Application note:

Extending the positioning range of the TMC429 (or TMC428) to 32 bit or more

TRINAMIC® Motion Control GmbH & Co. KG
GERMANY

www.trinamic.com

The TMC429 can be combined with the TMC26x and TMC389 microstepping drivers. As these drivers provide a high microstep resolution, the 24 bit position range might not be sufficient for a number of applications. This application note is meant to be a practical guideline for extending the positioning range.

1 Table of contents

2 Understanding the algorithm

The TMC429 and the TMC428 have 24 bit registers for position information. These are the registers *X_TARGET* and *X_ACTUAL* available for each of the three motor drivers. When doing a positioning movement, the motion direction is determined by the sign of the 24 bit difference between the both values. This way, the maximum displacement per move is (2^23)-1, i.e. 8388607.

The algorithm for position extension requires two additional registers within the microcontroller: A 32 bit target position (*XTARGET32*) and a 32 bit actual position (*XACTUAL32*).

Starting a positioning sequence:

1. *XTARGET32* 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 at least each 2 seconds assuming maximum TMC429 clock frequency (32MHz)

1. Each control cycle starts with reading *X_ACTUAL* from the TMC429.
2. *X_ACTUAL* becomes checked for an underflow or overflow, by comparing to the lower 24 bits of the old value stored in *XACTUAL32.* In case of an underflow, the upper 8 bits of *XACTUAL32* become decremented by 1. In case of an overflow, they become incremented by one. The lower 24 bits of *XACTUAL32* become updated with the value read from *X_ACTUAL*.
3. This step is required in positioning mode, only: *XTARGET32* becomes compared to *XACTUAL32*. In case, the difference is larger than 2^22, *X_TARGET* becomes set to *XACTUAL32* + 2^22, using the lower 24 bits of the result. In case the difference is smaller than -2^22, *X_TARGET* becomes set to *XACTUAL32* - 2^22 the same way. In case the difference is in between, *X_TARGET* becomes set to *XTARGET32* using the lower 24 bits.

As an additional extension, the position compare mechanism must be switched off in case the upper eight bits of *XACTUAL32* do not match the upper 8 bits of *XTARGET32*. When a position snapshot is triggered by a reference switch, the upper 8 bits of *XACTUAL32* need to be copied to the 32 bit copy of the snapshot register, too.

