Motion Control Products

Application note
Using the ABB motion drives for master encoder input

AN00241

Rev C (EN)

Leverage the power and flexibility of EtherCAT to connect one or more master encoders to your AC500 motion system via the auxiliary encoder channels available on the ABB motion servo drive products

Introduction

AC500 PLCs (PM585 and PM59x) can be used to perform real-time motion control of ABBs EtherCAT enabled servo drives. In some applications it is necessary to synchronize the motion of these drives to a master encoder (e.g. cut to length applications using a lay-on encoder to track the movement of the material to be cut to length).

This application note details how to use Automation Builder to define the hardware setup suitable for use of any available encoder channel on a connected MicroFlex e190 or MotiFlex e180 servo drive as a master encoder and how to then use the encoder value provided by the drive to create a (virtual/simulated) master axis in a PLCopen motion application. This offers some performance benefits over the use of a CD522 2 channel encoder module…

- Encoder input frequency extended up to 8MHz (quadrature), depending on encoder channel used, compared to 300kHz for the CD522
- Encoder position update synchronized with EtherCAT cycle (leading to smoother motion from geared axes)
- Ability to filter touchprobe (latch) data via drive parameters (leading to faster and simpler PLC application code development)

This document also details how the touchprobe function block included with the PS552-MC-E motion libraries can be used in combination with the e190 and e180 drives to latch a connected master encoder position. A sample Automation Builder project is included to illustrate all of the topics covered by this application note.

Pre-requisites

You will need to have the following to work through this application note and perform synchronized motion on an EtherCAT drive:

- Mint Workbench build 5860 or later (see new.abb.com/motion for latest downloads and support information)
- A MicroFlex e190 or MotiFlex e180 drive with build 5868 or later firmware
- A PC or laptop running Automation Builder 2.1.1 or later
- An installed (and licensed) copy of the ABB PLCopen motion control library (PS552-MC-E v3.2.0 or later)
- One of the following AC500 PLC processors…..PM585, PM590, PM591, PM592 or PM595 (PLC processors should be running firmware version 2.5.1 or later). The PM595 is provided with an integrated EtherCAT coupler (this should be running firmware version 4.2.32.2 or later). All other processors require a CM579-ECAT communication module (which must be running firmware version 2.6.9 or later, but ideally version 4.3.0.2 or later). Contact your local ABB PLC support team for details on how to check these requirements and update if necessary or visit http://new.abb.com/plc/programmable-logic-controllers-plcs and select the link for ‘Software’. For the purposes of the text in this application note we have assumed the use of a PM591 PLC with CM579-ETHCAT coupler
- Straight-through Ethernet cable (patch cable) to connect the EtherCAT coupler to the drive (do not switch between patch and cross-over cabling as this can affect the order of devices on EtherCAT which is critical)

- A copy of application note AN00205 (AC500 and motion drives - EtherCAT Getting Started Guide) and the Automation Builder PLC project that is included with it
- A copy of application note AN00242 and the export file that is included with it
- A RS422 (5v differential line driver) incremental encoder

To follow the basic steps to create a hardware configuration and write some PLC code that uses the encoder values from the drive only requires a PC or laptop running Automation Builder 2.1.1 or later, an installed copy of the PS552-MC-E motion control libraries and the export file included with application note AN00242. It is assumed the reader has a basic working knowledge of Mint Workbench, Automation Builder, CoDeSys and the AC500 PLC and that if you intend to perform motion the reader has read and understood the contents of application note AN00205, which is also available for download from new.abb.com/motion, and has commissioned an EtherCAT based servo drive (MicroFlex e190 or MotiFlex e180 for example) ready for use with the AC500 PLC.

Connecting the master encoder

MicroFlex e190 drives are provided with 3 encoder channels…

- Channel 0 – the universal encoder input (supporting encoder, encoder with halls, SSI, Biss, Smartabs, Endat 2.1, Endat 2.2, Sin/Cos and resolver via an external adaptor)
- Channel 1 – an incremental encoder formed by fast inputs 1 and 2 when setting ENCODERMODE(1) to support encoder type operation on these inputs
- Channel 2 – an incremental encoder achieved via the unused hall sensor inputs on the universal encoder input when encoder channel 0 is set for any of the digital encoder types (SSI, Biss, Smartabs, Endat 2.2) or for the external resolver adaptor

MotiFlex e180 drives are also provided with 3 encoder channels…

- Channel 0 – the motor feedback input (X13). Encoder support varies according to the particular feedback module fitted to the drive
- Channel 1 – an incremental encoder formed by fast inputs 1 and 2 when setting ENCODERMODE(1) to support encoder type operation on these inputs
- Channel 2 – an incremental encoder input (X11)

Any of these encoder channels can be utilized as a master encoder input providing it is available for connection/use (e.g. not being used for the motor feedback) and the signal levels from the encoder to be used are compatible with the encoder channel selected. Please refer to the appropriate drive installation manual or further details if necessary.

