Encoder Configuration

Application Note 107

This application note describes the encoder concept, related functionality and settings of *Triamec* drives.
This document is valid for firmware versions ≥ 4.22.

Table of Contents

Document	AN107_Encoder_EP
Version	024, 2024-04-16
Source	Q:\Doc\ApplicationNotes\
Destination	T:\doc\ApplicationNotes
Owner	mvx

1 Position Units

The units of an axis are specified using

Axes[].Parameters.PositionController.PositionUnit

Possible settings are currently meters (m), millimeters (mm), radians (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. Use Axis[].Commands.General.Event = ChangeUnits 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.

2 System Configuration

Hardware View: Up to four encoders can be connected, if a drive is equipped with two encoder *Option Modules* [2]. 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
- X11 Option encoder input for axis 1

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 in 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 in Encoders[0].
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 in Encoder[1].
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 in Encoders[0] and the option module of Axis[1] is available in Encoders[1].
OptionD	X10_Axis0Option →Axes[0]/Encoders[0] X11_Axis1Option →Axes[0]/Encoders[1] X20_Axis0Standard →Axes[1]/Encoders[0] X21_Axis1Standard →Axes[1]/Encoders[1]	The option module of Axis[0] is available in Encoders[0] and the option module of Axis[1] is available in 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*** encoder topology configuration: The software parameters (including the type) may only be configured once 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.

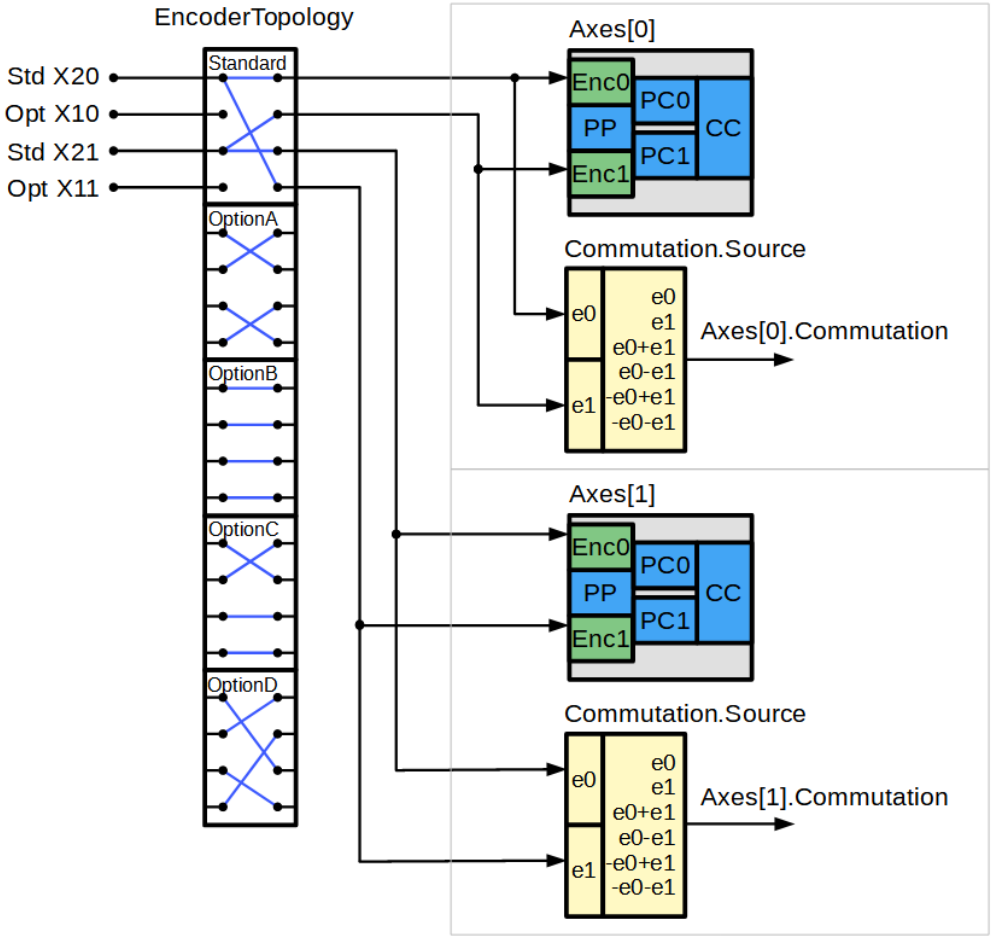

Figure 1: Routing of the EncoderTopology options.

