles] after fullstep release when no signal comparison should happen (only valid for dcStep with TMC26x)
		CL_VADD_EMF (0x000000)	U	23:0	Additional velocity value to calculate *V_MAX_CL_EMF* which is the encoder velocity where back EMF consideration reaches the maximum angle *CL_GAMMA* during closed loop operation.
W	0x62	*DC_LSPTM (0x00FFFFFF)*	U	31:0	dcStep low speed timer [# clock cycles]
		ENC_VEL_ZERO (0xFFFFFF)		23:0	Delay time [# clock cycles] after the last incremental encoder change to set *V_ENC_MEAN* = 0.
W	0x63	*ENC_VMEAN_WAIT (0x00)*	U	7:0	Delay period [# clock cycles] before the next current encoder velocity value becomes considered for mean encoder velocity calculation. **Set ENC_VMEAN_WAIT > 32 (Incremental encoder only)** Will be set automatically to *SER_PTIME* for SSI/SPI encoder or to *BISS_IN_BUSYS* for BiSS encoder
		SER_ENC_VARIATION (0x00)		7:0	Multiplier for maximum permitted serial encoder variation between consecutive absolute encoder requests. **(Absolute encoder only)**: Maximum permitted value = *ENC_VARIATION* / 256 · 1/8 · *ENC_IN_RES* (If *ENC_VARIATION* = 0 → Maximum permitted value = 1/8 · *ENC_IN_RES*)
		ENC_VMEAN_FILTER (0x0)		11:8	Filter exponent to calculate mean encoder velocity.
		ENC_VMEAN_INT (0x0000)		31:16	Encoder velocity update time [# clock cycles] (Incremental encoder only). Minimum value is set automatically to 256.
		CL_CYCLE (0x0000)		31:16	CL control cycle [# clock cycles] (Absolute encoder only). Will be set automatically to *fastest* possible cycle for ABN encoders
W	0x64	*SYNCHRO_SET (0x00)*	-	6:0	Set of switches for synchronization purposes (see next page for further information)
R	0x65	*V_ENC (0x00000000)*	S	31:0	Current encoder velocity [pps].
R	0x66	*V_ENC_MEAN (0x00000000)*	S	31:0	Current filtered encoder velocity [pps].

MISC					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
W	0x67	*VSTALL_LIMIT (0x00000000)*	U	23:0	Stop on stall velocity limit [pps]: Only above this limit an active stall leads to a stop on stall if enabled.
W	0x7C	*CIRCULAR_DEC (0x000)*	U	31:0	Decimal places (bit31=1digit) for circular movement if one revolution is not exactly mapped to an even number of µSteps per revolution
W	0x7D	*ENC_COMP_XOFFSET (0x0000)*	U	15:0	Start offset for triangular compensation in horizontal direction (as number between 0 and 1).
		ENC_COMP_YOFFSET (0x00)	S	23:16	Start offset for triangular compensation in vertical direction. |ENC_COMP_YOFFSET| ≤ 127
		ENC_COMP_AMPL (0x00)	U	31:24	Maximum amplitude for encoder compensation

19.25.1 Synchronization Configuration

REFERENCE SWITCH CONFIGURATION (0X64)				
R/W	**Addr**	***Reg name***	**Bit**	**Description**
W	0x64	*SYNCHRO_SET* *Default: 0x00*	0	*consider_mp1* 0 MP1 input pin will be not considered for synchronization purposes. 1 MP1 input pin will be considered for synchronization purposes.
			1	*consider_mp2* 0 MP2 inout pin will be not considered for synchronization purposes. 1 MP2 inout pin will be considered for synchronization purposes.
			2	*cl_fit_for_mp2* 0 CL_FIT_Flag will not be forwarded via MP2 inout pin 1 CL_FIT_Flag will be forwarded via MP2 inout pin.
			3	*mp2_as_wired_and* 0 MP2 output function could not be used as wired-and. 1 MP2 output function could be used as wired-and.
			4	*mp2_as_wired_or* 0 MP2 output function could not be used as wired-or. 1 MP2 output function could be used as wired-or.
			5	*rampgen_block_en* 0 Internal ramp generator will not be blocked if incoming synchronization signal is interpreted as blocked. 1 Internal ramp generator will be blocked if incoming synchronization signal is interpreted as blocked.
			6	*rampgen_block_en* 0 External step control will not be blocked if incoming synchronization signal is interpreted as blocked. 1 External step control will be blocked if incoming synchronization signal is interpreted as blocked.

