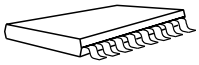
TRINAMIC
MICROCHIPS

# TMC 428 – DATA SHEET

## Intelligent Triple Stepper Motor Controller with Serial Peripheral Interfaces



TRINAMIC® Microchips GmbH
Deelbögenkamp 4C
D – 22297 Hamburg
GERMANY

T +49 - (0) 40 - 51 48 06 - 0
F +49 - (0) 40 - 51 48 06 - 60
www.trinamic.com
info@trinamic.com

## Features

The TMC428 is a miniaturized high performance stepper motor controller. It controls up to three 2-phase stepper motors. All motors can operate independently. The TMC428 allows up to 6 bit micro step resolution– which is up to 64 micro steps per full step –individually selectable for each motor. Once initialized, it performs all real time critical tasks autonomously based on target positions and velocities, which may be altered on-the-fly. So, an inexpensive micro controller together with the TMC428 forms a complete motion control system. The micro controller is free to do application specific interfacing and high level control functions. Both, the communication with the micro controller and with one to three daisy chained stepper motor drivers take place via two separate 4 wire serial peripheral interfaces. The TMC428 directly connects to SPI™* smart power stepper motor drivers.

- Controls up to three 2-phase stepper motors
- Serial 4-wire interface for µC with easy-to-use protocol
- Configurable interface for SPI™ motor drivers
- Different types of SPI™ stepper motor driver chips may by mixed within a single daisy chain
- Communication on demand minimizes traffic to the SPI™ stepper motor drivers chain
- Programmable SPI™ data rates up to 1 Mbit/s
- Wide range for clock frequency – can use CPU clock up to 16 MHz
- Internal 24 bit wide position counters
- Full step frequencies up to 20 kHz
- Read-out facility for actual motion parameters (position, velocity, acceleration) and driver status
- Individual micro step resolution of {64, 32, 16, 8, 4, 2, 1} micro steps via built-in sequencer
- Programmable 6 bit micro step table with up to 64 entries for a quarter sine wave period
- Built-in ramp generators for autonomous positioning and speed control
- On-the-fly alteration of target motion parameters (like position, velocity, acceleration)
- Automatic acceleration dependent current control (power boost)
- Power down mode (100 µA) with transparent wake-up for normal operation (typical 5 mA @ 16 MHz)
- 3.3V or 5V operation with CMOS / TTL compatible IOs (all inputs Schmitt-Trigger)
- Ultra small 16 pin SSOP package (optional 24 pin SOIC24 package)
- Integrated power-on-reset

* *SPI is Trademark of Motorola, Inc.*

Life support policy

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

# General Description

The TMC428 is a miniaturized high performance stepper motor controller with a low price for high volume automotive and industrial motion control applications as well. Once initialized, the TMC428 controls up to three independent 2-phase stepper motors from each other. The low price makes it attractive also for those applications, where only one or two stepper motors have to be controlled simultaneously.

The TMC428 performs all real time critical tasks autonomously. Thus a low cost micro controller is sufficient to perform the tasks of initialization, application specific interfacing, and to specify target positions and velocities. The TMC428 allows on-the-fly alteration of all target parameters. Read-back option for all internal registers simplifies programming. The TMC428 can perform up to $2^{23}$ steps respectively micro steps fully independent from the micro controller with its internal position counters. The step resolution– individually programmable for each stepper motor –ranges from full step, half step, up to 6 bit micro stepping (64 micro steps / full step) for precise positioning and noiseless stepper motor rotation. Optionally, the micro step table– common for all motors –can be adapted to motor characteristics to further enhance smooth, low toque ripple application.

The TMC428 uses serial interfaces for communication with the micro controller and for the stepper motor drivers. The serial interface for the micro controller uses a fixed length of 32 bits with a simple protocol, directly connecting to SPI™ interfaces. The serial interface to the stepper motor drivers is flexible configurable for different types– even from different vendors –with up to 64 bit length for the SPI daisy chain. Our own smart power stepper motor drivers TMC288 and TMC289 perfectly fit to the TMC428. Without additional hardware, drivers with same serial interface polarities of chip select and clock signals may be mixed in a single chain. To mix drivers with different serial interface polarities, additional inverters (e.g. 74HC04, 74HC14) are required. For those driver chips without serial data send back option, the variant of the TMC428 with two additional chip select outputs is proposed. The TMC428 sends data to the driver chain on demand only, which minimizes the interface traffic and reduces the power consumption.

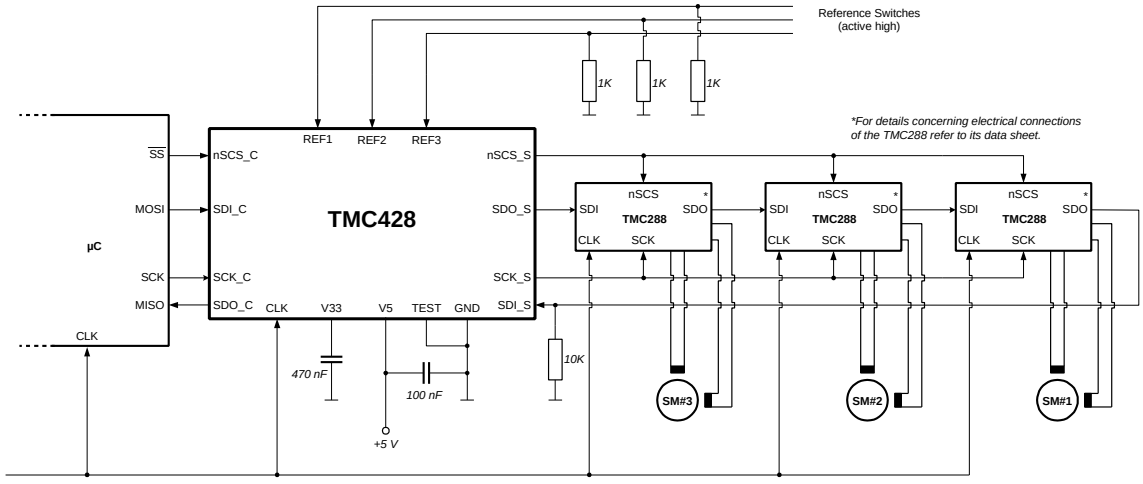



Figure 1: TMC428 application environment with TMC428 in SSOP16 Package

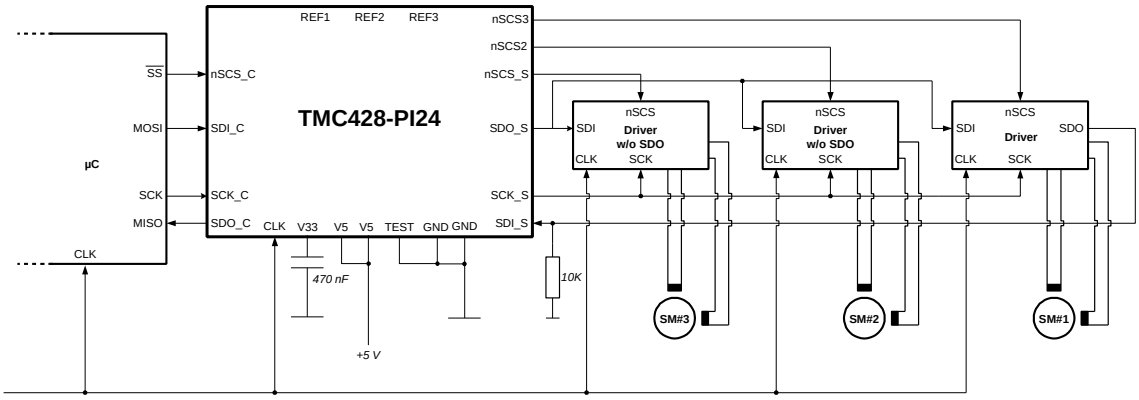



Figure 2 – Usage of Drivers without Serial Data Output (SDO) with TMC428 in SOIC24 Package

The maximum SPI™ data rate is the clock frequency divided by 16. The maximum step frequency depends on the total length of the datagrams send to the SPI™ stepper motor driver chain. At a clock frequency of 16 MHz, with a daisy chain of three SPI™ stepper motor drivers of 16 bit datagram length each, the maximum *full step* frequency is 16 MHz / 16 / ( 3 * 16 ) which is approximately 20 kHz. But, the micro step rate may be higher, even if the stepper motor driver loses micro steps due to SPI™ data rate limit, as long as the number of skipped micro steps is less then a full step. In this respect, one should remember, that at high step rates– respectively pulse rates –the differences between micro stepping and full step excitation vanishes.

The TMC428 has four different modes of motion, programmable individually for each stepper motor, named RAMPMODE, SOFTMODE, VELOCITYMODE, and HOLDMODE. For positioning applications the RAMPMODE is most suitable, whereas for constant velocity applications the VELOCITYMODE is. In RAMPMODE, the user just sets the position and the TMC428 calculates a trapezoid velocity profile and drives autonomously to the target position. During motion, the position may be altered arbitrarily. The SOFTMODE is similar to the RAMPMODE, but the decrease of the amount of velocity during deceleration is done with a soft, exponentially shaped velocity profile. In VELOCITYMODE, a target velocity is set by the user and the TMC428 takes into account user defined limits of velocity and acceleration. In HOLDMODE, the user sets target velocities, but the TMC428 ignores any limits of velocity and acceleration, to realize arbitrary velocity profiles, controlled completely by the user.

The TMC428 has capabilities to generate interrupts depending on different stepper motor conditions chosen by an interrupt mask. However, status bits send back automatically to the micro controller each time it sends data to the TMC428 are sufficient for polling techniques. Error condition handling of stepper motor drivers can be handled by read out option of data bits– up to 48 bits, which is sufficient for most stepper motor drivers – send back from the drivers to the TMC428.

Without any additional logic, in the default reference switch mode, the three reference switch inputs are defined as left side reference switches, one for each stepper motor. In another mode, the 1st reference input is defined as left reference switch input of motor number one, the 2nd reference input is defined as left reference switch input of motor number two, and the 3rd reference input is defined as right reference switch of stepper motor number one. In that mode, there is no reference switch input available for stepper motor three. With an external multiplexer 74HC157 any stepper motor may have a left and a right reference switch.

Serial stepper motor drivers provide different status bits (driver active, in-active, ...) and error bits (short to ground, wire open, ...). To have access to those error bits, datagrams with a total length up to 48 bits

send back from the stepper motor driver chain to the TMC428 are buffered within two 24 bit wide registers. The micro controller has direct access to these registers. Although, the TMC428 provides datagrams with up to 64 bits, only the last 48 bits send back from the driver chain are buffered for read out by the micro controller. This is because buffering of 3 time 16 bits is sufficient for a chain of three TMC288 stepper motor drivers (see Figure 1) and most other drivers– also from other vendors –sending back up to 16 bits.

From the user's point of view, the TMC428 consists of a set of registers, accessed by an micro controller (µC) via a serial interface in an uniform way. Each time, a µC sends a datagram to the TMC428, it simultaneously receives a datagram from the TMC428. Each datagram contains all necessary information to address and to select between read and write for both, the registers and the on-chip memory. This simplifies the communication with the TMC428 and makes the programming easy. Some micro controllers have a SPI™ hardware interface, which directly connects to the serial four wire micro controller interface of the TMC428. For micro controllers without SPI™ hardware interface one just has to write a subroutine sending a 32 bit vector via a serial four wire interface if called, returning that 32 bit vector received from the TMC428.

## Notation of Number Systems

Decimal numbers are used as usual without additional identification. Binary numbers are identified by a prefixed % character. Hexadecimal numbers are identified by a prefixed $ character. So, for example the decimal number 42 in the decimal system is written as %101010 in the binary number system, and it is written as $2A in the hexadecimal number system.

# Pinning

There are two package variants of the TMC428 available. The smaller SSOP16 package is sufficient for our TMC288 SPI™ stepper motor drivers with up to three drivers in a chain and for most SPI™ stepper motor drivers from other vendors. Some SPI™ stepper motor drivers from other vendors have no serial data output and can not simply be arranged in a daisy chain to drive more than one motor. So, the bigger SOIC24 package is provided for those stepper motor drivers from other vendors only. All inputs are Schmitt-Trigger. Possibly unused inputs (REF1, REF2, REF3, SDI_S) need to be connected to ground.

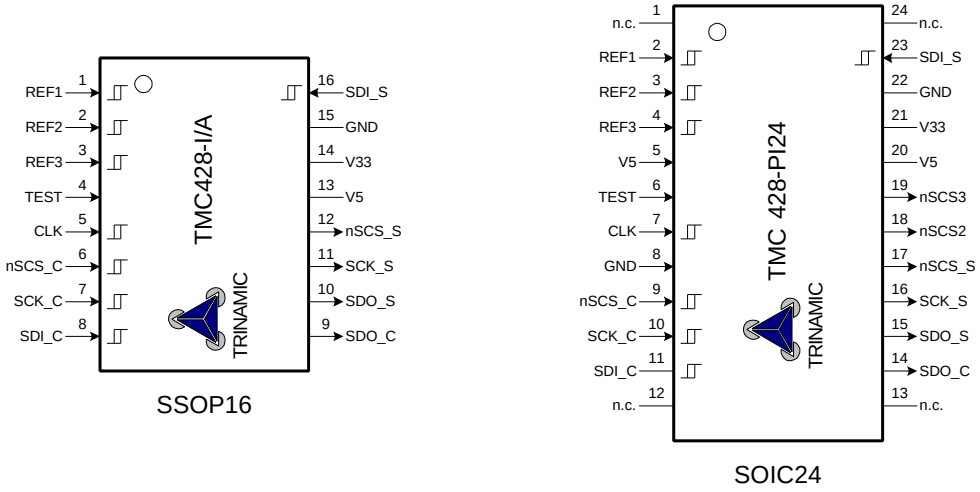



Figure 3: TMC428 pin out

| Pin | SSOP16 | SOIC24 | In/Out | Description |
|---|---|---|---|---|
| Reset | - | - | - | internal power-on reset |
| CLK | 5 | 7 | I | clock input |
| nSCS_C | 6 | 9 | I | low active SPI chip select input driven from µC |
| SCK_C | 7 | 10 | I | serial data clock input driven from µC |
| SDI_C | 8 | 11 | I | serial data input driven from µC |
| SDO_C | 9 | 14 | O | serial data output to µC input |
| nSCS_S | 12 | 17 | O | SPI chip select signal to stepper motor driving chain |
| nSCS2 | - | 18 | O | SPI chip select signal (SOIC24 package only) |
| nSCS3 | - | 19 | O | SPI chip select signal (SOIC24 package only) |
| SCK_S | 11 | 16 | O | serial data clock output to SPI stepper motor driver chain |
| SDO_S | 10 | 15 | O | serial data output to SPI stepper motor driver chain |
| SDI_S | 16 | 23 | I | serial data input from SPI stepper motor driver chain |
| REF1 | 1 | 2 | I | reference switch input 1 |
| REF2 | 2 | 3 | I | reference switch input 2 |
| REF3 | 3 | 4 | I | reference switch input 3 |
| V5 | 13 | 5,20 | | +5V supply / +3.3V supply |
| V33 | 14 | 21 | | 470 nF ceramic capacitor pin / +3.3V supply |
| GND | 15 | 8, 22 | | ground |
| TEST | 4 | 6 | I | must be connected to ground |
| n.c. | - | 1, 12, 13, 24 | - | not connected |

Table 1 - TMC428 pin out

# Functional Description and Block Diagram

From the user's point of view, the TMC428 consists of a set of registers of different units and on-chip RAM (see. Figure 4), accessed via the serial µC interface in an uniform way. The serial interface to the micro controller is easy to use. It uses just a simple protocol with fixed length datagrams for read and write access. The serial interface to the stepper motor driver chain has to be configured by an initialization sequence which writes the configuration into the on-chip RAM. Once configured the serial driver interface works autonomously. The internal multiple port RAM controller of the TMC428 autonomously takes care of access scheduling. So, the user may read and write both, registers and on-chip RAM at any time. The registers hold global configuration parameters and the motion parameters. The on-chip RAM stores the configuration of the serial driver interface and the micro step table.

The ramp generator monitors the motion parameters stored in its registers. If required, it generates step pulses automatically taking user defined motion parameter limits into account. The serial driver interface automatically sends datagrams to the stepper motor driver chain on demand. The ramp generator calculates velocity profiles controlling the pulse generator. The micro step unit (including sequencer) processes step pulses from the pulse generator– representing micro steps, half steps, or full steps depending on the selected step resolution –and makes the results available to the serial driver interface. The ramp generator also interfaces the reference switch inputs. Unused reference switches have to be connected to ground. A pull-down resistor is necessary at the SDI_S input of the TMC428 for those serial peripheral interface stepper motor drivers that set their serial data output to high impedance 'Z' while inactive.

