



Encoder Configuration

Application Note 107

Version	Date	Editor	Comment
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	2018-12-12	mvx	Chapters on position units and the BissB encoder.
009	2019-05-02	mvx	New parameters and commands for absolute encoders. New Tamagawa and Nikon.
011	2019-12-12	mvx	Extend position latch selectors with DigIn1..6 entries and option modules (FW>=4.7.6)
012	2020-12-21	mvx	New PositionUnit change function and simplified description for AbsoluteHoming
013	2021-01-20	dg	Description for new EventInput PositionError for homing and latching added.
014	2021-08-19	mvx	Describe early trigger. Describe homing from the TwinCAT HMI with the Tria-Link field bus.
015	2021-09-15	dg	Figure EncoderTopology added and DigitalBiss enhanced.

Document AN107_Encoder_EP
Version 015
Source Q:\doc\ApplicationNotes\
Destination T:\doc\ApplicationNotes
Owner mvx

Copyright © 2021
Triamec Motion AG
All rights reserved.

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.....	3		
2	Position Units.....	3		
3	System Configuration.....	4		
4	Not-Absolute Encoders.....	6		
4.1	Incremental RS422.....	6		
4.2	Incremental TTL.....	6		
4.3	Analog.....	6		
5	Absolute Encoders without sin/cos signals	7		
5.1	DigitalEndat.....	7		
5.2	DigitalBissB and DigitalBissC....	7		
5.3	DigitalTamagawa.....	7		
5.4	DigitalNikon.....	7		
6	Absolute Encoders with sin/cos signals. .	8		
6.1	Subresolution.....	8		
6.2	AnalogEndat.....	8		
6.3	AnalogBissB and AnalogBissC. .	8		
7	Position Latching.....	10		
7.1	Register Interface.....	10		
7.2	EtherCAT interface.....	12		
7.2.1.	Trigger individual or 24V	12		
7.2.2.	Trigger simultaneous X20	13		
7.2.3.	Trigger simultaneous X21	13		
7.2.4.	Trigger simultaneous X10	14		
7.2.5.	Trigger simultaneous X11	14		
7.2.6.	Using the NCI module...	15		
7.2.7.	Using the CNC module...	15		
7.3	Timing considerations.....	15		
8	Homing.....	17		
8.1	Homing Method Standard.....	17		
8.2	Absolute Encoder Homing.....	20		
8.3	Homing Method Immediate....	20		
8.4	Homing Method AtPosition....	20		
8.5	TwinCAT Homing with EtherCAT	21		
8.6	TwinCAT Homing with the Tria-Link fieldbus.....	22		
8.7	TwinCAT Absolute Encoders with CNC	22		
9	TwinCAT interface of old gen. drives.....	23		
9.1	Fast Encoder activation.....	23		
9.2	Endat.....	23		
10Touch Probe Sequence (CNC).....	24		
10.1PLC- Example.....	24		
10.1.1.	Channel Parameters....	24		
10.1.2.	Axis Parameters.....	24		
10.1.3.	Implementation.....	24		
10.2G-Code Example.....	27		
10.3Remarks	28		
11Tama Controlled Touch Probe.....	28		
11.1Un- coupling and Coupling.....	28		
11.1.1.	Preparation.....	29		
11.1.2.	Sequence.....	29		

1 Overview

This application note mainly describes the encoder concept of Triamec Drives. This is valid for firmware \geq 4.5.0. For the configuration of older generation drives using TwinCAT, see chapter 9.

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), degree (degree) and turns (turns). Changing this setting will only affect the display units in the TAM System Explorer and the conversion factor between drive and EtherCAT.

If the position unit is changed, all parameters with a relation to the position unit must also be adapted to match with the new scale. Use `Axis[].Commands.General.Event=ChangeUnits` (FW 4.9.0) to not only change the unit but also all parameters associated with this unit. Only Tama controller parameters need to be adapted manually in this case.

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 (see also Figure 1). 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. Preferred if using option module 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.