19.26 Transfer Registers

TRANSFER					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
W	0x68	ADDR_TO_ENC *(0x00000000)*	-	31:0	SPI encoder only: Address data permanently sent to get encoder angle data from the SPI encoder slave device. BiSS and SPI encoder: Address data sent from TMC4361 to the encoder for one-time data transfer.
W	0x69	*DATA_TO_ENC (0x00000000)*	-	31:0	SPI / BiSS configuration data sent from TMC4361 to the serial encoder for one-time data transfer.
R	0x6A	*ADDR_FROM_ENC (0x00000000)*	-	31:0	SPI encoder only: Repeated request data is stored here. BiSS and SPI encoder: SPI / BiSS address data received from the serial encoder as response of the one-time data transfer.
R	0x6B	*DATA_FROM_ENC (0x00000000)*	-	31:0	SPI / BiSS data received from the serial encoder as response of the one-time data transfer.
W	0x6C	*COVER_LOW (0x00000000)*	-	31:0	Lower configuration bits of SPI orders which should be sent from TMC4361 to the motor drivers via SPI output. If automatic cover transfer is enabled (automatic_cover = 1 and resulting *COVER_DATA_LENGTH* ≤ 32), *COVER_LOW* will be sent if |*VACTUAL*| crosses *SPI_SWITCH_VEL* **downwards**.
W	0x6D	*COVER_HIGH (0x00000000)*	-	31:0	Upper configuration bits of SPI orders which should be sent from TMC4361 to the motor drivers via SPI output. If automatic cover transfer is enabled (automatic_cover = 1 and resulting *COVER_DATA_LENGTH* ≤ 32), *COVER_HIGH* will be sent if |*VACTUAL*| crosses *SPI_SWITCH_VEL* **upwards**.
R	0x6E	*COVER_DRV_LOW (0x00000000)*	-	31:0	Lower configuration bits of SPI response received from the motor driver connected to the SPI output.
R	0x6F	*COVER_DRV_HIGH (0x00000000)*	-	31:0	Upper configuration bits of SPI response received from the motor driver connected to the SPI output.

19.27 SinLUT Registers

SINLUT					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
W	0x70	*MSLUT[0] (0xAAAAB554)*	-	31:0	Each bit defines the difference between consecutive values in the microstep look-up table MSLUT (in combination with MSLUTSEL).
	0x71	*MSLUT[1] (0x4A9554AA)*			
	0x72	*MSLUT[2] (0x24492929)*			
	0x73	*MSLUT[3] (0x10104222)*			
	0x74	*MSLUT[4] (0xFBFFFFFF)*			
	0x75	*MSLUT[5] (0xB5BB777D)*			
	0x76	*MSLUT[6] (0x49295556)*			
	0x77	*MSLUT[7] (0x00404222)*			
W	0x78	*MSLUTSEL (0x FFFF8056)*	-	31:0	Definition of the four segments within each quarter MSLUT wave.
R	0x79	*MSCNT (0x000)*	U	9:0	Current µStep position of the sine value.
W		*MSOFFSET (0x000)*		9:0	Microstep offset for PWM mode (TMC23x/24x only)
R	0x7A	*CURRENTA (0x000)*	S	8:0	Actual current value of coilA (sine values).
		CURRENTB (0x000)	S	24:16	Actual current value of coilB (sine90_120 values).
R	0x7B	*CURRENTA_SPI (0x000)*	S	8:0	Actual current value of coilA (sine values).
		CURRENTB_SPI (0x000)	S	24:16	Actual current value of coilB (sine90_120 values).
R	0x7C	*SCALE_PARAM (0x000)*	U	8:0	Actual used scale parameter.
W	0x7E	*START_SIN (0x00)*	U	7:0	Start value for sine waveform
		START_SIN90_120 (0xF7)	U	23:16	Start value for cosine waveform
		DAC_OFFSET (0x00)	U	31:24	Offset for absolute sine and cosine values which will be forwarded via SPI output as DAC output values.

