Encoder configuration for the TSD drive series

Application Note 107

Version	Date	Editor	Comment
003	2016-03-04	mvx	Concept based on the TSD series of drives
004	2016-04-19	mvx	Add EncoderTopology selector (FW2077)
005	2017-03-21	mvx	Add chapter on position latching , globalTrigger use and homing
006	2017-10-17	mvx	Guide to CNC homing, info on commutation state for endat save
007	2018-10-24	dg	Chapter about touch probe added
008	2018-12-20	mvx	Chapters on position units and the BissB encoder.

Document AN107_Encoder_EP
Version 008
Source C:\svnroot\doc\ApplicationNotes\
Destination T:\doc\ApplicationNotes
Owner mvx

Triamec Motion AG
Industriestrasse 49
6300 Zug / Switzerland

Phone +41 41 747 4040
Email info@triamec.com
Web www.triamec.com

Disclaimer

This document is delivered subject to the following conditions and restrictions:

- This document contains proprietary information belonging to Triamec Motion AG. Such information is supplied solely for the purpose of assisting users of Triamec products.
- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Information in this document is subject to change without notice.

Table of Contents

1 Overview

This application note mainly describes the encoder concept of the TSD series of Triamec Drives. For the configuration of older generation drives using TwinCAT, see chapter 8.

2 Position Units

As of firmware 4.2.0 the units of an axis are specified using

Axes[].Parameters.PositionController.PositionUnit.

Possible settings are currently meters (m), millimeters (mm), radiants (rad) and degree (degree). Changing this setting will only affect the display units in the TAM System Explorer and the conversion factor between drive and EtherCAT. The real scale is not automatically changed and must be specified in the encoder module using the pitch parameter. This must correspond to the position controller parameters.

3 System Configuration

Hardware View: Up to four encoders can be connected to the hardware if a drive is equipped with two encoder option modules. The drive input is named by the connector.

- X20 Standard encoder input for axis 0
- X21 Standard encoder input for axis 1
- X10 Option encoder input for axis 0 (options TOE1, TOE2)
- X11 Option encoder input for axis 1 (options TOE1, TOE2)

Software View: The dual loop concept allows two encoders feeding two position controllers for each axis. Each axis *i* can be configured separately. The parameters of an encoder software module *k* and its controller counterpart are at

- Axis[i].Parameters.PositionControllers.Encoders[k]
- Axis[i].Parameters.PositionControllers.Controllers[k]

The relationship between the hardware view and the software view is selected by a global ***General.Parameters.EncoderTopology*** (outside of the axis) using the following table. The first row is the default.

EncoderTopology	Hardware → Axis / Encoder	Notes
Standard	X20_Axis0Standard → Axes[0]/Encoders[0] X21_Axis1Standard → Axes[0]/Encoders[1] X21_Axis1Standard → Axes[1]/Encoders[0] X20_Axis0Standard → Axes[1]/Encoders[1]	There are no encoder option modules. The neighbor axis encoder is available as encoders[1]
OptionA	X10_Axis0Option → Axes[0]/Encoders[0] X20_Axis0Standard → Axes[0]/Encoders[1] X11_Axis1Option → Axes[1]/Encoders[0] X21_Axis1Standard → Axes[1]/Encoders[1]	The option modules are available as Encoders[0], which is used for commutation.
OptionB	X20_Axis0Standard → Axes[0]/Encoders[0] X10_Axis0Option → Axes[0]/Encoders[1] X21_Axis1Standard → Axes[1]/Encoders[0] X11_Axis1Option → Axes[1]/Encoders[1]	The option modules are available as Encoders[1]. Commutation is based on the standard encoders.
OptionC	X10_Axis0Option → Axes[0]/Encoders[0] X20_AxisStandard → Axes[0]/Encoders[1] X21_Axis1Standard → Axes[1]/Encoders[0] X11_Axis1Option → Axes[1]/Encoders[1]	The option module of axis 0 is available as Encoders[0], which is used for commutation. Commutation of axis 1 is based on the standard encoder.

Please note, that ***commutation*** is always based on *PositionController.Encoders[0]*.

Each encoder is set to a mode *type* (Analog, Incremental...) using the selector ***Parameters.PositionController.Encoders[].Type***, see next chapter. There is a constraint for the ***Standard*** encoderTopology configuration: The software parameters (including the type) may only be configured one's per hardware module. Lets assume, for example, *Axes[0].PositionController.Encoders[1].type* is set to *Analog*. Then the parameter *Axes[1].PositionController.Encoders[0].type* must be set to ***None***. Otherwise, the error ***EncoderConfigurationError*** is thrown.

4 Encoder Type

The encoder type is specified with *Parameters.PositionController.Encoders[].Type*.

4.1 Incremental RS422

This type uses the complementary A, $\overline{A}$, B, $\overline{B}$ inputs of the encoder for line counting. See figure 1 for the positive counting direction.

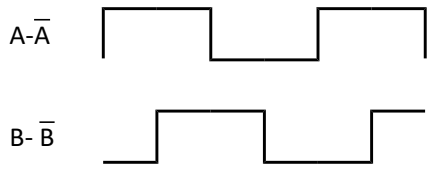


Figure 1: Positive counting direction for incremental encoders.

4.2 Incremental TTL

This type uses the single ended TTL inputs encIo0 and encIo1 for line counting. The direction is the same as in the RS422 case when using encIo0 as A-$\overline{A}$ and encIo1 as B-$\overline{B}$.

