Motion Control Products

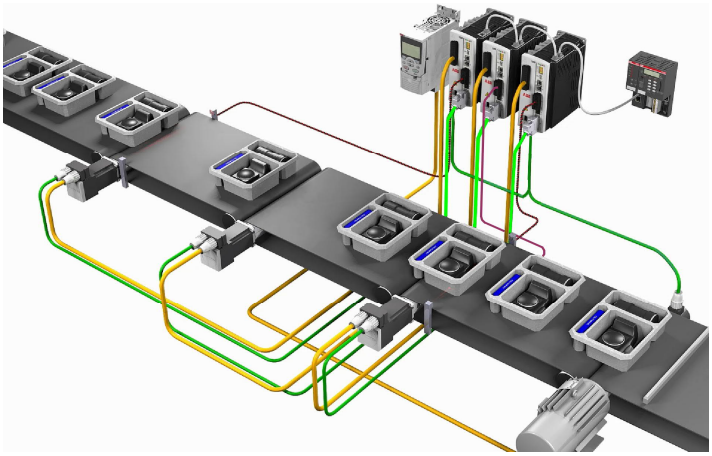
Application note
Product spacing (PLCopen motion)

AN00248

Rev C (EN)

ABB PLCopen motion provides a range of function blocks to simplify applications where products require offsets or phase shifting of position.

Typical applications would be “smart belts” or “phasing conveyors” often used in packaging machines to ensure randomly spaced products are spaced out regularly ready for the next part of the packaging process.



Introduction

This application note provides an insight into some of the techniques which can be used to easily implement a product spacing machine such as that illustrated in the picture above.

The code that accompanies this application note is targeted at the hardware illustrated by the ‘Product Spacing’ animation. With minor changes the same code could be adapted to run on any motion capable AC500 PLC with the relevant PS552-MC motion libraries available to the programmer. This same code could be modified for a simple system requiring control of just a single phasing belt.

In other application notes such as AN00205 (*AC500 - EtherCAT quick start guide)* and in AN00241 (Using motion drive encoder channels for master encoder input) we cover the basic principles which are referenced in this application note. These should be read and understood before using this document.

The phasing belt system described by this application note would normally be provided as an intermediate piece of equipment sitting between an upstream (infeed) conveyor, delivering product at irregular intervals, and a downstream flighted conveyor, delivering products to the next part of the process with a regular spacing. In the illustration above the flighted conveyor is shown as being under control of an ABB ACS drive, but in most cases, where the phasing system is an independent system, the flighted conveyor would be controlled by another machine/system. In either case an encoder signal (from the flighted conveyor motor or driven directly by the conveyor itself) is required for the phasing system to use as a master position reference.

The phasing system comprises three servo axes, each of which drives a conveyor belt:

- Primary phasing belt
- Intermediate belt
- Secondary phasing belt

Each axis operates in Cyclic Synchronous Position (CSP) mode over EtherCAT, profiled by a motion-capable AC500 PLC.

Product spacing example

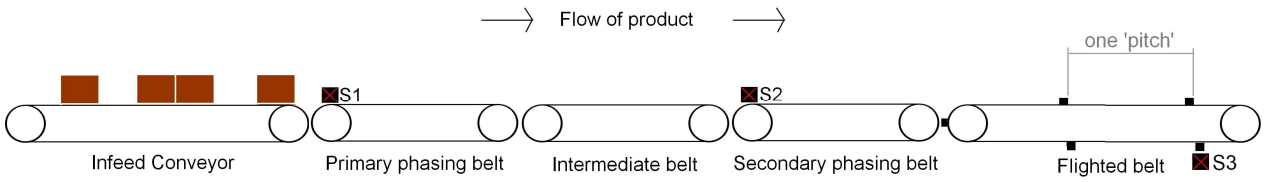
The product spacing example (available on request from cn-motionsupport@cn.abb.com) requires the following hardware to be able to exercise the example:

- ABB AC500 motion-capable PLC (PM585, PM59x with FW **2.5.1** or later)
- CM579-ETHCAT communication card (with FW **2.6.9** or late)
- Three MicroFlex e190 AC servo drives (with FW **5868** or later), one for each phasing belt and one for the intermediate belt. In the program example the intermediate belt is implemented as a virtual axis.
- Two OPT-MF-200 Encoder breakout option (for master (flighted) conveyor position latching).
- Two ABB AC servo motors (the example assumes SmartABS encoders with 131072 counts per revolution resolution)
- Two NPN photo reflective photocell (for product detection) sensors (wired to the relevant input DIN 1 on the two e190 drives fitted with the OPT-MF-200 which will be used to provide a high speed capture of the Flighted conveyor belt position as a product travels across the phasing belts)
- One NPN photo proximity flight sensor (S3) on the flighted conveyor to detect the 'zero' position of the conveyor flight.
- A 'master encoder'. For us to latch the positon of the flighted below we must mount an encoder on it which is coupled to the belt drive. This is typically an incremental encoder wired to each of the OPT-MF-200 cards (arranged so that one rev, or a whole number of revs, of the encoder occurs in one pitch of the flighted conveyor's travel).

For our example we will assume the phasing system has no control over the flighted conveyor/belt, it just monitors the position of this conveyor via the encoder coupled to it.

Principle of operation

Products arrive at the primary phasing belt irregularly spaced (or grouped together). The primary phasing belt is geared to the flighted conveyor (at the outfeed) at a ratio whereby the primary phasing belt travels one product length for every pitch travelled by the flighted conveyor.



To achieve this ratio the system must know the **default product length,** so this is entered into the system either via an HMI or passed from another machine via an appropriate fieldbus for example.