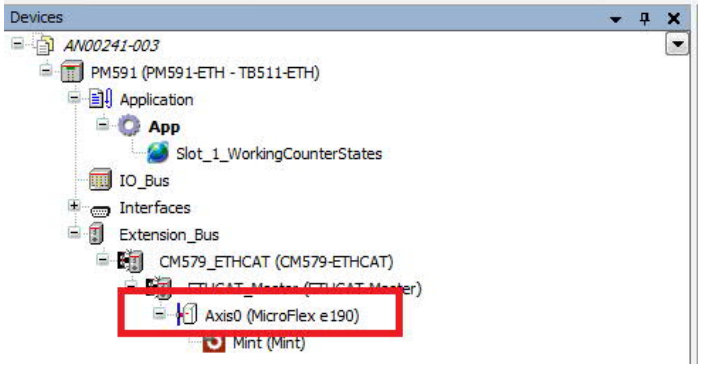
For our application note example we will assume a MicroFlex e190 drive is being used to control an ESM series motor fitted with a Smartabs feedback device and that a RS422 differential line driver encoder is connected to encoder channel 2 via the Encoder 2 connector at X7 on the drive. The principles are identical for a MotiFlex e180 so we won't explain using that separately, the process should be obvious enough.

Automation Builder – Adding the additional encoder mapping

Throughout this application note we will assume that the reader has opened the Automation Builder project supplied as part of AN00205 and we will just illustrate the additional steps needed to add and configure/use the encoder input on the drive. Alternatively a ready-made example project is available as part of this application note.

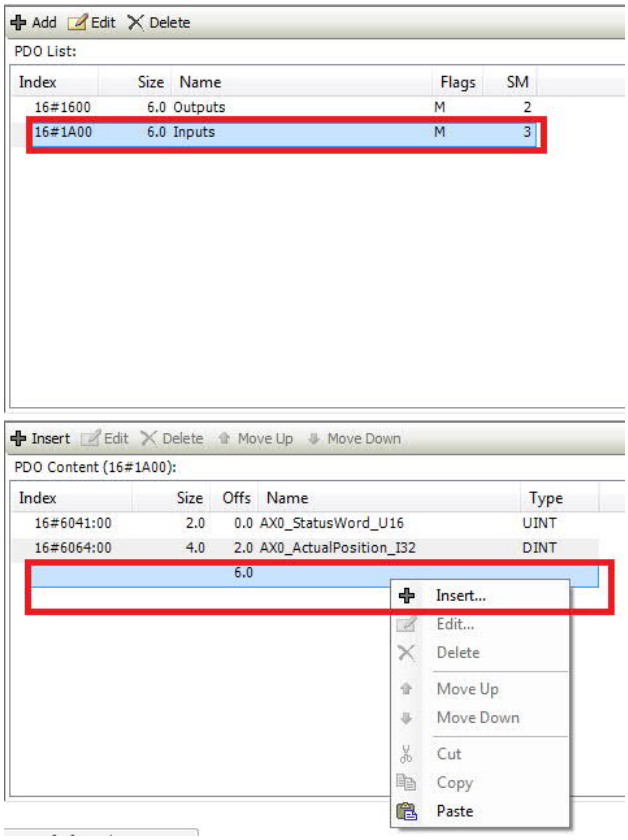
Open the PLC project provided as part of AN00205 from the Automation Builder 'File>Open Project…' menu and once opened select 'File>Save Project As…' to give your project a new name (to avoid damaging the original project).

Expand the Devices tree if necessary and in turn expand the Extension_Bus icon and all its sub-elements until you reach the icon for the MicroFlex e190 drive…

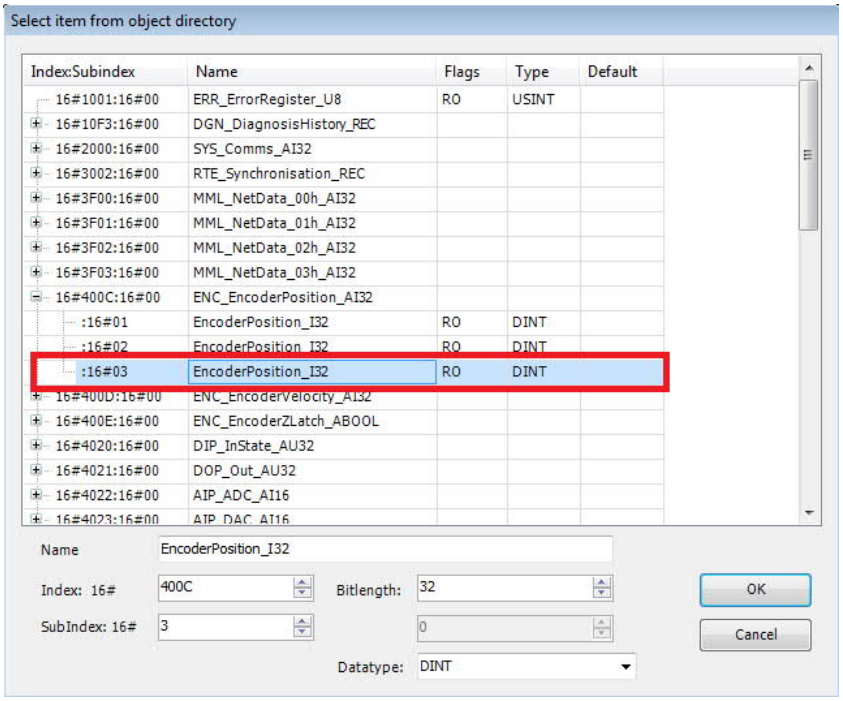


Double-click this icon to display the settings in the right-hand pane. “Enable Expert Settings” needs to be selected on the General tab (this should already be the case if you have used the project from AN00205 as the starting point or if you have opened the example included with this application note).

