Motion Control Products

Application note
Using the CD522 module for master encoder input

AN00239

Rev B (EN)

Use a CD522 2 channel encoder module in combination with an AC500 PLC to provide up to 2 master encoder channels for use in PLCopen motion applications

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 the 2 channel encoder module (CD522) and how to then use the encoder value(s) provided by this module to create a (virtual/simulated) master axis in a PLCopen motion application. For information on how to use the fast encoder latching features of this module please refer to application note AN00240.

Pre-requisites

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

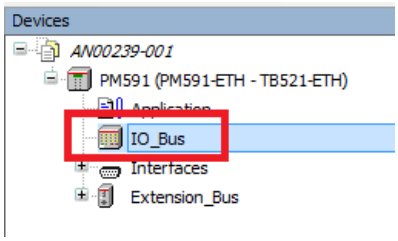
- Mint Workbench build 5812 or later (see new.abb.com/motion for latest downloads and support information)
- A MicroFlex e150 or MotiFlex e180 drive with build 5812 (e150) / 5819 (e180) or later firmware
- A PC or laptop running Automation Builder 1.2 or later
- An installed copy of the ABB PLCopen motion control library (PS552-MC-E v3.1.0 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 e150/e180 EtherCAT Getting Started Guide) and the Automation Builder PLC project 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 CD522 module only requires a PC or laptop running Automation Builder 1.2 or later and an installed copy of the PS552-MC-E motion control libraries. It is assumed the reader has a basic working knowledge of 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 e150 or MotiFlex e180 for example) ready for use with the AC500 PLC.

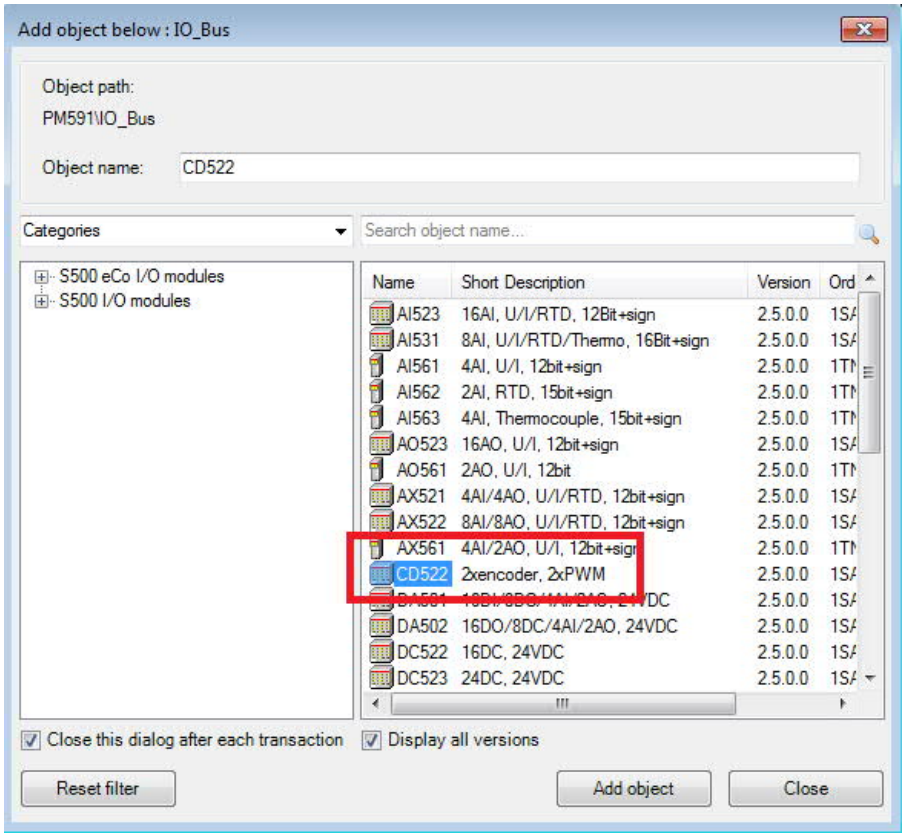
Automation Builder – Adding the CD522 module

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 CD522 module. Alternatively a ready-made example project is available as part of this application note.