4.3 Analog

This type uses A, $\overline{A}$, B, $\overline{B}$ as analog inputs

$$A - \overline{A} = V_{ss} \cos(\varphi)$$
$$B - \overline{B} = V_{ss} \sin(\varphi)$$
$$\varphi = 2\pi \frac{position}{pitch}$$

with V_{ss}=1.0V.

4.4 AnalogEndat

This type initializes the position using an Endat absolute encoder. After initialization the function is identical to the *"Analog"* encoder type and uses the analog sine and cosine inputs for counting. The initialization takes place during first commit:

- In a persistent system, commit is done after parameter load before starting any tama programs and before responding to requests from a control system.
- Otherwise, the commit is done after loading the configuration.

Endat specific information is shown in ***Signals.PositionController.Encoder[].DigitalEncoder***.

If two encoders are used for one axis, only one can be in the *analogEndat* mode: There is only one pathPlanner for both encoders. SetPosition will set both encoders. Therefore it does not make sense to have two encoders setting the initial position.

Usually an analog encoder has a higher resolution than its absolute Endat information. If the initial position from Endat was just used as entry for setPosition, the position would be subject to the bad endat resolution. Therefore, we use the endat information just for the line count and the analog sine cosine information is used for the subResolution. See difference between ***Counts*** and ***CountsExt*** in *Signals.PositionControllers.Encoders[].DigitalEncoder.Counts* is the raw position of the endat device in number of counts. *CountsExt* is the extended (fractional) position in units of counts.

Please be aware that Endat positions use cos(-phi) and sin(-phi) and are therefore phase shifted by 180°. The signals *Counts* and *CountsExt* discussed before are based on this endat definition with endat shift and endat count units. The position *Encoders[].DigitalEncoder.Position* which is (together with the offset) loaded using setPosition uses the Analog Encoder Definition of the shift in the last chapter and its units are based on the ***pitch*** setting.

See also chapter 5 below on saving the reference position in the encoder persistency flash.

See application note AN122 for commutation when persistency data are available.

4.5 DigitalEndat

This type reads the digital cyclic bus information into the encoder without using the analog sin/cos inputs. Prefer analog or analogEndat if analog inputs are available.

4.6 DigitalBissB

This type reads the digital cyclic bus information into the encoder without using the analog sin/cos inputs. Prefer analog if analog inputs are available.

Use the parameter ***dataFormat*** to specify the number of bits of the encoder: "12-24" denotes an encoder with 12 bits multiturn and 24 bits single turn. Currently BissB encoders run at 5MHz and 20

kHz cyclic update rate. Contact Triamec Motion AG if your encoder requires different conditions.

The single turn setting (see above) corresponds to one motor turn. Use the ***pitch*** parameter to specify the encoder scale in units of positionUnits per turn. Set

Axes[].Parameters.Motor.EncoderCountsPerMotorReolution = 1

Example ***Axes[].Parameters.PositionController.PositionUnits*** = Degree

Axes[].Parameters.PositionController.Encoders[0].Pitch = 360

Specifies an axis scale in degrees without any gear in between.

5 Persistency

An absolute encoder (Endat) supplies an absolute position defined by the encoder manufacturer. This is usually not the position required in the machine coordinate system. The machine manufacturer defines the zero of each axis. This reference position of an axis is derived by a homing or calibration procedure during machine setup at the machine manufacturer place. Then the position offset between absolute encoder and machine coordinate system is stored.

5.1 Procedure

During setup of a machine, the drive parameters are loaded and made persistent. This ***drive persistency*** does not contain axis specific offset positions because they may differ between different machines of the same type.

After loading the drive parameters, the drive searches for encoders of absolute encoder type (Endat). Since the encoder hardware does not contain offsets at first, it will show the error ***NoPersistencyData***. This error can be acknowledged and the machine manufacturer continues taking the reference position and activating it using ***set position***.

Now the manufacturer uses the command ***Save*** in ***Commands.PositionController.EncoderPersistency*** to save the calibration offset (***encoder persistency***) and the commutation offset into the encoder hardware. Please note that saving is only possible if the commutation state is valid. This means that the axis can be enabled or not, but it must have been enabled ones before saving is possible.

There are some extended functions, which may be beneficial in special situations: With the command ***Invalidate***, the persistency data may be cleared. The operator will get the error NoPersistencyData during next startup. The command ***Read*** reads the endat data into the *DigitalEncoder* register but does not change the encoder position. With ***ReadAndActivate***, the data is read and activated using setPosition.

5.2 Endat Flash Map

The following background information is not necessary for machine manufacturers but may be beneficial in special cases.

The position offset is stored in the OEM flash part of the endat encoder. It starts at the first available memory location and spans a region of 10 registers of 16bits. It contains a checksum and a key word for identification. Without this measure, a virgin flash could have been interpreted as containing persistency data even though it was never saved.

6 Position Latching

Encoder positions can be latched by a digital input trigger. Applications use this feature for referencing (homing) or measurement purposes. The position latching takes place in the FPGA for optimal timing and accuracy. We describe the register interface (Chapter 6.1), the EtherCAT interface (chapter 6.2) and the timing.