The interrupt controller continuously watches reference switches and ramp generator conditions and generates an interrupt if required. To save pins, the interrupt signal is multiplexed to the SDO_C signal. This signal becomes the low active interrupt signal called nINT while nSCS_C is high (see Figure 5). So, if the micro controller disables the interrupt during access to the TMC428 and enables the interrupt otherwise, the multiplexed interrupt output of the TMC428 behaves like a dedicated interrupt output. For polling, the TMC428 sends the status of the interrupt signal to the micro controller with each datagram.

To drive a stepper motor to a new target position, the user just overwrites the target position of that stepper motor by sending a datagram to the TMC428. To run a stepper motor with a target velocity, the user just writes it into a register assigned to a stepper motor.

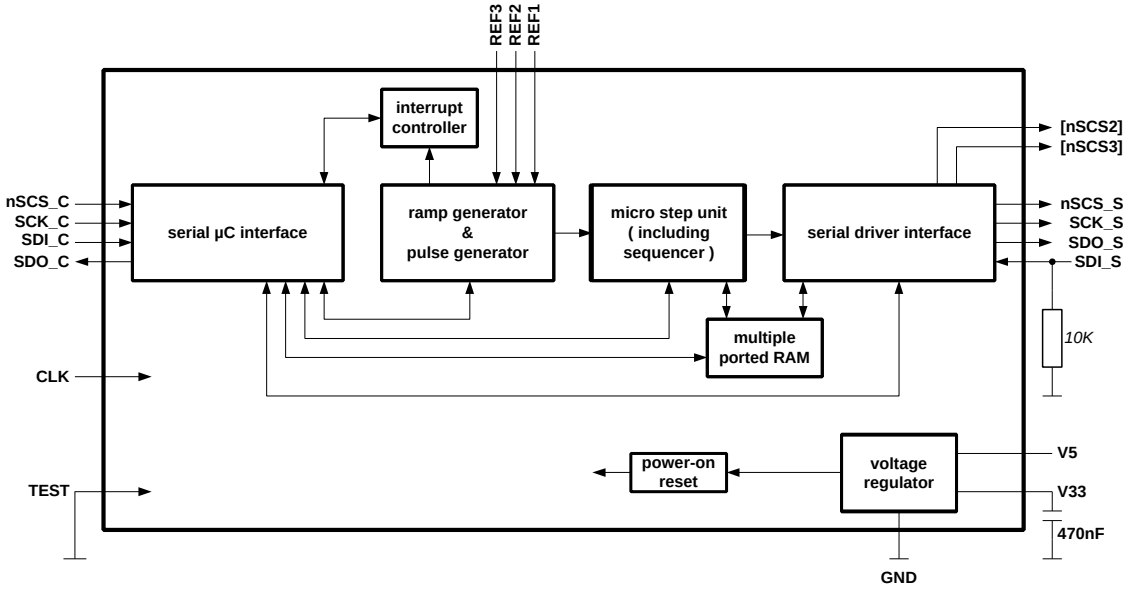



Figure 4: TMC428 functional block diagram

# Serial Peripheral Interfaces

The four pins named SCS_C, SCK_C, SDI_C, SDO_C form the serial micro controller interface of the TMC428. The communication between the micro controller and the TMC428 takes place via 32 bit datagrams of fixed length. Concerning communication, the µC is the master and the TMC428 is the slave, with the TMC428 in turn being the master for the stepper motor driver daisy chain. Similar to the micro controller interface, the TMC428 uses a four wire serial interface for communication with the stepper motor driver daisy chain. The four pins named SCS_S, SCK_S, SDO_S, SDI_S form the serial stepper motor driver interface. Stepper motor drivers with parallel inputs can be used in connection with the TMC428 with some additional glue logic.

## Serial Peripheral Interface for µC

The serial micro controller interface of the TMC428 behaves as a simple 32 bit shift register shifting serial data SDI_C in with the rising edge of the clock signal SCK_C and copying the content of the 32 shift register with the rising edge of the selection signal nSCS_C into a buffer register of 32 bit length. The serial interface of the TMC428 sends back data read from registers or read from internal RAM back immediately via the signal SDO_C. It processes serial data synchronously to the clock signal CLK.

Because of the on-the-fly processing of the input data stream, the serial micro controller interface of the TMC428 accepts the serial data clock signal SCK_C with at least a duration (tSCKCL + tSCKCH = 3*tCLK + 3 * tCLK) of a total number of six clock cycles of CLK as outlined in the timing diagram Figure 5. The data signal from the micro controller changes with the falling level of the serial data clock input SCK_C. The maximum duration (tSCKCL + tSCKCH) of the serial data clock signal SCK_C is unlimited. But three clock cycles is the lower limit for the low level (tSCKCL ≥ 3 * tCLK) of the serial data clock SCK_C and for the high level (tSCKCH ≥ 3 * tCLK) it.

A complete serial datagram frame has a fixed length of 32 bit. While the data transmission from the micro controller to the TMC428 is idle, the low active serial chip select input nSCS_C and also the serial data clock signal SCK_C are set to high. While the signal nSCS_C is high, the TMC428 assigns the status of the internal low active interrupt signal named nINT to the serial data output SDO_C (Figure 5). The serial data input SDI_C of the TMC428 has to be driven by the micro controller. In contrast to other SPI™ compatible devices, the SDO_C signal of the TMC428 is always driven. It will never be in high impedance 'Z'.

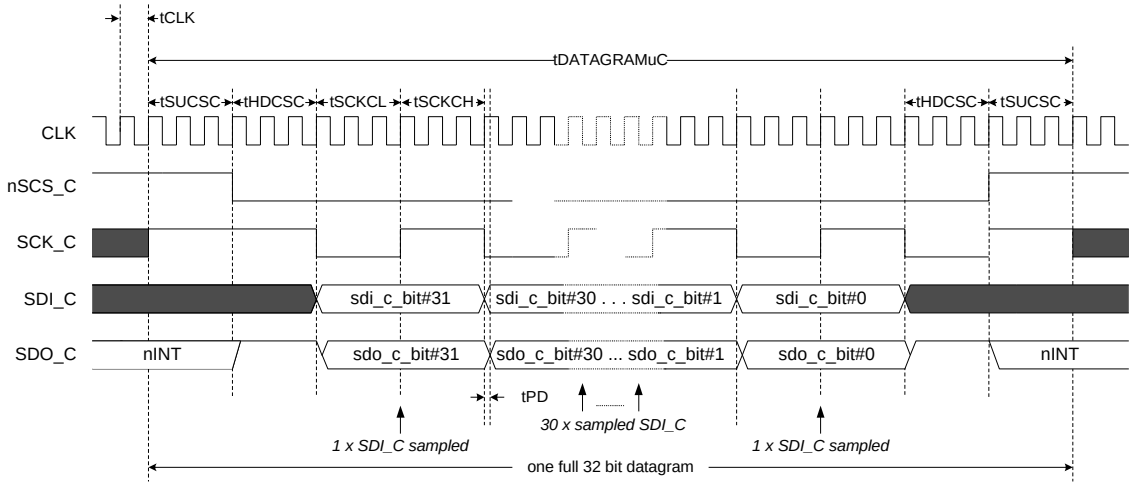



Figure 5 - Timing diagram of the serial µC interface

The signal nSCS_C has to be assigned at least three clock cycles (tSUCSC) high before sending a 32 bit datagram. To initiate a transmission, the signal nSCS_C has to be set to low. Three clock cycles later (tHDCSC) the serial data clock may go low. The most significant bit (MSB) of a 32 bit wide datagram comes first and the least significant bit (LSB) is transmitted as the last one. To complete the data transmission the serial data clock SCK_C has to set to high first and at least three clock cycles later, the signal nSCS_C has to be assigned to high. So, nSCS_C and SCK_C change in opposite order from low to high at the end of a transmission as these signals change from high to low at the beginning.

### Automatic Power-On Reset

The TMC428 performs an automatic power-on reset (see ). To be sure, that the power-on reset has been completed before starting communication with the TMC428, one should wait at least for 10 µs before sending the first datagram, which is approximately one datagram at 16 MHz clock frequency (tDATAGRAM16MHzMin = (1+32+1) * 6 / 16 MHz = 12.75 µs).

### Serial Peripheral Interface to Stepper Motor Driver Chain

The timing of the serial stepper motor interface is similar to that of the micro controller interface. It directly connects to SPI™ smart power stepper motor drivers. The bit mapping is configurable individually for each stepper motor driver chip of the daisy chain. From the micro controllers point of view, it simply sends a fixed sequence of datagrams to the TMC428 to initialize it after power-up. Once initialized by the micro controller, the TMC428 autonomously generates the datagrams for the stepper motor driver daisy chain without any additional interventions of the micro controller.

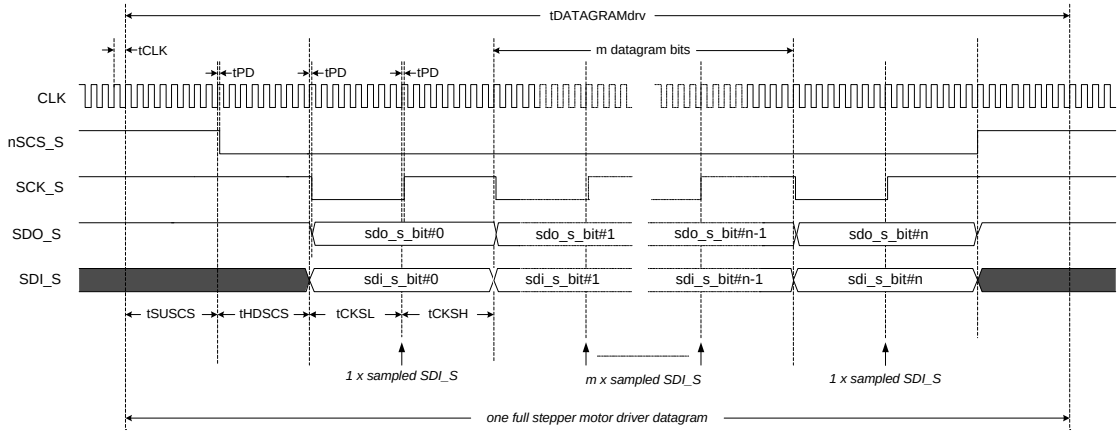



Figure 6: Timing diagram of the serial stepper motor driver interface

The bit mapping for each stepper motor driver is composed of so called *primary signal bits* provided by the micro step unit of the TMC428 individually for each stepper motor. Each primary signal bit is represented by a five bit code word called *primary signal code*. The order of primary signal bits to be send to the stepper motor driver daisy chain is defined by the order of primary signal code words in the configuration RAM area. To distinguish different stepper motor drivers, an additional bit called *next motor bit* (NxM-Bit) is prefixed to the five bit wide primary signal code words. So, the total data word width is six bit. Each NxM-Bit effects an incrementation of an internal stepper motor address until processing– sending serially datagram bits –of the last stepper motor is completed. For this, a parameter called LSMD (last stepper motor driver) has to be programmed during initialization after power up. So, the codes written into the serial interface configuration RAM area represent the mapping of control signals provided by the micro step units to control bits of the drivers. It might be noted here, that configuring the serial driver interface is much easier as it might seem here. It is explained in detail, illustrated by examples below.

The timing of the serial driver interface is programmable in a wide range. The clock divider provides 16 up to 512 clock cycles (tCLK) for a serial driver interface data clock period. The default duration of a clock period (tSCKCL+tSCKCH) of the signal nSCS_S is 16+16=32 clock periods of the clock signal CLK. The minimal duration of a serial interface clock period (tSCKCL+tSCKCH) is 8+8=16 clock cycles of signal CLK as outlined in Figure 6. Also, the polarities of the signals nSCS_S and SCK_S are programmable to use driver chips from other vendors with inverted polarities without additional glue logic.

The input SDI_S of the serial driver interface must always be driven to a defined level. So, to avoid high impedance ('Z') at that input pin, a pull-up resistor or a pull-down resistor of 10 KΩ is necessary at that input, if the stepper motor driver chain is idle.

| Symbol | Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| tSUCSC | Setup Clocks for nSCS_C | 3 | | ∞ | CLK periods |
| tHDCSC | Hold Clocks for nSCS_C | 3 | | ∞ | CLK periods |
| tSCKCL | Serial Clock Low | 3 | | ∞ | CLK periods |
| tSCKCH | Serial Clock High | 3 | | ∞ | CLK periods |
| tDAMAGRAMuC | Datagram Length | 3+3 + 32*6 + 3+3 = 204 | | ∞ | CLK periods |
| tDAMAGRAMuC | Datagram Length | 12.75 | | ∞ | µs |
| fCLK | Clock Frequency | 0 | | 16 | MHz |
| tCLK | Clock Period tCLK = 1 / fCLK | 62.5 | | ∞ | ns |
| tPD | CLK-rising-edge-to-Output Propagation Delay | | 5 | | ns |

Table 2 - Timing Characteristics of the Serial Microcontroller Interface

| Symbol | Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| tSUSCS | | 8 | 16 | 256 | CLK periods |
| tHDSCS | | 8 | 16 | 256 | CLK periods |
| tCKSL | | 8 | 16 | 256 | CLK periods |
| tCKSH | | 8 | 16 | 256 | CLK periods |
| tDAMAGRAMdrv | Datagram Length | 8+8+1*16+8+8=48 | | 512+64*512+512= 33792 | CLK periods |
| tDAMAGRAMdrv | Datagram Length | 3 | | 2112 | µs |
| fCLK | Clock Frequency | 0 | | 16 | MHz |
| tCLK | Clock Period tCLK = 1 / fCLK | 62.5 | | ∞ | ns |
| tPD | CLK-rising-edge to Outputs Delay | | 5 | | ns |

Table 3 - Timing Characteristics of the Serial Stepper Motor Driver Interface

Datagram Structure

The micro controller (µC) communicates with the TMC428 via the four wire (nSCS_C, SCK_C, SDI_C, SDO_C) serial interface. Each datagram send to the TMC428 via the pin SDI_C and each datagram received from the TMC428 via the pin SDO_C is 32 bit long. The first bit send is the MSB (most significant bit named sdi_c_bit#31 at Figure 5). The last bit send is the LSB (least significant bit named sdi_c_bit#0 at Figure 5). During reception of a datagram, the TMC428 immediately sends back a datagram of the same length to the micro controller. That datagram send back is the result of the request given by the datagram from the micro controller.

A request to read out one register of the TMC428 immediately turns back a datagram with the contents of that register addressed by the datagram send from the micro controller. In case of writing data into registers, the TMC428 sends back 8 status bits and 24 data bits set to '0'. Datagrams send from the micro controller to the TMC428 have the form:

| 32 bit DATAGRAM send from a µC to the TMC428 via pin SDI_C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |

The 32 bit wide datagrams send to the TMC428 are assorted in four groups of bits: RRS (register RAM select) selecting either registers or on-chip RAM; ADDRESS bits addressing memory within the register set or within the RAM area; RW (read write) bit distinguishing between read access and write access; DATA bits for write access– for read access these bits are *don't care* and should be set to '0'. Different internal registers of the TMC428 have different lengths. So, for some registers only a subset of these 24 data bit is really necessary, and unnecessary data bits should be set to '0'. Some addresses access more than a single register simultaneously. In that cases, unnecessary data bits should also be set to '0'.

The 32 bit wide datagrams received by the µC from the TMC428 are assorted in two groups of bits: STATUS BITS and DATA BITS. The status bits, send back with each datagram, carry the most important information about internal states of the TMC428 and the settings of the reference switches. These datagrams have the form:

| 32 bit DATAGRAM send back from the TMC428 to a µC via pin SDO_C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| STATUS BITS | | | | | | | | DATA BITS | | | | | | | | | | | | | | | | | | | | | | | |
| nINT | CDGW | SM3 | | SM2 | | SM1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | RS3 | xEQt3 | RS2 | xEQt2 | RS1 | xEQt1 | | | | | | | | | | | | | | | | | | | | | | | | |

The status bit nINT is the internal low active interrupt controller output signal. Handling of interrupt conditions without using interrupt techniques is possible by polling this status bit. The interrupt signal is also directly available at the SDO_C pin of the TMC428 if nSCS_C is high. The pin SDO_C may directly be connected to an interrupt input of the micro controller. Because the SDO_C / nINT output is multiplexed,

the micro controller has to disable its interrupt input while it sends a datagram to the TMC428, because SDO_C signal– driven by the TMC428 –alternates during datagram transmission.