3 Analog Encoders

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.

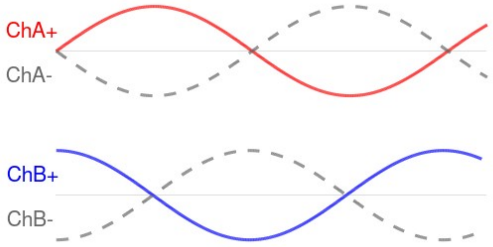

Figure 2: Positive counting direction for analog encoders.

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

The parameters are set as for incremental encoders (chapter 4).

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

4 Incremental 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

In case of linear axes, the value for EncoderCountsPerMotorRevolution has to be rounded to the nearest integer value.

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

4.1 Incremental RS422

This type uses the complementary *ChA+*, *ChA-* and *ChB+*, *ChB-* inputs of the encoder for line counting. See figure 3 for the positive counting direction.

The same configuration applies for the *Incremental RS422 Fast* type.

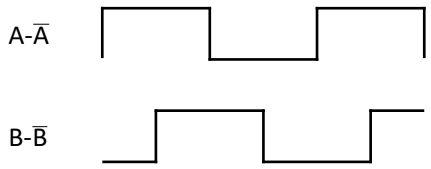

Figure 3: Positive counting direction for incremental encoders.

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 *EncIn0* as channel A and *EncIn1* as channel B.

5 Digital 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	Rotary	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.

In case of linear axes, the value for EncoderCountsPerMotorRevolution has to be rounded to the nearest integer value.

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

5.1 Digital EnDat

This type reads the cyclic position data using the *EnDat* protocol. The position update rate is limited to 50kHz.
Prefer *AnalogEndat* if sin/cos signals are provided by the encoder (see also chapter 6.2).

5.2 Digital BiSS

This type reads the cyclic position data using the *BiSS* protocol. The position update rate is limited to 20kHz.
Prefer *AnalogBissB* if sin/cos signals are provided by the encoder (see also chapter 6.3).

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.***

Per default, *BiSS-C* encoders run at 2MHz clock speed. Other clock speeds are available by appending the corresponding string to the DataFormat: "-F4MHz" for 4MHz clock, or "-F5MHz" for 5MHz clock.

5.3 Digital Tamagawa

Same setup as for *BiSS* encoders, see chapter 5.2. The position update rate is limited to 20 kHz.

5.4 Digital Nikon

Use a data format string "16-20-F2.5MHz" to specify an encoder with 16 bits multi-turn resolution, 20 bits single turn resolution and a clock frequency of 2.5 MHz.

Available clock frequencies are 2.5 MHz and 4 MHz with corresponding position update rates of 20kHz and 50 kHz.

6 Digital Encoders with sin/cos signals

These encoders contain 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 analog encoders, see chapter 3.

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 digital resolution. Therefore, we use the digital information just for line counting and the analog sin/cos information for the sub-resolution. The alignment fails if the digital information is not consistent with the analog information. The error *EncoderSubResolutionError* will be shown in this case.

As of firmware 4.7, this alignment step 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 Analog EnDat

This type reads the absolute position using the *EnDat* format (see 5.1) and the analog sin/cos signals for precise and fast position updates.

6.3 Analog BissB and BissC

This type reads the absolute position using the *BiSS* format (see 5.2) and the analog sin/cos signals for

precise and fast position updates.

The unit of this encoder type is based on 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 multi-turn and ST=24 bits single turn, will not be specified with “12-24”, as for a digital-only *BiSS*. Instead the multi-turn part is the sum of MT+K and the single turn part is the difference of ST-K, resulting in DataFormat = “23-13”.