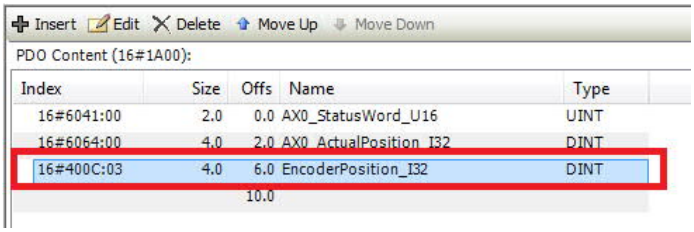
We need to add a new PDO mapping for Encoder channel 2 (to be read from the drive by the PLC) so select the ‘Expert Process Data’ tab in the right hand pane, click on ‘Inputs’ in the top right window and then right click the blank line underneath the PDO mapping for ‘AX0_ActualPosition_I32’ in the bottom right hand window…



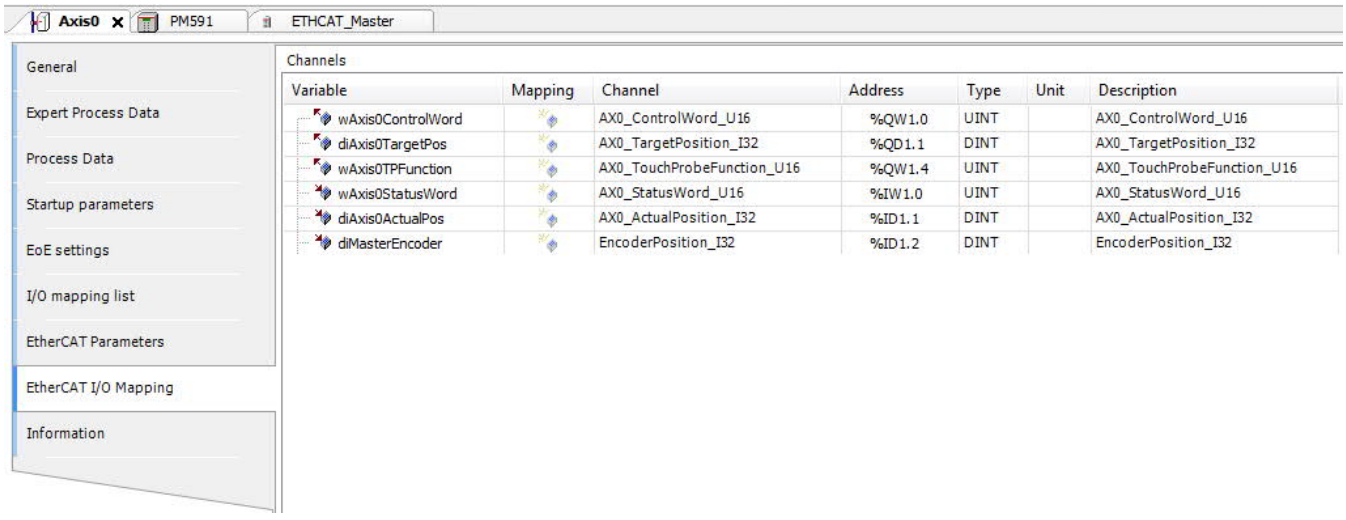
Now click on ‘Insert…’ and select Object 14#400C sub index 16#03 from the list of available objects (Object 16#400C is ‘Encoder position’ and the subindexes relate to each encoder channel…..subindex 16#01 is channel 0, subindex 16#02 is channel 1 and subindex 16#03 is channel 2).



Click OK and this object will be added to the list of PDO mappings from the drive to the PLC.



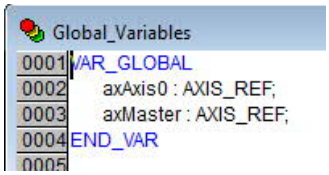
We now need to assign a name to this mapping so click on the 'EtherCAT I/O Mapping' tab in the right hand pane and assign a name to the Encoder Position mapping. We called ours diMasterEncoder as shown below…..



Save the project and launch CoDeSys (accept the confirmation to update the configuration). We can now create our master axis and use this mapped encoder to set the position of our master axis.