3 An implementation in C

A sample implementation is shown here. It uses the TMC429 library provided with the TMC429+TMC26x-EVAL and extends this so that the 32 bit positioning extension can be used in a transparent way. For this purpose, the routines Write428Int and Read428Int are extended in a way that accessing the XTARGET or XACTUAL register will access the 32 bit variables holding the 32 bit position values instead. The routine Check32BitExtension() is provided to be called on a regular basis (at least every two seconds), and the routine TrackPosition() is just helper function for the Check32BitExtension() routine.

```
typedef struct
{
  int XTarget;   //Target position (32 Bit)
  int XActual;   //Actual position (32 Bit)
  int XOld;      //last 24 bit position (for position counting)
  UCHAR ExtPosFlag;  //TRUE when positioning more than 8388607 steps
} TExtPos;

TExtPos ExtendedPositions[3];  //32 bit position registers for each axis

void Write428Int(UCHAR Address, int Value)
{
  UCHAR Write[4], Read[4];

  UCHAR Motor;
  int Value2;

  Motor=(Address & 0x60)>>5;
  if(Address<0x60)
  {
    switch(Address & 0x1e)
    {
      case IDX_XTARGET:
        //When changing XTARGET the value will be corrected in a way that
        //at first we don’t move more than 8388607 microsteps.
        //During moving the value of XTARGET will be adapted successively
        //in Check32BitExtension() until we are less than 8388607
        //microsteps away from our 32 bit target position.

        ExtendedPositions[Motor].XTarget=Value;
        if(abs(ExtendedPositions[Motor].XTarget-
            ExtendedPositions[Motor].XActual)>8388607)
        {
          if(ExtendedPositions[Motor].XTarget>ExtendedPositions[Motor].XActual)
            Value=ExtendedPositions[Motor].XActual+8388607;
          else
            Value=ExtendedPositions[Motor].XActual-8388607;

          ExtendedPositions[Motor].ExtPosFlag=TRUE;
        }
        else ExtendedPositions[Motor].ExtPosFlag=FALSE;
        break;

      case IDX_XACTUAL:
        //When changing XACTUAL the value of XTARGET will be corrected in a way that
        //at first we don’t move more than 8388607 microsteps.
        //During moving the value of XTARGET will be adapted successively
        //in Check32BitExtension() until we are less than 8388607
        //microsteps away from our 32 bit target position.
        ExtendedPositions[Motor].XActual=Value;
        ExtendedPositions[Motor].XOld=Value & 0x00ffffff;
        if(abs(ExtendedPositions[Motor].XTarget-ExtendedPositions[Motor].XActual)>8388607)
        {
          if(ExtendedPositions[Motor].XTarget>ExtendedPositions[Motor].XActual)
            Value2=ExtendedPositions[Motor].XActual+8388607;
          else
            Value2=ExtendedPositions[Motor].XActual-8388607;

          Write[0]=IDX_XTARGET|MOTOR_NUMBER(Motor)<<5;
          Write[1]=Value2 >> 16;
          Write[2]=Value2 >> 8;
          Write[3]=Value2 & 0xff;
          ReadWrite428(Read, Write);

          ExtendedPositions[Motor].ExtPosFlag=TRUE;
        }
        else ExtendedPositions[Motor].ExtPosFlag=FALSE;
        break;
    }
  }

  Write[0]=Address;
  Write[1]=Value >> 16;
  Write[2]=Value >> 8;
  Write[3]=Value & 0xff;

  ReadWrite428(Read, Write);
}

void TrackPosition(UCHAR Axis, int New24BitPosition)
{
  int Diff24_1;
  int Diff24_2;

  if(New24BitPosition>ExtendedPositions[Axis].XOld)
  {
    Diff24_1=New24BitPosition-ExtendedPositions[Axis].XOld;
    Diff24_2=0xffffff - Diff24_1 + 1;
    if(Diff24_1<=Diff24_2)
    {
      ExtendedPositions[Axis].XActual+=Diff24_1;
    }
    else
    {
      ExtendedPositions[Axis].XActual-=Diff24_2;
    }
  }
  else if(New24BitPosition<ExtendedPositions[Axis].XOld)
  {
    Diff24_1=ExtendedPositions[Axis].XOld-New24BitPosition;
    Diff24_2=0xffffff-Diff24_1+1;
    if(Diff24_1<=Diff24_2)
    {
      ExtendedPositions[Axis].XActual-=Diff24_1;
    }
    else
    {
      ExtendedPositions[Axis].XActual+=Diff24_2;
    }
  }

  ExtendedPositions[Axis].XOld=New24BitPosition;
}

void Check32BitExtension(UCHAR Axis)
{
  UCHAR Read[4], Write[4];
  int Actual24BitPosition;
  int Value;
  UCHAR RampMode;

  //Check the actual ramping mode
  Write[0] = Axis<<5|IDX_REFCONF_RM|TMC428_READ;
  ReadWrite428(Read, Write);
  RampMode=Read428[3];

  //Check actual 24 bit position and update "virtual" 32 bit position register.
  //In this routine, Write428Int() or Read428Int() must NOT be used!
  Write[0]=Axis<<5|IDX_XACTUAL|TMC428_READ;
  ReadWrite428(Read, Write);

  Actual24BitPosition=(Read[1]<<16)|(Read[2]<<8)|(Read[3]);
  TrackPosition(Axis, Actual24BitPosition);

  //The 24 bit target position register will be adapted successively to the 32 bit target
  //position when there is a move of more than 8388607 microsteps in progress.
  if(ExtendedPositions[Axis].ExtPosFlag && RampMode==RM_RAMP)
  {
    if(abs(ExtendedPositions[Axis].XTarget-ExtendedPositions[Axis].XActual)>8388607)
    {
      if(ExtendedPositions[Axis].XTarget>ExtendedPositions[Axis].XActual)
        Value=ExtendedPositions[Axis].XActual+8388607;
      else
```

```
        Value=ExtendedPositions[Axis].XActual-8388607;
    }
    else
    {
      ExtendedPositions[Axis].ExtPosFlag=FALSE;
      Value=ExtendedPositions[Axis].XTarget;
    }
    //Don’t use Write428Int() here!
    Write428Datagram(Axis<<5|IDX_XTARGET, Value >> 16, Value >> 8, Value & 0xff);
  }
}
```

4 Revision history

4.1 Documentation revision

Version	**Date**	**Author** **BD=Bernhard Dwersteg** **OK=Olav Kahlbaum**	**Description**
0.1	2011-DEC-24	BD, OK	First version
0.2	2012-JAN-16	OK	Sample code added

table 1: Documentation revisions