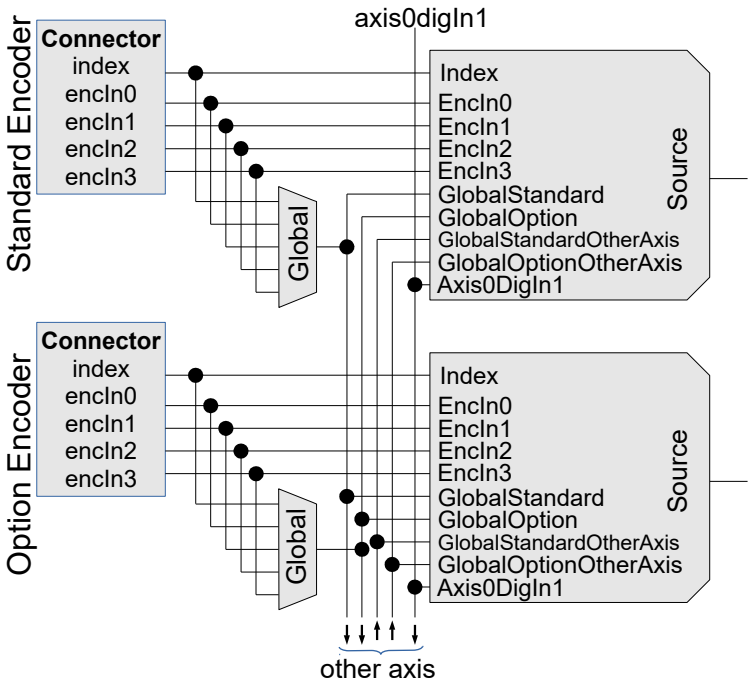


Figure 2: Schematic of the position latch triggers and there selectors for one axis.

6.1 Register Interface

The configuration of the latching modules is done in *Axes[].Commands.PositionController. The two modules "PositionLatchStandard" and "PositionLatchOption"* correspond to their respective encoder connector with the following registers ([1])

- *Source*
  Specifies the digital input trigger source
- *FallingEdge*
  FALSE is rising edge, TRUE is falling edge
- GlobalSource
  Specifies a global trigger for an axis, which can be used by all modules for simultaneous triggering.

These settings are commands, i.e. they are not stored in the persistent parameter flash. Using the latching module is illustrated for the standard encoder of an axis:

- Set ***Commands.PositionController.PositionLatchStandard.Enable*** to TRUE to start searching
- The signal *Signals.PositionController.PositionLatchStandard.State* ([2])
  changes from "Disabled" to "Preparing" and then stays on "Search" until the trigger is received.
- After latching, the position is saved to *Signals.PositionController.PositionLatchStandard.Position* and the signal *Signals.PositionController.PositionLatchStanard .State changes to "Found".*
- Set ***Commands.PositionController.PositionLatchStandard.Enable*** FALSE for a minimum time of 0.1ms.
- The signal *Signals.PositionController.PositionLatchStandard.State*
  changes to "Disabled" and the module is ready for the next sequence.

1 Before firmware 2085, the PositionLatchSource and PositionLatchFalling where parameters of the encoder modules which had to be commited and globalTrigger was not available.

2 Before firmware 2085, the state of a position latch unit was available as a boolean "positionLatchDone".

6.2 EtherCAT interface

With EtherCAT the above sequence can be controlled by direct register access as shown in the last chapter. Alternatively, the TwinCAT touch probe module can be used.

The touch probe cyclic interface of an axis is activated using the slot mechanism of TwinCAT. Double click the EtherCAT drive in TwinCAT. Go to the tab “Slots” and choose the axis. Choose “touch probe” instead of “standard”. This will add the additional signals *TouchProbeStatus* (0x60b9), *TouchProbePosition1Pos* (0x60BA) and *TouchProbePosition1Neg* (0x60bb). It also adds the command word *TouchProbeFunction* (0x60b8). Activate the configuration.

In the PLC section, include the library “Tc2_MC2” and the following declarations

```
touch                : MC_TouchProbe;
axis0    AT %I*      : AXIS_REF;
trigger              : TRIGGER_REF;
```

Then add the following code

```
axis0.ReadStatus();
trigger.EncoderID       := 1;                          // 1..255
trigger.TouchProbe      := TouchProbe1;                // E_TouchProbe
trigger.SignalSource    := SignalSource_DriveDefined;  // E_SignalSource
trigger.Edge            := RisingEdge;                 // E_SignalEdge, RisingEdge
trigger.Mode            := TOUCHPROBEMODE_SINGLE;      // E_TouchProbeMode
trigger.ModuloPositions := FALSE;
touch(Axis := axis0, TriggerInput := trigger);
```

Finally wire the input axis0 to the NCI module axis component MC.NCTOPLC_AXIS_REF_OLD3.

Limitations:

Only single capture mode is available.

Selection of the source:

The trigger source of the touch probe unit does not comply with CAN specifications. Use the (none-persistent) register ***0x23F4*** for axis0 and ***0x2BF4*** for axis 1 instead. Use the EtherCAT startup list or write the COE register from PLC code. The value of this register reflects the four selectors in Figure 2 and configures standard and option encoder at ones. Calculate the sum of the following constants:

Source	***Standard encoder***	***Option encoder***
Index	0x0000’0000	0x0000’0000
EncIn0	0x0000’0002	0x0002’0000
EncIn1	0x0000’0003	0x0003’0000
EncIn2	0x0000’0004	0x0004’0000
EncIn3	0x0000’0005	0x0005’0000
GlobalTrigger of the standard encoder	0x0000’000A	0x000A’0000
Global trigger of the option encoder	0x0000’000B	0x000B’0000

GlobalTrigger of the standard encoder of the other axis	0x0000'000C	0x000C'0000
Global trigger of the option encoder of the other axis	0x0000'000D	0x000D'0000
Axis0 DigIn1	0x0000'000E	0x000E'0000

If a global trigger is chosen in the source table above, the global trigger input is selected as follows

GlobalTrigger	***Standard encoder***	***Option encoder***
Index	0x0000'0100	0x0010'0000
EncIn0	0x0000'0200	0x0020'0000
EncIn1	0x0000'0300	0x0030'0000
EncIn2	0x0000'0400	0x0040'0000
EncIn3	0x0000'0500	0x0050'0000

By adding the four values in these two tables, both encoders are set up and can than be used with the touch probe function block from TwinCAT. Please note: If a global trigger of the other axis is chosen as source, this global trigger must be set in the CAN register of the respective axis.