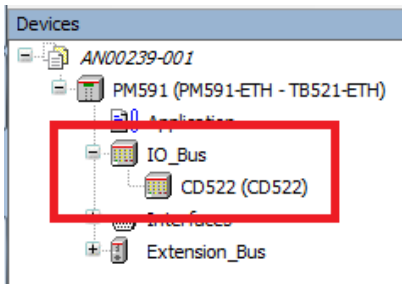
Open the PLC project provided as part of AN00205 from the Automation Builder 'File>Open Project…' menu and once opened select 'File>Save Project As…' to give your project a new name (to avoid damaging the original project). Once saved, right click the 'IO_Bus' icon in the Devices tree…



…and from the right-click menu select 'Add object'. A dialogue will appear allowing the user to select from a list of all possible S500 I/O modules that can be added to the AC500 I/O bus. Select CD522 from this list and click on the 'Add object' button…

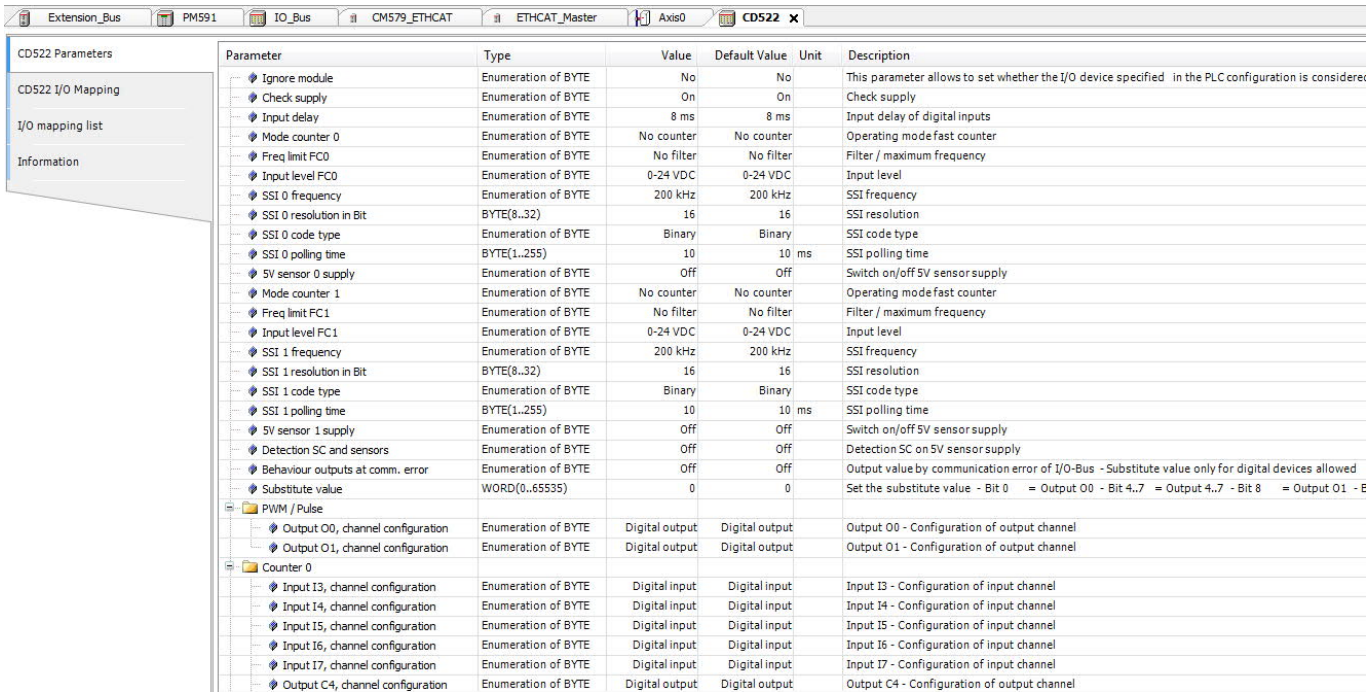


The module will now appear in the Devices tree as shown below…



Note that if you are using a different PLC project that already includes some I/O modules you will need to ensure that the order of the modules in the Devices tree matches the physical order of the devices on the I/O bus. If the new CD522 module is out of position in the tree you can select it and drag it to the correct position in the tree.