For initialization purposes, the TMC428 enables direct communication between the micro controller and the stepper motor driver chain by sending a so called *cover datagram*. The position cover_position and actual length cover_len of a cover datagram is specified by writing them into a common register. Writing an up to 24 bit wide cover datagram to the register cover_datagram will fade in that cover datagram into the next datagram send to the stepper motor driver chain. As a default setting, the TMC428 only sends datagrams on demand. Optionally, continuous update – periodic sending of datagrams to the stepper motor driver chain – is also possible. So, the status bit named CDGW (cover datagram waiting) is a handshake signal for the micro controller in regard to the datagram covering mechanism. This feature is necessary to enable direct data transmission from a micro controller to the stepper motor driver chips for initialization purposes.

The status bits RS3, RS2, RS1 represent the settings of the reference switches. But, the reference switch inputs REF3, REF2, REF1 are not mapped directly to these status bits. Rather, the reference switch inputs may have different functions, depending on programming (see page 20). The three status bits xEQt3, xEQt2, xEQt1 indicate individually for each stepper motor, if it has reached its target position. The status bits RS3, RS2, RS1 and bits xEQt3, xEQt2, xEQt1 can trigger an interrupt or enable simple polling techniques.

Simple Datagram Examples

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

*0*110011000000000000000000100100011

to the TMC428. With inactive interrupt (nINT=1), no cover datagram waiting (CDGW=0), all reference switches inactive (RS3=0, RS2=0, RS1=0), and all stepper motors at target position (xEQt3=1, xEQt2=1, xEQt1=1) the status bits would be %*10010101* the TMC428 would send back the 32 bit datagram:

*10010101*000000000000000000000000

To read (rw=1) back that register write before, one would have to send the 32 bit datagram

*0*110011000000000000000000000000

to the TMC428 and would get back from it the datagram

*10010101*000000000000000100100011.

Write (rw=0) access to on-chip RAM (rrs=1) to an address %111111 occurs similar to register access, but with rrs=1. To write two 6 bit data words %100001 and %100011 to successive pair-wise RAM addresses %1111110 and %1111111 (%100001 to %1111110 and %100011 to %1111111) which are commonly addressed by one datagram (see pages 13 and 29), one would have to send the datagram

*1*111111000000000001000110010000 1.

To read (rw=1) from that on-chip memory address, one would have to send the datagram
*1*111111000000000000000000000000.

## Address Space Partitions

The address space is partitioned in different ranges. Each of the up to three stepper motors has a set of registers individually assigned to it, arranged within a contiguous address space. An additional set of registers within the address space holds some global parameters common for all stepper motors. One dedicated global parameter register is essential for the configuration of the serial four wire stepper motor driver interface. One half of the on-chip RAM address space holds the configuration parameters for the stepper motor driver chain. The other half of the on-chip RAM address space is provided to store a micro step table if required. The first seven datagram bits (*sdi_c_bit#31* and *sdi_c_bit#30 ... sdi_c_bit#25*, respectively *RRS* and *ADDRESS*) address the whole address space of the TMC428.

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

Table 4 - TMC428 address space partitions

The stepper motors are controlled directly by writing motion parameters into associated registers. Only one register write access is necessary to change a target motion parameter. E.g. to change the target position of one stepper motor, a micro controller has to send only one 32 bit datagram to the TMC428. The same is true for changing a target velocity. Some parameters are composed as a single data word at a single address. Those parameters– initialized once and unchanged during operation –have to be changed commonly. Access to on-chip RAM addresses takes always place to two successive RAM addresses. So, always two data words are modified with each write access to the on-chip RAM. Once initialized after power-up, the content of the RAM is usually left unchanged.

### Read and Write

Read and write access is selected by the RW bit (sdi_c_bit#24) of the datagram send from the µC to the TMC428. The on-chip configuration RAM and the registers are writeable with read-back option. Some addresses are read-only. Write access (RW=0) to some of those read-only registers triggers initialization.

### Register Set

The register address mapping is given in Table 5 on page 14. These registers are initialized internally during power-up. During power-up initialization, the TMC428 sends no datagrams to the stepper motor driver chain. *Please note*: Before writing target parameters to the register set, the RAM has to be initialized first.

### RAM Area

The RAM address mapping is given in Table 12 page 30. The on-chip RAM is *NOT* initialized internally during power-up. The RAM has to be initialized by the micro controller first after power-up.

32 bit DATAGRAM send from a µC to the TMC428 via pin SDI_C

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |

| RRS | smda | | IDX | | | | RW | DATA |
|---|---|---|---|---|---|---|---|---|
| 0 | 00 | | 0 | 0 | 0 | 0 | | x_target |
| | | | 0 | 0 | 0 | 1 | | x_actual |
| | | | 0 | 0 | 1 | 0 | | v_min |
| | | | 0 | 0 | 1 | 1 | | v_max |
| | | | 0 | 1 | 0 | 0 | | v_target |
| | | | 0 | 1 | 0 | 1 | | v_actual |
| | | | 0 | 1 | 1 | 0 | | a_max |
| | | | 0 | 1 | 1 | 1 | | a_actual |
| | 01 | | 1 | 0 | 0 | 0 | | is_agtat, is_aleat, is_v0, a_threshold |
| | | | 1 | 0 | 0 | 1 | | 1, pmul, pdiv |
| | 10 | | 1 | 0 | 1 | 0 | | lp, ref_conf, rm |
| | | | 1 | 0 | 1 | 1 | | interrupt_mask, interrupt_flags |
| | | | 1 | 1 | 0 | 0 | | pulse_div, ramp_div, usrs |
| | | | 1 | 1 | 0 | 1 | | dx_ref_tolerance |
| | | | 1 | 1 | 1 | 0 | | x_latched |
| | | | 1 | 1 | 1 | 1 | | |
| | 11 | | JDX | | | | | common registers (SMDA=11) |
| | | | 0 | 0 | 0 | 0 | | datagram_low_word |
| | | | 0 | 0 | 0 | 1 | | datagram_high_word |
| | | | 0 | 0 | 1 | 0 | | cover_position, cover_len |
| | | | 0 | 0 | 1 | 1 | | cover_datagram |
| | | | 0 | 1 | 0 | 0 | | |
| | | | 0 | 1 | 0 | 1 | | |
| | | | 0 | 1 | 1 | 0 | | |
| | | | 0 | 1 | 1 | 1 | | |
| | | | 1 | 0 | 0 | 0 | | power-down |
| | | | 1 | 0 | 0 | 1 | | |
| | | | 1 | 0 | 1 | 0 | | |
| | | | 1 | 0 | 1 | 1 | | |
| | | | 1 | 1 | 0 | 0 | | |
| | | | 1 | 1 | 0 | 1 | | |
| | | | 1 | 1 | 1 | 0 | | l3, r3, l2, r2, l2, r1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | | mot1r, refmux, cont_update, clk2_div, cs_ComInd, polarities (DAC_AB, FD_AB, PH_AB, SCK_S, nSCS_S), LSDM |

Table 5 - TMC428 register address mapping

# Register Description

The registers contain binary coded numbers. Some are unsigned, positive numbers, some are signed numbers in two's complement, and some are just bit vectors or bit vectors of single flags.

x_target (IDX=%0000)

This register holds the current target position in units of full steps respectively micro steps. The unit of the target position depends on the setting of the associated micro step resolution register usrs. If the difference x_target - x_actual is not zero then the TMC428 moves the stepper motor that the difference becomes zero. Both, target position x_target and current position x_actual (usually not necessary) may be altered on the fly. To move from one position to another, the ramp generator of TMC428 automatically generates ramp profiles (step pulses with defined frequencies) in consideration of velocity limits v_min and v_max and acceleration limit a_max.

Note: The registers x_target, x_actual, v_min, v_max, and a_max are initialized with zero after power up. Thus, no step pulses are generated because motion is prohibited. Setting a_max to zero during motion of the stepper motor results in the inability of the stepper motor to stop, because it cannot change its velocity.

x_actual (IDX=%0001)

The current position of each stepper motor is available by read out of the registers called x_actual. The actual position can be over written by the micro controller. This feature is for reference switch calibration under control of the micro controller. If a reference switch position has been determined, the actual position is set to zero at the reference switch position.

v_min (IDX=%0010)

This register holds the absolute value of the velocity where the stepper motor can be stopped abruptly. It should be set greater than zero. This value allows to reach the target position faster because the stepper motor is not slowed down below v_min before the target is reached. Also consider, that due to the finite numerical representation of integral relations, the target position can not be reached exactly, if the calculated velocity is less than one. So, setting v_min to one assures reaching each target position exactly. The unit of velocity parameters (v_max, v_target, and v_actual) is steps per time unit. The time unit is defined by the parameter pulse_div. The pulse frequencies in unit steps per second depends on the clock frequency of the clock signal at pin CLK of the TMC428.

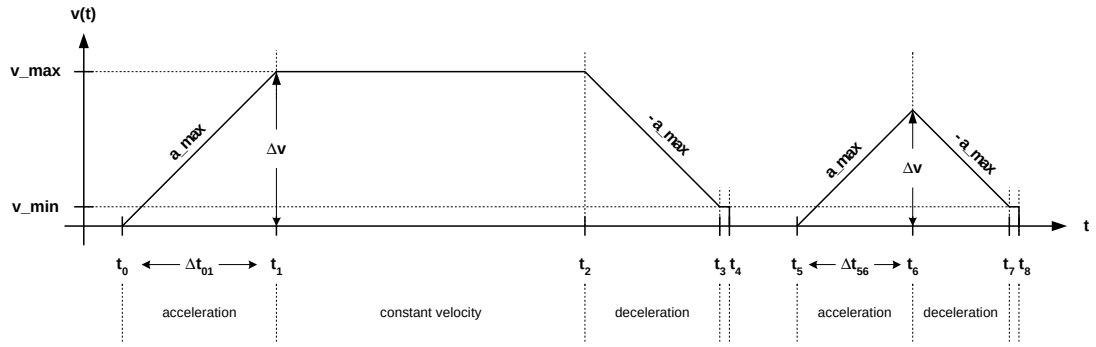



Figure 7 - Velocity ramp parameters and velocity profiles

v_max (IDX=%0011)

This is the limit of the velocity. The absolute value of velocity– which may be positive or negative –will not exceed this limit, except the limit is changed during motion to a value below the current velocity. Note: To set target position x_target and current position x_actual to an equivalent value (e.g. to set both to zero at a reference point), the assigned stepper motor should be stopped first, and the parameter v_max should be set to zero to hold the assigned stepper motor at rest before writing into the register x_target and x_actual.

v_target (IDX=%0100)

In modes RAMP_MODE and SOFT_MODE this register holds the current target velocity calculated internally by the ramp generator. In mode VELOCITY_MODE a target velocity can be written into this register. Then the associated stepper motor accelerates until it reaches the target velocity specified. Changing velocity occurs in consideration of motion parameter limits in VELOCITY_MODE. In HOLD_MODE the register can be overwritten but it is ignored.

v_actual (IDX=%0101)

This read only register holds the current velocity of the associated stepper motor. Internally, the ramp generator of the TMC428 processes with 20 bits while only 12 bits can be read out as v_actual. So, an actual velocity of zero *read out* by the micro controller means that the current velocity is in an interval between zero and one. Because of this, the actual velocity should not be used to detect a stop of a stepper motor. For stop detection there is a dedicated bit within the interrupt register, which can simply be read out by the micro processor or generate an interrupt. But, if one writes zero to register v_actual, which is possible in HOLD_MODE only, the associated stepper motor stops immediately, because hidden bits are set to zero with each write access to the register v_actual. In HOLD_MODE only, this register is a read-write register. In HOLD_MODE, motion parameters are ignored and the micro controller has the full control to generate a ramp. In that mode, the TMC428 only handles the micro stepping and datagram generation for the associated stepper motor of the daisy chain.

a_max (IDX=%0110)

The maximum acceleration is defined by this register. The unit of the acceleration parameters (a_max, a_actual, a_threshold) is change of velocity per time unit divided by 256. Note: In contrast to the time unit of the velocity parameters defined by pulse_div, the time unit of the acceleration parameters is defined by the parameter ramp_div (see page 22). The change of pulse frequencies expressed as *change of velocity per second* also depends on the frequency of the clock signal at pin CLK of the TMC428.

a_actual (IDX=%0111)

The actual acceleration, which the TMC428 actually applies to a stepper motor can be read out by the micro controller from this register for monitoring purposes. Internally, it is updated with each clock. The actual acceleration is used to select scale factors for the coil currents. The returned value a_actual is the smoothed internal acceleration. This smoothing avoids oscillations of the readout value. Thus the returned a_actual values should not be used directly for high precision applications.

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

These parameters represent current scaling values $I_s$ and are applied to the motor depending on the ramp phase: The parameter is_agtat is applied if the acceleration (a) it greater than (gt) a threshold acceleration ($a_t$ ). This is to increase current during strong acceleration. The parameter is_aleat is applied if the acceleration is lower equal (le) than the threshold acceleration. This is the nominal motor current. The third parameter is_v0 is applied if the stepper motor is at rest, to save power, to keep it cool, and to avoid noise probably caused by chopper drivers. The parameter a_threshold is the threshold used to compare with the current acceleration to select the current scale factor. The three parameters is_agtat, is_aleat, and is_v0 are bit vectors of three bit width. One of these is selected conditionally and assigned to an interim bit vector i_scale. The current scaling factor $I_s$ is defined in Table 6.

| i_scale | | | $I_s$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 / 8 |
| 0 | 1 | 0 | 2 / 8 |
| 0 | 1 | 1 | 3 / 8 |
| 1 | 0 | 0 | 4 / 8 |
| 1 | 0 | 1 | 5 / 8 |
| 1 | 1 | 0 | 5 / 8 |
| 1 | 1 | 1 | 7 / 8 |

Table 6 - Current Scale Factors

One of the three scale factors is_agtat, is_aleat, and is_v0 is selected according to Table 7. If the velocity is zero, the parameter is_v0 is used for scaling. If the velocity is not zero, either is_aleat or is_agtat is used for scaling, depending on the absolute value of the acceleration and the acceleration threshold a_threshold.

| | | |
|---|---|---|
| v = 0 | | $I_s$ := is_v0 |
| v ? 0 | $\lvert a \rvert = a_{threshold}$ | $I_s$ := is_aleat |
| | $\lvert a \rvert > a_{threshold}$ | $I_s$ := is_agtat |

Table 7 - Current Scale Selection Scheme

The automatic motion dependent current scale feature of the TMC428 is provided primarily for micro step operational mode. But it may also be applied for full step or half step drivers, if those provide current control bits. For those drivers, one could initialize the micro step tabular with a constant function, square function or sine wave using the two most significant DAC bits.

pmul & pdiv (IDX=%1001)

The stepper motors are driven with a trapezoidal velocity profile, which may become triangular if the maximum velocity is not reached (see Figure 7, 15). Depending on the difference between the target position x_target and the actual position x_actual, the ramp generator continuously calculates target velocities v_target for the pulse generator (see Figure 8, page 18). The pulse generator then generates (micro) step pulses taking into account the motion parameter limits (v_min, v_max, a_max). With a target velocity proportional to the difference of target position x_target and current position x_actual,

the stepper motor approaches the target position. This also works, if the target position is changed during motion. The stepper motor moves to a target position until the difference between the target position x_target and the current position x_actual vanishes.