19.28 Version Registers

VERSION					
R/W	**Addr**	***Reg name (default)***	**S/U**	**Bit**	**Description**
R	0x7F	*Version No (0x0001)*	U	15:0	Current TMC4361 version number

20 Absolute Maximum Ratings

The maximum ratings may not be exceeded under any circumstances. Operating the circuit at or near more than one maximum rating at a time for extended periods shall be avoided by application design.

Parameter (VCC = 3.3V nominal → TEST_MODE = 0V)	Symbol	Min	Max	Unit
Supply voltage	V_{CC}	3.0	3.6	V
Input voltage IO	V_{IN}	-0.3	3.6	V

Parameter (VCC = 5V nominal → TEST_MODE = 1.8V)	Symbol	Min	Max	Unit
Supply voltage	V_{CC}	4.8	5.2	V
Input voltage IO	V_{IN}	-0.3	5.2	V

21 Electrical Characteristics

21.1 DC Characteristics Operating Conditions

DC characteristics contain the spread of values guaranteed within the specified supply voltage range unless otherwise specified. Typical values represent the average value of all parts measured at +25°C. Temperature variation also causes stray to some values. A device with typical values will not leave Min/Max range within the full temperature range.

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Extended temperature range	T_{COM}		-40°C		125	°C
Nominal core voltage	V_{DD}			1.8		V
Nominal IO voltage	V_{DD}			3.3 / 5.0		V
Nominal input voltage	V_{IN}		0.0		3.3 / 5.0	V
Input voltage low level	V_{INL}	V_{DD} = 3.3V / 5V	-0.3		0.8 / 1.2	V
Input voltage high level	V_{INH}	V_{DD} = 3.3V / 5V	2.3 / 3.5		3.6 / 5.2	V
Input with pull-down		V_{IN} = V_{DD}	5	30	110	µA
Input with pull-up		V_{IN} = 0V	-110	-30	-5	µA
Input low current		V_{IN} = 0V	-10		10	µA
Input high current		V_{IN} = V_{DD}	-10		10	µA
Output voltage low level	V_{OUTL}	V_{DD} = 3.3V / 5V			0.4	V
Output voltage high level	V_{OUTH}	V_{DD} = 3.3V / 5V	2.64 / 4.0			V
Output driver strength	I_{OUT_DRV}	V_{DD} = 3.3V / 5V		4.0		mA

21.2 Power Dissipation

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Static power dissipation	PD_{STAT}	All inputs at VDD or GND V_{DD} = 3.3V / 5V			1.1 / 1.7	mW
Dynamic power dissipation	PD_{DYN}	All inputs at VDD or GND f_{CLK} variable V_{DD} = 3.3V / 5V			2.3 / 3.7	mW / MHz
Total power dissipation	PD	f_{CLK} = 16 MHz V_{DD} = 3.3V / 5V			37.9 / 60.3	mW

21.3 General IO Timing Parameters

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operation frequency	f_{CLK}	$f_{CLK} = 1 / t_{CLK}$	4.2*)	16	30	MHz
Clock Period	t_{CLK}	Rising edge to rising edge	33.5	62.5		ns
Clock time low			16.5			ns
Clock time high			16.5			ns
CLK input signal rise time	t_{RISE_IN}	20 % to 80 %			20	ns
CLK input signal fall time	t_{FALL_IN}	80 % to 20 %			20	ns
Output signal rise time	t_{RISE_OUT}	20 % to 80 % load 32 pF		3.5		ns
Output signal fall time	t_{FALL_OUT}	80 % to 20 % load 32 pF		3.5		ns
Setup time for SPI input signals in synchronous design	t_{SU}	Relative to rising clk edge	5			ns
Hold time	t_{HD}	Relative to rising clk edge	5			ns

*) The lower limit for f_{CLK} refers to the limits of the internal unit conversion to physical units. The chip will also operate at lower frequencies.