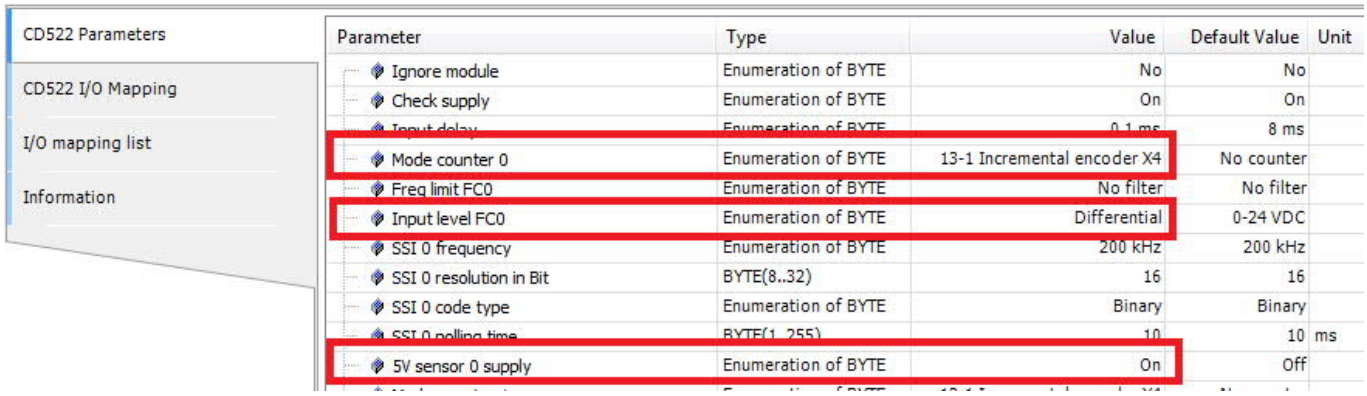
Now double-click the CD522 icon, the right hand pane will display four tabs/pages of configuration information for the module…



Parameter	Type	Value	Default Value	Unit	Description
Ignore module	Enumeration of BYTE	No	No		This parameter allows to set whether the I/O device specified in the PLC configuration is considered
Check supply	Enumeration of BYTE	On	On		Check supply
Input delay	Enumeration of BYTE	8 ms	8 ms		Input delay of digital inputs
Mode counter 0	Enumeration of BYTE	No counter	No counter		Operating mode fast counter
Freq limit FC0	Enumeration of BYTE	No filter	No filter		Filter / maximum frequency
Input level FC0	Enumeration of BYTE	0-24 VDC	0-24 VDC		Input level
SSI 0 frequency	Enumeration of BYTE	200 kHz	200 kHz		SSI frequency
SSI 0 resolution in Bit	BYTE(8..32)	16	16		SSI resolution
SSI 0 code type	Enumeration of BYTE	Binary	Binary		SSI code type
SSI 0 polling time	BYTE(1..255)	10	10	ms	SSI polling time
5V sensor 0 supply	Enumeration of BYTE	Off	Off		Switch on/off 5V sensor supply
Mode counter 1	Enumeration of BYTE	No counter	No counter		Operating mode fast counter
Freq limit FC1	Enumeration of BYTE	No filter	No filter		Filter / maximum frequency
Input level FC1	Enumeration of BYTE	0-24 VDC	0-24 VDC		Input level
SSI 1 frequency	Enumeration of BYTE	200 kHz	200 kHz		SSI frequency
SSI 1 resolution in Bit	BYTE(8..32)	16	16		SSI resolution
SSI 1 code type	Enumeration of BYTE	Binary	Binary		SSI code type
SSI 1 polling time	BYTE(1..255)	10	10	ms	SSI polling time
5V sensor 1 supply	Enumeration of BYTE	Off	Off		Switch on/off 5V sensor supply
Detection SC and sensors	Enumeration of BYTE	Off	Off		Detection SC on 5V sensor supply
Behaviour outputs at comm. error	Enumeration of BYTE	Off	Off		Output value by communication error of I/O-Bus - Substitute value only for digital devices allowed
Substitute value	WORD(0..65535)	0	0		Set the substitute value - Bit 0 = Output O0 - Bit 4..7 = Output 4..7 - Bit 8 = Output O1 - B
PWM / Pulse					
Output O0, channel configuration	Enumeration of BYTE	Digital output	Digital output		Output O0 - Configuration of output channel
Output O1, channel configuration	Enumeration of BYTE	Digital output	Digital output		Output O1 - Configuration of output channel
Counter 0					
Input I3, channel configuration	Enumeration of BYTE	Digital input	Digital input		Input I3 - Configuration of input channel
Input I4, channel configuration	Enumeration of BYTE	Digital input	Digital input		Input I4 - Configuration of input channel
Input I5, channel configuration	Enumeration of BYTE	Digital input	Digital input		Input I5 - Configuration of input channel
Input I6, channel configuration	Enumeration of BYTE	Digital input	Digital input		Input I6 - Configuration of input channel
Input I7, channel configuration	Enumeration of BYTE	Digital input	Digital input		Input I7 - Configuration of input channel
Output C4, channel configuration	Enumeration of BYTE	Digital output	Digital output		Output C4 - Configuration of output channel