PLC application – Adding the code for the master axis

The first thing we need to do is add a new AXIS_REF to the code for our master axis. Go to the 'Resources' tab and open the Global_Variables window. We can now add a new AXIS_REF definition for our master axis. We named ours 'axMaster' as shown below…



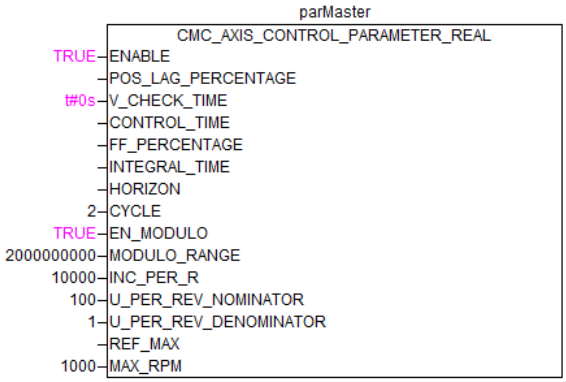
It is logical that the master axis position/profile would require updating before calculating the target position for any geared axes so we will add the code required for our master axis to the beginning of our EtherCAT_Control program. Open the EtherCAT_Control program (or whatever your program called by the EtherCAT task is named) and select network 0001 and right click the grey area to the left. Select 'Network before' to insert a new network at the start of the program.

Add a new network 0001 and include a CMC_AXIS_CONTROL_PARAMETER_REAL function block in this network (we named ours parMaster). This block will set the various control parameters for our master axis (e.g. scaling and modulo). The table below details the input parameters that must be set – all other input parameters can be left blank…

Input parameter	Description	Value/assignment	Data type
ENABLE	Enables processing of the function block	True	BOOL
V_CHECK_TIME	Delay time for velocity monitoring	T#0s	TIME
CYCLE	Cycle time for axis update. This should match the EtherCAT cycle time we've set for the EtherCAT master in the Automation Builder Devices tree	2	LREAL
EN_MODULO	This parameter should be set TRUE if the axis is to be treated as a modulo axis (i.e. if there a wrap on the axis position at a predefined range or if the axis position will wrap past the 32 bit position boundary eventually such as with a continuously rotating unidirectional axis). If the axis only moves forwards/backwards within the 32 bit position range and there is no requirement for modulo functionality then set this parameter FALSE. For this application note we will set it TRUE to create a master axis that runs in the forward direction continuously (e.g. if the master encoder were being used in conjunction with a unidirectional conveyor)	TRUE	BOOL
MODULO_RANGE	Used in combination with EN_MODULO. This parameter sets how many encoder counts there are in one modulo cycle. If EN_MODULO is set to FALSE or if there is no requirement to define a modulo/cycle size for the master axis then it is suggested to set this value close, but not equal, to the maximum size (2147483647)…e.g. 2000000000. For an application where the master axis has a repeating cycle (e.g. a virtual master axis for a die cutting or printing machine) set this value to the number of encoder counts in the master axis cycle (this will depend on the resolution of the master encoder and the gearing to this encoder from the mechanics). For our example we will assume we don't have a master cycle so we will set this parameter to 2000000000	2000000000	DINT
INC_PER_R	Number of encoder counts in one revolution of the master encoder (in quadrature). For our example we will assume a 2500 line encoder (so 10000 quadrature counts)	10000	DWORD

U_PER_REV_NOMINATOR	Together with the DENOMINATOR parameter below these two values define how many user units the master axis moves for one rev of the master encoder. For our example we will assume that one rev of the master encoder equates to 100mm of travel of the master axis so we can use 100 and 1 for these two parameters to express this	100	DINT
U_PER_REV_DENOMINATOR	See above	1	DINT
MAX_RPM	Defines the maximum speed (in revolutions per minute) the master axis will travel. In our example the maximum master speed is 60m/min (1000mm/sec). As our earlier scaling was set for 100mm of travel for 1 encoder rev, this results in a maximum speed of 10 revs per second (600rpm). We will set 1000rpm to allow some leeway (which could be required if we were to perform some phasing type motion on the master axis for example)	1000	WORD

The screenshot below shows our completed function block…



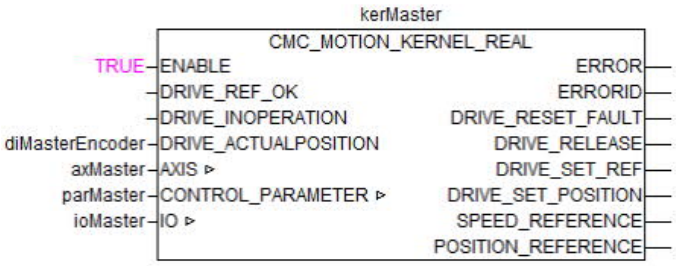
Now, in the same way we use a Kernel function block for a real axis, we must now add a CMC_MOTION_KERNEL_REAL function block to the program (i.e. the profiler for the master axis) so add a new network after the control parameter block and include the Kernel function block (we named ours parMaster).

The table below details the required input parameters to this block…

Input parameter	**Description**	**Value/assignment**	**Data type**
ENABLE	Enables processing of the function block	TRUE	BOOL
DRIVE_ACTUALPOSITION	The position of the master axis, in this case taken from the PDO mapped encoder wired to the remote drive	diMasterEncoder	DINT
AXIS	Which AXIS_REF the Kernel block relates to	axMaster	AXIS_REF
CONTROL_PARAMETER	Which CMC_AXIS_CONTROL_PARAMETER_REAL function block the Kernel block relates to	parMaster	CMC_AXIS_CONTROL_PARAMETER_REAL
IO	Defines an axis IO structure (see AN00205 for further details). In this example enter the text ‘Master_IO’ and declare this variable as a type CMC_AXIS_IO (as was done for the real axis in AN00205)	ioMaster	CMC_AXIS_IO

When using real axes it is necessary to assign at least one of the output parameters of the Kernel function block (i.e. SPEED_REFERENCE or POSITION_REFERENCE) to the relevant EtherCAT PDO mapped variable associated with the drive. In this case there is no real hardware associated with the master axis (apart from the encoder connected to the remote drive) so there is no requirement to assign variables to any of the Kernel output parameters. The only exception could be the ERROR output (and ERRORID) should the user wish to detect faults occurring with processing of the master axis. For this simple example we have decided to ignore the ERROR output, but in a real application it is recommended that this is incorporated into the system's error handling logic.