22 Modifications as regards TMC4361 (old version)

1. Limitations as regards the usage of trapezoidal ramps are obsolete
2. S-Ramps: Erroneous behavior by setting VMAX=0 during positioning ramp is fixed → for security reasons, keep old handling (switch to velocity mode, then set VMAX = 0, then wait for VATUAL=0, then switch to positioning mode again
3. STPIN/DIRIN support with adjustable direct and indirect external control including free adjustable gearing factor (and selectable input filter settings)
4. dcStep support for TMC26x and TMC21xx/51xx
5. Extensive synchronization support:
    a. Automatic switch between slave mode (external control) and internal ramp mode to ease slave/master switches during an active ramp
    b. INTR and TARGET_REACHED pins with selectable PU/PD output functionality for connecting several motion controllers
    c. Masterless start signal synchronization mode selectable
6. Substantial expansion of Pipeline possibilities:
    a. Besides XTARGET, POS_COMP, GEAR_RATIO and GENERAL_CONF selectable for pipeline usage (ease also synchronization)
7. Shadow registers available for pipelining and synchronizing ramp parameters.
8. Circular movement (limited range for XACTUAL: -X_RANGE <= XACTUAL < X_RANGE) with automatic overflow calculation. Extra settings are provided to support also number of µSteps per revolutions which are not even integers (uneven numbers and decimal places possible)
    a. Virtual stop position could be used to define a blocking area in the circle. The path from XACTUAL to XTARGET will not cross this area.
    b. Automatic count of executed revolutions (=REV_CNT)
    c. ABN support of circular movements
9. Velocity mode for Closed Loop operation mode
10. Automatic fullstep switching for TMC21xx/51xx in S/D mode
11. Automatic PWM switching for TMC24x (low velocity: PWM mode, higher: SPI mode)
12. Target_reached generation is now also dependent on position deviation during closed loop operation
13. After starting with VSTART, ASTART is also available for another starting acceleration value than AMAX
14. Motor direction reversible by setting a switch
15. Compare possibilities expanded
    a. XACTUAL/ENC_POS vs POS_COMP/X_LATCH/X_HOME or POS_COMP vs REV_CNT
16. X_HOME now read/write, not only readable
17. ENC_CONST now read/write, not only readable (if automatic ENC_CONST calculation is not feasible)
18. ENC_POS could now set to any defined value if ENC_POS should be cleared at n-event
19. Encoder velocity calculation is now also available for serial encoders
20. Sleep Timer event exchanged with RST event
21. Version register available
22. SPI mode available where only Cover datagrams will be sent, even during motion
23. Velocity dependent SPI cover datagram transfer