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

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

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

$$p = p\_mul / p\_div$$

with

$$p\_mul = \{128, 128+1, 128+2, 128+3, ..., 128+127\}$$

and

$$p\_div = \{2^3, 2^4, 2^5, ..., 2^{14}, 2^{15}, 2^{16}\}.$$

Instead of direct storage of the parameters p_mul and p_div, the TMC428 stores two parameters called pmul and pdiv, with

$$p\_mul = 128 + pmul \quad \text{and} \quad p\_div = 2^{3+pdiv} = 2\text{^}(3+pdiv)$$

where

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

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

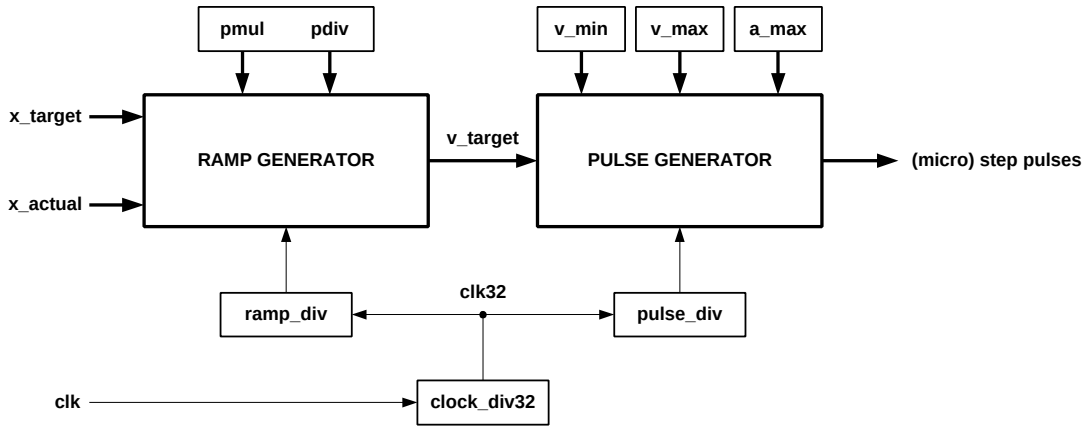



Figure 8 - Ramp Generator and Pulse Generator

The parameter p has to be calculated for a given acceleration. This calculation is not done by the TMC428 itself, because this task has to be done only once for a given acceleration limit. The acceleration limit is a stepper motor parameter, which is usually fixed in most applications. If the acceleration limit has to be changed nevertheless, the micro controller could do this task or one could provide a pair of p_mul and p_div in a memory for each acceleration limit a_max required.

### How to Calculate p_mul and p_div respectively pmul and pdiv

The proportionality factor p = p_mul / p_div depends on the acceleration limit a_max. So, a pair of p_mul / p_div has to be calculated once for each proposed acceleration limit a_max. There may exist more than one valid pair of p_mul and p_div for a given a_max.

To accelerate, the ramp generator with each time step accumulates the acceleration value to the actual velocity. Internally, the absolute value of the velocity is represented by 11+8 = 19 bits, while only the most significant 11 bits and the sign are used as input for the pulse generator. So, there are $2^{11}$ = 2048 values possible to specify a velocity, ranging from 0 to 2047. The ramp generator accumulates a_max divided by $2^8$ = 256 at each time step to the velocity during acceleration phases. So, the acceleration from velocity = 0 to maximum velocity = 2047 spans over 2048* 256 / a_max pulse generator clock pulses. Within that acceleration phase, the pulse generator generates S = ½ * 2048* 256 / a_max * T steps for its (micro) step unit. The parameter T is the clock divider ratio T = $2^{rampdiv}$ / $2^{pulsdiv}$ = $2^{rampdiv-pulsdiv}$= 2^(ramp_div-pulse_div). During acceleration, the velocity has to be increased until the velocity limit v_max is reached or deceleration is required to reach the target position exactly (see Figure 7). The TMC428 automatically decelerates, if required using the difference between current position and target position and the proportionality parameter p, which has to be p = 2048 / S. With this, one gets p = 2048 / ( ( ½ * 2048* 256 / a_max ) * 2^(ramp_div-pulse_div) ) . This expression can be simplified to

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

To avoid overshooting, the parameter p_mul should be made approximately up to 10% smaller than calculated. If the proportionality parameter p is too small, the target position will be reached slower, because the slow down ramp starts earlier. The target position is approached with minimal velocity v_min, whenever the internally calculated target velocity becomes less than v_min. With a good parameter p the minimal velocity v_min is reached a couple of steps before the target position. With parameter p set a little bit to large and small v_min overshooting of one step respectively one micro step may occur. Decrementation of the parameter pmul avoids such one-step overshooting.

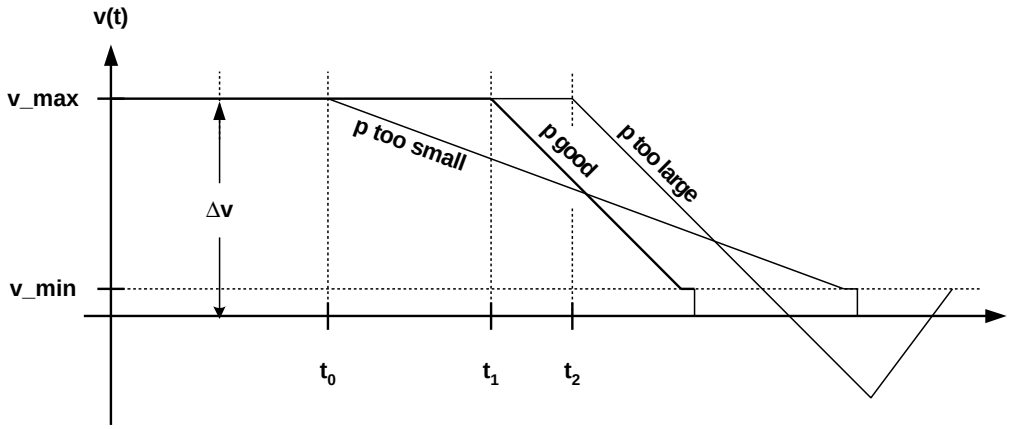



Figure 9 - Proportionality Parameter p and Outline of Velocity Profile(s)

To represent the parameter p = p_mul / p_div = (128+pmul) / 2^(3+pdiv) one just has to find a pair of pmul and pdiv that approximates p, with pmul in range 0 ... 127 and pdiv one of {8, 16 , 32 ,64 ,128 , 256, 512, 1024, 2048, 4096, 8192, 16384, 32786, 65536}. There are only 128 * 14 = 1792 pairs of (pmul, pdiv). So, one can simply try all possible pairs (pmul, pdiv) with a program and choose one pair. To find one pair, one has to calculate

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

and

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

and

q = p / p'

for each pair (pmul , pdiv) and select one pair for that 1.0 < q < 0.95 is valid. So, the value q interpreted as a function q(a_max, ramp_div, pulse_div, pmul, pdiv) gives the quality criterion required. Although q = 1.0 indicates that (pmul , pdiv) perfectly represents the desired p for a given a_max, this could cause overshooting because of finite numerical precision. In case of high resolution micro stepping, overshooting of one micro step is negligible, if it is below the micro step resolution of a stepper motor. To avoid overshooting, one can use pmul-1 instead of the selected pmul. An example program in C language can be found on page 47.

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

The bit vectors ref_conf and ramp_mode are accessed via a common address, because these parameters normally are initialized only once. The bit called lp (latched position) is a read only status bit. The TMC428 has three reference switch inputs. Without additional hardware, three reference switches are available. Per default, each reference switch input is assigned individually to each stepper motor as a left reference switch. The reference switch input REF3 can alternatively be assigned as the right reference switch of stepper motor number one. In that configuration *a left and a right reference switch* is assigned to stepper motor one, a left reference switch is assigned to stepper motor two, and no reference switch is assigned to stepper motor three. The bit named mot1r in the stepper motor global parameter register (rrs=1 & address=111111) selects one of these configurations. With additional hardware, up to six reference switches– a left and a right one assigned to each stepper motor –are supported (see Figure 11). The additional hardware is just a 74HC157, where three of four 2-to-1-multiplexers are used. The feature of multiplexing is controlled by the bit named refmux in the stepper motor global parameter register (rrs=1 & address=111111).

The two least significant bits on this address named ramp_mode (rm) select one of the four possible stepping modes:

00 : RAMP_MODE
01 : SOFT_MODE
10 : VELOCITY_MODE
11 : HOLD_MODE

The mode called RAMP_MODE is proposed as the default mode for positioning tasks, where the VELOCITY_MODE is the default for applications, where stepper motors have to be driven precisely with constant velocity. The SOFT_MODE is similar to the standard RAMP_MODE except that the target position is approached quite slowly, but this feature could be useful for those applications where vibrations at the target positions have to be minimized. The HOLD_MODE is proposed for motion control applications, where the ramp generation is completely controlled by the micro controller.

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

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

| 32 bit DATAGRAM send from a µC to the TMC428 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | smda | | 1 | 0 | 1 | 0 | | | | | | | | | | | | | lp | ref_conf | | | | | | | | | | rm | |
| 0 | | | | | | | | | | | | | | | | | | | latched position (waiting) | REF_RnL | SOFT_STOP | DISABLE_STOP_R | DISABLE_STOP_L | | | | | | | 00 : RAMP, 01 : SOFT, 10 : VELOCITY, 11 : HOLD |

Table 8 - lp & ref_conf & ramp_mode (rm)

A reference switch indicates a reference position within a given tolerance. The tolerance can be programmed in units of steps respectively micro steps. When a reference switch is reached, the actual position can be stored automatically. The read-only status bit lp (latch position waiting) indicates the status of this automatic storage of the current position triggered by a reference switch tagged event.

interrupt_mask & interrupt_flags (IDX=%1011)

The TMC428 provides one interrupt register of eight flags for each stepper motor. Each interrupt bit can be disabled individually. The interrupt status nINT is low active. So, if an interrupt condition of one stepper motor becomes active, nINT is set to low ('0') and it remains low until all interrupt bits– selected by interrupt masks –of all stepper motors become inactive. If the interrupt status is inactive, nINT is high ('1'). The interrupt status is mapped to the most significant bit (31) of each datagram send back to the µC (see page 10) and it is available at the SDO_C pin of the TMC428 if the pin nSCS_C is low.

Demultiplexing of the multiplexed interrupt status signal at the pin SDO_C could be done using additional hardware. But it is not necessary if the micro controller always disables its interrupt when it sends a datagram to the TMC428 and enables its interrupt input when sending of the datagram is completed.

| 32 bit DATAGRAM send from a µC to the TMC428 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | smda | | 1 | 0 | 1 | 1 | | | | | | | | | | interrupt mask | | | | | | | | interrupt flags | | | | | | | |
| | | | | | | | | | | | | | | | | MASK_STOP_RIGHT_HIGH | MASK_STOP_LEFT_HIGH | MASK_STOP_RIGHT_LOW | MASK_STOP_LEFT_LOW | MASK_STOP | MASK_REF_MISS | MASK_REF_WRONG | MASK_POS_END | INT_STOP_RIGHT_HIGH | INT_STOP_LEFT_HIGH | INT_STOP_RIGHT_LOW | INT_STOP_LEFT_LOW | INT_STOP | INT_REF_MISS | INT_REF_WRONG | INT_POS_END |

Table 9 - interrupt register & interrupt mask

An interrupt flag is set to '1' if its assigned interrupt condition occurs and the corresponding interrupt mask is set ('1'). Interrupt flags are reset to '0' by a write access (RW='0') to the interrupt register address (IDX=%1010) with a '1' at the position of the bit to be cleared. A '0' at the corresponding position within the datagram used to reset interrupt flags leaves the corresponding interrupt flag untouched.

If an end position is reached and the interrupt mask MASK_POS_END is '1' the bit named INT_POS_END is set to one. If the reference switch becomes active out of the reference switch tolerance range– defined by the dx_ref_tolerance register –the interrupt flag INT_REF_WRONG is set if its interrupt mask bit MASK_REF_WRONG is set. The interrupt flag INT_REF_MISS is set if the reference switch is inactive at the 0 position and the mask MASK_REF_MISS is enabled. The INT_STOP flag is set, if the reference switch has forced a stop and if the interrupt mask MASK_STOP is set. The INT_STOP_LEFT_LOW flag is set if the reference switch changes from high to low and if the interrupt mask bit MASK_STOP_LEFT_LOW is set. The interrupt flag INT_STOP_RIGHT_LOW similar to INT_STOP_LEFT_LOW but for the right reference switch. The INT_STOP_LEFT_HIGH indicates that the left reference switch input changes from low to high if the mask MASK_STOP_LEFT_HIGH is set. The INT_STOP_RIGHT_HIGH indicates it for the right reference switch if the mask MASK_STOP_LEFT_HIGH is set.

pulse_div & ramp_div & usrs (IDX=%1100)

The frequency of the external clock signal (see Figure 3, CLK) is divided by 32 (see Figure 8, clk_div32). This clock drives two programmable clock dividers for the ramp generator and for the pulse generator. The pulse generator clock– defining the maximum step pulse rate –is determined by the parameter pulse_div. The pulse rate R (step frequency respectively micro step frequency) is given by

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

where f_clk[Hz] is the frequency of the external clock signal. The parameter velocity is in range 0 to 2047. The pulse generator of the TMC428 generates one step pulse with each pulse generator clock pulse if the velocity is set to 2047. The change ΔR in the pulse rate per time unit (step frequency change per second – the acceleration) is given by

ΔR[Hz/s] = f_clk[Hz] * a_max / 256 / ( 2^ramp_div * 2048 * 32 ).

The parameter a_max is in range 0 to 2047. So, the parameter ramp_div scales the acceleration parameter a_max, where the parameter pulse_div scales the velocity parameters.

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

Table 10 – micro step resolution selection (usrs) parameter

The three bit wide parameter usrs (µ step resolution selection) determines the micro step resolution for its associated stepper motor according to Table 10. There is a individual set of 6 DAC bits proposed for each of the two phases (coils) for current control to provide up to 64 micro steps per full step. Depending on the micro step resolution, a sub set of 6 DAC bits are significant. Using full stepping, the current amplitude is constant for both phases (coils) of a stepper motor and the polarity of one phase (coil) changes with each full step. The micro step counters are initialized by 0 during power-on reset. With each micro step an associated counter accumulates the programmed micro step resolution value usrs.

dx_ref_tolerance (IDX=%1101)

To allow the motor to drive near the reference point, it is possible to exclude a range of steps from the stop switch function. This parameter is important to disable stopping forced by reference switches during reference position search. The parameter affects interrupt conditions as described before.

x_latched (IDX=%1110)

This read-only register stores the actual position if a change of the reference switch is detected. The reference switch is defined by the bit REF_RnL of the configuration register lp & ref_conf & ramp_mode (IDX=%1010). To initialize this position storage mechanism one simply has to write to the (read-only) register. Then the actual position is saved in the register with the next change of the reference switch status. The bit lp signals if latching of the position is pending.

### Unused Address (IDX=%1111)

This register address (idx=1111) within each stepper motor register block {smda=00, 01, 10} is unused. Writing to this register has no effect. Reading this register gives back the actual status bits and 24 data bits set to '0'.

## Global Parameter Registers

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

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

The TMC428 stores datagrams send back from the stepper motor driver chain with a total length of up to 48 bits. The register datagram_low_word holds the lower 24 bits of this 48 bits and the register datagram_high_word holds the higher 24 bits of the 48 bits. These both registers together form a 48 bit shift register, where the data from pin SDI_S are shifted left into it with each datagram bit send to the stepper motor driver chain via the signal SDO_S. A write to one of these read-only register addresses initializes these registers, to update its contents with the next received datagram from the drivers chain.

### cover_pos & cover_len (JDX=%0010)

The TMC428 provides direct sending of datagrams from the micro controller to the stepper motor drivers. This may be necessary for initialization of different driver chips and useful for reconfiguration purposes. A datagram with up to 24 bits can be transferred to the stepper motor driver by covering one datagram sent to the drivers chain. The parameter cover_pos defines the position of the first datagram bit to be covered by the cover_datagram (JDX=%0011) of length cover_len. In contrast to the datagram numbering order of bits the position count for the cover datagram starts with 0. The cover_datagram bits indexed from cover_len-1 to 0 cover the datagram sent to the drivers chain.

> Important: *A step bit used to control stepper motor drivers must not be covered.*

This is because the coverage of a step bit would cause losing that associated step if the step bit is active. The TMC428 stores cover_pos+1 instead of cover_pos due to internal requirements. So, one writes cover_pos but reads back cover_pos+1.

### cover_datagram (JDX=%0011)

This register holds up to 24 bit of a cover datagram. A cover datagram covers the next datagram sent to the stepper motor driver chain. If no datagrams are sent to the drivers chain, the cover datagram is sent immediately. The status of the cover datagram is mapped to the status bits sent back with each datagram (see page 10, CDGW). This status bit is also available for readout of cover_pos & cover_len (JDX=%0010), where CDGW the most significant data bit (23). An example for the cover datagram is given in Figure 10. In that example 7 bits cover 7 bits of a 48 bit datagram from bit number 39 to bit number 46.