The screenshot below shows our finished network…



All of the code needed to create the master axis is now completed and the axMaster AXIS_REF can now be used as the master axis for all of the available multi-axis PLCopen motion functions available within the PS552-MC-E motion libraries (e.g. MC_GearIn, MC_GearInPos, MC_CamIn etc…).

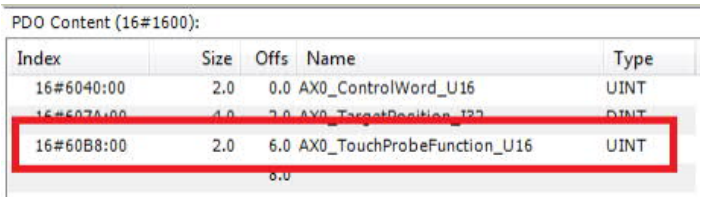
Note that in all cases, when using the master axis derived from the remote encoder, it is necessary to use 'mcActualValue' as the 'MasterValueSource' when using the multi-axis function blocks.

PLC application – Configuring and using touchprobe to capture master encoder position

MicroFlex e190 and MotiFlex e180 drives are provided with two touchprobe (fast position capture) objects that are associated with the two fast digital inputs on the drives. Touchprobe 1 is always associated with digital input 1 and touchprobe 2 is always associated with digital input 2. By default a PLC touchprobe function block will cause the drive to capture axis **position**, but from firmware version 5860 onwards it became possible to also configure these drives to make a captured **encoder** value available to the PLC over EtherCAT. Please refer to application note AN00221 for further information about the use of the touchprobe function block and available process data objects.

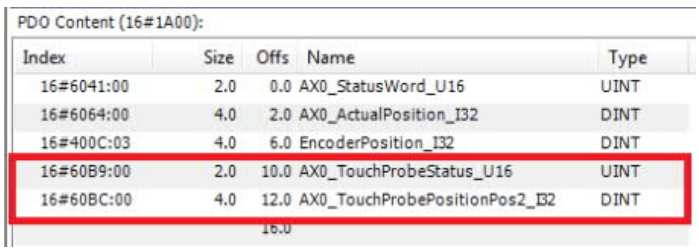
For this application note we will use digital input 2 (and hence touchprobe 2) on the drive to capture our master encoder value. We're only interested in the captured value from a rising edge on input 2 so the screenshots below show the output and intput PDO mappings we need to add to our drive configuration in Automation Builder…

Outputs:


PDO Content (16#1600):

Index	Size	Offs	Name	Type
16#6040:00	2.0	0.0	AX0_ControlWord_U16	UINT
16#60B8:00	2.0	6.0	AX0_TouchProbeFunction_U16	UINT

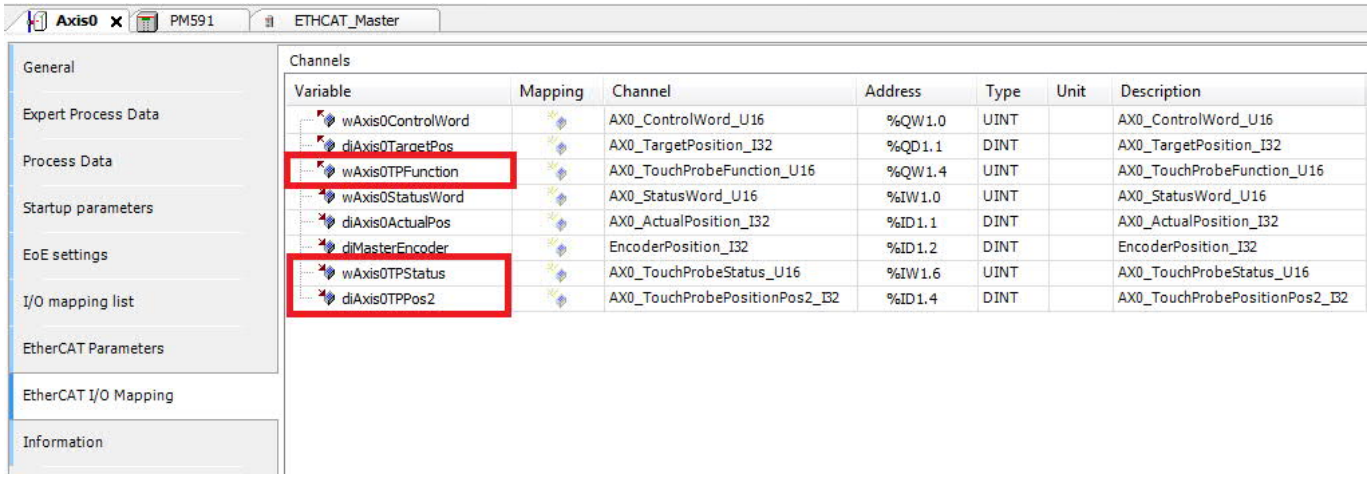
Inputs:


PDO Content (16#1A00):

Index	Size	Offs	Name	Type
16#6041:00	2.0	0.0	AX0_StatusWord_U16	UINT
16#6064:00	4.0	2.0	AX0_ActualPosition_I32	DINT
16#400C:03	4.0	6.0	EncoderPosition_I32	DINT
16#60B9:00	2.0	10.0	AX0_TouchProbeStatus_U16	UINT
16#60BC:00	4.0	12.0	AX0_TouchProbePositionPos2_I32	DINT

If you need to use digital input 1 or the negative edge (or both edges) of one of the fast inputs then simply add the necessary PDO mappings to access this data (again, refer to AN00221 for further information if required).

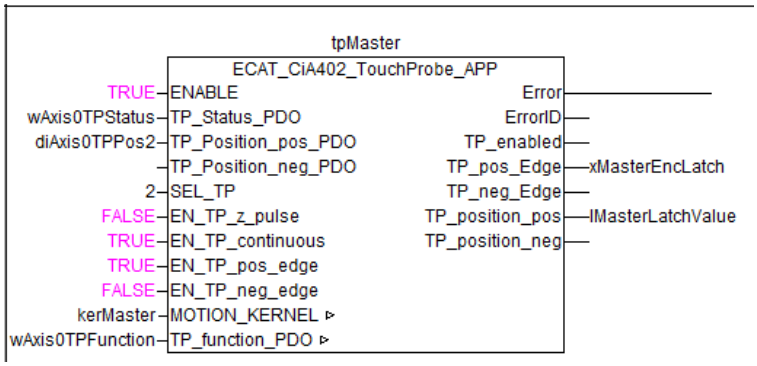
We now need to give suitable names to our additional PDO mappings via the 'EtherCAT I/O Mapping' tab for our drive. The screenshot below shows how we named our variables…



Make sure you are logged out of the PLC, add the required PDO mappings to the Automation Builder configuration as shown above and then right click the PLC application icon and select 'Create configuration data' to force the CoDeSys global variables to be updated to include definitions for our new PDO mappings.

Switch to the CoDeSys editor, we can now start to add the touchprobe function block to the PLC code. As is highlighted by AN00221, it is necessary to place the touchprobe function block in a program unit that is called by the EtherCAT related task. In our example our program unit named 'prgEtherCATControl' can be used. Insert a new network into the PLC program after the network containing the master axis KERNEL function block and add an ECAT_CiA402_TouchProbe_APP block to this new network.

We need to configure this function block to continuously look for touchprobe values from touchprobe 2 on the drive using the rising edge of digital input 2. The screenshot below shows our completed network…



Note how we have used the names we allocated earlier to our touchprobe PDO mappings for the function block input parameters requiring PDO references. We also added a Boolean variable (xMasterEncLatch) to indicate the presence of a new latch value (this is only set TRUE for one program cycle so your application code may need to consider setting/latching this variable if necessary) and a LREAL variable (lMasterLatchValue) to store the actual latch data. Note that the link to the master axis' KERNEL ensures that this latch data is presented in scaled user units and not raw encoder counts.

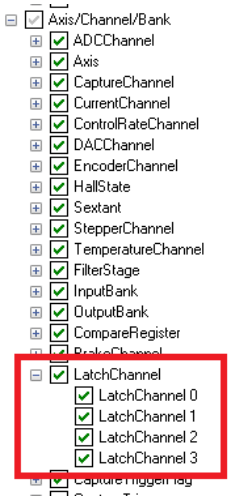
As we mentioned earlier, by default the drive is configured to latch **position** of an axis. So in order for our touchprobe function block to present **encoder** data instead we must now configure our e190 or e180 drive to do this. There are two ways to achieve this…

1. We can use the drive parameters to configure operation of the drive's touchprobe/latch channels (e.g. using Mint Workbench to configure these settings and then store these drive parameters permanently)
2. We can use EtherCAT object writes (i.e. SDO access) from the PLC program to modify the drive's configuration – with this method the configuration is usually left as "volatile" (so the drive defaults back to its original configuration whenever the PLC program stops, resets or if the drive is power cycled) and the PLC application code determines when the drive configuration needs to be adjusted

In practice a combination of the two methods above is often used. The user may wish to use the drive's parameters to permanently configure touchprobe 2 to operate as a latch of encoder channel 2 using a latch inhibit value (filter to allow subsequent fast inputs to be ignored until the encoder has travelled a specified distance since the previous latch) and may then use SDO access from the PLC to allow the machine operator to adjust this latch inhibit value at runtime to suit different operating conditions.