The CD522 2 channel encoder module supports a variety of encoder types (e.g. 5v/RS422 differential incremental encoder, HTL incremental encoder, SSI encoder etc…). For the purposes of this application note we will assume a RS422 incremental encoder will be used and wired to encoder channel 0 on the CD522 module via input terminals 1.0 (/A0), 1.1 (/B0), 2.0 (A0), 2.1 (B0), 1.3 (5V0) and 1.4 (0V). Please consult the Automation Builder help system for further information about the CD522 module capabilities and how to wire this module for all supported encoder types.

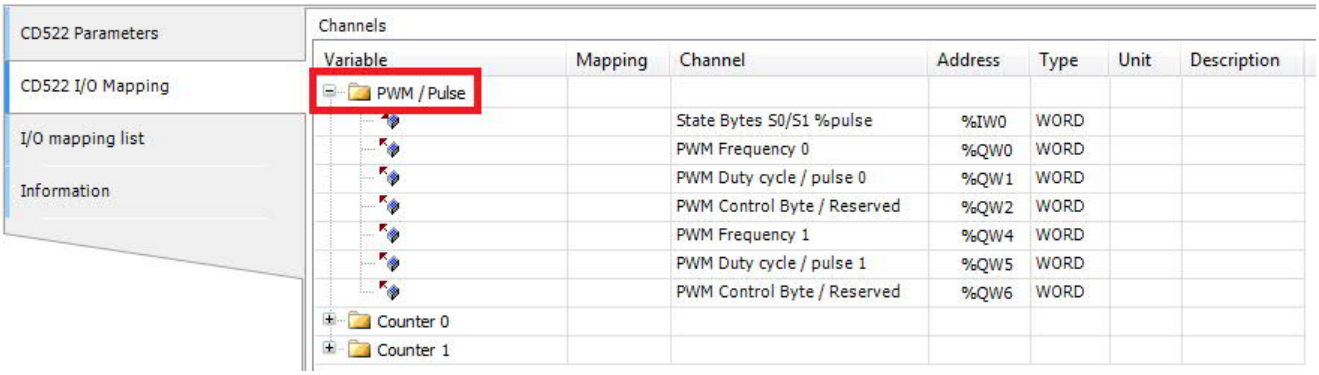
The ‘CD522 Parameters’ tab in the right hand pane can now be used to configure the encoder, for our system we need to setup a quadrature encoder counter (i.e. the number of encoder lines/pulses will be multiplied by 4 as the CD522 will count every encoder edge), tell the module we are using a differential encoder input and turn on the CD522’s 5 volt output for channel 0 to power our encoder…



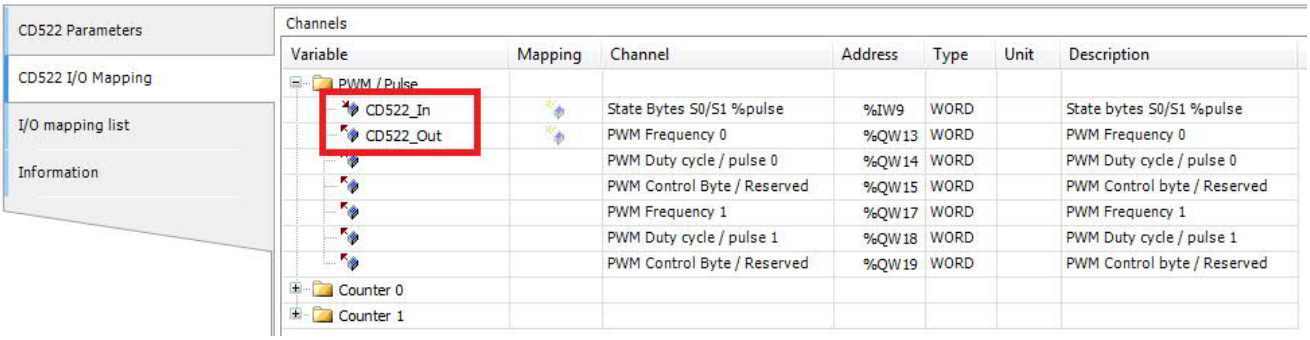
The mode and input level settings will vary depending on the type of encoder you are using so please consult the Automation Builder Help system for further details about these options if necessary.