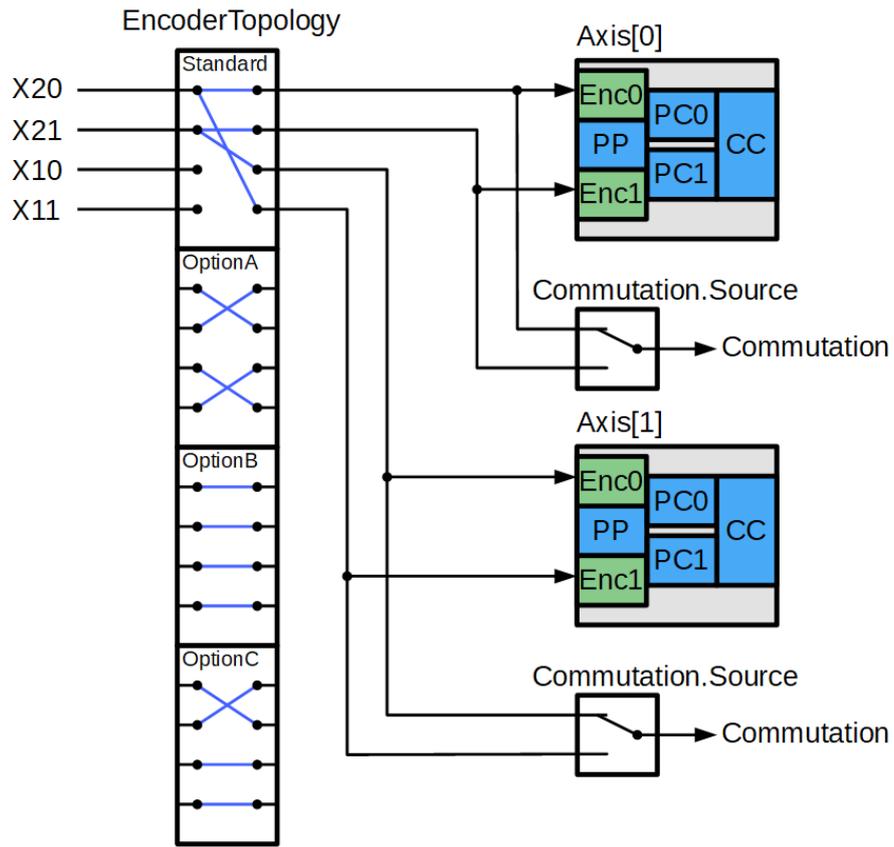


Figure 1: Routing of the EncoderTopology options.

4 Not-Absolute Encoders

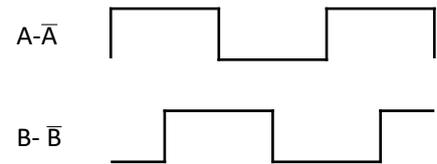
These encoders are characterized by their period. An example of the encoder scale

Axes[].Parameters...	Rotative with 2048 cycles per turn (one cycle is four quadrants)	Linear with 20μm period and a pole-pair distance of 25mm
Axis units in this example	Degrees without any gear in between.	Millimeter without any gear in between.
Motor.EncoderCountsPerMotorRevolution	2048	1250 (set pole pairs to 1)
PositionController.PositionUnits	Degree	mm
PositionController.Encoders[].Pitch	0.17578125 (=360/2048)	0.020

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

4.1 Incremental RS422

This type uses the complementary A, \bar{A} , B, \bar{B} inputs of the encoder for line counting. See figure 2 for the positive counting direction.



4.2 Incremental TTL

This type uses the single ended TTL inputs encln0 and encln1 for line counting. The direction is the same as in the RS422 case when using enclo0 as A- \bar{A} and enclo1 as B- \bar{B} .

Figure 2: Positive counting direction for incremental encoders.

4.3 Analog

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

$$A - \bar{A} = V_{ss} \cos(\varphi)$$

$$B - \bar{B} = V_{ss} \sin(\varphi)$$

$$\varphi = 2\pi \frac{\text{position}}{\text{pitch}}$$

with $V_{ss}=1.0V$.

5 Absolute Encoders without sin/cos signals

These types read the digital cyclic bus information into the encoder without using the analog sin/cos inputs. An example of the encoder scale:

<i>Axes[].Parameters...</i>	Rotative	Linear with 20nm resolution and a pole-pair distance of 25mm
	Assuming no gear in between. (The single turn setting corresponds to one motor turn.)	Millimeter without any gear in between.
<i>Motor.EncoderCountsPerMotorRevolution</i>	1	1250000
<i>PositionController.PositionUnits</i>	Degree	mm
<i>PositionController.Encoders[].Pitch</i>	360.0	0.000020

An absolute encoder supplies an absolute position. This can be used for homing and commutation functions. See chapter 8 for homing. For persistent commutation of absolute encoders, see "ServoDrive-SetupGuide_EP012.pdf".

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

5.1 DigitalEndat

This type reads the cyclic position using the Endat protocol. Its update rate is limited to 50kHz. Prefer analogEndat if analog inputs are available.

5.2 DigitalBissB and DigitalBissC

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

Use the parameter ***Encoders[].DataFormat*** to specify the number of bits for multi turn (MT) and single turn (ST) by using the following syntax "[MT]-[ST]". For example "12-24" denotes an encoder with multi turn MT=12 bits and single turn ST=24 bits. **Choose ST=0 for linear encoders.** Currently, BissB and BissC encoders run at 5MHz and 20 kHz cyclic update rate. Contact Triamec Motion AG if your encoder requires different conditions.

5.3 DigitalTamagawa

Same setup as for DigitalBissB. The position loop rate is 20 kHz.

5.4 DigitalNikon

Same setup as for DigitalBissB. Use a dataformat string "16-20-F2.5MHz" to specify an encoder with 16 bits multiturn and 20 bit single turn resolution and a clock frequency of 2.5 MHz. Available frequencies are 2.5 MHz and 4 MHz with corresponding position loop rates of 20kHz and 50 kHz.