23 Layout Example

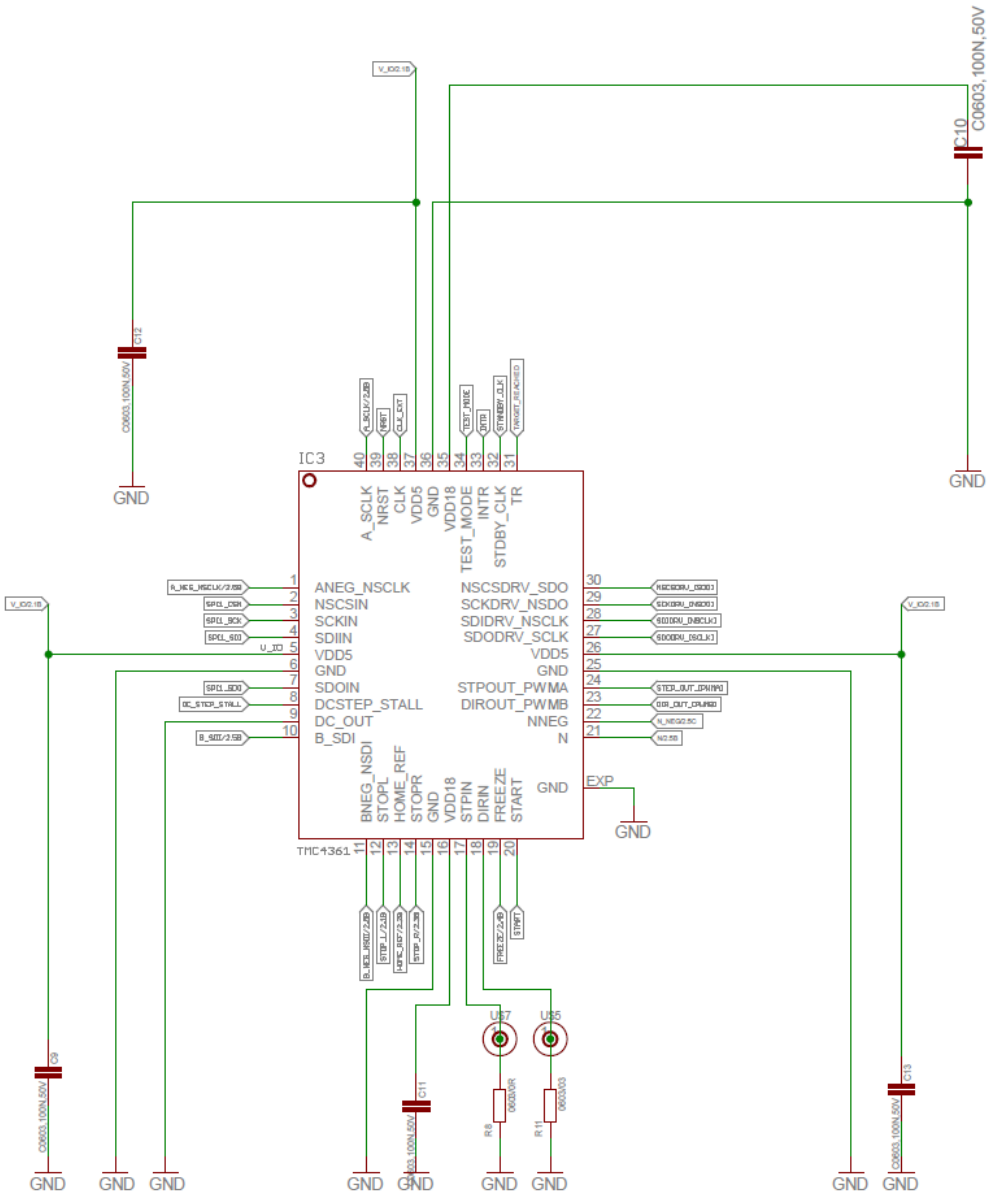


Figure 23.1 Internal circuit diagram for layout example

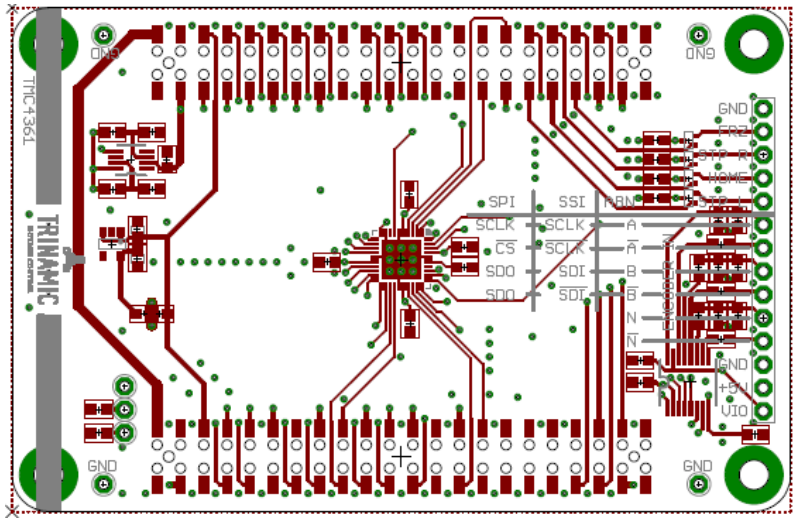


Figure 23.2 Components (assembly for application with encoder)