At this stage these are the only parameter settings required to allow our RS422 encoder to operate, we will revisit this window in Automation Builder later in this document when we discuss the use of the CD522 module’s fast input for latching encoder position.

Now select the ‘CD522 I/O Mapping’ tab and expand the folder labelled ‘PWM / Pulse’…



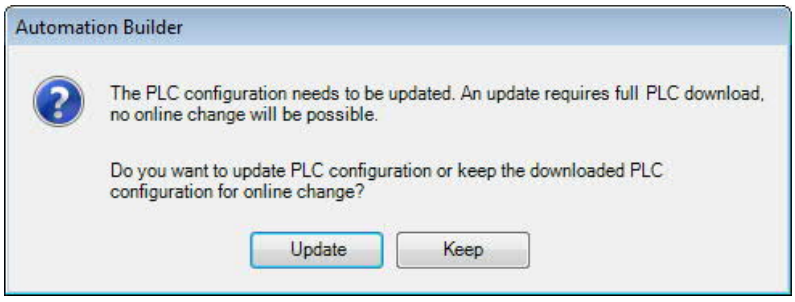
The first two variables within this folder are 'State Bytes S0/S1 %pulse' (the first input) and 'PWM Frequency 0' (the first output). We must give names to these two variables so that global variables are created based on these when the PLC configuration is created later. Our PLC program will then utilise pointers to these variables as a means of addressing the CD522 module – this is described in the following section of this document in more detail. For now create some names for the two variables, we called ours 'CD522_In' and 'CD522_Out' as shown below…



Configuration of the CD522 module within Automation Builder is now complete so at this point be sure to save your project.

PLC application – Adding the CD522 function block

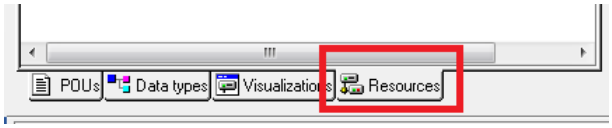
Double-click the PLC application icon in Automation Builder to launch the PLC programming environment. Automation Builder will let you know that the PLC hardware configuration has changed and therefore the configuration files need updating before any code can be downloaded….

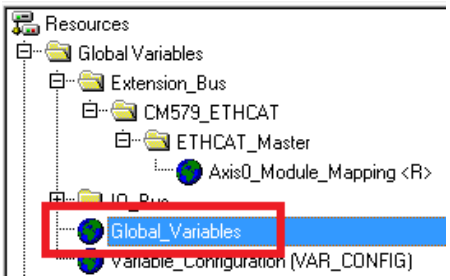


Click 'Update' to accept this and let it create the new configuration (which includes creating the two global variables we added for the CD522 module). The latest installed library for the CD522 module will be automatically included in the Library Manager as part of this process so there is no need to add any additional library files to the project as a result of adding this module.

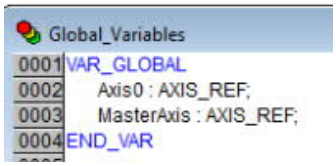
We are going to add an encoder function block to our system that will eventually be used to set the position of a (simulated) master axis within the PLC application. PLCopen motion function blocks that provide 'geared' moves (e.g. MC_CamIn, MC_GearIn, MC_GearInPos) all require a master **axis** to be specified so it is not possible to only use the necessary encoder function block without also creating a master axis.

As we are using the project created for application note AN00205 as a starting point for this application note our PLC code already contains a definition for the slave axis in the Global_Variables section of the Resources tree. Click on the 'Resources' tab in CoDeSys and then double-click the 'Global_Variables' icon…



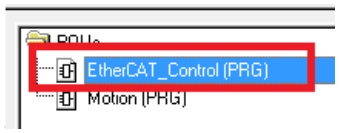


The Global_Variables window will open and we can now add a new AXIS_REF definition for our master axis. We named ours 'MasterAxis' as shown below…