6 Absolute Encoders with sin/cos signals

These encoders contain a sin/cos signals in addition to the serial absolute position information. The absolute position is initially loaded from the serial data stream. After initialization, the position is determined using the sin/cos information same as with analog encoders.

Note: The *pitch* and *EncoderCountsPerMotorRevolution* setting of these encoders is specified as for non-absolute encoders, see chapter 4.

An absolute encoder supplies an absolute position. This can be used for homing and commutation functions. See chapter 8 for homing. For persistent commutation of absolute encoders, see "ServoDrive-SetupGuide_EP012.pdf".

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 or during reset of an encoder error.

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

6.1 Subresolution

Usually an analog encoder has a higher resolution than its absolute digital information. If the initial position was just used as entry for `setPosition`, the position would be subject to the bad digital resolution. Therefore, we use the digital information just for the line count and the analog sine cosine information is used for the subResolution. The alignment fails if the digital information is not consistent with the analog A/B information. The error "EncoderSubResolutionError" will be shown in this case.

As of firmware 4.7, this check must be enabled using the DataFormat register. Choose "M1" for AnalogEndat and a suffix "-M1" for analogBiss. If AnalogBiss is set zero, the alignment of digital data and analog data is lost and this function cannot be used anymore.

6.2 AnalogEndat

Please be aware that Endat positions use $\cos(-\phi)$ and $\sin(-\phi)$ and are therefore phase shifted by 180°.

6.3 AnalogBissB and AnalogBissC

This type reads the digital absolute position using the Biss-B format (see 5.2) and then uses the analog sin/cos signals for precise and fast position update.

The natural unit of this encoder type is not "one turn" but one sin/cos cycle! Therefore find the number of bits K required for "sin/cos periods per turn", for example $K=11$ bits for an encoder with 2048 sin/cos periods. This has to be taken into account in the dataFormat string as follows: A Biss encoder with $MT=12$ bits multiturn and $ST=24$ bits single turn, will not be specified with "12-24" as for a DigitalBissB, but with "23-13". Here the multiturn part is the sum $MT+K$ and the single turn part is the difference $ST-K$.

7 Position Latching

Encoder positions can be latched by a digital input trigger. Applications can 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 7.1), the EtherCAT interface (chapter 7.2) and the timing.

7.1 Register Interface

Some inputs can be used as global trigger to simultaneously latch positions of axis 0 and 1.

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 used for this module (see Figure 3).
- *Type*:
Defines if *RisingEdge* or *FallingEdge* pulls the trigger.
- *Global*:
Can be used to define an input of this module as a global trigger (see Figure 3). The global trigger can be used by all modules which allows for example simultaneous triggering. Set *Global* to *None* if no input of this module is used as a global trigger.
For example if *Encln0* of the option encoder of axis 0 should be used as trigger to latch the position of the standard encoder of axis 1 set:
 - ♦ *Axis[0].Commands.PositionController.PositionLatchOption.Global = Encldx0*
 - ♦ *Axis[1].Commands.PositionController.PositionLatchStandard.Source = GlobalOptionOtherAxis*
- *PositionErrorThreshold*:
If the source register is set to *PositionError* the position latch will be triggered in case the absolute value of the master position error exceeds this threshold.
- *CurrentThreshold*:
If the source register is set to *Current* the position latch will be triggered in case the absolute value of the controller current exceeds this threshold.

These settings are commands, i.e. they are not stored persistent.

Using the latching module is illustrated below for the standard encoder of an axis:

- Set the registers *Source*, *Global* and *Type*.
- 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

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

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

the signal `Signals.PositionController.PositionLatchStandard.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.