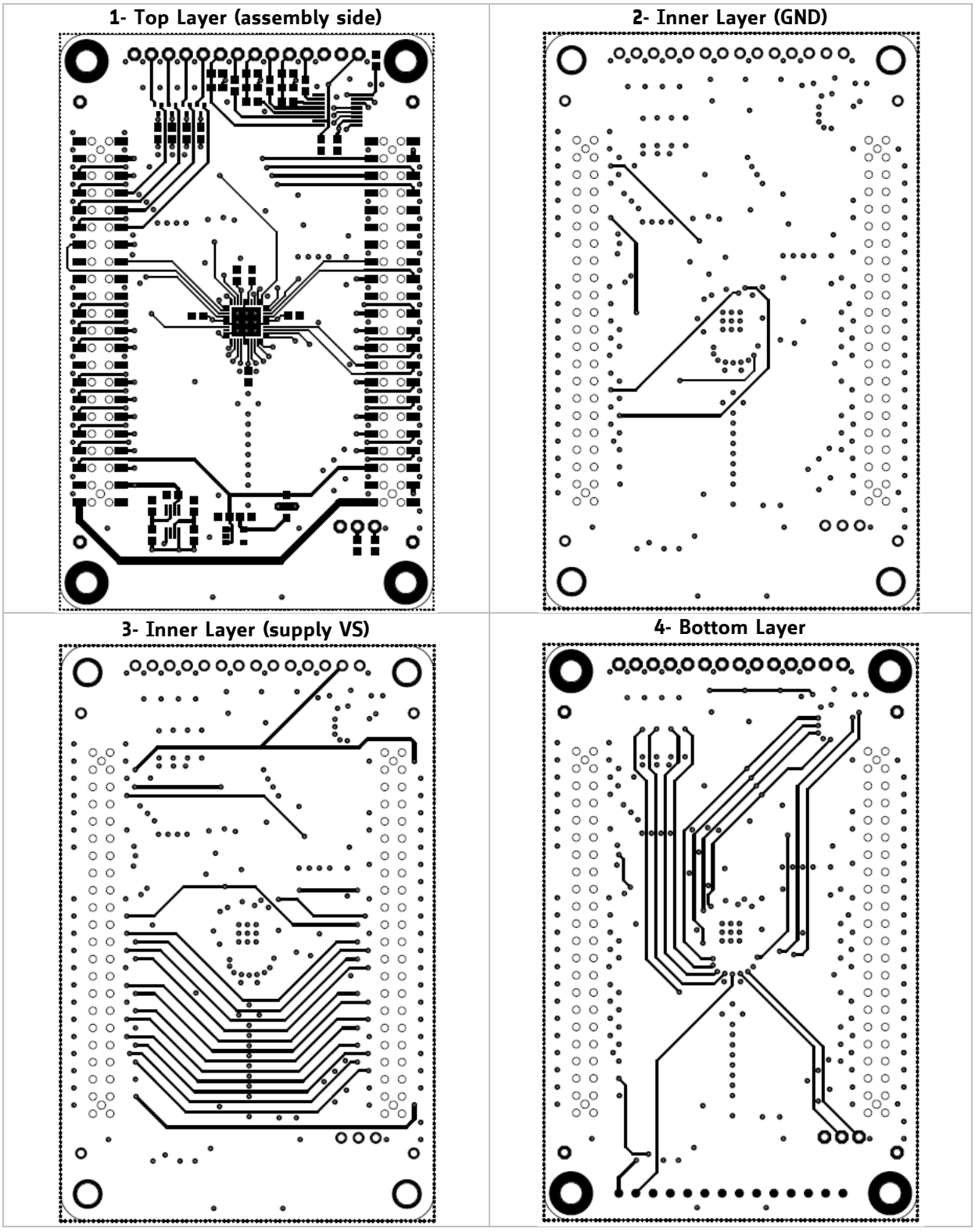
