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.
016	2021-12-07	sm	Extract Homing chapter to AN141, general formatting

Document	AN107_Encoder_EP
Version	016
Source	Q:\doc\ApplicationNotes\
Destination	T:\doc\ApplicationNotes
Owner	mvx

Copyright © 2021
Triamec Motion AG
All rights reserved.

Triamec Motion AG
Lindenstrasse 16
6340 Baar / 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....5
4.1 Incremental RS422....5
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 Encoder diagnostics and error config....9
8 Position Latching....9
8.1 Register Interface....9
8.2 EtherCAT interface....12
8.2.1. Trigger individual or 24V....12
8.2.2. Trigger simultaneous X20....13
8.2.3. Trigger simultaneous X21....13
8.2.4. Trigger simultaneous X10....14
8.2.5. Trigger simultaneous X11....14
8.2.6. Using the NCI module....15
8.2.7. Using the CNC module....15
8.3 Timing considerations....15
9 TwinCAT interface of old gen. drives....17
9.1 Fast Encoder activation....17
9.2 Endat....17
10 Touch Probe Sequence (CNC)....18
10.1 PLC-Example....18
10.1.1. Channel Parameters....18
10.1.2. Axis Parameters....18
10.1.3. Implementation....18
10.2 G-Code Example....21
10.3 Remarks....22
11 Tama Controlled Touch Probe....22
11.1 Un-coupling and Coupling....22
11.1.1. Preparation....23
11.1.2. Sequence....23
References....24

1 Overview

This application note mainly describes the encoder concept of Triamec Drives. This is valid for firmware >= 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_Axis0Standard → 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.

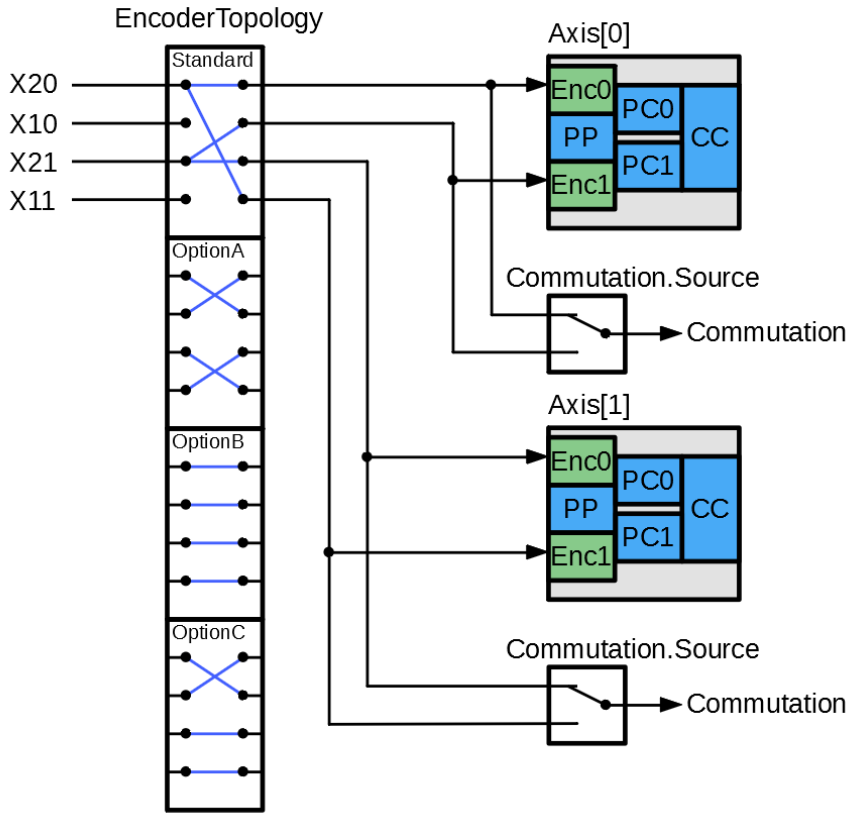

Figure 1: Routing of the EncoderTopology options.

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, $\overline{A}$, B, $\overline{B}$ 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 encIn0 and encIn1 for line counting. The direction is the same as in the RS422 case when using encIo0 as A-$\overline{A}$ and encIo1 as B-$\overline{B}$.