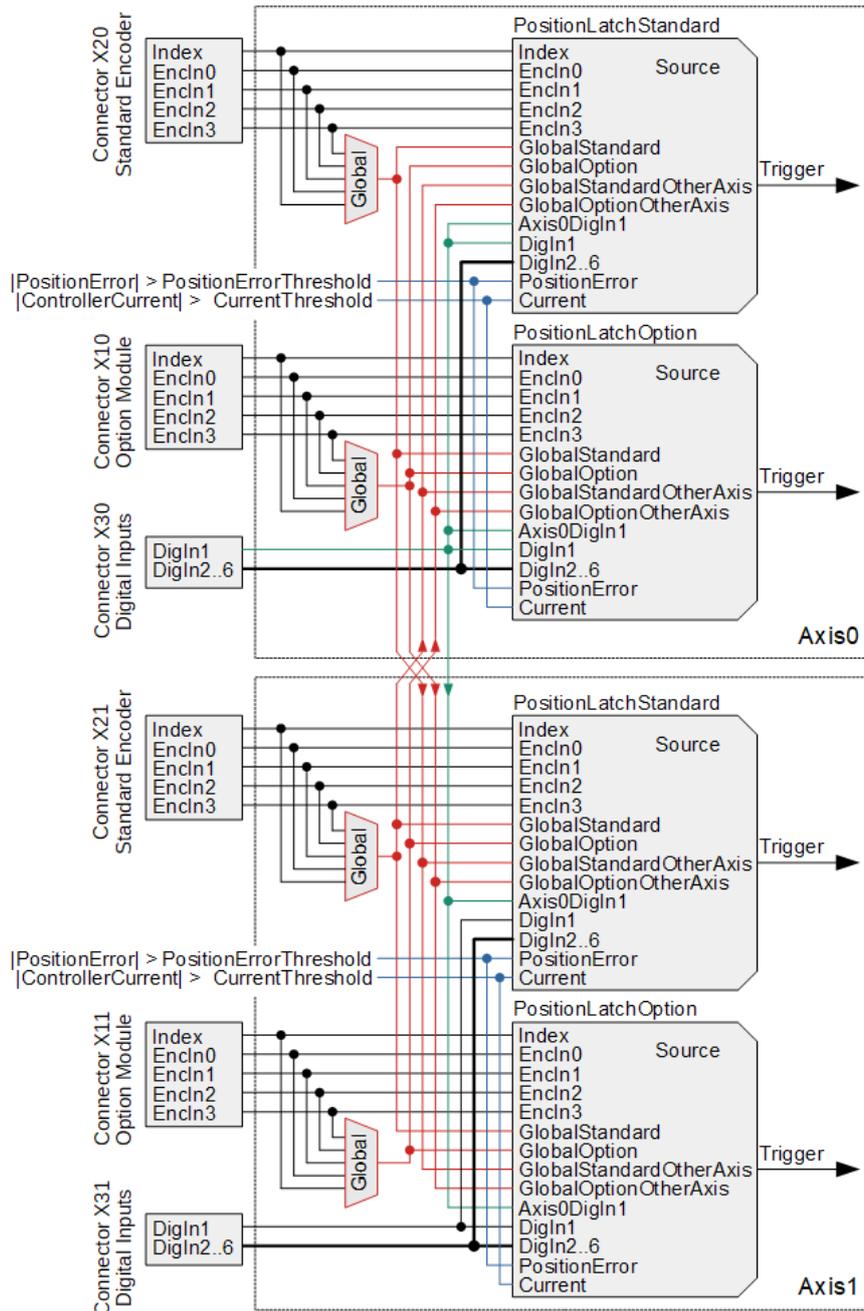


Figure 3: Schematic of the position latch triggers and theirs selectors.

7.2 EtherCAT interface

With EtherCAT the above sequence can be controlled by direct register access using the registers shown in the previous section. 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. For the axis module, 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).

Limitations: Only single capture mode is available.

The trigger source of the touch probe unit does not comply with CAN specifications. Use the (none-persistent) COE register **0x23F4** for axis0 and **0x2BF4** for axis 1 to specify the trigger source instead. For this, the EtherCAT startup list can be used or the COE registers can be written by the PLC program. The value of these registers define the four selectors of each axis in in Figure 3. The following sections show what values should be entered for these registers. After setting these registers, activate the configuration.

First decide, if the two axes should be triggered individually (Section 7.2.1) or simultaneous from a common trigger source (Sections 7.2.2 to 7.2.5). A special case is a simultaneous trigger from a digital 24V input. This case can be handled as described in the section for individual triggering (Section 7.2.1) .

7.2.1. Trigger individual or 24V

Choose this chapter, if the two axes should be triggered individually or if they are triggered from a 24V input source. For each axis, choose between latching standard encoder positions or option encoder positions. Then choose the trigger source. Enter the value obtained from the following table into the register 0x23F4 to set the source for axis 0 or 0x2BF4 to set the source for axis 1:

Trigger Source	Position from the Standard encoder	Position from the Option encoder
Index	0x0000'0000	0x0000'1000
Encln0	0x0000'0002	0x0000'1002
Encln1	0x0000'0003	0x0000'1003
Encln2	0x0000'0004	0x0000'1004
Encln3	0x0000'0005	0x0000'1005
Axis0 DigIn1 (fast input see note * below)	0x0000'000E	0x0000'100E
DigIn1 (see note * below)	0x0000'0014	0x0000'1014
DigIn2	0x0000'0015	0x0000'1015
DigIn3	0x0000'0016	0x0000'1016
DigIn4	0x0000'0017	0x0000'1017
DigIn5	0x0000'0018	0x0000'1018
DigIn6	0x0000'0019	0x0000'1019
PositionError	0x0000'001E	0x0000'101E
Current	0x0000'001F	0x0000'101F



(*) The 24V input DigIn1 of axis0 is a faster input than the other digital 24V inputs. Prefer this input for time critical latching and specify input "Axis0DigIn1" instead of "DigIn1" in this case.