6.3 Timing considerations

The 24V digital inputs X30 and X31 are slow and not recommended for fast position latching. The reaction time (jitter) is 100µs. Only the digital input AuxIn1 of axis 0 has a fast response time.

The fast digital TTL inputs ***encIo0*** to ***encIo3*** are available at every encoder connector. The timing accuracy (jitter) is 0.2µs, dominated by the encoder path signal read. Besides of the accuracy there is also a systematic delay. While the delay on the trigger input is small (0.01µs) there is a significant delay of 7µs due to hardware and software filtering in the encoder path. This delay can be compensated by taking into consideration the speed of the axis in the search phase.

7 Homing

This describes the homing module of the TSD series of drives. After a general introduction, we describe the supported homing methods.

The homing consists of four phases

- The first search phase is typically used to move into a marker, but various triggers can be chosen.
- Then a relocate move is used to get to the position, where the next search should start.
- The second search phase typically searches for an encoderIndex, but various triggers can be chosen.
- The move to home phase moves the axis to its final position.

The homing parameters of an axis are at ***Axes[].Parameters.Homing***

- Method The homing method
- FirstSearchMove This folders contains the parameters for the first search
- RelocateMove This folders contains the parameters for the relocate move
- SecondSearchMove This folders contains the parameters for the second search
- MoveToHomePositionThis folders contains the parameters for the final move to the home position

Please note that the ***home position*** is not the ***referencePosition***. The reference position is the position the encoder is set to at the found position of the second search move. The home position is the position where the axis will move to after successful homing. The home position is a parameter which will be the same for all machines of a series. The reference position is a command value received from the control system before homing and is typically calibrated during machine assembly.

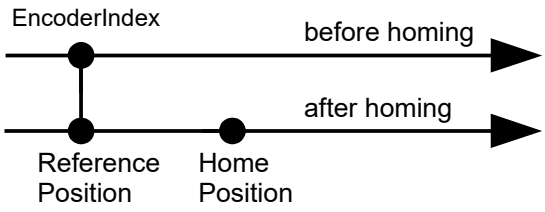


Figure 3: Difference between the reference position and the home position.

The folders with the homing phases “FirstSearch” and “SecondSearch” contain parameters

- EventInput The trigger input to be searched for
- ActiveLow Choose TRUE if the event is active low
- SignedMaxDistance The maximum distance a search will move if no trigger event is found.
- DynamicReduction Values smaller than 1.0 will reduce the pathplanner velocity settings.

The sign of “SignedMaxDistance” has a special meaning. If the trigger is not active before the move, the axis will move into the direction indicated by the sign of this parameter. Otherwise, the direction will be reversed.

The folder with the homing phase “relocate move” contains the same parameter “DynamicReduction“ plus

- Distance The signed distance to move.

The folder with the homing phase “moveToHome” contain the same DynamicReduction parameter plus

- Position The absolute position for the final move.

The homing commands of an axis are at ***Axes[].Commands.Homing***

- Command — Use ***Start*** to start homing, and ***Stop*** to stop any homing ongoing and related moves
- TestNotEnabled — This allows testing the homing triggers by manually moving the axis. The states will behave as usual, but no motion commands are issued.
- ReferencePosition — The reference position as received from the control system.

The homing signal of an axis is the homing state ***Axes[].Signals.General.HomingState***. It indicates not only the phases of a homing, but also homing errors.

7.1 Homing Method Immediate

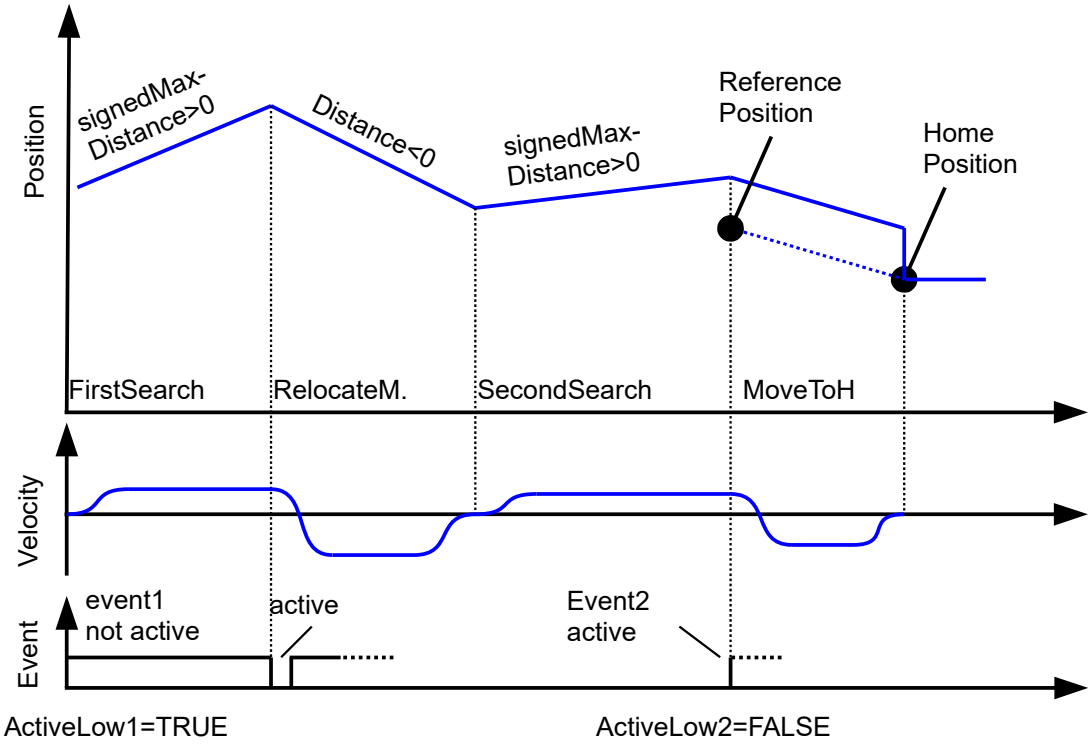
This homing method sets Homing Done without any action. There is no set position taking place. Final state is ***homingDone***.

