Application note:

Extending the positioning range of the TMC5031, TMC5062, TMC5xxx to rotational movements or more than 32 bit

TRINAMIC® Motion Control GmbH & Co. KG
GERMANY

www.trinamic.com

The TMC5XXX family features a 32 bit motion range, which will fit most applications. However some applications may need a larger motion range due to extensive gearing. The same is valid for drives, which operate rotational, but may also be required to do a positioning within a rotation. This application note is meant to be a practical guideline for extending the positioning range.

1 Table of contents

2 Extending the Positioning Range

The TMC5XXX family has 32 bit registers for position information. These are the registers *X_TARGET* and *X_ACTUAL* available for each motor driver. When doing a positioning movement, the motion direction is determined by the sign of the 32 bit difference between the both values. This way, the maximum displacement per move is (2^31)-1, i.e. 2,147,483,647 steps, or 8,388,608 fullsteps at 256 microstepping, or 41,943 rotations of a 200 step motor or 11.6 hours of rotation of this motor at 1RPS.

2.1 Algorithm

The algorithm for position extension requires two additional registers within the microcontroller: A 64 (or 48) bit target position (*XTARGET64*) and a 64 bit actual position (*XACTUAL64*).

Starting a positioning sequence:

1. *XTARGET64* becomes updated with the with the new target position
2. The regular service procedure becomes executed

Regular service procedure:

This procedure becomes executed on a regular base. Taking into account maximum velocity, it needs to be executed only at a minimum rate of a few times per hour, or for example whenever the position value is to be used.

1. Each control cycle starts with reading *X_ACTUAL* from the TMC5XXX.

2. *X_ACTUAL* becomes checked for an underflow or overflow, by comparing to the lower 32 bits of the old value stored in *XACTUAL64.* In case of an underflow, the upper 32 bits of *XACTUAL64* become decremented by 1. In case of an overflow, they become incremented by one. The lower 32 bits of *XACTUAL64* become updated with the value read from *X_ACTUAL*.

3. This step is required in positioning mode, only: *XTARGET64* becomes compared to *XACTUAL64*. In case, the desired position difference is larger than 2^30, *X_TARGET* becomes set to *XACTUAL64* + 2^30, using the lower 32 bits of the result. In case the difference is smaller than -2^30, *X_TARGET* becomes set to *XACTUAL64* - 2^30 the same way. In case the difference is in between, *X_TARGET* becomes set to *XTARGET64* using the lower 24 bits.

As an additional extension, the position compare mechanism (if used) must be switched off in case the upper eight bits of *XACTUAL64* do not match the upper 32 bits of *XTARGET64*. When a position snapshot is triggered by a reference switch, the upper 32 bits of *XACTUAL64* need to be copied to the 64 bit copy of the snapshot register, too.

2.2 Conclusion

This way, the motion is executed in displacements of maximum 2^30. The displacement is kept with each execution of this service procedure, thus the motion will not end before the full displacement has been executed. The additional expense in processing time is extremely low, as the algorithm is not real time critical.

3 Rotational Movements

A rotational movement is possible by using the velocity mode of the motion controller. But, when a certain position in a rotation has to be addressed, the motion controller must be switched back to positioning mode. The actual position has to be modulo-divided by the number of microsteps per rotation in order to determine the optimum target position. In case the rotational movement has been active for a long time, any unknown overflow of the number range would lead to a loss of the position information within the rotation. Therefore it is important to keep track of the overflow or to increase the positioning range to a range which can never be overflown (e.g. 48 or 64 bit). The algorithm shown sums up the displacement generated by the overflows in order to avoid the necessity for 64 bit modulo division. Therefore, the number of fractional rotations per overflow is calculated and added / subtracted for each overflow or underflow.

3.1 Understanding the basic concept

To help imagination, it is the same task as keeping track of the valve position of a bicycle cycling on a circle lane, e.g. 400m circumference. It can be determined from the bicycle position on the circle lane, when the circumference of the wheel is known. Additionally, the number of rounds (in positive or negative direction) if required, unless the circumference of the wheel is an integer quotient of 400m. Whenever the bicycle is at a certain position, the position of the valve can be determined by calculating x meters (measured from the start) modulo the circumference of the wheel. In order to avoid calculation with large numbers, add the offset determined by 400 meters modulo the circumference to the calculation whenever a round has been completed. This way, you always keep track of the valve position without having to count the number of rounds.

3.2 Algorithm

Poll the motor position in a low sequence in order to be sure that you no overflow or underflow of the 32 bit position counter *X_ACTUAL* is missed. Treat *X_ACTUAL* as a positive 32 bit number.

A new variable called *OFFSET* will be used to keep track of the displacement per overflow. It becomes initialized to 0.

Calculate the constant *DISPLACEMENT_CONST* to give the number of microsteps done additionally to an integer count of physical rotations after the overflow of the position range:

DISPLACEMENT_CONST = 2^32 modulo (the number of microsteps per physical rotation)

1. Read out *X_ACTUAL* on a regular basis. Check if the two MSBs bits 31 and 30 have changed (when compared to the last read out) from binary 00 to binary 11 (underflow) or from 11 to 00 (overflow).
   Whenever *X_ACTUAL* overflows, increment *OFFSET* by *DISPLACEMENT_CONST.*
   Whenever *X_ACTUAL* underflows, decrement *OFFSET* by *DISPLACEMENT_CONST.*

   Hint: Any two adjacent bits can be used, e.g. bits 23 and 22, to reduce the number range for subsequent operations. Adapt DISPLACEMENT_CONST accordingly.

2. To avoid the offset becoming larger than the number of microsteps per physical rotation calculate *OFFSET=OFFSET* modulo (the number of microsteps per physical rotation).

Now, the absolute position within a physical rotation is known by adding *OFFSET* to *X_ACTUAL* and calculating the result modulo (the number of microsteps per physical rotation). The offset required to add to the new target position *X_TARGET* representing the zero position within the rotation is the integer division result of the same.

Using these two results, a motion within the actual rotation can be executed at any time.

4 Documentation revision

Version	Date	Author BD=Bernhard Dwersteg	Description
1.0	2013-DEC-05	BD	First version

table 1: Documentation revisions