7 Encoder Diagnostics

For diagnostics, the following bits are available:

Bit	Name	Description
0x001	AmplitudeError	The sin/cos amplitude is smaller than 25%
0x002	AmplitudeWarning	The sin/cos amplitude is smaller than 50%
0x004	DigitalFlag1	The encoder specific flag1 in the datagram is set
0x008	DigitalFlag2	The encoder specific flag2 in the datagram is set
0x010	DigitalCrc	The datagram checksum is not valid
0x020	DigitalCommunication	Datagram frame error
0x040	OverrangePhaseA	The analog encoder phase A voltage is out of the ADC range
0x080	OverrangePhaseB	The analog encoder phase B voltage is out of the ADC range
0x100	QuadrantError	The analog AB quadrant changed from either 0↔2 or 1↔3 (FW>=4.21)

Table 1: Encoder Error Bits

The 0 to 1 transition of a bit in this list will trigger the following actions:

- Increment the diagnostic counter signal, matching the encoder input: Axes[].Signals.PositionController.Encoders[].Diagnostics.Counters.
- Depending on how the ErrorMask is configured, the bit either triggers a warning or an error (see next chapter).
- Generate a log entry.

7.1 Mask Configuration

Three masks are configurable to define the behavior of the diagnostic bits in Table 1.

Mask	Prefix	Description
Inversion	I	Change the trigger from rising edge to falling edge (since FW 4.19).
Warning	W	The trigger throws a warning (since FW 4.19), see [3]: 4738, 4740, 4742, 4744, 4746, 4748, 4750
Error	E	The trigger throws an error, see [3]: 6848, 6850, 6852, 6854, 6856, 6858, 6860

Table 2: Encoder diagnostic mask types

The diagnostic masks are set up with encoder type dependent defaults. The currently active mask is displayed in Axes[].Signals.PositionController.Encoders[].Diagnostics.ErrorMask.

To change the mask, set the configuration string in the register:
Axes[].Parameters.PositionControllers.Encoders[].DataFormat.

Define the string by adding up the hex values in Table 1 and prepend the letter according to Table 2. If there are other values in the register already, append a – (minus) before the mask string.

For analog encoders, the default error mask is **0x1C1** (FW>=4.22).

Example 1:

- No value in DataFormat before.
- Trigger error on *QuadrantError*, *AmplitudeError*, *OverrangePhaseA* and *OverrangePhaseB*

error mask string results in **E1C1**:

Register	Actual	Prepare
DataFormat		E1C1

Example 2:

- *BiSS-C* configuration value in DataFormat.
- Trigger error on *DigitalFlag1*

error mask string results in **-E004**:

Register	Actual	Prepare
DataFormat	12-24	12-24-E004

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.

8.1 Register Interface

Some inputs can be used as global trigger to simultaneously latch positions of Axes[0] and Axes[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:

- ***Source**:*
  Specifies the digital input trigger source used for this module (see Figure 4).
- ***Type**:*
  Defines if *RisingEdge* or *FallingEdge* sets the trigger.
- ***Global**:*
  Can be used to define an input of this module as a global trigger (see Figure 4). 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 *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 = EncIn0
  - 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.
- ***ExternalInputBit0***:
  If the source register is set to ***ExternalInputBit0***, the position latch will be triggered by bit0 of the register Axes[].Commands.General.ExternalInput. This register can be cyclically written by the control system or Tama to allow latching or homing based on external events.

The settings listed above are commands and therefore not stored persistently.

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

- Set the registers Commands.PositionController.PositionLatchStandard.Source, Global and Type.
- Set Commands.PositionController.PositionLatchStandard.Enable to True to start searching.
- The signal Signals.PositionController.PositionLatchStandard.State.
  changes from Disabled to Preparing and then stays on Search until the trigger is received.
- The signal Signals.PositionController.PositionLatchStandard.Trigger shows the actual state of the trigger input. A special feature will reveal EMI (see also chapter 8.3).
- 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.