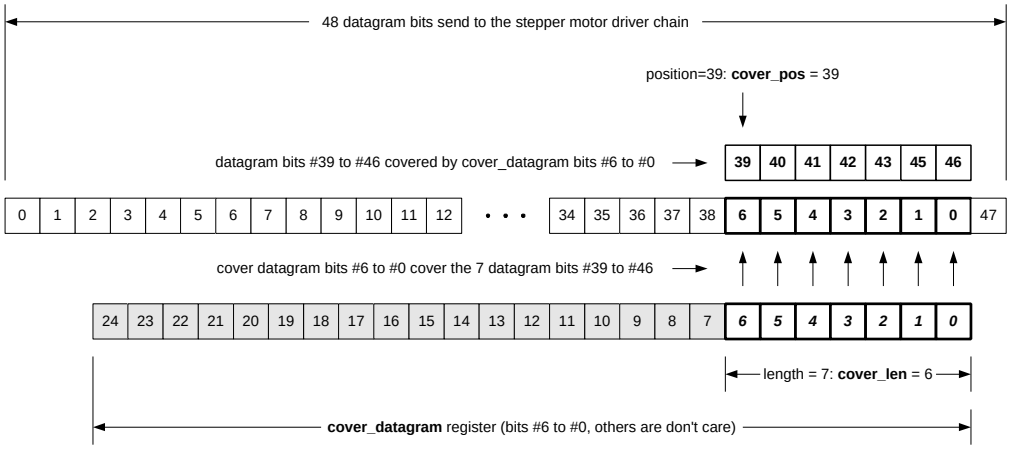



Figure 10: Cover Datagram Example

Unused Addresses (JDX={%0011, %0100, %0101, %0110, %0111})

There are unused addresses before the power_down address within the address space of the global parameter registers.

power_down (JDX=%1000)

A write to the register address named power_down sets the TMC428 into the power down mode until it detects a falling edge at the pin nSCS_C.

Unused Addresses (JDX={%1001, %1010, %1011, %1100, %1101})

There are also unused addresses after the power_down address within the address space of the global parameter registers.

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

The current state of all reference switches– demultiplexed internally by the TMC428 if left and right reference switches are used –can be read from this read-only register. The bit named continuous_update of the Stepper Motor Global Parameter Register (JDX=%1111) is important for reading out of reference switches as explained below.

Stepper Motor Global Parameter Register (JDX=%1111)

Last but not least, this register holds different configuration bits for the stepper motor drivers chain. The absolute address (RRS & ADDRESS) of the stepper motor global parameter register is %0111111 0 ($7E). The stepper motor global parameter register holds different configuration bits together (see Table 11). For the datagram configuration the number of stepper motor drivers is important. It is represented by the

parameter lsmd (last stepper motor driver). The two bit wide parameter lsmd has to be set to %00 for one stepper motor driver, %01 for two stepper motor drivers, and %10 for three stepper motor drivers. Five bits are proposed to change polarities to opposite polarity. The selection signal for the stepper motor driver chain named nSCS_S is controlled by the polarity bit named polarity_nscs_s. The nSCS_S signal is low active if this bit is set to '0' it is high active if this bit is set to '1'. The polarity of the stepper motor driver chain clock signal SCK_S is defined by the bit polarity_sck_s. If this bit is '0' the clock polarity is according to Figure 6 on page 3. The clock signal SCK_S is inverted If it is set to '1'. The bit polarity_PH_AB defines the polarity of the phase bits for the stepper motor. Inverting this bit changes the rotation direction of the associated stepper motor. The bit polarity_FD defines the polarity of the fast decay controlling bit. If it is '0' fast decay is high active and if it is '1' fast decay is low active. The bit named polarity_DAC_AB defines the polarity of the DAC bit vectors. If it is '0' the DAC bits are high active and if it is '1' the DAC bits are inverted – low active.

The bit named csCommonIndividual defines either a single chip select signal nSCS_S is used common for all stepper motor driver chips (resp. TMC288 / TMC289) or three chip select signals nSCS_S, nCS2, nCS3 are use to select the stepper motor driver chips individually. This feature is available only for the TMC428 within the larger SOIC24 package (TMC428-PI24) where the two additional chip select signals nCS2, nCS3 are available (see Figure 2). The one common chip select signal nSCS_S is used if the bit named csCommonIndividual is '0'.

| 32 bit DATAGRAM send from a µC to the TMC428 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | | | | mot1r | refmux | | | | continuous_update | clk2_div | | | | | | | | csCommonIndividual | polarities | | | | | lsmd |
| | | | | | | | | | | | | | | | | | | | | | | | | | polarity_DAC_AB | polarity_FD | polarity_PH_AB | polarity_sck_s | polarity_nscs_s | last stepper motor driver | |

Table 11 - Stepper Motor Global Parameter Register

The seven bits named clk2_div determine the clock frequency of the stepper motor driver chain clock signal SCK_S. The frequency f_sck_s[Hz] of the stepper motor driver chain clock signal SCK_S is f_sck_s[Hz] = f_clk[Hz] / ( 2 * (clk2_div+1) ). A value of 127 (%1111111, $7F) is the upper limit for the parameter clk2_div. With clk2_div = 127 the clock frequency of SCK_S is at minimum. Due to internal processing, a value of 7 (%0000111, $07) is the lower limit for the clock divider parameter clk2_div. With clk2_div = 7 the clock frequency of SCK_S is at maximum. Due to internal processing, the frequency of SCK_S does not become higher for clk2_div < 7, but the signal SCK_S becomes asymmetric with respect to it's duty cycle. An asymmetric duty cycle may cause male function of stepper motor drivers, where stepper motor driver chips may work correctly in particular at low clock frequencies of CLK.

So, the range of clk2_div is {7, 8, 9, . . ., 125, 126, 127}. The default value after power-on reset is clk2_div = 15. The clock frequency of SCK_S should be set as high as possible by choice of the parameter clk2_div in consideration of the data clock frequency limit defined by the slowest stepper motor driver chip of the daisy chain. If step frequencies reach the order of magnitude of the maximum datagram frequency– determined by the clock frequency of SCK_S and by the datagram length –the step frequencies may vary, which is an inherent property of that serial communication. Either if variations of step frequencies are acceptable or not depends on the application. Using high resolution micro stepping driver chips– as provided by TMC289 / TMC288 driver chips –avoids this problem.

The TMC428 sends datagrams to the stepper motor driver chain on demand if continuous_update is '0'. This reduces the communication traffic. The reference switches are processed while datagrams are send to the stepper motor driver chain only. If reference switches are configured to stop associated stepper motors automatically, the configuration bit continuous_update must be set to '1' to force periodic sending of datagrams to the stepper motor driver chain and to sample the reference switches periodically, also if all stepper motors are at rest. With this, a stepper motor restarts if its associated reference switch becomes inactive. Without continuous update, a stepper motor stopped automatically by a reference switch would stay at rest until a datagram is send to the stepper motor driver chain, also if its reference switch becomes inactive. Than, the relevant stepper motor can be moved in the opposite direction of the reference switch or in can be moved in both directions by disabling the automatic stop function.

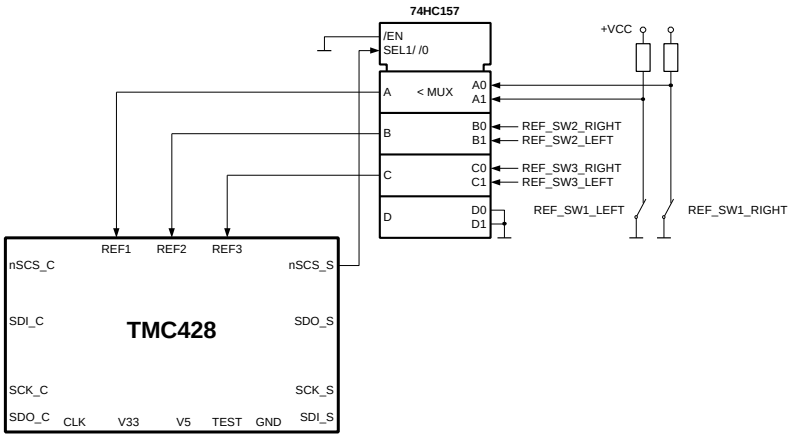



Figure 11 - Reference Switch Multiplexing with 74HC157 (refmux=1)

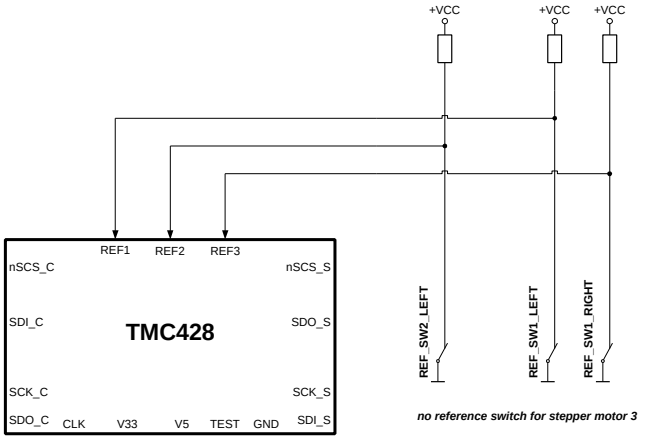



Figure 12 – Two-One-Null Reference Switch Configuration for mot1r=1 (and refmux=0)

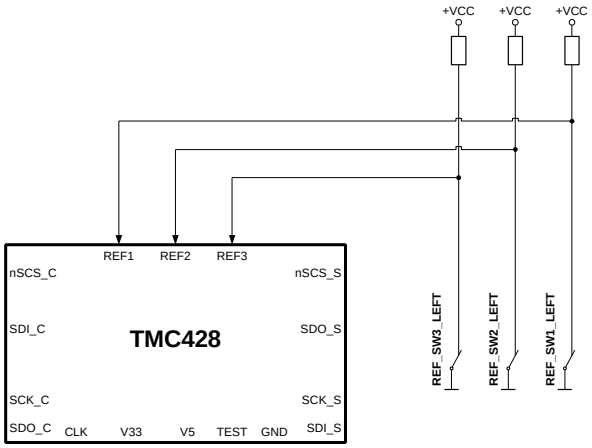



Figure 13 - Left-Side-Only Reference Switch Configuration for mot1r=0 (and refmux=0)

If continuous_update is '1', internal status bits are updated periodically also if all stepper motors are at rest. Additionally, the chip select signal nSCS_S for the stepper motor driver chain is also the control signal for a multiplexer in case of using the reference switch multiplexing option (see Figure 11). So, the continuous_update must be set to '1' if automatic stop by reference switches is enabled, if six multiplexed reference switches are used, and to get the states of reference switches while all stepper motors are at rest.

The bit named refmux must be set to '1' to enable reference switch multiplexing (see Figure 11). If reference switch multiplexing is enabled, and the mot1r is ignored. The default value after power-on reset of refmux is '0'. If refmux is '0', the association of the reference switch inputs REF1, REF2, REF3 depends on the setting of the configuration bit mot1r. The power-on default value of mot1r is '0'. With that default value, REF1 is associated to the left reference switch of stepper motor #1, REF2 is associated to the left reference switch of stepper motor #2, and REF3 is associated to the left reference switch of stepper motor #3.

If mot1r is set to '1' the input REF1 is also associated to the left reference switch of stepper motor #1. REF2 is also associated to the left reference switch of stepper motor #2. But, the input REF3 is associated to the *right reference switch* of stepper *motor #1* and no reference switch input is associated to stepper motor number#3 (see Figure 12).

After power-on-reset, per default refmux=0 and mot1r=0 selects the single reference switch configuration outlined in Figure 13, where each reference switch input (REF1, REF2, REF3) is assigned individually to one each stepper motor as the left reference switch.

### Simultanous Start of up to Three Stepper Motors

Starting stepper motors simultanously can be achifed by sending successive datagrams starting the stepper motors. If the delay between those datagrams is of the magnitude of some micro seconds, the stepper motors can be considered as started simultanously. Feedding the reference switch signals through the microcontoller (Figure 14) allows exact simultanous start of the stepper motors under software control.

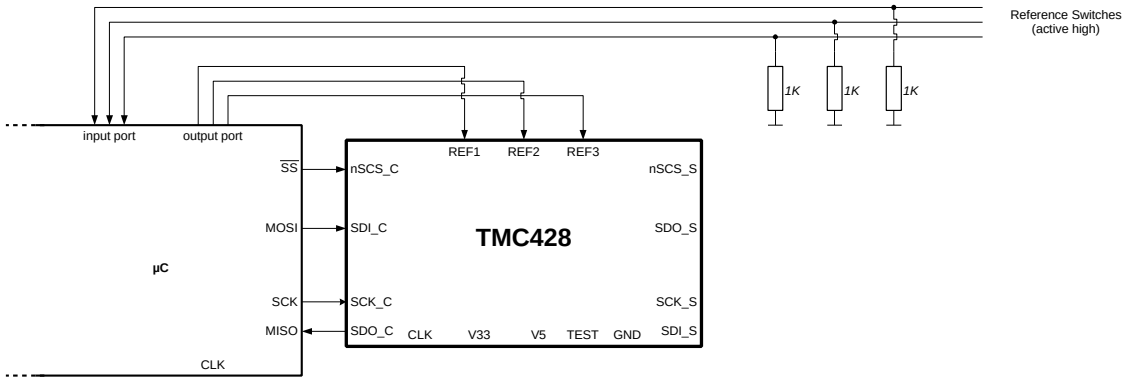



Figure 14 - Reference Switch Soft Control for Exact Simultanous Stepper Motor Start

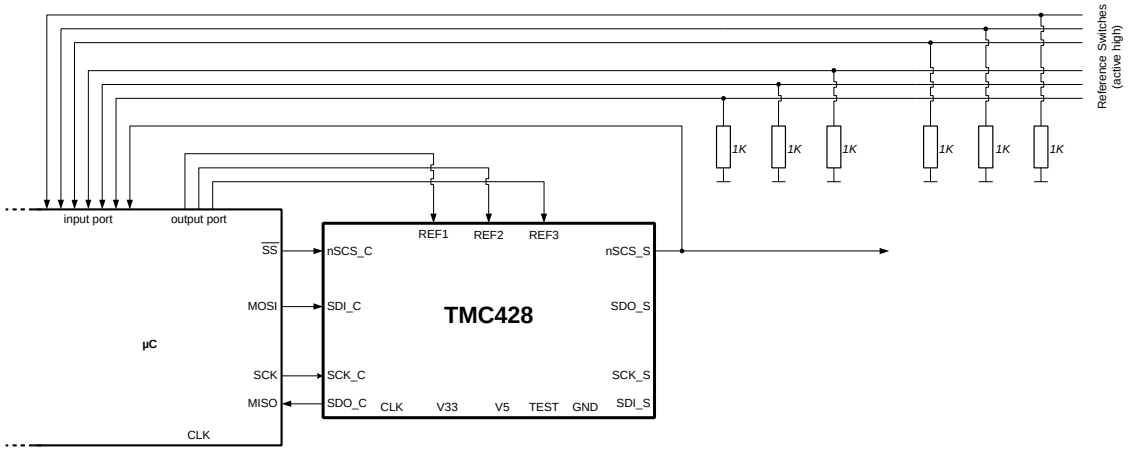



Figure 15 - Reference Switch Soft De-Multiplexing and Soft Control for six Reference Switches

## RAM Address Partitioning and Data Organization

The on-chip RAM capacity is 128 x 6 bit. These 128 on-chip RAM cells of 6 bit width are addressed via 64 addresses of 2 x 6 bit (see Table 12). So, from the point of view of addressing the on-chip RAM via datagrams, the address space enfolds 64 addresses of 24 bit wide data, where only 2 x 6 = 12 bits are relevant. These 64 addresses are partitioned– selected by the RRS (Register RAM Select, datagram bit 31)– into two address ranges of 32 addresses. The registers of the TMC428 are addressed with RRS='0'. The on-chip RAM is addressed with RRS='1'. The 64 on-chip RAM addresses are partitioned into two separated ranges by the most significant address bit of the datagram (bit 30).

The first 32 addresses are proposed for the configuration of the serial stepper motor driver chain. Each of these 32 addresses stores two configuration words, composed of the so called NxM (Next Motor) bit together with the 5 bit wide primary signal code. While sending a datagram, the primary signal code words are read internally beginning with the first address of the driver chain datagram configuration memory range. Each primary signal code word selects a signal proposed by the micro step unit. If the NxM bit is '1' an internal stepper motor addressing counter is incremented. If this internal counter is equivalent to the lsmd (last stepper motor driver) parameter, the datagram transmission is finished and the counter is preset to %00 for the next datagram transmission to the stepper motor driver chain.