7.2.2. Trigger simultaneous X20

Here the trigger for both axes is a 3.3V input of the axis0 standard encoder connector X20. Choose the trigger source. Choose between latching axis 0 standard encoder positions or option encoder positions. Enter the value obtained from the following table into the register 0x23F4

Trigger Source	Position from the Standard encoder	Position from the Option encoder
Index	0x0000'010A	0x0000'110A
Encln0	0x0000'020A	0x0000'120A
Encln1	0x0000'030A	0x0000'130A
Encln2	0x0000'040A	0x0000'140A
Encln3	0x0000'050A	0x0000'150A

Choose between latching axis 1 standard encoder positions or option encoder positions. Then enter the value obtained from the following table into the register 0x2BF4

Source	Position from the Standard encoder	Position from the Option encoder
GlobalTrigger of the standard encoder of the other axis	0x0000'000C	0x0000'100C

7.2.3. Trigger simultaneous X21

Here the trigger for both axes is a 3.3V input of the axis1 standard encoder connector X21. Choose between latching axis 0 standard encoder positions or option encoder positions. Enter the value obtained from the following table into the register 0x23F4

Source	Position from the Standard encoder	Position from the Option encoder
GlobalTrigger of the standard encoder of the other axis	0x0000'000C	0x0000'100C

Next choose the trigger source. Choose between latching axis 1 standard encoder positions or option encoder positions. Then enter the value obtained from the following table into the register 0x2BF4

Trigger Source	Position from the Standard encoder	Position from the Option encoder
Index	0x0000'010A	0x0000'110A
Encln0	0x0000'020A	0x0000'120A
Encln1	0x0000'030A	0x0000'130A
Encln2	0x0000'040A	0x0000'140A

Encln3	0x0000'050A	0x0000'150A
--------	-------------	-------------

7.2.4. Trigger simultaneous X10

Here the trigger for both axes is a 3.3V input of the axis0 option encoder connector X10. Choose between latching axis 0 standard encoder positions or option encoder positions. Then choose the trigger source. Enter the value obtained from the following table into the register 0x23F4

<i>Trigger Source</i>	<i>Position from the Standard encoder</i>	<i>Position from the Option encoder</i>
Index	0x0000'010B	0x0000'110B
Encln0	0x0000'020B	0x0000'120B
Encln1	0x0000'030B	0x0000'130B
Encln2	0x0000'040B	0x0000'140B
Encln3	0x0000'050B	0x0000'150B

Choose between latching axis 1 standard encoder positions or option encoder positions. Then enter the value obtained from the following table into the register 0x2BF4

<i>Source</i>	<i>Position from the Standard encoder</i>	<i>Position from the Option encoder</i>
GlobalTrigger of the option encoder of the other axis	0x0000'000D	0x0000'100D

7.2.5. Trigger simultaneous X11

Here the trigger for both axes is a 3.3V input of the axis1 option encoder connector X11. Choose between latching axis 0 standard encoder positions or option encoder positions. Enter the value obtained from the following table into the register 0x23F4

<i>Source</i>	<i>Position from the Standard encoder</i>	<i>Position from the Option encoder</i>
GlobalTrigger of the option encoder of the other axis	0x0000'000D	0x0000'100D

Next choose the trigger source. Choose between latching axis 1 standard encoder positions or option encoder positions. Then enter the value obtained from the following table into the register 0x2BF4

<i>Trigger Source</i>	<i>Position from the Standard encoder</i>	<i>Position from the Option encoder</i>
Index	0x0000'010B	0x0000'110B
Encln0	0x0000'020B	0x0000'120B
Encln1	0x0000'030B	0x0000'130B
Encln2	0x0000'040B	0x0000'140B
Encln3	0x0000'050B	0x0000'150B

7.2.6. Using the NCI module

In the PLC section, include the library "Tc2_MC2" and the following declarations

```
touch                : MC_TouchProbe;
axis0    AT %I*      : AXIS_REF;           // if not using triamec sample code
trigger              : TRIGGER_REF;
```

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. If the triamec sample code is used, the AXIS_REF is already included as "axis0.nci". Delete the row "axis0" in the declaration section above and use "axis0.nci" instead of "axis0" in the code above.

7.2.7. Using the CNC module

Configure the following CNC axis parameters for each axis used for measuring

```
kenngr.messachse      1      ; Axis engaged in measurement travel ( P-AXIS-00118)
kenngr.measure.signal DRIVE
```

The following sequence starts a move with maximum end position 25 and then moves to the found position

```
#MEAS MODE[1]                ; measurement type (P-CHAN-00057)
G100 X30 F1
G01 X V.A.MESS.X F10
```

7.3 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 **encln0** to **encln3** 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 ($0.01\mu\text{s}$) there is a significant delay of $7\mu\text{s}$ for TTL inputs 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.

8 Homing

This chapter describes the homing module. After a general introduction, we describe the supported homing methods.

There are two classes of homing methods. The motion based method “Standard” derives the homing position from a move sequence. The second class is for absolute Encoders. These can use their absolute position information for homing. An absolute position of an absolute encoder is usually a position aligned to the motor phasing, 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 may be stored in various ways.

The zero of an axis may be derived in various ways, using **Parameters.Homing.Method**: The axis zero is received

- **Standard** by a homing search move, started by the control system
- **AbsoluteEncoderXOffsetEncoder** from encoder persistency data of encoder[X].
- **AbsoluteEncoderXOffsetDrive** from drive persistency data and the encoder[X] position.
- **Immediate**
- **At Position**

If two absolute encoders are used for one axis, only one can be the position reference. This is why the encoder number X has to be specified for absoluteEncoder based methods.