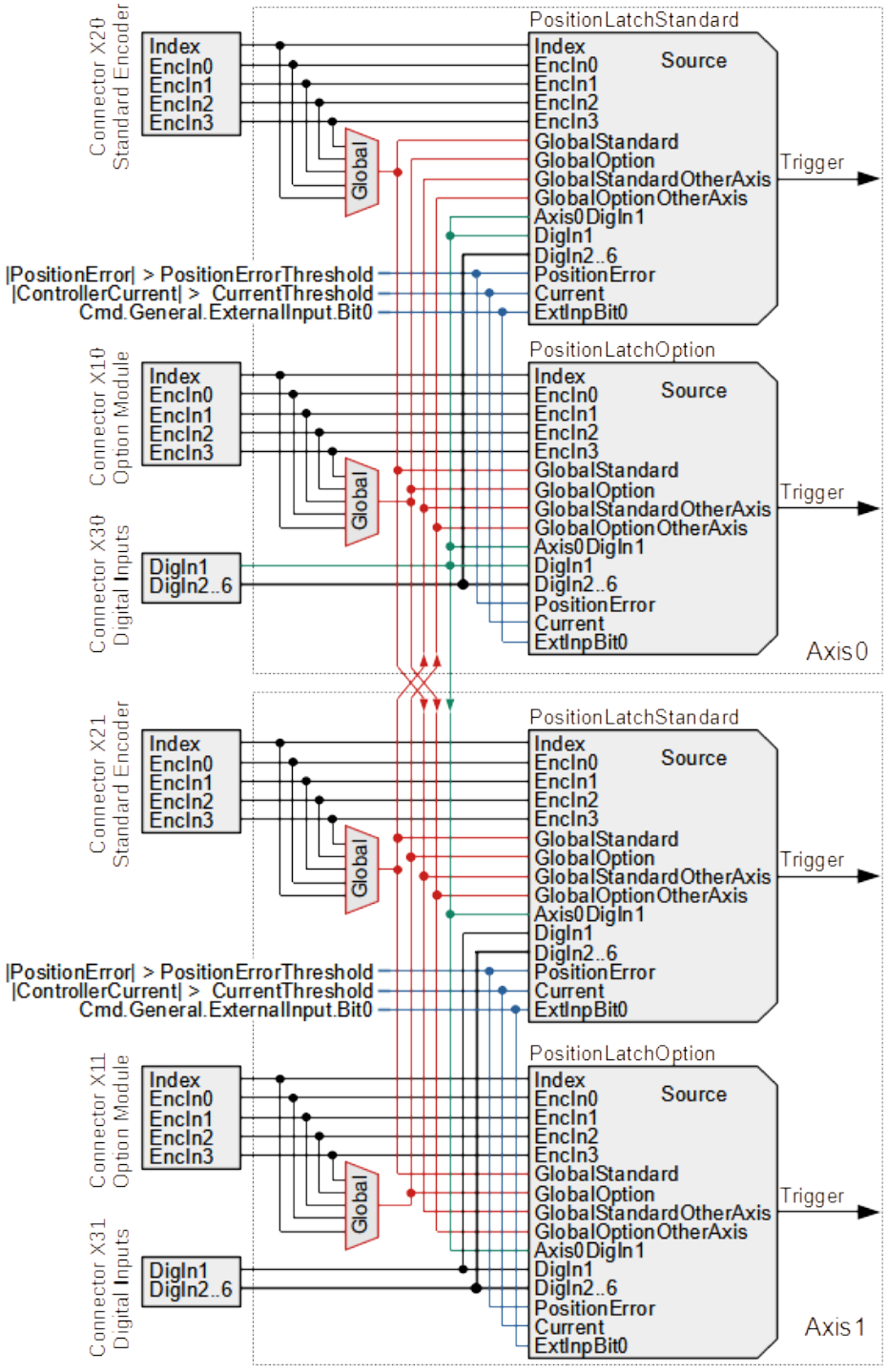

Figure 4: Schematic of the position latch triggers and theirs selectors.

8.2 Timing considerations

The 24V digital inputs *X30* and *X31* are slow and not recommended for fast position latching. The reaction time (jitter) is 300µs for *DigIn1&2* and 1200µs *DigIn3..6*. Only the digital input *DigIn1* of axis 0 has a fast response time similar to the EncIn, see below.

The fast digital *TTL* inputs *EncIn0* to *EncIn3* are available at every encoder connector.

The timing accuracy (jitter) of the EncIn and DigIn1 of axis0 is 0.2µs, dominated by the encoder path signal read. Besides of the accuracy there is also a systematic delay. The delay on the trigger input is small (0.01µs for EncIn and 0.1µs for Axis0.DigIn1). 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 for by considering the speed of the axis in the search phase.

8.3 EMI Detection Feature

The *Position Latch Unit* has a built in tool to detect electromagnetic interference (*EMI*) on the configured trigger source. If there is any spurious peak or noise on the electric input signal, the trigger state will toggle every 100µs. This results in a square wave trigger signal.