The second 32 addresses are proposed to store the micro step table, which might be a quarter sine wave period as a basic approach or the quarter period of an periodic function optimized for micro stepping of a given stepper motor type. Different stepper motors may step with different micro step resolutions, but the micro step look up table (LUT) is the same for all stepper motors controlled by one TMC428. Any quarter wave period stored in the micro step table is expanded automatically to a full period wave together with its 90° phase shifted wave.

| 32 bit DATAGRAM send from a µC to the TMC428 via pin SDI_C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | data @ odd RAM addresses | | | | | | | | data @ even RAM addresses | | | |
| 1 | 0 | 32 x (2x6 bit) driver chain datagram configuration range | | | | | | | | | | | | | | | | NxM_1 | signal_codes | | | | | | | NxM_0 | signal_codes | | | |
| 1 | 1 | 32 x (2x6 bit) quarter period sine wave LUT range | | | | | | | | | | | | | | | | quarter sine wave values (amplitude) | | | | | | | | quarter sine wave values (amplitude) | | | |

Table 12 - Partitioning of the On-Chip RAM address space

## Stepper Motor Driver Datagram Configuration

Different control signals are required to drive 2-phase stepper motors. What kind of signals are required, depends on the choice of stepping mode– full step, half step, micro step – and on additional options proposed by the used stepper motor driver chips. So, the TMC428 primarily provides a full set of control signals individually for each of the up to three stepper 2-phase stepper motors respectively stepper motor driver chips of the daisy chain. Mnemonics for primary signal codes are given in Table 13.

The control signals for each of the two coils of a 2-phase stepper motor are 6 bits for a DAC controlling the current of a coil, a phase polarity bit, and a fast decay bit for those stepper motor driver chips with a fast decay feature for the coil current. These signals are available individually for each coil (COIL A and COIL B). Fixed configuration bits– for example to select a current range of a driver –are proposed (named Zero and One). Additionally, step and direction bits are provided. One unique 5 bit code word– named primary signal code –is assigned to each primary control signal (see Table 13).

| MNEMONIC | PRIMARY SIGNAL CODE | | FUNCTION | |
|---|---|---|---|---|
| | hex | bin | | |
| DAC_A_0 | $00 | %00000 | DAC A, bit 0 (LSB) | COIL A |
| DAC_A_1 | $01 | %00001 | DAC A, bit 1 | |
| DAC_A_2 | $02 | %00010 | DAC A, bit 2 | |
| DAC_A_3 | $03 | %00011 | DAC A, bit 3 | |
| DAC_A_4 | $04 | %00100 | DAC A, bit 4 | |
| DAC_A_5 | $05 | %00101 | DAC A, bit 5 (MSB) | |
| PH_A | $06 | %00110 | phase polarity bit A | |
| FD_A | $07 | %00111 | fast decay bit A | |
| DAC_B_0 | $08 | %01000 | DAC B, bit 0 (LSB) | COIL B |
| DAC_B_1 | $09 | %01001 | DAC B, bit 1 | |
| DAC_B_2 | $0A | %01010 | DAC B, bit 2 | |
| DAC_B_3 | $0B | %01011 | DAC B, bit 3 | |
| DAC_B_4 | $0C | %01100 | DAC B, bit 4 | |
| DAC_B_5 | $0D | %01101 | DAC B, bit 5 (MSB) | |
| PH_B | $0E | %01110 | phase polarity bit B | |
| FD_B | $0F | %01111 | fast decay bit B | |
| Zero | $10 | %10000 | constant ‘0’ | |
| One | $11 | %10001 | constant ‘1’ | |
| Direction | $12 | %10010 | 0 : up / 1 : down resp. counter clockwise / clockwise | |
| Step | $13 | %10011 | step bit for step/direction control of drivers | |
| *UNUSED (these codes may be used for future devices)* | $14 | %10100 | ‘1’ for TMC428-I, TMC428-A, TMC428-PI24 | |
| | $15 | %10101 | | |
| | $16 | %10110 | | |
| | $17 | %10111 | | |
| | $18 | %11000 | | |
| | $19 | %11001 | | |
| | $1A | %11010 | | |
| | $1B | %11011 | | |
| | $1C | %11100 | | |
| | $1D | %11101 | | |
| | $1E | %11110 | | |
| | $1F | %11111 | | |

Table 13 - Primary Signal Codes

The micro step unit (including sequencer) provides the full set of control signals for three stepper motor driver chips. A subset out of these control signals is selected by the stepper motor driver datagram configuration, which is stored within the first 32 addresses– but up to 64 values –of the on-chip RAM (see Table 12, page 30). The stepper motor drivers are organized in a daisy chain. So the addressing of the stepper motor driver chips within the daisy chain is by its position.

As mentioned before, the TMC428 sends datagrams to the stepper motor driver chain on demand. To guarantee the integrity of each datagram send to the stepper motor driver chain, the status of all primary control signals are buffered internally before sending. Afterwards, the transmission starts with selection of the buffered primary control signals of the first motor (smda=%00) by reading the first primary signal code word (even data word at on-chip RAM address %00000) from on-chip configuration RAM area. The primary signal codes select the primary signals provided for the first stepper motor. The first stepper motor is addressed until the NxM (next motor) bit is read from on-chip configuration RAM. The stepper motor driver address is incremented with each NxM='1' if the current stepper motor driver address is below the parameter lsmd (last stepper motor driver). If the stepper motor driver address is equivalent to the lsmd parameter, a NxM='1' indicates the completion of the transmission. With that, the stepper motor driver address counter of the serial interface is reinitialized to %00 and the unit waits for the next transmission request.

So, the order of primary signal codes in the on-chip RAM configuration area determines the order of datagram bits for the stepper motor driver chain, whereas the prefixed NxM bit determines the stepper motor driver positions. If no NxM bit with a value of '1' is stored within the on-chip RAM, the TMC428 will send endless. So, the on-chip RAM has to be configured first. After power-on reset, the registers of the TMC428 are initialized, so that no transmission of datagrams to the stepper motor driver chain is required. Access to on-chip RAM is always possible, also during transmission of datagrams to the driver chain.

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

All registers are initialized by the automatic power-on reset. The registers are initialized, that stepper motors are at rest. The on-chip RAM is not initialized by the power-on reset. Writing to registers may involve action of the stepper motor units initiated by the TMC428 resulting in sending datagrams to the stepper motor driver chain. Those datagrams have a random power-on configuration of the on-chip-RAM. So, before writing any motion control register– respectively position or velocity –the on-chip RAM must be initialized first.

### An Example of a Stepper Motor Driver Datagram Configuration

The following example demonstrates, how to configure the datagram and shows what has to be stored within the on-chip RAM to represent the desired configuration. In the example a driver chain of three stepper motor drivers is proposed. The first stepper motor driver has a serial interface of 12 bits length, the second driver has a length of 8 bits, and the last driver has a length of 10 bits (see Table 14). The corresponding content of the configuration on-chip RAM is outlined in Table 15. The datagrams, to be send from the micro controller to the TMC428, to store that configuration are outlined in Table 16.

| | example datagram configuration for sending from TMC428 to stepper motor driver daisy chain | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| position within datagram | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| driver | driver # 1 (SMDA=%00) | | | | | | | | | | | | driver # 2 (SMDA=%01) | | | | | | | | driver #3 (SMDA=%10) | | | | | | | | | |
| NxM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| mnemonic of primary signal | PH_B | DAC_B_4 | DAC_B_3 | DAC_B_2 | DAC_B_1 | DAC_B_0 | PH_A | DAC_A_4 | DAC_A_3 | DAC_A_2 | DAC_A_1 | DAC_A_0 | Zero | Zero | Zero | One | One | One | Step | Direction | FD_B | FD_A | PH_B | DAC_B_2 | DAC_B_1 | DAC_B_0 | PH_A | DAC_A_2 | DAC_A_1 | DAC_A_0 |

Table 14 - Datagram Example

| on-chip RAM configuration for the example | | | | | | |
|---|---|---|---|---|---|---|
| address | data | NxM & signal code word | | mnemonic | position | |
| | | NxM | signal code word | | | |
| $00 | $0E | 0 | $0E | PH_B | 0 | driver#1 |
| $01 | $0C | 0 | $0C | DAC_B_4 | 1 | |
| $02 | $0B | 0 | $0B | DAC_B_3 | 2 | |
| $03 | $0A | 0 | $0A | DAC_B_2 | 3 | |
| $04 | $09 | 0 | $09 | DAC_B_1 | 4 | |
| $05 | $08 | 0 | $08 | DAC_B_0 | 5 | |
| $06 | $06 | 0 | $06 | PH_A | 6 | |
| $07 | $04 | 0 | $04 | DAC_A_4 | 7 | |
| $08 | $03 | 0 | $03 | DAC_A_3 | 8 | |
| $09 | $02 | 0 | $02 | DAC_A_2 | 9 | |
| $0A | $01 | 0 | $01 | DAC_A_1 | 10 | |
| $0B | $20 | 1 | $00 | DAC_A_0 | 11 | |
| $0C | $10 | 0 | $10 | Zero | 12 | driver#2 |
| $0D | $10 | 0 | $10 | Zero | 13 | |
| $0E | $10 | 0 | $10 | Zero | 14 | |
| $0F | $11 | 0 | $11 | One | 15 | |
| $10 | $11 | 0 | $11 | One | 16 | |
| $11 | $11 | 0 | $11 | One | 17 | |
| $12 | $13 | 0 | $13 | Step | 18 | |
| $13 | $32 | 1 | $12 | Direction | 19 | |
| $14 | $0F | 0 | $0F | FD_B | 20 | driver#3 |
| $15 | $07 | 0 | $07 | FD_A | 21 | |
| $16 | $0E | 0 | $0E | PH_B | 22 | |
| $17 | $0A | 0 | $0A | DAC_B_2 | 23 | |
| $18 | $09 | 0 | $09 | DAC_B_1 | 24 | |
| $19 | $08 | 0 | $08 | DAC_B_0 | 25 | |
| $1A | $06 | 0 | $06 | PH_A | 26 | |
| $1B | $02 | 0 | $02 | DAC_A_2 | 27 | |
| $1C | $01 | 0 | $01 | DAC_A_1 | 28 | |
| $1D | $20 | 1 | $00 | DAC_A_0 | 29 | |
| With Ismd = %10 the (third) NxM bit at address $1D (position 29) finishes the datagram transmission | | | | | | |

Table 15 - RAM Contents for the Datagram Example

| binary datagram specification | : | hexadecimal datagram |
|---|---|---|
| 10000000----------001100--001110 | : | 80000c0e |
| 10000010----------001010--001011 | : | 82000a0b |
| 10000100----------001000--001001 | : | 84000809 |
| 10000110----------000100--n00110 | : | 86000406 |
| 10001000----------000010--000011 | : | 88000203 |
| 10001010----------100000--000001 | : | 8a002001 |
| 10001100----------010000--010000 | : | 8c001010 |
| 10001110----------010001--010000 | : | 8e001110 |
| 10010000----------010001--010001 | : | 90001111 |
| 10010010----------110010--010011 | : | 92003213 |
| 10010100----------000111--001111 | : | 9400070f |
| 10010110----------001010--001110 | : | 96000a0e |
| 10011000----------001000--001001 | : | 98000809 |
| 10011010----------000010--000110 | : | 9a000206 |
| 10011100----------100000--000001 | : | 9c002001 |

Table 16 – Configuration Datagram Sequence Specification for the Datagram Example

| % binary datagram representation | $ hexadecimal datagram |
|---|---|
| % 10 00000 0 000000 00 001100 00 001110 | $ 80 00 0c 0e |
| % 10 00001 0 000000 00 001010 00 001011 | $ 82 00 0a 0b |
| % 10 00010 0 000000 00 001000 00 001001 | $ 84 00 08 09 |
| % 10 00011 0 000000 00 000100 00 000110 | $ 86 00 04 06 |
| % 10 00100 0 000000 00 000010 00 000011 | $ 88 00 02 03 |
| % 10 00101 0 000000 00 100000 00 000001 | $ 8a 00 20 01 |
| % 10 00110 0 000000 00 010000 00 010000 | $ 8c 00 10 10 |
| % 10 00111 0 000000 00 010001 00 010000 | $ 8e 00 11 10 |
| % 10 01000 0 000000 00 010001 00 010001 | $ 90 00 11 11 |
| % 10 01001 0 000000 00 110010 00 010011 | $ 92 00 32 13 |
| % 10 01010 0 000000 00 000111 00 001111 | $ 94 00 07 0f |
| % 10 01011 0 000000 00 001010 00 001110 | $ 96 00 0a 0e |
| % 10 01100 0 000000 00 001000 00 001001 | $ 98 00 08 09 |
| % 10 01101 0 000000 00 000010 00 000110 | $ 9a 00 02 06 |
| % 10 01110 0 000000 00 100000 00 000001 | $ 9c 00 20 01 |

Table 17 - Datagrams Specified in Table 16 (with '-' (don't cares) replaced by '0')

## Initialization of the Micro Step Look-Up-Table

The TMC428 provides a look-up-table (LUT) of 64 values of 6 bit for micro stepping. The micro step LUT can be adapted by storing an arbitrary quarter period of a periodic function to individual stepper motor characteristics. It is common to uses one period of a sine wave function for micro stepping. With that function, the current of one phase is controlled with the sine function where the other phase is controlled with the cosine function.

To initialize the LUT for micro stepping one simply has to load a quarter sine wave period into the micro step LUT within the on-chip RAM area. Two successive values of the sine wave function are included in one datagram similar to the primary signal code words for the stepper motor driver chain configuration. The TMC428 automatically expands the quarter sine wave period to a full sine and cosine function. The necessary data values y(i) to represent a ¼ sine wave period for the micro step LUT are defined by

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

where the conditional replacement y(i) := 63 for y(i) > 63 has to be done. With this, one gets the following 64 values y(i) = { 0, 2, 3, 5, 6, 8, 9, 11, 12, 14, 16, 17, 19, 20, 22, 23, 24, 26, 27, 29, 30, 32, 33, 34, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 56, 57, 58, 59, 59, 60, 60, 61, 61, 62, 62, 62, 63, 63, 63, *64*, *64*, *64*, *64*, *64* }.

| 32 bit DATAGRAM send from a µC to the TMC428 via pin SDI_C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RRS | ADDRESS | | | | | | RW | DATA | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | $(x)_{10}$ | data @ odd RAM addresses | | | | | | $(x)_{10}$ | data @ even RAM addresses | | | | | | | | | | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 0 | 0 | 0 | 1 | | | 5 | 0 | 0 | 0 | 1 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | |
| | | 0 | 0 | 0 | 1 | 0 | | | 8 | 0 | 0 | 1 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | |
| | | 0 | 0 | 0 | 1 | 1 | | | 11 | 0 | 0 | 1 | 0 | 1 | 1 | 9 | 0 | 0 | 1 | 0 | 0 | 1 | | | | | | | | | |
| | | 0 | 0 | 1 | 0 | 0 | | | 14 | 0 | 0 | 1 | 1 | 1 | 0 | 12 | 0 | 0 | 1 | 1 | 0 | 0 | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | 1 | 1 | 0 | 1 | 1 | | | 62 | 1 | 1 | 1 | 1 | 1 | 0 | 62 | 1 | 1 | 1 | 1 | 1 | 0 | | | | | | | | | |
| | | 1 | 1 | 1 | 0 | 0 | | | 63 | 1 | 1 | 1 | 1 | 1 | 1 | 63 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |
| | | 1 | 1 | 1 | 0 | 1 | | | *63* | 1 | 1 | 1 | 1 | 1 | 1 | 63 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |
| | | 1 | 1 | 1 | 1 | 0 | | | *63* | 1 | 1 | 1 | 1 | 1 | 1 | *63* | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |
| | | 1 | 1 | 1 | 1 | 1 | | | *63* | 1 | 1 | 1 | 1 | 1 | 1 | *63* | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | |

Table 18 - scheme of ¼ sine wave period with 6 bit resolution and 64 ( 32 x 2 ) values