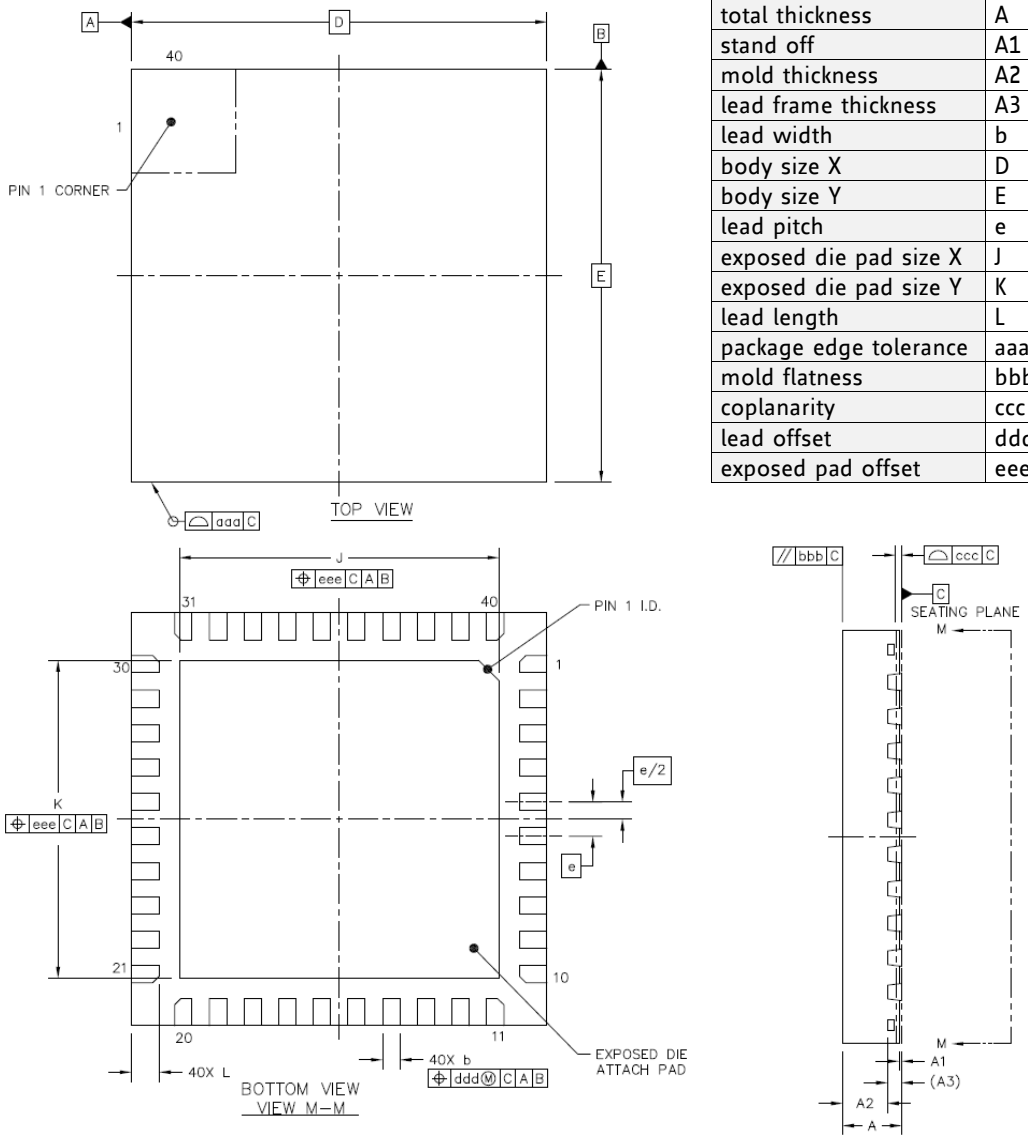