Homing commands of an axis are at **Axes[].Commands.Homing.Command**:

- **Start** start a motion based homing sequence, and
- **Stop** stop any motion based homing ongoing and related moves
- **SaveEncoder** save persistency data to an absolute encoder (axis must be disabled) and
- **InvalidateEncoder** delete the persistency data of an absolute encoder (axis must be disabled)

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.

8.1 Homing Method Standard

The motion based 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** set to **Standard** for this type.
- **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

- **MoveToHomePosition** This folder 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.

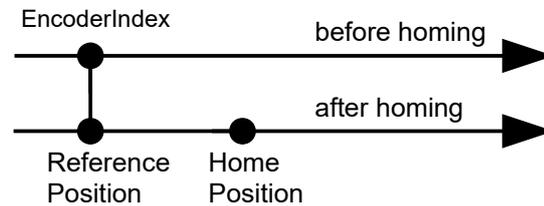


Figure 4: Difference between the reference position and the home position.

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.

In case the **EventInput** register is set to **PositionError** the event will be triggered in case the absolute value of the master position error exceeds the threshold configured in register **PositionErrorThreshold**. This event input is suitable for example to detect a hard-stop.

The first and second search checks for an early trigger. It assumes the trigger condition is not reached within the first 5ms of the search. If the trigger is found earlier, it throws an early trigger error.

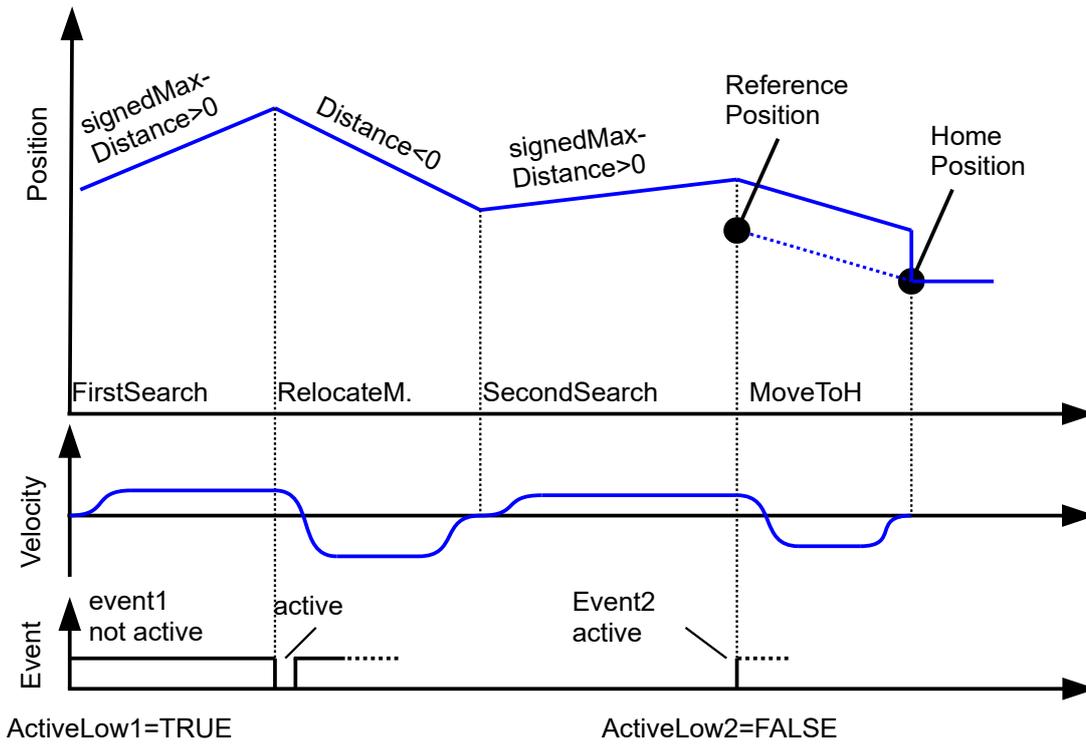
The folder with the homing phase “relocate move” contains the same parameter “DynamicReduction” plus

- **Distance** The signed distance to move.