The last five values (which are calculated to be 64) have to be replaced by 63. With this replacement one finally gets y(i) = { 0, 2, 3, 5, 6, 8, 9, 11, 12, 14, 16, 17, 19, 20, 22, 23, 24, 26, 27, 29, 30, 32, 33, 34, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 56, 57, 58, 59, 59, 60, 60, 61, 61, 62, 62, 62, 63, 63, 63, *63*, *63*, *63*, *63*, *63* }. These 64 values– representing a quarter sine period in the interval [ 0 ... π/4 [ which is expanded automatically by the TMC428 to a full sine cosine period –have to be send to the on-chip RAM of the TMC428 by 32 datagrams:

| % binary representation of the datagram | decimal represented pair of values (separated by & character) | $ hexadecimal representation |
|---|---|---|
| % 11 00000 0 00000000 00 000010 00 000000 | 2 & 0 | $ C0 00 02 00 |
| % 11 00001 0 00000000 00 000101 00 000011 | 5 & 3 | $ C2 00 05 03 |
| % 11 00010 0 00000000 00 001000 00 000110 | 8 & 6 | $ C4 00 08 06 |
| % 11 00011 0 00000000 00 001011 00 001001 | 11 & 9 | $ C6 00 0B 09 |
| % 11 00100 0 00000000 00 001110 00 001100 | 14 & 12 | $ C8 00 0E 0C |
| % 11 00101 0 00000000 00 010001 00 010000 | 17 & 16 | $ CA 00 11 10 |
| % 11 00110 0 00000000 00 010100 00 010011 | 20 & 19 | $ CC 00 14 13 |
| % 11 00111 0 00000000 00 010111 00 010110 | 23 & 22 | $ CE 00 17 16 |
| % 11 01000 0 00000000 00 011010 00 011000 | 26 & 24 | $ D0 00 1A 18 |
| % 11 01001 0 00000000 00 011101 00 011011 | 29 & 27 | $ D2 00 1D 1B |
| % 11 01010 0 00000000 00 100000 00 011110 | 32 & 30 | $ D4 00 20 1E |
| % 11 01011 0 00000000 00 100010 00 100001 | 34 & 33 | $ D6 00 22 21 |
| % 11 01100 0 00000000 00 100101 00 100100 | 37 & 36 | $ D8 00 25 24 |
| % 11 01101 0 00000000 00 100111 00 100110 | 39 & 38 | $ DA 00 27 26 |
| % 11 01110 0 00000000 00 101010 00 101001 | 42 & 41 | $ DC 00 2A 29 |
| % 11 01111 0 00000000 00 101100 00 101011 | 44 & 43 | $ DE 00 1C 1B |
| % 11 10000 0 00000000 00 101110 00 101101 | 46 & 45 | $ E0 00 2E 2D |
| % 11 10001 0 00000000 00 110000 00 101111 | 48 & 47 | $ E2 00 30 2F |
| % 11 10010 0 00000000 00 110010 00 110001 | 50 & 49 | $ E4 00 32 31 |
| % 11 10011 0 00000000 00 110100 00 110011 | 52 & 51 | $ E6 00 34 33 |
| % 11 10100 0 00000000 00 110110 00 110101 | 54 & 53 | $ E8 00 36 35 |
| % 11 10101 0 00000000 00 111000 00 110111 | 56 & 55 | $ EA 00 38 37 |
| % 11 10110 0 00000000 00 111001 00 111000 | 57 & 56 | $ EC 00 39 38 |
| % 11 10111 0 00000000 00 111011 00 111010 | 59 & 58 | $ EE 00 3B 3A |
| % 11 11000 0 00000000 00 111100 00 111011 | 60 & 59 | $ F0 00 3C 3B |
| % 11 11001 0 00000000 00 111101 00 111100 | 61 & 60 | $ F2 00 3D 3C |
| % 11 11010 0 00000000 00 111110 00 111101 | 62 & 61 | $ F4 00 3E 3D |
| % 11 11011 0 00000000 00 111110 00 111110 | 62 & 62 | $ F6 00 3E 3E |
| % 11 11100 0 00000000 00 111111 00 111111 | 63 & 63 | $ F8 00 3F 3F |
| % 11 11101 0 00000000 00 111111 00 111111 | *63* & 63 | $ FA 00 3F 3F |
| % 11 11110 0 00000000 00 111111 00 111111 | *63* & *63* | $ FC 00 3F 3F |
| % 11 11111 0 00000000 00 111111 00 111111 | *63* & *63* | $ FE 00 3F 3F |

Table 19 - Datagrams for Initialization of a Quarter Sine Wave Period Micro Step Look-Up-Table

These 32 datagrams (Table 19) are sufficient for all programmable micro step resolutions. If micro stepping is proposed for at least one stepper motor, these 32 datagrams have to be send once to the TMC428 for initialization of the micro step table after power-on reset. The initialization of the micro step look-up-table is not necessary, if full stepping is used for *all* stepper motors. The On-Chip RAM is not initialized during power-on reset. So, the full initialization of the whole micro step look-up-table is recommended to avoid trouble caused by missing look-up table entries. Additionally, a fully initialized micro step look-up-table allows the selection of individual micro step resolutions for different stepper motors.

Micro Step Enhancement

Sine cosine micro stepping is not sufficient for all types of real stepper motors not even for those stepper motors labeled as optimized for micro stepping operation. A periodic trapezoidal or triangular function similar to a sine function or a superposition of these function as a replacement of the pure sine wave function (Figure 16) is a better choice for different types of stepper motors. Taking the physics of stepper motors into account, the choice of the function for micro stepping can be simply determined by a single shape parameter $s$ as explained below. The programmability of the micro step look-up table of the TMC428 also during operation– on-the-fly alteration of parameters –provides a simple and effective facility to attune micro stepping to a given type of two-phase stepper motor. Enhanced micro stepping requires accurate current control. So, stepper motor driver chips with enabled and well tuned fast decay operational mode– as our TMC288 / TMC289 smart power drivers provide –are necessary to be used.

Non-Linearities resulting from magnetic field configuration determined by shapes of pole shoes, ferromagnetic characteristics, and other stepper motor characteristics effect non-linearity in micro stepping of real stepper motors. The non-linearity of micro stepping causes micro step positioning displacements, vibrations and noise, which can be reduced dramatically with an adapted micro step table.

Nevertheless sine cosine micro stepping is a good first order approach for microstepping. The micro step enhancement possible with the TMC428 bases on replacement of the look-up table initialization function sin(φ ) used for sine cosine micro stepping by a function with the shape parameter $s$. A quarter sine wave period is the basic approach for initialization of the micro step look up table . A quarter of a trapezoidal function or a quarter of a triangular function is chosen depending on the shape parameter $s$ for a given stepper motor type.

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

The look-up table ($f(\varphi)$ ) of the TMC428 enfolds a quarter period ($0 = \varphi < \pi/2$) only. This quarter period is expanded to a full period ($0 = \varphi < 2\pi$ ) and the phase shifted companion function value ($f(\varphi - \pi/2)$ ) is added automatically by the TMC428 during operation. So, for reach function value ($f(\varphi)$ ) one automatically gets a pair of function values {$f(\varphi)$; $f(\varphi - \pi/2)$} respectively {sin(φ ); cos(φ )}. This automatic expansion of the TMC428– primary proposed for sine cosine microstepping ($f(\varphi) = \sin(\varphi)$ ) –also works fine with other micro step controlling functions $f_s$.

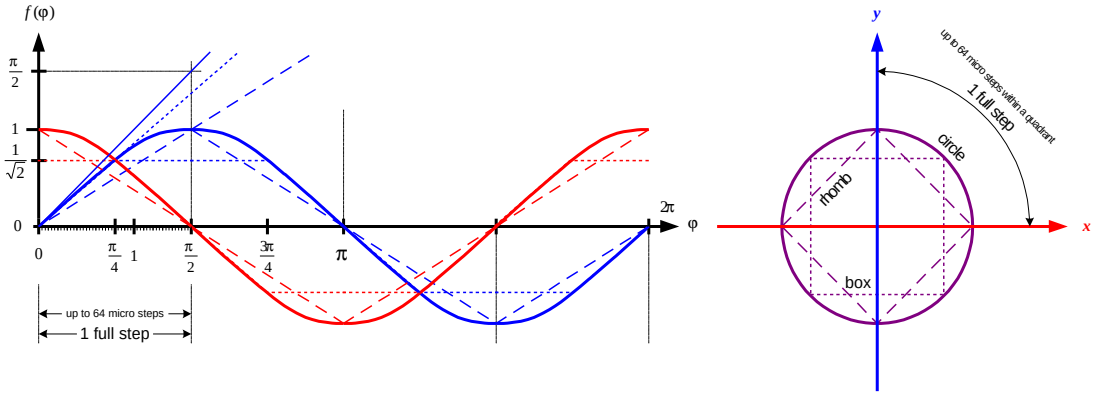



Figure 16 - micro step enhancemant by introduction of a shape function $f_s(\varphi)$

The shape parameter $s$ selects one of three functions $f_{box}(f)$, $f_{circle}(f)$, $f_{rhomb}(f)$ respectively a superposition of two of them. The shape parameter $s = 0$ selects the function $f_{circle}(f)$ which is the sine function $\sin(\varphi)$ as used for sine cosine micro stepping. With this, one gets the unit circle ($r$=1.0) by transformation to cartesian coordinates {$y = \sin(\varphi)$; $x = \cos(\varphi)$} as outlined in Figure 16, a shape parameter $s$ = +1.0 results in a box, and a shape parameter $s$ = -1.0 results in a rhomb. Other values except those, result in something between box and circle respectively something between circle an rhomb.

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

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

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

$$f_{r\,\mathrm{hom}\,b}(\varphi) = \frac{2}{\pi} \cdot \varphi$$

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

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

So, the shape parameter $s$ selects the type of function and it also provides a continuous transition between circle and box respectively circle and rhomb. To estimate, what function would be best for a given type of stepper motor, one can try micro stepping based on different shape parameters $s$ by downloading different micro step tables on-the-fly into the TMC428 during motion of a stepper motor. For calculation of data for the micro step look-up table of the TMC428, one has to replace $\varphi \rightarrow \varphi_i$ *ranging from 0 to* $\pi/2$ for the quarter period by

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

The amplitude of the shape function $f_s$ ($f_i$) has to be limited to the range of 0.0 to 1.0 respectively to the range of 0 to 63 for the on-chip RAM as described in the beginning of the micro stepping section.

### Partial look-up table initialization option

A partially initialized micro step table may be sufficient, if all stepper motors– except those driven in full step mode –are proposed to use the same micro step resolution constantly before a single micro step is processed. But with a partial initialized micro step look-up table, the micro step resolution *must not be changed* anyway. So, a partially initialized look-up table should be taken into account only, if it is a must because of small memory of proposed used micro controller. Instead of partial initialization of the look-up table of the TMC428, initialization the look-up table with a triangular function $f_{rhomb}(\varphi)$ would be a much better choice.

# Package Outlines and Dimensions

Shrink Small Outline Package with 16 Pins (SSOP16, 150 MIL ) of TMC428-I and TMC428-A

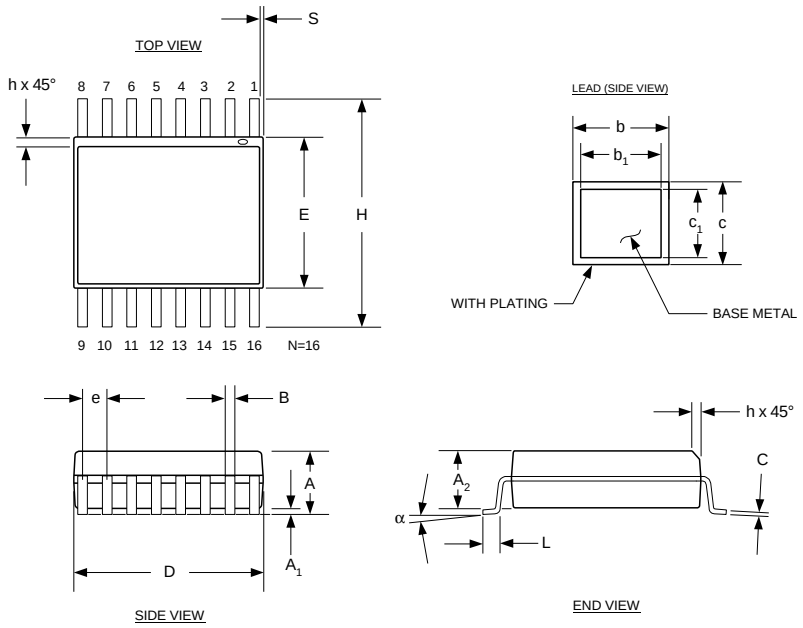



Figure 17 – Package Outline Drawing SSOP16, 150 MILS

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

Table 20 - Dimensions of Package SSOP16, 150 MILS (Note: BSC ≈ Best Case)

Small Outline Package with 24 Pins (SOIC24) for TMC428-PI24

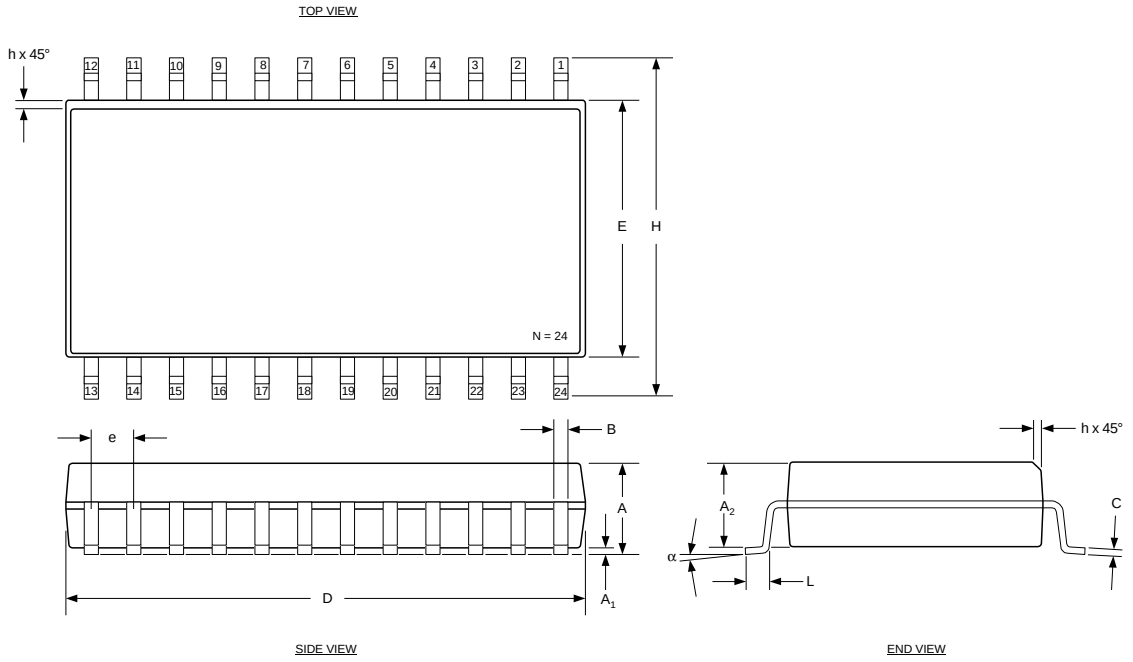



Figure 18 - Package Outline Drawing SOIC24, 300 MILS

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

Table 21 - Dimensions of Package SOIC24, 300 MILS (Note: BSC ≈ Best Case)

## Marking

| | |
|---|---|
| Product Name | TMC428-I |
| Product ID (at top and bottom) | 56563A |
| Package Name | SSOP16 – 150 MILS |
| Internal Package Name | SSOP16_A |
| Date Code | YYWW (year YY and week WW) |
| Lot Number (at bottom only) | XXX |
| Logo | No |
| Real Size (see note below) |  |
| Zoomed Size | **TMC428-I**<br>**Trinamic**<br>**56563A** |

| | |
|---|---|
| Product Name | TMC428-A |
| Product ID (at top and bottom) | 56563A |
| Package Name | SSOP16 – 150 MILS |
| Internal Package Name | SSOP16_A |
| Date Code | YYWW (year YY and week WW) |
| Lot Number (at bottom only) | XXX |
| Logo | No |
| Real Size (see note below) |  |
| Zoomed Size | **TMC428-A**<br>**Trinamic**<br>**56563A** |

| | |
|---|---|
| Product Name | TMC424-PI24 |
| Product ID | 56563A |
| Package Name | SOIC 24 – 300 MILS |
| Internal Package Name | SO024_B |
| Date Code | YYWW (year YY and week WW) |
| Lot Number | XXX |
| Logo | Yes |
| Real Size (see note below) |  |
| Zoomed Size |  **TMC 428-PI24**<br>**Trinamic ®**<br>**56563A**<br>**XXX YYWW** |

Note: Proposed to be of ''Real Size'' if printed with scale of 100% on paper of DIN-A4 format – but the printed size may differ depending on the printer.