To analyze the connected input for *EMI*, the following steps are recommended.

- First configure the trigger signal source as described in chapter 8.1.
- Depending on the configured trigger signal, scope one of the following signals.
  - Axes[].Signals.PositionController.PositionLatchStandard.Trigger, or
  - Axes[].Signals.PositionController.PositionLatchOption.Trigger
- Check if the trigger signal shows square waves by recording it with the scope.
- It Is recommended to analyze the signal with and without enabled motors. If the enabled motor shows interference on the trigger signal, check the motor cable shielding and sensor cable shielding.

9 Touch Probe

9.1 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 drive in TwinCAT. Go to the tab **Slots** and select the axis. Assign an option including **touch probe**. This will add the additional signals *TouchProbeStatus (0x60b9)*, *TouchProbePosition1Pos (0x60ba)* and *TouchProbePosition1Neg (0x60bb)*. It also adds the command word *TouchProbeFunction (0x60b8)*.

Note Only single capture mode is available.

The trigger source of the touch probe unit does not comply with *CAN* specifications. Use the (non-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 4. 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 9.1.1) or simultaneous from a common trigger source (Sections 9.1.2 to 9.1.5). 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 9.1.1).

9.1.1. 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
Axis0DigIn1 (fast input)[1]	0x0000'000E	0x0000'100E
DigIn1	0x0000'0014	0x0000'1014
DigIn2	0x0000'0015	0x0000'1015
DigIn3	0x0000'0016	0x0000'1016

1 The 24V input DigIn1 of axis 0 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.

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

9.1.2. 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

9.1.3. 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

9.1.4. 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

9.1.5. 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

9.1.6. Using the NCI module

In the *PLC* section, include the library `Tc2_MC2` and the following declarations.

```
axis0 AT %I*    : AXIS_REF;              // if not using triamec sample code
touch           : MC_TouchProbe;
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.NcToPlc` to the *NCI* module axis component `ToPlc`.

If the *Triamec* library is used, the AXIS_REF is already included as `axis0.nci`. Delete the line for `axis0` in the declaration section above and use `axis0.nci.ReadStatus();` instead in the code above.

9.1.7. 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 X25 F1
G01  X V.A.MESS.X F10
```

9.2 Tria-Link Interface (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 re-couple the axis.
See chapter 9.3 for more information about an advanced touchdown detection controlled by a Tama program.

9.2.1. PLC Example

Channel Parameters

In this example, the M-functions 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 axes.

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 `Axis0DigIn1` 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
         positionLatchReg2.Source:= TL_ConstAxisParPosCtrlEncLatchSrc.Axis0DigIn1;
         positionLatchReg2.EdgeFalling:= FALSE;
         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);
```

9.2.2. G-Code Example

The following example uses the `G310` command to execute the touch probe move [4]. 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:
...
...
```

9.2.3. 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 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.

9.3 Tama Controlled Stop

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

```
if (Register.Axes_0.Signals.PositionController.PositionLatchStandard.State >= PositionLatchState.Search &&
    Register.Axes_0.Signals.PositionController.PositionLatchStandard.Trigger &&
    Register.Application.Variables.Booleans[0])
{
    Register.Axes_0.Commands.PathPlanner.Command = Triamec.TriaLink.PathPlannerCommand.EmergencyStop;
}
```

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

9.3.1. Uncoupling and Coupling

To avoid an error when an axis uncouples, 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 interpolator the following command has to be added to the G-code after touchdown detection.

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

- The axis-parameter *P-AXIS-00258* has to be set to 1:

  ```
  kenngr.tracking_offset_remain   1  ( P-AXIS-00258 : Keep position offset after tracking )
  ```

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

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

- Therefore, override the command `TorquePermission` before the coupling will be broken. E.g.

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

- The following command must remain TRUE until CNC has stopped.

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

- Therefore, override of the `followMe` state. 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 `followMe` mode. E.g.:

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

Sequence

- Wait for the *CNC* to request the preparation of the touch probe move by an 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 move (e.g. G310) and set `overrideTorquePermission` to `TRUE`.
- The touchdown signal has to be provided to the *CNC*.
- Wait until the 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 M-function acknowledged and the sequence is done in this case.
- If a touchdown is detected, set `overrideAxisTracking` to `TRUE`.
- Wait until all axes are in tracking mode (follow up mode).
- Set `overrideAxisTracking` to `FALSE` and reactivate the coupling by setting the couple command from `FALSE` to `TRUE`.
- Wait until all axes are coupled.
- Set `executeTouchdown` and `overrideAxisTracking` to `FALSE`, then acknowledge the M-function.