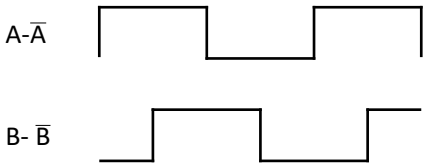

Figure 2: Positive counting direction for incremental encoders.

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. For diagnostics and the definition of any error reaction, consider chapter 7.

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:

Axes[].Parameters	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.
.Motor.EncoderCountsPerMotorRevolution	1	1250000
.PositionController.PositionUnits	Degree	mm
.PositionController.Encoders[].Pitch	360.0	0.000020

An absolute encoder supplies an absolute position. This can be used for homing and commutation functions. For persistent commutation of absolute encoders, see [1]. For diagnostics and the definition of any error reaction, consider chapter 7.

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. For persistent commutation of absolute encoders, see [1]. For diagnostics and the definition of any error reaction, consider chapter 7.

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 Encoder diagnostics and error config

For diagnostics, consider the following bits in ***Axes[].Signals.PositionController.Encoders[].ErrorFlags***

- 0x001 AmplitudeError : The sinCos amplitude is smaller than 25%
- 0x002 AmplitudeWarning : The sinCos amplitude is smaller than 50%
- 0x004 DigitalFlag1
- 0x008 DigitalFlag2
- 0x010 Crc
- 0x020 Communication
- 0x040 OverrangePhaseA
- 0x080 OverrangePhaseB

The 0 to 1 transition of a bit in this list will increment the corresponding diagnostic counter in Axes[].Signals.PositionController.Encoders[].Diagnostics.Counters. Additionally, it will generate a log entry. This entry is either a warning or an error, depending on the mask Axes[].Signals.PositionController.Encoders[].Diagnostics.ErrorMask.

The error mask is typically 0x01 for Analog Encoders and 0x0C0 for DigitalEncoders. To activate, for example, the additional errors OverrangePhaseA and B of an Analog encoder use Axes[].Parameters.PositionControllers.Encoders[].DataFormat = ***"E0C1"***. If this string is not empty, add ***"-EHHH"*** to the end of the string, where HHH is the hex representation of the bits in the above list. This feature is available for firmware > 4.13.7.

8 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 8.1), the EtherCAT interface (chapter 8.2) and the timing.

8.1 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.

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

For example if *EncIn0* 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 = ***EncIdx0***
- 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 the signal Signals.PositionController.PositionLatchStandard.State changes to ***Found***.
- Set Commands.PositionController.PositionLatchStandard.Enable to ***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.