7.2 Homing Method AtPosition

This homing method sets the actual position to the commanded value referencePosition without any move. Final state is ***homingDone***.

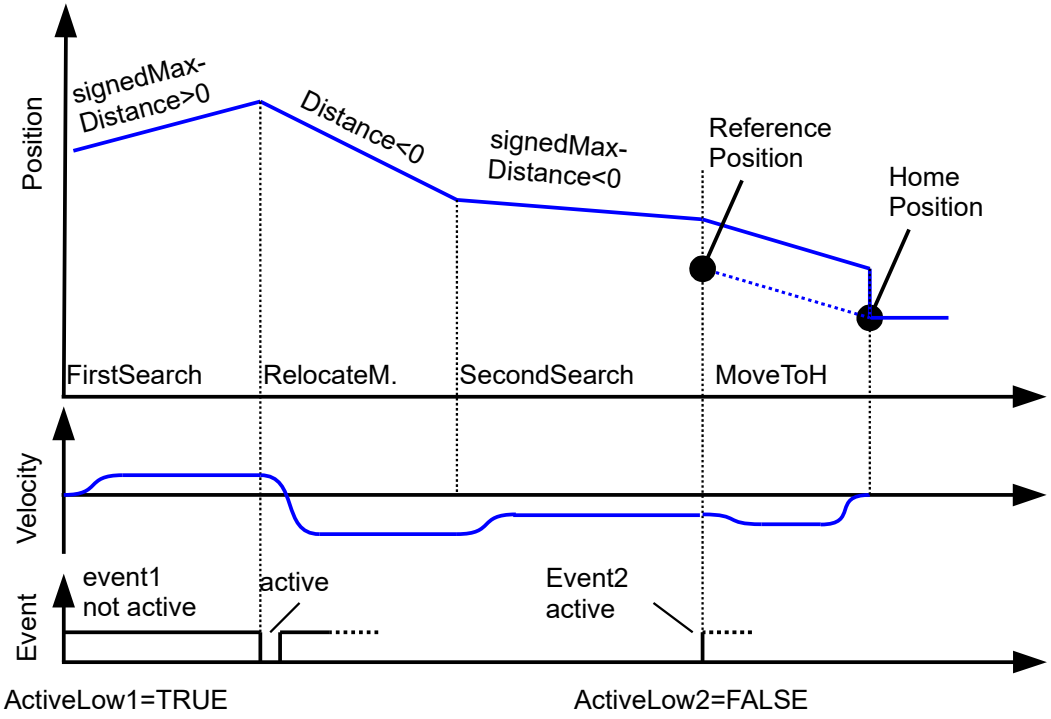
7.3 Homing Method Standard

The first example is a move to the positive limit marker followed by a reverse move and then move again to the positive direction to search the encoderIndex. Please note that the end position of the relocate move is attained in the axis mode standstill.

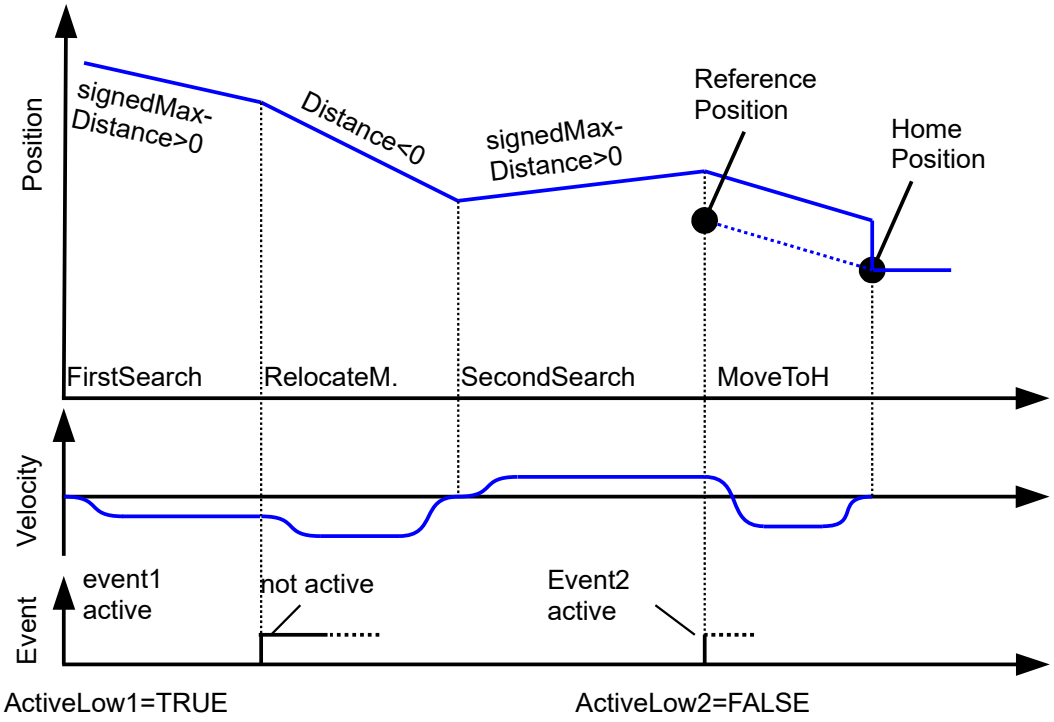


The second example is a move to the positive end marker followed by a reverse relocation move and