10 Switching Encoders

There are applications where different encoders are needed for two tasks on the same axis. When switching encoders, also the controls have to adapt. This chapter describes the setup requirements and the switching procedure.

10.1 Setup and Requirements

The main requirement is that two encoders are connected and set up so that:

- Both encoders route to the same axis (see Chapter 2), and
- Both encoders are configured in Parameters.PositionController.Encoders[0] and Encoders[1].

With both encoders configured, also both Controllers[0] and Controllers[1] are active. The axis must be fully commissioned with both controllers tuned correctly.

WARNING As soon as position controller parameters are set, the control system becomes active.

10.1.1. Checklist for Commissioning

- Set the General.Parameters.EncoderTopology matching the wiring and the application.
- Commission and tune Axes[0] with Encoders[0] and Controllers[0].
- Set Axes[].Parameters.PositionController.Encoders[0].Type to None.
- Set the parameters for Encoders[1].
- Set Axes[].Parameters.Motor.EncoderCountsPerMotorRevolution to match Encoders[1].
- Set Axes[].Parameters.Commutation.Source to Encoder1.
- Tune the position controller Controllers[1].

NOTE By setting the Encoders[i].Type to None, also the corresponding Controllers[i] is disabled.

10.2 Switch Encoders in Process

To switch encoders, parameter changes and commit commands are necessary. This can be established manually with the *TAM System Explorer*. To command an encoder switch through software, the same sequence is necessary, as if done manually. The sequence can be commanded with all supported control methods, which are:

- A main controller (*PLC*), connected through a field bus (*EtherCAT* or *Tria-Link*).
- A *Tama* implementation on the drive.
- An application using the *TAM SDK*, connected through any of the available interfaces.

10.2.1. Checklist for Sequence Implementation

The parameter indexes in the following checklist refer to an example that switches from Encoders[0] to Encoders[1] on Axes[0].

- Disable the axis.
- Set Axes[0].Parameters.Motor.EncoderCountsPerMotorRevolution to match Encoders[1].
- Set Axes[0].Parameters.Commutation.Source to Encoder1.
- Set Axes[0].Parameters.PositionController.Encoders[0].Type to None.
- Set Axes[0].Parameters.PositionController.Encoders[1].Type to the corresponding type.
- Set Axes[0].Parameters.PositionController.MasterPositionSource to Encoder1.
- Commit CurrentController changes.
- Commit PositionController changes.
- Enable the axis.
- Run the homing routine.

WARNING Switching encoders and/or controllers invalidates the homing, as the position cannot be tracked. Therefore it is mandatory to run a homing routine after the encoder switching.

References

[1] “Servo Drive Setup Guide”, ServoDrive-SetupGuide_EP018.pdf, Triamec Motion AG, 2022.

[2] “Option Modules Manual”, HWTO_OptionModulesManual_EP013.pdf, Triamec Motion AG, 2022.

[3] “Drive Messages”, AN102_DriveMessages_EP005.pdf, Triamec Motion AG, 2023.

[4] “Measure with interruption and jump (G310) (Types 5, 6)”, https://infosys.beckhoff.com/content/1033/tf5200_programming_manual/206197003.html, 30.06.2023

Revision History

Version	Date	Editor	Comment
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
017	2022-03-04	sm	Update EncoderTopology image
018	2022-04-22	mvx	Describe Latch trigger signal
019	2022-06-07	sm	Update template, Add chapter for EMI detection feature.
020	2023-01-09	sm	add chapter “Switch Encoders”, update to current firmware requirements.
021	2023-03-16	dg	New EncoderTopology OptionD introduced.
022	2023-06-30	sm	Introduce encoder diagnostics mask “Inversion” and “Warning” with FW 4.19
023	2023-09-12	mvx, sm	Timing of Axis0DigIn1. Latching with ExternalInputBit0, restructure TouchProbe chapters
024	2024-04-15	mvx	New bit “QuadrantError” in encoder diagnostics with FW 4.22

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.