Using Mint Workbench to configure the touchprobe settings

Connect to the MicroFlex e190 or MotiFlex e180 drive using Mint Workbench and from the left hand screen selection bar click on 'Parameters' to view the drive's parameter table. In the parameter tree scroll down to the Axis/Channel/Bank section and expand the 'Latch channel' section of the tree…



The drive has four latch channels that are assigned to the touchprobe objects as follows:
LatchChannel 0 = Touchprobe 1 positive edge operation
LatchChannel 1 = Touchprobe 1 negative edge operation
LatchChannel 2 = Touchprobe 2 positive edge operation
LatchChannel 3 = Touchprobe 2 negative edge operation

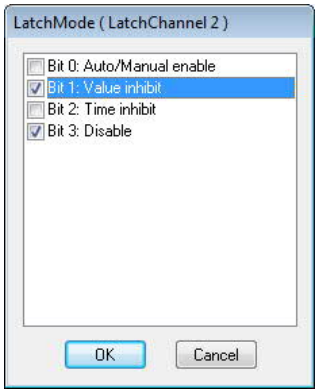
For our example in the application note we are using a positive edge on digital input 2 (i.e. touchprobe 2) to capture the master encoder value, so click on the LatchChannel 2 icon in the tree to display the parameters relating to this in the right hand pane of the parameters window

You should find that your parameters for Latch Channel 2 look something like this at this point…

Parameter	Active
LatchEnable (LatchChannel 2)	FD 0
LatchInhibitTime (LatchChannel 2)	FD 0 ms
LatchInhibitValue (LatchChannel 2)	FD 0.0000
LatchMode (LatchChannel 2)	FD 0x0008
LatchSource (LatchChannel 2)	FD Axis position
LatchSourceChannel (LatchChannel 2)	FD 0
LatchTriggerChannel (LatchChannel 2)	C 2
LatchTriggerEdge (LatchChannel 2)	FD Positive edge
LatchTriggerMode (LatchChannel 2)	FD Digital input
LatchValue (LatchChannel 2)	RO 0.0000

You can see that, by default, this channel is configured to capture Axis position (of Axis 0, as set by LatchSourceChannel) from a positive/rising edge of digital input 2 (where 2 is set by the LatchTriggerChannel).
When the PLC program runs and the Touchprobe function block is enabled the PLC uses the Touchprobe Function PDO (that we mapped earlier) to set LatchEnable to 1, to set bit 0 of LatchMode to 1 and to clear bit 3 of LatchMode. Together these actions result in the LatchChannel being enabled and every rising edge on digital input 2 will result in the drive passing the 'LatchValue' for channel 2 back to the PLC's Touchprobe function block via the TP2Pos PDO we mapped earlier.

We need to latch encoder channel 2 (our master encoder) instead, so click on the current LatchSource setting (Axis position) and change this to read 'Encoder value'. Now change the value for LatchSourceChannel from 0 to 2 (to indicate encoder channel 2).
If you want to use the drive's inbuilt ability to filter latches click on the existing LatchMode setting and select the 'Value inhibit' or 'Time inhibit' bit as required (leave the 'Disable' bit as this is controlled by the PLC automatically)….



See LATCHMODE in the Mint Help system for further details about automatic filtering of latch data.

Now enter a value for the LatchInhibitValue. As we are latching encoder this value is in encoder units (scaled by ENCODERSCALE(2) in this case on the drive). For this example we will assume this encoder channel has an ENCODERSCALE of 1 (so it is operating in encoder counts) so we might enter a value of 10000 for example to setup a filter distance of 10000 counts.

After all of these actions your parameters should look something like this…

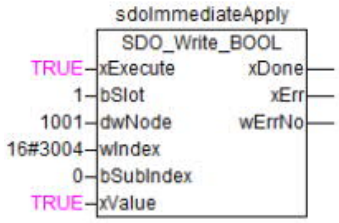
Parameter	Active
LatchEnable (LatchChannel 2)	FD 0
LatchInhibitTime (LatchChannel 2)	FD 0 ms
LatchInhibitValue (LatchChannel 2)	C 10000.0000
LatchMode (LatchChannel 2)	C 0x000A
LatchSource (LatchChannel 2)	C Encoder value
LatchSourceChannel (LatchChannel 2)	C 2
LatchTriggerChannel (LatchChannel 2)	C 2
LatchTriggerEdge (LatchChannel 2)	FD Positive edge
LatchTriggerMode (LatchChannel 2)	FD Digital input
LatchValue (LatchChannel 2)	RO 0.0000

Now select Tools>Store Drive Parameters from the Workbench top menu to save these settings in the drive's non-volatile parameter area.

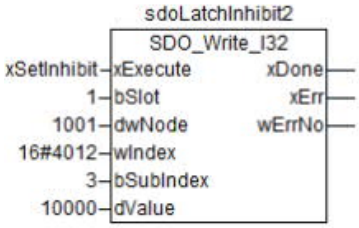
Using PLC code to configure the touchprobe settings

Instead of (or as well as) setting the drive parameters permanently as described in the previous section it is also possible to modify drive parameters at runtime using SDO object writes from the PLC. Application note AN00242 fully describes how to use the PLC to perform SDO object writes (and reads) and includes an export file that, when imported, will provide your PLC application with ready-made function blocks to access the various objects within the connected servo drives.