then continue in the negative direction to search the encoderIndex. Since the relocation move and the secondSearch move are into the same direction, the axis will not stop in between for optimal motion behaviour. For technical reasons, the relocation move is a continuous move, not a discrete move.

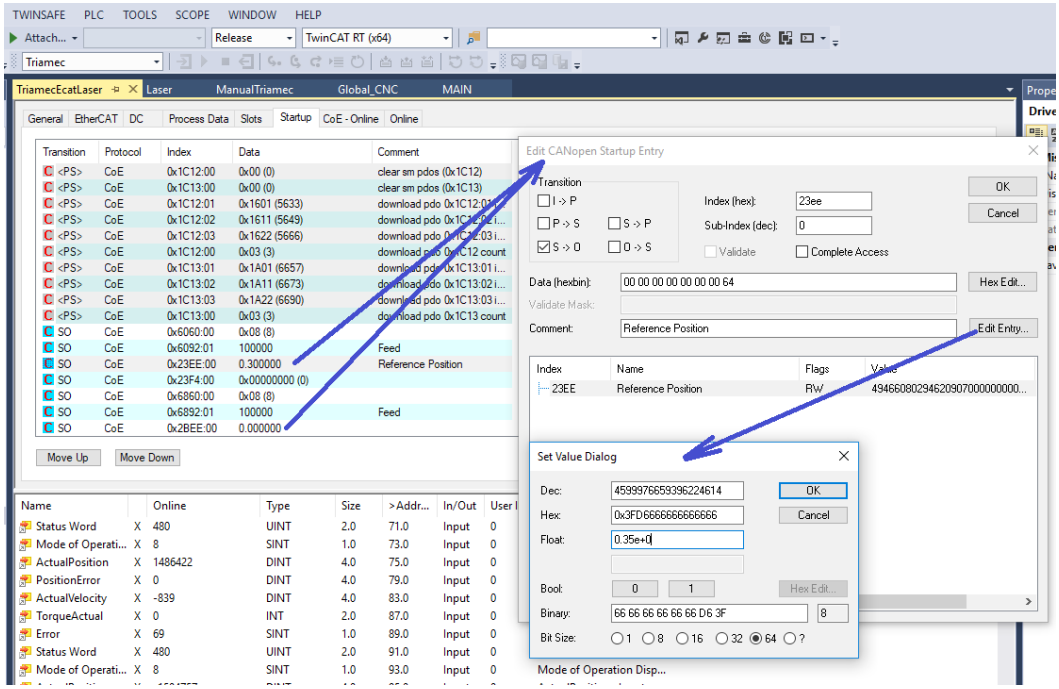


The third sample was obtained with the same parameters as the second sample. This time the marker was already occupied when starting. Therefore the first search moves into the opposite direction.



7.4 TwinCAT Homing with EtherCAT

The reference position can be setup as shown in the following figure.



7.5 TwinCAT Homing with CNC

To setup homing with the TwinCAT CNC module, make sure the cyclic data of the drive contain the objects 0x6060 and 0x6061 (Mode of operation and mode of operation display). This is standard with the ESI file Triamec1.1.xml and newer as delivered with the sample code package 1.1.1.

The parameter list of the CNC axis module must contain the following entries for correct homing behavior:

kenngr.device_id	8	(turn off the target reached check before homing)
kenngr.set_refpos_mode	ABSOLUT	(P-AXIS-00278 : Modes for setting the homing pos)
kenngr.set_refpos_offset	0	(P-AXIS-00279 : [0.1um] or [10-4degree] Offset
kenngr.homing_type	DRIVE_CONTROLLED	(P-AXIS-00299 : Homing type

The homing sequence is then started using the G-code G74. The following sample program starts homing of Z first. When finished homing of X and Y start together:

```
G74 Z1 X2 Y2
```

The position of the reference marker 0x23EE (axis0) and 0x2BEE (axis1) may vary from machine to machine due to calibration considerations. In the sample codes, this position is set to 0.0 in the startup list. The type is a double with metric units (m or rad) as used during drive setup. It might be overwritten by COE register write commands as described in the “Triamec TwinCAT EtherCAT Quick Startup Guide”.

7.6 TwinCAT Absolute Encoders with CNC

With the following entry in the axis data, the CNC immediately sets reference done without the need for homing:

kenngr.abs_pos_gueltig 1 (P-AXIS-00014 : Absolute measurement system)

8 TwinCAT interfaces: old generation drives

This describes TwinCAT Encoder interfaces for TSx51 and TSPxxx types of drives.

8.1 Fast Encoder activation

To activate the fast encoder mode, add the following declaration to MAIN_SLOW

triamec\TcHmiPro\TcApplication\bin\Debug\System

```
encoderConfig : TL_EncoderConfig;
```

and its code

```
encoderConfig.Execute := gAxis[4].ready;
encoderConfig.station      := gAxis[4].MC_axis.station;
encoderConfig.fastencoder  := TRUE;
encoderConfig(Trialink:=Trialink);
```

be aware, that using fast encoder requires the extension TAD5 be mounted to the analog encoder input.