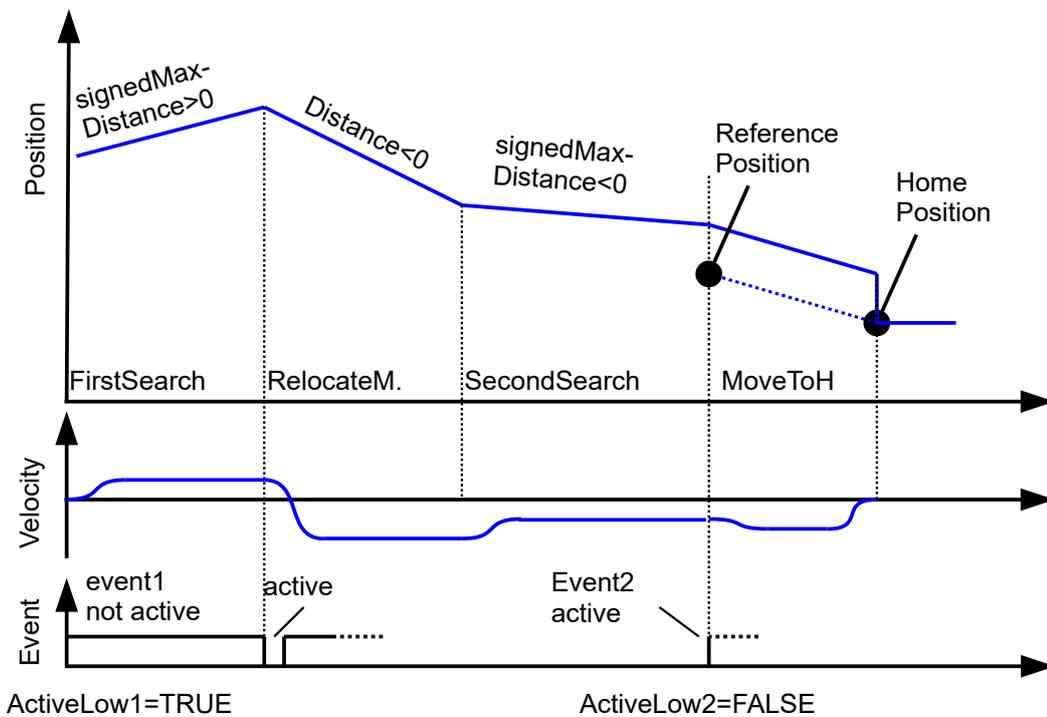
The folder with the homing phase **moveToHome** contain the same **DynamicReduction** parameter plus

- **Position** The absolute position for the final move.

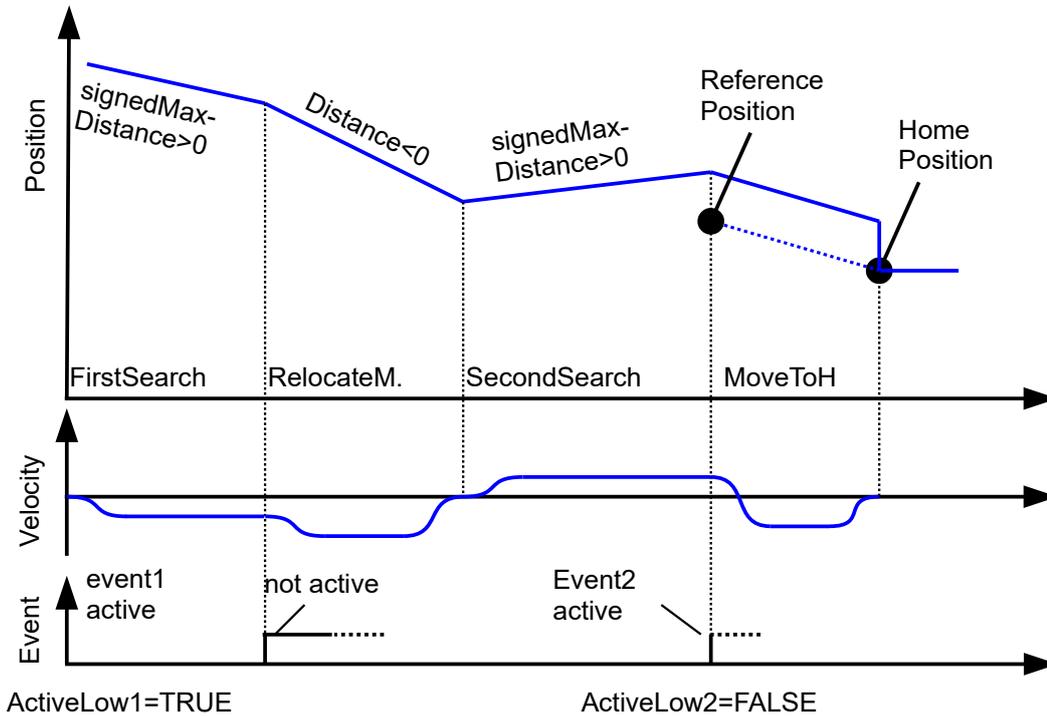
The homing of an axis is started with the command Start of **Axes[.].Commands.Homing.Command**. It can be interrupted with the command **Stop** of the same register. The register **TestNotEnabled** in the same folder allows testing the homing triggers by manually moving the axis. The states will behave as usual, but no motion commands are issued. **ReferencePosition** is the reference position as received from the control system. 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 encoder index. 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.



EventInput

8.2 Absolute Encoder Homing

The persistent position offset between absolute encoder and machine coordinate system can be stored in two ways. In the encoder itself or in the drive:

If the homing method **AbsoluteEncoderXOffsetEncoder** is selected, the position offset is loaded from the encoder EEPROM when the encoder is activated (after loading a configuration, resetting from an encoder error or during startup). If there is no such information in the encoder, the error **NoPersistencyData** is shown. The position offset is saved to the encoder with the command **SaveEncoder** ⁽³⁾ of the register **Command.Homing.Command**. Please note that this is currently only possible in Disabled state.