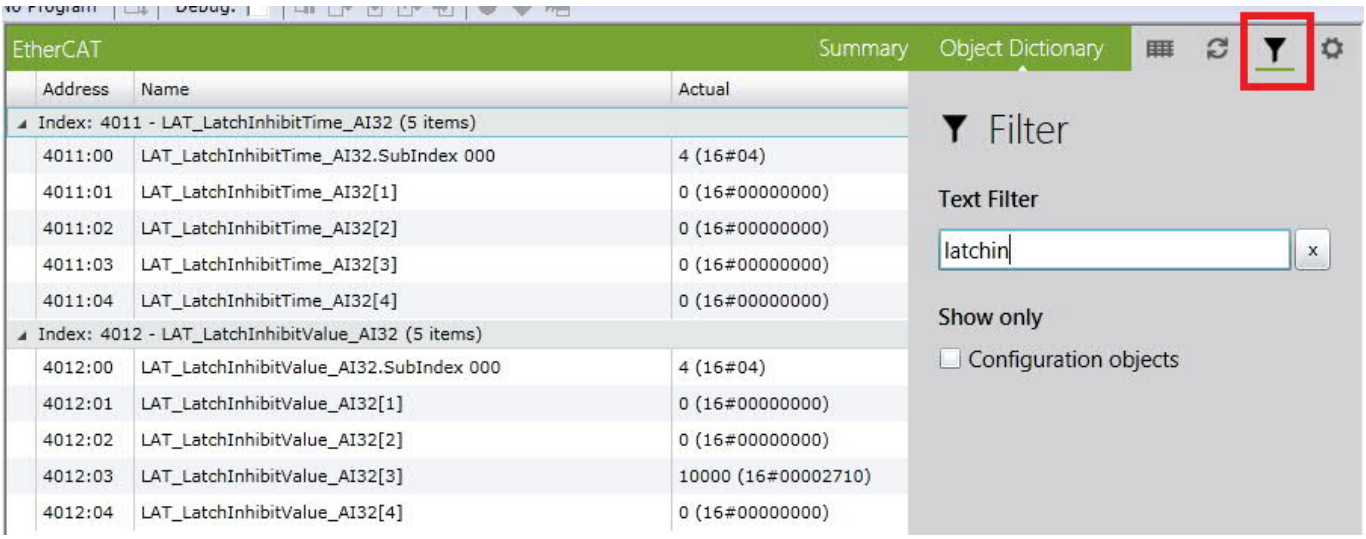
If you don't already have these function blocks available to you ensure you have copied the 'SDO Access.exp' file that is included with application note AN00242 onto your hard drive and then select 'Project > Import…' from the CoDeSys menu. As detailed by AN00242, it is necessary to set the Immediate Apply object (16#3004 subindex 0) to TRUE in order for data written to the EtherCAT objects to be passed across to the drive parameter table. The screenshot below shows how this might look in a PLC program written in FBD (note that the inputs xExecute are rising edge triggered so a TRUE on these inputs results in the SDO being called once only, at initial program start)…



Once the immediate apply is set TRUE the PLC code can then update touchprobe settings as required. For example, the screenshot below shows how the PLC code may set a value of 10000 for LATCHINHIBITVALUE(2)….



Note that this parameter is controlled by EtherCAT object 16#4012 subindex 03 (LAT_LatchInhibitValue_AI32). If you are unsure which object to use you can use the Filter option in the Workbench Object Dictionary viewer to search for entries. For example, in the screenshot below we started typing 'latchin' to look for objects containing this text.…



As we mentioned earlier, if you're using the drive for a master encoder you are unlikely to change the settings for LATCHSOURCE and LATCHSOURCECHANNEL, and the hardware connections will probably determine the LATCHTRIGGERCHANNEL, so it's most likely that you will use the Parameter Viewer to configure the "fixed settings" and SDO access for settings that the operator might want to adjust to suit a particular machine configuration.

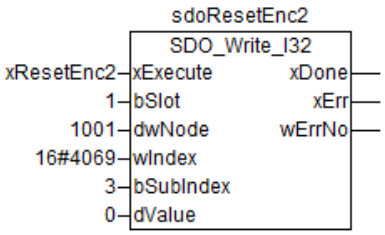
Setting a new master encoder value

The mapped encoder value (object 0x400C, subindex 3 in the case of Encoder channel 2) is read only, so it is not possible for the PLC to write a value to this object. However, it may be desirable to reset the master encoder value at some point during the application (e.g. if the encoder is being used by the drive itself for triggering various SENTINEL channels that may be configured).

To allow this to be done an additional encoder related object (0x4069 ENC_ForcePosition_AI32) is included – the subindexes to this object match the subindexes of the Encoder position object (0x400C). So for example, writing to 0x4069 subindex 3 will result in 0x400C subindex 3 (i.e. encoder 2) being modified.

Object 0x4069 is not PDO mappable, it can only be accessed via an SDO write. As the object is of data type I32 the SDO_Write_I32 function block from application note AN00242 can be used to write a new encoder value.

An example showing how to set encoder 2 to zero is shown below…



Note that if you want to retain the relationship between the master encoder value and the master axis position it is necessary to reset the master axis position (using MC_SetPosition) at the same time the master encoder value is modified.

Contact Us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC

EtherCAT® is a registered trademark and patented technology, licenced by Beckhoff Automation GmbH, Germany