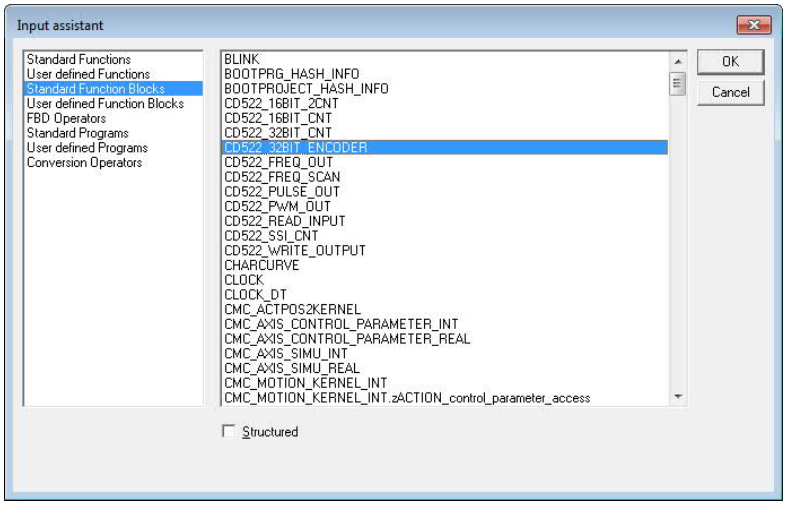
Our slave axes is updated (profiled) by a CMC_MOTION_KERNEL_REAL function block that resides in a program element called by our EtherCAT task (see application note AN00205 for further details). It is logical that our master axis (and hence the CD522 encoder) must also be updated at the same rate and therefore the logic for this must also be within a program element called by the EtherCAT task. We could create a new program and add this to the list of programs called by the EtherCAT task which might help the user to identify which program elements are responsible for which axis for example, but for this application note we will add the required code to our existing program that is triggered from the EtherCAT coupler…..this program is named EtherCAT_Control.

Double-click the EtherCAT_Control icon from within the POU explorer in CoDeSys to open this program.



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 CD522 encoder module and its corresponding axis to the beginning of our EtherCAT_Control program. 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.

The first function block we will add to the program is named CD522_32BIT_ENCODER. Insert a new block into network 0001 and type this name (or use the Input assistant via keyboard shortcut 'F2' to find this block – easiest if you turn off the 'Structured' view and look in the 'Standard Function Blocks' section)…

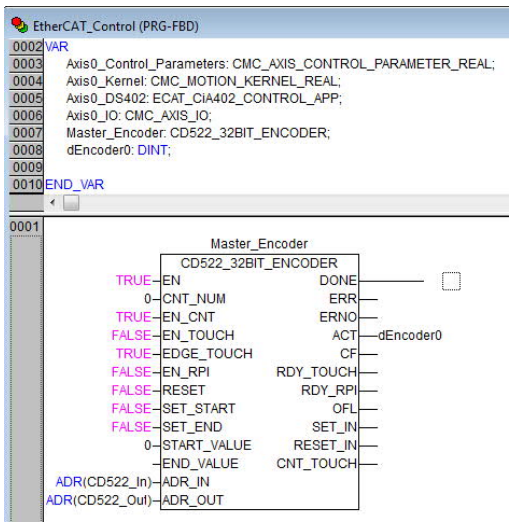


Give this instance of the CD522_32BIT_ENCODER function block a name (we called ours Master_Encoder) and a declare this variable. The table below details the input and output parameters available for this function block and indicates what value should be entered or variable should be assigned for each for this example…

Input parameter	**Description**	**Value/assignment**	**Data type**
EN	Enables processing of the function block	TRUE	BOOL
CNT_NUM	Which encoder channel on the CD522 the block relates to	0	WORD
EN_CNT	Enables the encoder counter for the selected channel	TRUE	BOOL
EDGE_TOUCH	Sets which edge of the fast input for the selected encoder channel is used to latch encoder position (False = falling edge, True = rising edge)	TRUE	BOOL
EN_RPI	Enables the reference point initiator (setting the counter to the start value from a Z pulse)	FALSE	BOOL
RESET	Resets the encoder counter	FALSE	BOOL
SET_START	Sets the encoder counter to the start value	FALSE	BOOL
SET_END	Sets the encoder counter to the end value	FALSE	BOOL
START_VALUE	The value to use with SET_START and/or EN_RPI	0	DINT
END_VALUE	If the encoder counter reaches END_VALUE then FB output CF is set to True	Leave blank	DINT
ADR_IN	Pointer to the start of the CD522 module's inputs	ADR(CD522_In)	POINTER TO STRUCTCD522IN
ADR_OUT	Pointer to the start of the CD522 module's outputs	ADR(CD522_Out)	POINTER TO STRUCTCD522OUT
Output parameter	**Description**	**Value/assignment**	**Data type**
DONE	Set on completion of abortion of processing	Leave blank	BOOL
ERR	Indicates if a processing error has occurred	Leave blank	BOOL
ERNO	The error code for any processing errors	Leave blank	WORD
ACT	The actual encoder count value for the selected channel	dEncoder0	DINT
CF	Output is set true if the encoder count reaches END_VALUE	Leave blank	BOOL
RDY_TOUCH	Set True for one scan when a new latched value is available	Leave blank	BOOL
RDY_RPI	Set True for one scan when the RPI operation is done	Leave blank	BOOL
OFL	Set True when the actual value passes from -1 to 0 or from 0 to -1	Leave blank	BOOL
SET_IN	Set True if one of the CD522 module's inputs is configured as a set input	Leave blank	BOOL
RESET_IN	Set True if one of the CD522 module's inputs is configured as a reset input	Leave blank	BOOL
CNT_TOUCH	The latched encoder value resulting from operation of the relevant fast latch input on the CD522 module	Leave blank	DINT