8.2 Endat

To activate Endat 2.1 for axis 1, add this declaration and code to MAIN_SLOW

```
Endat : TL_EndatActivate
```

and

```
Endat.Execute := gAxis[1].ready;
Endat.Offset := 0;
Endat(axis:=gAxis[1].MC_axis, Trialink:=Trialink);
```

Note that this requires drive units in m or radian.

9 Touch Probe Sequence (CNC Controlled)

This section describes how the touch probe functionality can be implemented in the PLC code and executed by G code. With this example the move of the axis during touchdown is controlled by the CNC. Therefore the axis is always in coupled state and no synchronization of the G code interpreter, the NC interpolation and the actual position is required to recouple the axis.

See chapter 10 for more information about an advanced touchdown detection controlled by a Tama program.

9.1 PLC-Example

Channel Parameters

In this example, the M-funcitons M200 and M201 are used for the synchronization between G-code and PLC. The synchronization type (P-CHAN-00041) of the M-functions is set to MVS_SVS = 0x00000002: “Output of M-function to PLC before motion block, Synchronization before motion block”:

```
m_synch[200]  0x00000002      ( MVS_SVS Touch Probe init)
m_synch[201]  0x00000002      ( MVS_SVS Touch Probe done)
```

Axis Parameters

All axes involved in the measuring must be identified as a measuring axis (P-AXIS-00118):

```
kenngr.messachse                1             # Massachse
```

Measurement method (P-AXIS-00516) defines the source of the measuring signal.

```
kenngr.measure.signal           PLC
```

With this setting the touch move stops if all of the measuring axes detect touchdown. Therefore the probing signal has to be applied to all measurement axis.

Implementation

The following global constants are involved in the example:

```
VAR_GLOBAL
   ...
   Trialink         : TL_Trialink2;
   gAxis            : ARRAY [1..N_AXIS] OF TL_AxisSlow;
   CNCSystem        : ST_CncSystem;
   gTouchdown       : BOOL;
   ...
END_VAR
```

The main functionality is implemented in the following sample code “FB_TouchProbe”. With this example axis 1 is used as measurement axis (idxMAxis:= 1) and AuxIn1 as input for the probe signal.

```
VAR_INPUT
   Execute                  : BOOL;             // set Execute FALSE and TRUE to reset
```

```
END_VAR
VAR_OUTPUT
   Error                    : BOOL;              // error flag
   ErrorId                  : UDINT;             // error id
END_VAR

VAR
   stateTouchProbe          : USINT;             // state machine state
   positionLatch            : TL_PositionLatch;  // used for TS drive-generation
   positionLatchReg2        : TL_PositionLatchReg2;// used for TSD drive-generation
   iAxis                    : USINT;             // axis counter
   executePositionLatch     : BOOL;              // execute position latch
   positionLatchSearch      : BOOL;              // flag is set if latching is ready
   positionLatchDone        : BOOL;              // flag is set if latching is done
   positionLatchPosition    : LREAL;             // position when touch down was detected - valid if
                                                 // positionLatchDone
END_VAR
VAR CONSTANT
   idxMAxis                 : USINT := 1;        // index of measurement axis
END_VAR
```

```
// call position latch function block
// depending on the drive generation a different function block is used
IF gAxis[idxMAxis].MC_axis.register2.supported THEN // TSD drive generation
   positionLatchReg2(Execute:=executePositionLatch, Trialink:=Trialink, axis:=
      gAxis[idxMAxis].MC_axis);
   Error:=  positionLatchReg2.Error;
   ErrorId:= positionLatchReg2.ErrorID;
   gTouchdown:= positionLatchReg2.Found;
   positionLatchSearch:= positionLatchReg2.Search;
   positionLatchDone:= positionLatchReg2.Done;
   positionLatchPosition:= positionLatchReg2.Position;
ELSE  // TS drive generation
   positionLatch(Execute:=executePositionLatch, Trialink:=Trialink, axis:= gAxis[idxMAxis].MC_axis);
   Error:=  positionLatch.Error;
   ErrorId:= positionLatch.ErrorID;
   gTouchdown:= positionLatch.Found;
   positionLatchSearch:= positionLatch.Search;
   positionLatchDone:= positionLatch.Done;
   positionLatchPosition:= positionLatch.Position;
END_IF

// state machine
CASE stateTouchProbe OF
   0: // idle - wait for M-function
      IF NOT Error AND CNCSystem.Channel[CHAN].M[200].bState_rw THEN
         // setup and execute position latch
         IF gAxis[idxMAxis].MC_axis.register2.supported THEN // TSD drive generation
            positionLatchReg2.Source:= TL_ConstAxisParPosCtrlEncLatchSrc.AuxIn1;
            positionLatchReg2.EdgeFalling:= FALSE;
```

```
            ELSE  // TS drive generation
               positionLatch.Source:=  TL_Config.ReferenceFirstInput.AuxIn1;
               positionLatch.EdgeFalling:= FALSE;
            END_IF
            executePositionLatch:= TRUE;
            stateTouchProbe:= stateTouchProbe+1;
         END_IF
      1:  // wait until position latch is ready
         IF positionLatchSearch THEN
            // clear M-function
            CNCSystem.Channel[CHAN].M[200].bState_rw:= FALSE;
            stateTouchProbe:= stateTouchProbe+1;
         ELSIF Error OR NOT Execute THEN
            stateTouchProbe:= 100;
         END_IF
      2:  // wait until G310 is finished - indicated with M201
         IF CNCSystem.Channel[CHAN].M[201].bState_rw THEN
            // if this state is reached, G310 move generated a position latch or reached end of move
            IF positionLatchDone THEN
               // position latched - evaluate position
               // ...
               stateTouchProbe:= 100;
            ELSE
               // end of move reached - reset
               executePositionLatch:=FALSE;
               stateTouchProbe:= 100;
            END_IF
         ELSIF Error OR NOT Execute THEN
            stateTouchProbe:= 100;
         END_IF
      100: // reset and clean up
         IF NOT Error OR NOT Execute THEN
            CNCSystem.Channel[CHAN].M[200].bState_rw := FALSE;
            CNCSystem.Channel[CHAN].M[201].bState_rw  := FALSE;
            executePositionLatch:=FALSE;
            stateTouchProbe:= 0;
         END_IF
      ELSE
         stateTouchProbe := 0;
   END_CASE
```