If the homing method **AbsoluteEncoderXOffsetDrive** is selected, the position offset is loaded from the drive persistent data when the encoder is activated (after loading a configuration, resetting from an encoder error or during startup). The position offset is saved to the drive persistency when saving the drive persistency.

8.3 Homing Method Immediate

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

8.4 Homing Method AtPosition

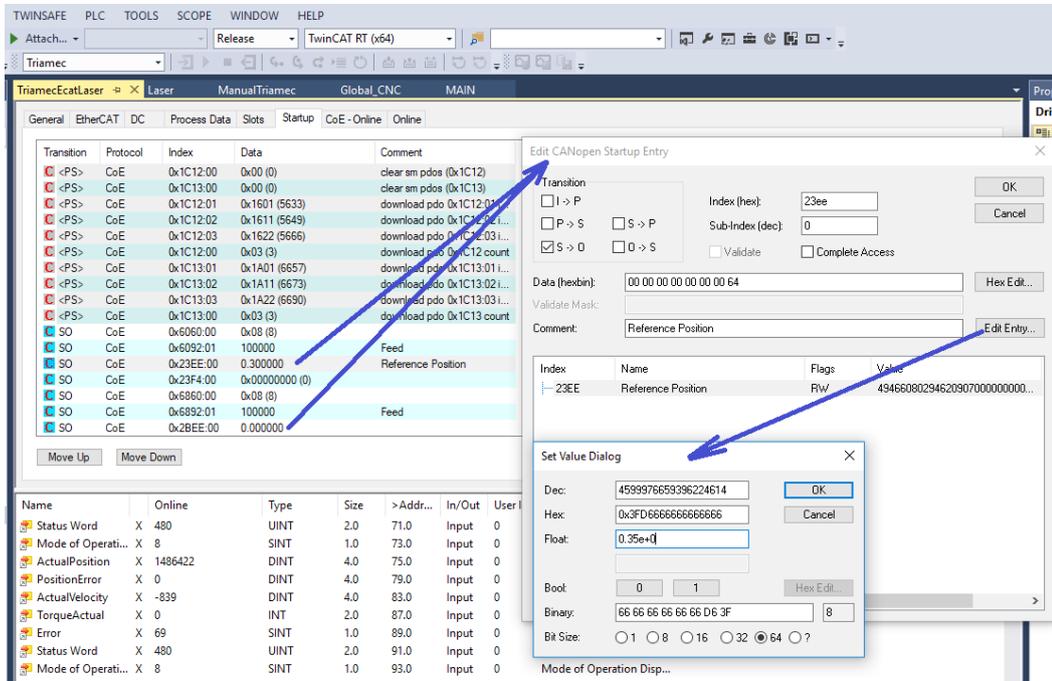
This homing method sets the actual position to the commanded value referencePosition without any move. Final state is **homingDone**.

³ With the command **InvalidateEncoder** the persistency data of the encoder may be cleared.

8.5 TwinCAT Homing with EtherCAT

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

! For correct use of EtherCAT Startup variables with 64 bit Floating point precision, use ESI descriptions Triamec1.8.xml or newer.



Homing from the CNC module

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".

8.6 TwinCAT Homing with the Tria-Link fieldbus

The homing procedure of an axis is started with a one-time call

```
gAxis[].referenceStart := TRUE;
```

while the HMI is in the mode Automatic or MDI.

Please note that (in contrast to the EtherCAT fieldbus), the standard CNC command G74 is not functional. Consider using the HMI function "Reference" by adding the following code in the PRG_CNCChannelHandler before the call to CNCChannel:

```
IF PLCMachineMode[nChan].Homing THEN
  PLCMachineMode[nChan].Homing := FALSE;
  AxisGroup.referenceStart := TRUE;
END_IF
```

Homing from the CNC module

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

8.7 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)
------------------------	---	---

9 TwinCAT interface of old gen. drives

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

9.1 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.

9.2 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.

10 Touch Probe Sequence (CNC)

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 11 for more information about an advanced touchdown detection controlled by a Tama program.

10.1 PLC-Example

10.1.1.Channel Parameters

In this example, the M-functons 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)
```

10.1.2.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.

10.1.3.Implementation

The following global constants are involved in the example:

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

The main functionality is implemented in the following sample code “FB_TouchProbe”. With this exam-

ple 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 old drive-generation
  positionLatchReg2 : TL_PositionLatchReg2; // used for new 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 // new 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 // old 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 // new drive generation
        positionLatchReg2.Source:= TL_ConstAxisParPosCtrlEncLatchSrc.AuxIn1;
        positionLatchReg2.EdgeFalling:= FALSE;
      ELSE // old 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
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);

```

10.2 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:
...
...
```

10.3 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.

11 Tama Controlled Touch Probe

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.

11.1 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:

11.1.1.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;
```

11.1.2.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.