Figure 23.3 Layout example

24 Package Mechanical Data

24.1 Dimensional Drawings

Attention: Drawings not to scale. All values in millimeters.



Parameter	Ref	Min	Nom	Max
total thickness	A	0.8	0.85	0.9
stand off	A1	0	0.035	0.05
mold thickness	A2	-	0.65	0.67
lead frame thickness	A3		0.203 REF	
lead width	b	0.2	0.25	0.3
body size X	D		6 BSC	
body size Y	E		6 BSC	
lead pitch	e		0.5 BSC	
exposed die pad size X	J	4.52	4.62	4.72
exposed die pad size Y	K	4.52	4.62	4.72
lead length	L	0.35	0.4	0.45
package edge tolerance	aaa		0.1	
mold flatness	bbb		0.1	
coplanarity	ccc		0.08	
lead offset	ddd		0.1	
exposed pad offset	eee		0.1	

Figure 24.1 Dimensional drawings

24.2 Package Codes

Type	Package	Temperature range	Code & marking
TMC4361	QFN40 (RoHS)	-40°C ... +125°C	TMC4361-LA

25 Disclaimer

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.

Information given in this data sheet is believed to be accurate and reliable. However no responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties which may result from its use.

Specifications are subject to change without notice.

All trademarks used are property of their respective owners.

26 ESD Sensitive Device

The TMC4361 is an ESD sensitive CMOS device sensitive to electrostatic discharge. Take special care to use adequate grounding of personnel and machines in manual handling. After soldering the devices to the board, ESD requirements are more relaxed. Failure to do so can result in defect or decreased reliability.

27 Table of Figures

28 Revision History

28.1 Document Revisions

Version	Date	Author HS = Hagen Sämrow SK = Stephan Kubisch SD = Sonja Dwersteg	Description
2.52	2014-APR-09	HS	Transfer from TMC4361old manual (v1.04) and first adjustments and explanations of the extended feature set
2.53	2014-AUG-04	HS	Test of new FMS: Power consumption increases slightly → power safety margins after new measurement increased
2.60	2014-Aug-20	HS	Further adaptations till chapter 12.5
2.61	2014-Aug-25	HS	Register overview updated Modifications (as regards TMC4361old) overview generated
2.62	2014-Aug-27	HS	Problems with virt_stop_mode="00" and S-Ramps occurred during bow phase B1 and B12 → **virt_stop_mode="00" prohibited for now** Further investigations in progress Until a workaround is achieved Furthermore, insert Attention for not changing RAMPMODE if v≠0 (esp. changing from velocity to pos mode is crucial)
2.62	2014-Sep-27	HS	Update of 0x63 register double assignments for Incremental ←→Absolute encoders
2.62	2014-Sep-30	HS	Product key newly assigned to TMC4361-LA (former key: TMC4361-LI) to exhibit the extended temperature range
2.65	2014-Oct-02	HS	FREEZED to FROZEN (correct now) changed Limits for On-the-fly changes expressed VMAX=0 Failure added
2.65	2014-Oct-02	HS	VDD5 changed to VCC - Images adapted - Electrical spec adapted - Remarks added for VCC = 5V → TEST_MODE have to be connect to VDD1V8!
2.65	2014-Nov-20	HS	Beautifying, some minor adaptations
2.65	2014-Dec-01	HS	Hint for 220Ohm resistors in series if START is connected to another TMC4361
2.66	2015-Jan-21	HS	Limitations for Shadow ramp transfer during motion added
2.66	2015-Feb-03	HS	Hints for ENC_WAIT and V_ENC_FILTER added
2.67	2015-Apr-01	HS	Register descriptions corrected
2.68	2015-Apr-14	HS	Changed CL_BETA=255 (from 256) for best performance

Table 28.1 Document Revisions