# Characteristics

| Symbol | Parameter | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| VDD3 | DC Supply Voltage | Voltage at Pin V33 in 3.3V mode | -0.3 | 3.6 | V |
| VI3 | DC Input Voltage, 3.3 V I/Os | | -0.3 | VDD3 + 0.3 | V |
| VO3 | DC Output Voltage, 3.3 V I/Os | | -0.3 | VDD3 + 0.3 | V |
| VDD5 | DC Supply Voltage | Voltage at Pin V5 | -0.3 | 5.5 | V |
| VI5 | DC Input Voltage, 5V I/Os | Continuous DC Voltage | -0.3 | VDD5 + 0.3, 5.5 max | V |
| VO5 | DC Output Voltage, 5V I/Os | Continuous DC Voltage | -0.3 | VDD5 + 0.3, 5.5 max | V |
| VESD | ESD Voltage at any I/O Pin | Human Body Model according to MIL-STD-883, with $R_c$ = 1 - 10 MΩ, $R_D$ = 1.5 KΩ, and $C_s$ = 100 pF. | | ±2000 | V |
| VESD5 | ESD Voltage at Pin V5 | | | | V |
| VESD33 | ESD Voltage at Pin V33 | | | | V |
| IMAXIO | Maximum Current into any Input or Output Terminal | One pin at a time | | 10 | mA |
| TEMPIND | Operating Free Air Temperature Range | Industrial | -40 | +85 | ℃ |
| TEMPAUT | Operating Free Air Temperature Range | Automotive | -40 | +125 | ℃ |
| TSG | Storage Temperature | | -60 | +150 | ℃ |

Table 22 - Absolute Maximum Ratings

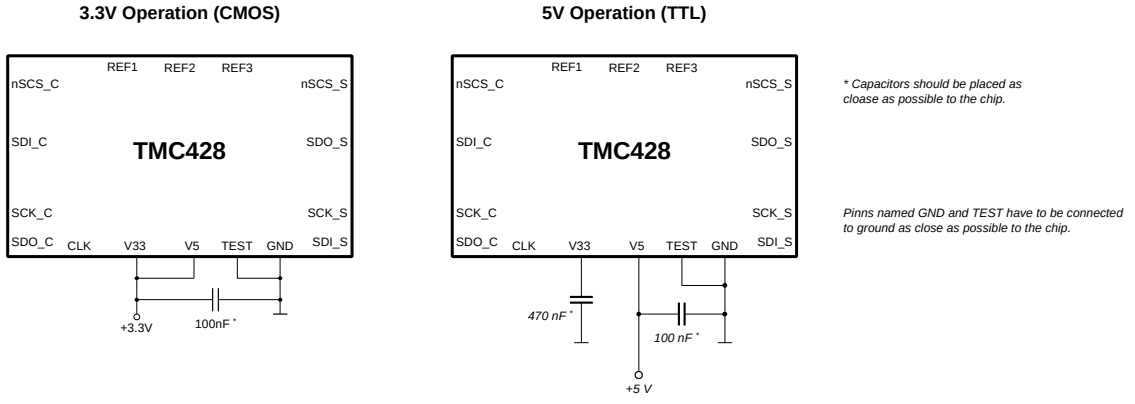



Figure 19 - 3.3V Operation (CMOS) vs. 5V Operation (TTL)

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| ILC | Input Leakage Current | | | | 10 | µA |
| CIN | Input Capacitance | | | 7 | | pF |

Table 23 – Characteristics

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| VDD3 | DC Supply Voltage | | 3.0 | 3.3 | 3.6 | V |
| VI3 | DC Input Voltage | | 0 | | VDD3 | V |
| VIL3 | Low Level Input Voltage | Pin TEST only | 0 | | 0.3 x VDD3 | V |
| VIH3 | High Level Input Voltage | Pin TEST only | 0.7 x VDD3 | | VDD3 + 0.3 | V |
| VLTH3 | Low Level Input Voltage Threshold | All Inputs except TEST | 1.025 | | 1.126 | V |
| VHTH3 | High Level Input Voltage Threshold | All Inputs except TEST | 1.675 | | 1.725 | V |
| VHYS3 | Schmitt-Trigger Hysteresis | | 0.550 | | 0.700 | V |
| VOL3 | Low Level Output Voltage | IOL = 0.3 mA | | | 0.1 | V |
| VOH3 | High Level Output Voltage | IOH = 0.3 mA | VDD3 – 0.1 | | | V |
| VOL3 | Low Level Output Voltage | IOL = 2 mA | | | 0.4 | |
| VOH3 | High Level Output Voltage | IOH = 2 mA | VDD3 – 0.4 | | | V |

Table 24 - DC Characteristics – 3.3V Supply Mode

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| VDD5 | DC Supply Voltage | | 4.5 | 5 | 5.5 | V |
| VI5 | DC Input Voltage | | 0 | | VDD5 | V |
| VIL5 | Low Level Input Voltage | Pin TEST only | 0 | | 0.3 x VDD5 | V |
| VIH5 | High Level Input Voltage | Pin TEST only | 0.7 x VDD5 | | VDD5 + 0.3 | V |
| VLTH5 | Low Level Input Voltage Threshold | All Inputs except TEST | 1.025 | | 1.126 | V |
| VHTH5 | High Level Input Voltage Threshold | All Inputs except TEST | 1.675 | | 1.725 | V |
| VHYS5 | Schmitt-Trigger Hysteresis | | 0.550 | | 0.700 | V |
| VOL5 | Low Level Output Voltage | IOL = 0.3 mA | | | 0.1 | V |
| VOH5 | High Level Output Voltage | IOH = 0.3 mA | VDD5 – 0.1 | | | V |
| VOL5 | Low Level Output Voltage | IOL = 4 mA | | | 0.4 | |
| VOH5 | High Level Output Voltage | IOH = 4 mA | VDD5 – 0.4 | | | V |

Table 25 - DC Characteristics – 5V Supply Mode

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| ISC16MHZ | Supply Current | f = 16 MHz at Tc=25°C | | | | mA |
| ISC4MHZ | Supply Current | f = 4 MHz at Tc=25°C | | 1,25 | | mA |
| ICS0MHZ | Supply Current | f = 0 MHz at Tc=25°C | | | | µA |
| IPDN25C | Power Down Current | Power Down Mode at Tc=25°C, 5V Supply | | 70 | | µA |
| IPDN85C | Power Down Current | Power Down Mode at Tc=85°C, 5V Supply | | | | µA |
| IPDN125C | Power Down Current | Power Down Mode at Tc=125°C, 5V Supply | | | | µA |

Table 26 - Power Dissipation

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| fCLK | Operation Frequency | fCLK = 1 / tCLK | 0 | 4 | 16 | MHz |
| tCLK | Clock Period | Raising Edge to Raising Edge of CLK | 62,5 | | ∞ | ns |
| tCLK_L | Clock Time Low | | 25 | | ∞ | ns |
| tCLK_H | Clock Time High | | 25 | | ∞ | ns |
| tRISE_I | Input Signal Rise Time | 10% to 90% except TEST pin | | | ∞ | ns |
| tFALL_I | Input Signal Fall Time | 90% to 10% except TEST pin | | | ∞ | ns |
| tRISE_O | Output Signal Rise Time | 10% to 90% | | 2 | | ns |
| tFALL_O | Output Signal Fall Time | 90% to 10% | | 4 | | ns |
| tSU | Setup Time | relative to falling clock edge at CLK | 1 | | | ns |
| tHD | Hold Time | relative to falling clock edge at CLK | 1 | | | ns |
| tPD | Propagation Delay Time | 50% of rising edge of the clock CLK to the 50% of the output | 1 | 5 | | ns |

Table 27 - General Timing Parameters

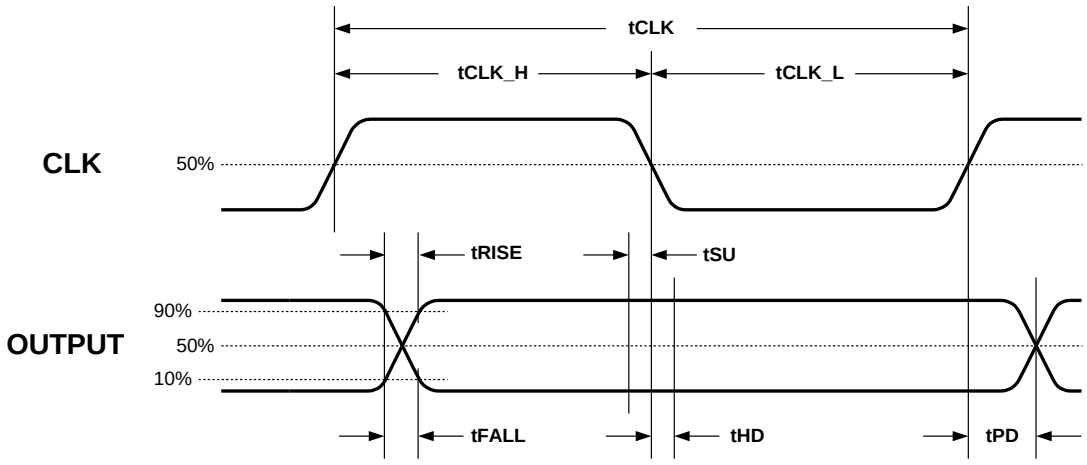



Figure 20 - General Timing Parameters

## On-Chip Voltage Regulator

The on-chip voltage regulator delivers a 3.3V supply for the core under a 30 mA load with a 4.5 minimum input voltage. An external 470 nF ceramic capacitor has to be connected between the V33 pin (see Figure 19, page 42) and ground, with connections as short as possible. The regulator is internally stable. The output capacitor is only needed for power supply filtering, not for feedback loop stabilization. The ripple voltage VRIPPLE on pin V33 is approximately

$$VRIPPLE \approx I\_average * tCLK / C\_load$$

where I_average is the average current, tCLK is the clock period, C_load is capacity of the capacitor. A capacity between 33 nF to 470 nF (perhaps 100 nF) are sufficient, depending on the ripple requirements. The technology is more important: x7r ceramic capacitors shall be preferred, in parallel with 470 pF to 1 pF c0g. Tantalum capacitors should be avoided, because of their poor high frequency behavior.

Additionally, an external 100 nF ceramic capacitor (CBLOCK) has to be connected between pin V5 and ground– with connections as short as possible –in 5V operational mode. In 3.3V operational mode an external 100 nF ceramic capacitor (see Figure 19) is necessary only between pin V33 and ground, with connections as short as possible.

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| TRANGEREG | Temperature range | Industrial | -40 | | 85 | °C |
| VDD5REG | Supply voltage vdd5 | 5 V Operational Mode | 4.5 | 5 | 5.5 | V |
| CBLOCK | Block capacitor | 5 V Operational Mode, x7r ceramic capacitor | | 100 | | nF |
| VDD3REG | Supply voltage vdd3 | 3.3 V Operational Mode | 2.9 | 3.3 | 3.6 | V |
| ICCNLREG | Current consumption | no load | | 50 | 100 | µA |
| IREG | Output current on vdd3 | max. 10 mA internal load by core | | | 20 | mA |
| tSREG | Startup time | no external capacitor connected | | | 20 | µs |
| tSREGC | Startup time | C_load = 470 nF | | | 150 | µs |
| TDRFT | Temperature drift | | | | 300 | ppm / °C |
| VRIPPLE | Ripple on vdd3 | C_load = 470 nF, I_load = 10mA + 20 mA | | 100 | | mV |
| CREG | External capacitor | On V33 pin, x7r ceramic | | 470 | | nF |
| CREG_RANGE | External capacitor range | On V33 pin, x7r ceramic, necessary capacity depending on required VRIPPLE | 33 | 470 | | nF |
| COPTIONAL | Optional capacitor | Additional Parallel Capacitor, c0g ceramic | 1 | | 470 | pF |
| PSRRDC | power supply ripple rejection | DC | | 50 | | dB |

Table 28 - Characteristics of the on-chip Voltage Regulator

## Power-On-Reset

The TMC428 is equipped with a static and dynamic reset with internal hysteresis (see Figure 21). So, it performs an automatic reset during power-on. If the power supply voltage goes below a threshold, an automatic power on reset is performed also.

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| VDD | Power supply range | | 3.0 | 3.3 | 3.6 | V |
| Temp | Temperature | | -40 | 25 | 85 | ℃ |
| Vop | Reset on/off | | | | 0.59 | V |
| Voff | Reset off | | 1,58 | 2,13 | 2,85 | V |
| Von | Reset on | | 1.49 | 1.98 | 2.70 | V |
| tRESPOR | Reset time of on-chip power-on-reset | | 2.14 | 3.31 | 5.52 | µs |
| Pd | Power dissipation after Tres | | 22.2 | 33 | 55.6 | µA |
| Vop125C | Reset on/off | T = -40°C | | | | V |
| Voff125C | Reset off | T = -40°C | | | | V |
| Von125C | Reset on | T = -40°C | | | | V |
| tRESPOR125C | Reset time of on-chip power-on-reset | T = -40°C | | | | µs |
| Pd125C | Power dissipation after Tres | T = -40°C | | | | µA |
| Vop125C | Reset on/off | T = 125°C | | | | V |
| Voff125C | Reset off | T = 125°C | | | | V |
| Von125C | Reset on | T = 125°C | | | | V |
| tRESPOR125C | Reset time of on-chip power-on-reset | T = 125°C | | | | µs |
| Pd125C | Power dissipation after Tres | T = 125°C | | | | µA |

Table 29 - Characteristics of the on-chip Power-On-Reset

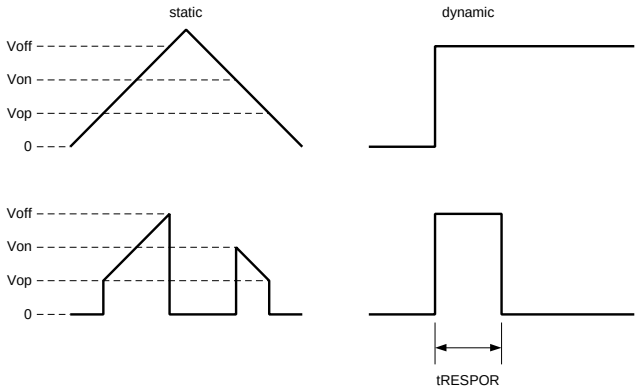



Figure 21 - Operating Principle of the power-on-reset

## Example for Calculation of p_mul and p_div for the TMC428

```
/* PROGRAM EXAMPLE 'pmulpdiv.c' : How to Calculate p_mul & p_div for the TMC428 */

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

void CalcPMulPDiv(int a_max, int ramp_div, int pulse_div,
                  int *p_mul, int *p_div, double *PIdeal, double *PBest)
{
  int   d, d_best, m, m_best;
  double q_ideal, q_best, q;

  q_ideal = a_max /  (pow(2, ramp_div-pulse_div)*128.0);
  q_best  =  -1;
  d_best  =   0;
  m_best  = 128;

  for(d=0; d<=13; d++)
  {
    m = (int) (q_ideal*pow(2, d+3));

    if(m>127 && m<256)
    {
      q = m / pow(2, d+3);

      if(fabs(q-q_best) < fabs(q_ideal-q_best) || q_best<0)
      {
        q_best = q;
        d_best = d;
        m_best = m;
      }
    }
  }

  *p_mul = m_best;
  *p_div = d_best;

  *PIdeal = q_ideal;
  *PBest = q_best;
}

int main(int argc, char **argv)
{
    int a_max=0, ramp_div=0, pulse_div=0, p_mul, p_div;
    double pideal, pbest;
    char **argp;

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

        if ( (!strcmp(*argv,"-a")) ) sscanf(*argp,"%d",&a_max);
        else if ( (!strcmp(*argv,"-r"))) sscanf(*argp,"%d",&ramp_div);
        else if ( (!strcmp(*argv,"-p"))) sscanf(*argp,"%d",&pulse_div);
      }
    }
    else
    {
      fprintf(stderr,"\n  USAGE : pmulpdiv -a <a_max> -r <ramp_div> -p <pulse_div>}\n"

                     "  EXAMPLE : pmulpdiv -a 865 -r 9 -p 7}\n");
      return 1;
    }

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

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

    printf("p_mul = %3.3d\np_div = %3d\n\np_ideal = %.15f\np_best = %.15f\n",
            p_mul, p_div, pideal, pbest);

    return 0;
}

/* ---------------------------------------------------------------------------- */
```

# Table of Figures



# Table of Tables



# Table of Contents