To provide the probing signal from the PLC to the CNC the HLI is used. Therefore the following code is added to the TL_CNC_AX function block. It is important to apply the probing signal to all measurement axes.

```
VAR
   ...
   pAxis              : POINTER TO High_Level_Interface_Ax;
   ...
END_VAR
```

```
pAxis^.lr_mc_control.probing_signal.enable_w:= TRUE;
pAxis^.lr_mc_control.probing_signal.command_w:= gTouchdown;
```

The touch probe function block is executed if the following command is called with the TASK_SLOW.

```
VAR
   ...
   touchProbe               : FB_TouchProbe;
   ...
END_VAR
```

```
// Execute toch probe state machine
touchProbe(Execute:=Enabled);
```

9.2 G-Code Example

The following G code example uses the G310 command to execute the touch probe move (see Programming Manual 4.1.10.5 from ISG). The M-function M200 is used to prepare the drive for the position latch and M201 is used to evaluate the result of the measurement by the PLC .

```
; do something
...
...

; move to start position
G1 X100 Y200 Z10

; initialize touchdown detection
M200                          ; execute PLC code to prepare position latch

; execute touch probe move
#MEAS MODE[5]                 ; measurement type is 5 (P-CHAN-00057)
G310 G1 Z-1  $GOTO N10:       ; start touch probe move

; no touchdown detected during G310 move
M201                          ; execute user specific code to reset
...
...
$GOTO N20:

; touchdown detected during G310 move
N10:
M201                          ; execute PLC code to evaluate the measurement

; continue
N20:
...
...
```

9.3 Remarks

- If the axis parameter kenngr.measure.signal is set to PLC_FIRST_EVENT, the implementation

would be simplified as the probing signal just has to be applied to one of the measuring axes to stop the move. But if PLC_FIRST_EVENT is used, a reset after the search move causes a 20s delay because of a bug in the ISG library. Therefore PLC_FIRST_EVENT should not be used.

10 Tama Controlled Touch Probe Sequence

This section covers some aspects of the touch probe functionality if the touchdown causes the axis to release the coupling. This is for example the case, if a stop-command is executed by the Tama program at touchdown.

In this case synchronization of the G code interpreter, the NC interpolation and the actual position is required to recouple the axis.

10.1 Un-coupling and Coupling

To avoid an error if the coupling of an axis is canceled while an NC program is running and to re-couple the axis, the following sequence has to be executed:

Preparation

- To synchronize G-code interpreter and NC-intepolator the following command has to be added to the G-code after touchdown detection.

```
#CHANNEL INIT [ACTPOS]
```

- The following axis-parameter has to be configured:

```
kenngr.tracking_offset_remain = 1
```

- In normal case TorquePermission will be reset if the coupling is broken.

```
 pAxis^.lr_mc_control.torque_permission.command_w:= TorquePermission
```

  Therefore the command TorquePermission has to be overridden before the coupling will be broken. E.g.

```
TorquePermission:= axes[iAxis].coupled OR gSimulate OR overrideTorquePermission;
```

- The command

```
pAxis^.lr_mc_control.follow_up.command_w := AxisTracking
```

  must remain TRUE until CNC has stopped, override of the follow_me state is required. E.g.:

```
AxisTracking:= axes[iAxis].followMe AND NOT executeTouchdown AND NOT (gSimulate AND enable)
OR overrideAxisTracking;
```

- To synchronize the actual position with the NC-interpolator, the NC has to be set to follow up mode. E.g.:

```
gCncAx[iAxis].AxisTracking:= axes[iAxis].followMe AND NOT executeTouchdown AND NOT (gSimulate
AND enable) OR overrideAxisTracking;
```

Sequence

- Wait until the CNC request the preparation of the touch probe move by a M-function.
- Set executeTouchdown to TRUE and prepare the touch probe move.

- When the touch probe move is ready, acknowledge the M-function so the CNC can execute the touch probe move (e.g. G310) and set overrideTorquePermission to TRUE.
- The touchdown signal has to be provided to the CNC.
- Wait until G310 move is done which is indicated by the CNC with an other M-function.
- If no touchdown is detected, the sequence is finished and executeTouchdown and overrideTorquePermission has to be set to FALSE and the the M-function acknowledged and the sequence is done in this case.
- If touchdown is detected, overrideAxisTracking has to be set to TRUE.
- Wait until all axes are in tracking mode (follow up mode).
- Set overrideAxisTracking to FALSE and re-couplele the axes by reactivate the coupling by setting the couple command from FALSE to TRUE;
- Wait until all axes are coupled.
- Set executeTouchdown and overrideAxisTracking to FALSE and acknowledge the M-function.