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

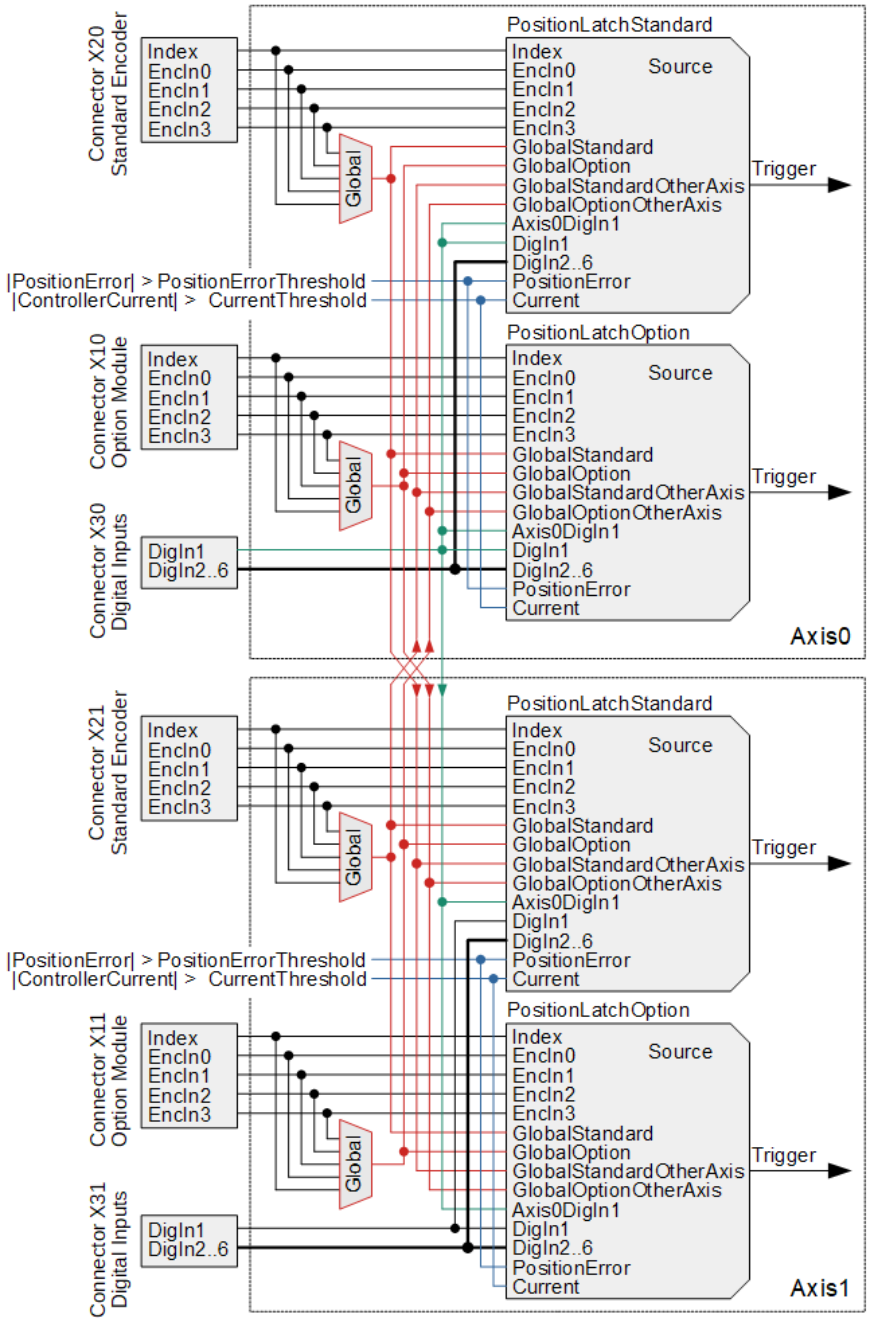

Figure 3: Schematic of the position latch triggers and theirs selectors.

8.2 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 8.2.1) or simultaneous from a common trigger source (Sections 8.2.2 to 8.2.5). 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 8.2.1) .

8.2.1. Trigger individual or 24V

Choose this configuration, 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
EncIn0	0x0000’0002	0x0000’1002
EncIn1	0x0000’0003	0x0000’1003
EncIn2	0x0000’0004	0x0000’1004
EncIn3	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.

8.2.2. 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
EncIn0	0x0000'020A	0x0000'120A
EncIn1	0x0000'030A	0x0000'130A
EncIn2	0x0000'040A	0x0000'140A
EncIn3	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

8.2.3. 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
EncIn0	0x0000'020A	0x0000'120A
EncIn1	0x0000'030A	0x0000'130A
EncIn2	0x0000'040A	0x0000'140A

EncIn3	0x0000’050A	0x0000’150A

8.2.4. 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

Trigger Source	***Position from the Standard encoder***	***Position from the Option encoder***
Index	0x0000’010B	0x0000’110B
EncIn0	0x0000’020B	0x0000’120B
EncIn1	0x0000’030B	0x0000’130B
EncIn2	0x0000’040B	0x0000’140B
EncIn3	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

Source	***Position from the Standard encoder***	***Position from the Option encoder***
GlobalTrigger of the option encoder of the other axis	0x0000’000D	0x0000’100D

8.2.5. 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

Source	***Position from the Standard encoder***	***Position from the Option encoder***
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

Trigger Source	***Position from the Standard encoder***	***Position from the Option encoder***
Index	0x0000’010B	0x0000’110B
EncIn0	0x0000’020B	0x0000’120B
EncIn1	0x0000’030B	0x0000’130B
EncIn2	0x0000’040B	0x0000’140B
EncIn3	0x0000’050B	0x0000’150B

8.2.6. 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.

8.2.7. 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
```

8.3 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 ***encIn0*** to ***encIn3*** 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 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.

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-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)
```

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;
   CNCSystem        : 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
```

```
   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.

References

[1] “Servo Drive Setup Guide, TSD and TSP Series”, ServoDrive-SetupGuide_EP0013.pdf, Triamec Motion AG, 2021.