Note the use of the ADR operator to define the pointers to the CD522 module's inputs and outputs (the ADR operator acts on the variables that we defined earlier as part of the CD522 Parameter setup). When assigning 'dEncoder0' as the output variable for the ACT function block output be sure to set the data type for this variable as DINT.

Here's what our function block looked like when we completed this process…



Save the file so you don't lose any of these changes. If you like you can Login to the PLC and run the program to check the operation of the encoder block. As the connected master encoder rotates you should see the value of dEncoder0 change accordingly – if this doesn't happen recheck all of the settings you made earlier and the physical connection of the encoder to the CD522 module.

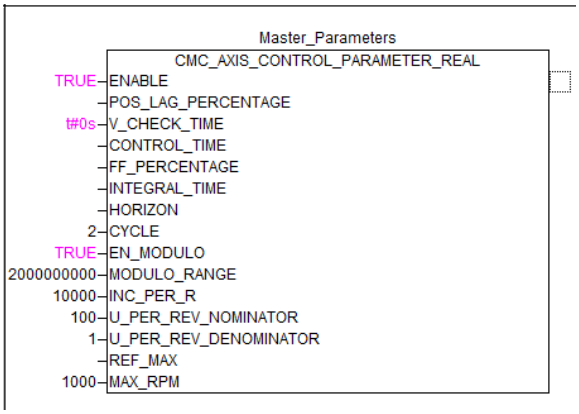
If you need to use the second encoder input channel on the CD522 module then a second instance of the CD522_32BIT_ENCODER function block must be included in the application code. Please consult the Automation Builder help system for further information about the CD522 module if required.

At this point we are now ready to use the CD522 encoder value to set the position of our master axis. Add a new network 0002 and include a CMC_AXIS_CONTROL_PARAMETER_REAL function block in this network (we named ours Master_Parameters). 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 previously set in AN00205 for the real axes	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	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 to something approaching, 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…

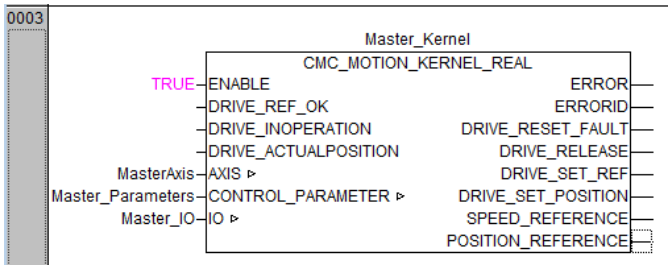


Now, in the same way we use a Kernel function block for a real axis, we must now add a CMC_MOTION_KERNEL_REAL function block to the program (i.e. the profiler for the master axis) so add a new network after the control parameter block and include the Kernel function block (we named ours Master_Kernel). 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
AXIS	Which AXIS_REF the Kernel block relates to	MasterAxis	AXIS_REF
CONTROL_PARAMETER	Which CMC_AXIS_CONTROL_PARAMETER_REAL function block the Kernel block relates to	Master_Parameters	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)	Master_IO	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 CD522 module) 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…



You may have noticed that we omitted to include an input for DRIVE_ACTUALPOSITION. Eventually this input needs to be linked to the CD522 encoder function block's ACT output (i.e. the master encoder count). However, there are some considerations to be given to how this encoder value is passed to the Kernel…

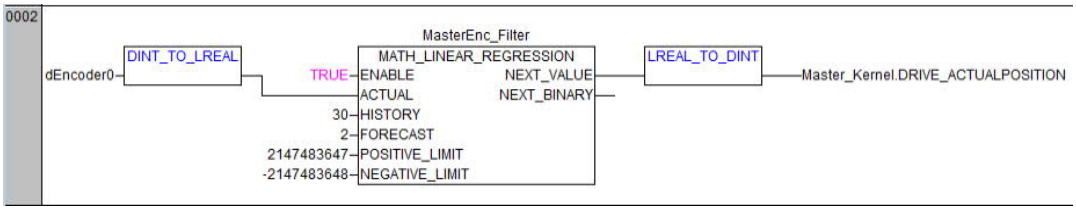
The CD522 module sits on the AC500 PLC's IO bus and as such the encoder value is only updated at a slow rate (e.g. every 4ms). As a result the resulting position/speed measurement of the master encoder can be quite 'granular' in nature (i.e. "noisy") and any slave axes performing geared motion based on the master axis may exhibit some increase in audible noise as a result of trying to use this encoder value as a reference (in extreme cases the slave axis may even trip with a motor overload due to the high accelerations associated with trying to follow a noisy master reference).

It is therefore necessary to "filter" the encoder value and provide smoothed encoder reference positions to the master axis' Kernel. This is achieved using the MATH_LINEAR_REGRESSION function block which is included as part of the Maths Functions library included with the PS552-MC-E motion control libraries.
Navigate to the Resources tab in CoDesys, open the Library Manager and add MathFunctions_AC500_Vxx.lib to your current project (where xx will be the current version of this library – e.g. 23). Once this has been added, add a new network to the EtherCAT_Control program - after the network containing the CD522_32BIT_ENCODER function block – this is where we will filter the actual encoder value. Insert a MATH_LINEAR_REGRESSION function block into this new network (we named ours MasterEnc_Filter).

The following table details the input and output parameters for the MATH_LINEAR_REGRESSION function block…

Input parameter	**Description**	**Value/assignment**	**Data type**
ENABLE	Enables processing of the function block. Changes to FORECAST can be made whilst the block is enabled, but changes to HISTORY must be loaded with a new rising edge on the ENABLE input	TRUE	BOOL
ACTUAL	The actual value to be filtered (in our case, the output from the CD522_32BIT_ENCODER function block)	dEncoder0 (cast from DINT to LREAL)	LREAL
HISTORY	The number of samples to be stored to calculate a moving average (typically in the range 10-50)	30	LREAL
FORECAST	The number of cycles to use to calculate a feed-forward correction to compensate for lag in the CD522 encoder value (typically in the range 1-4)	2	BOOL
POSITIVE_LIMIT	Maximum value that should be output by the block (in our case we don't want to affect the range of the CD522 encoder output, we only want to smooth it, so we will set the maximum value the CD522 encoder block produces	2147483647	DWORD
NEGATIVE_LIMIT	Minimum value that should be output by the block (in our case we don't want to affect the range of the CD522 encoder output, we only want to smooth it, so we will set the minimum value the CD522 encoder block produces	-21474836478	DINT
Output parameter	**Description**	**Value/assignment**	**Data type**
NEXT_VALUE	The resulting smoothed decimal value	Master_Kernel.DRIVE_ACTUALPOSITION (cast from LREAL to DINT)	LREAL
NEXT_BINARY	The resulting smooth value expressed in binary	Leave blank	DWORD

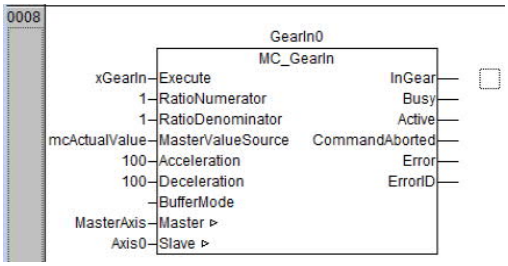
Note how the output of this function block is assigned to the actual position for our master axis. The screenshot below shows how this should look in the application code…



All of the code needed to create the master axis is now completed and the MasterAxis AXIS_REF can now be used as the master axis is 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 CD522 encoder, it is necessary to use 'mcActualValue' as the 'MasterValueSource' when using the multi-axis function blocks.

Example:



For further information about the operation of the multi-axis motion function blocks and how these are used in combination with a master axis please refer to the Automation Builder help system for further information.

For details on how to use the fast position latches on the CD522 encoder module please refer to application note AN00240.

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