A **sensor** (**S1**) on the primary phasing belt detects the leading edge of the incoming products and the speed/position of the belt is adjusted such that if the product were then to travel through the rest of the machine without any further adjustments being made the product would drop into the flighted conveyor in the correct position.

To achieve this result the system is provided with a parameter to adjust the '**primary target position**', i.e. where the flighted conveyor should be within one cycle when the product is detected on the primary phasing belt. The system is also provided with a parameter to adjust the **corrective primary phasing belt distance** over which the speed/position correction is made. Again, these parameters can either be entered via a local HMI or passed to the PLC from any other system via a fieldbus.

The intermediate conveyor has to travel at the same speed as the flighted conveyor (at a 1:1 ratio) constantly. This ratio serves two purposes; it ensures that if the product leaving the primary phasing belt is correctly placed that this placement remains correct and it also opens up a gap between the incoming products.

The secondary phasing belt has to travel at the same base speed as the flighted conveyor (at a 1:1 ratio) but its speed will increase or decrease dependant on the correction required This ensures that if the product leaving the intermediate belt is correctly placed that this placement remains correct and it also maintains the gap between the incoming products. If a correction needs to be made, then it can be done so from a starting speed that matches the intermediate conveyor thus not jerking the product too badly.

The secondary phasing belt is also fitted with a **sensor** (**S2**) that detects the leading edge of the products. The speed/position of the secondary belt is adjusted to ensure the products really do end up in the correct place on the flighted conveyor. These

adjustments are usually very small as they are only correcting small errors incurred during transfer of the products between the upstream belts.

To achieve this result the system is provided with a parameter to adjust the '**secondary target position**', i.e. where the flighted conveyor should be within one cycle when the product is detected on the secondary phasing belt. The system is also provided with a parameter to adjust the **corrective secondary phasing belt distance** over which the speed/position correction is made. Again, these parameters can either be entered via a local HMI or passed to the system via the fieldbus option card.

This example shows a machine with two phasing belts. Typically, as the machine speed increases the number of phasing belts also increases and rather than applying the whole correction on the primary belt the products are gradually brought into position over a number of belts. This limits the changes in speed needed on each belt and helps to avoid disturbance to the product itself. The number of belts that may be needed depends on things like product pitch, conveyor length vs speed, response time of system but two belts is a common configuration – one to do 'most' of the correction, the second to make any minor corrections.

In order for the PLC to know the absolute position of the flighted conveyor (within each pitch) it is necessary for the system to use a sensor (**S3**) on the flighted conveyor (placed to detect each flight) to initially reset the flighted conveyor position to zero (this only needs to be done once after power up). If there is one revolution of the encoder per flight pitch it may also be possible to use the encoder Z pulse for this function). In our example we use the DI0 of the intermediate conveyor drive to read the sensor status.

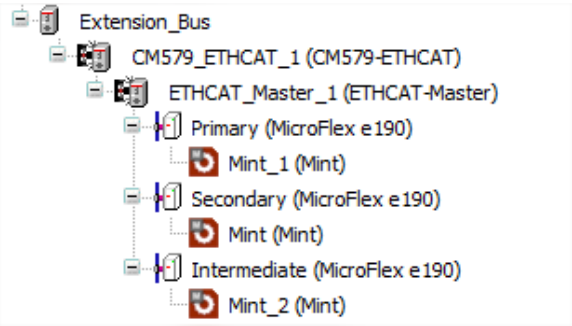
The sample code illustrates these main principles of operation. Depending on the placement of the incoming product some of the outgoing flights may remain empty (the downstream machine need to be able to cope with 'missing product'). All that is important is that incoming products eventually end up placed exactly within a particular flight.

Automation Builder - EtherCAT hardware configuration

In the example application we use the following axes…

Flighted belt	Virtual axis	: Position of this axis is derived from the encoder that would be fitted on this belt
Primary phasing belt	MicroFlex e190	
Intermediate belt	MicroFlex e190	
Secondary phasing belt	MicroFlex e190	

So for our example project we have added three MicroFlex e190 drives to the CM579-ETHCAT EtherCAT coupler in the device tree;



The following generic objects have been PDO mapped to the three belt drive and given appropriate variable names for both drives;

Object Description	Primary Drive Variables	Secondary Drive Variables	Intermediate Drive Variables
AX0_ControlWord_U16	PrimPhaseConvCW	SecondaryPhaseConvCW	IntermediateConvCW
AX0_TargetPosition_I32	PrimPhaseConvPosRef	SecondaryPhaseConvPosRef	IntermediateConvPosRef
AX0_TouchProbeFunction_U16	wAxis0TPFunction	wAxis0TPFunction_2	N/A
AX0_StatusWord_U16	PrimPhaseConvSW	SecondaryPhaseConvSW	IntermediateConvSW
AX0_ActualPosition_I32	PrimPhaseConvPosAct	SecondaryPhaseConvPosAct	IntermediateConvPosAct
AX0_TouchProbeStatus_U16	wAxis0TPStatus	wAxis0TPStatus_2	N/A
AX0_TouchProbePositionPos1_I32	diAxis0TPPos1	diAxis0TPPos1_2	N/A
AX0_TouchProbePositionNeg1_I32	diAxis0TPNeg1	diAxis0TPNeg1_2	N/A

AX0_TouchProbePositionPos2_I32	diAxis0TPPos2	diAxis0TPPos2_2	N/A
AX0_TouchProbePositionNeg2_I32	diAxis0TPNeg2	diAxis0TPNeg2_2	N/A
EncoderPosition_I32	diMasterEncoder	N/A	N/A
InputState_U32	N/A	N/A	IntermediateConvInputs

For more information on AC500 EtherCAT configuration go to: new.abb.com/drives/low-voltage-ac/motion - follow the links for 'Support' and open Application Note AN00205 from the 'Documents' section of the support site.

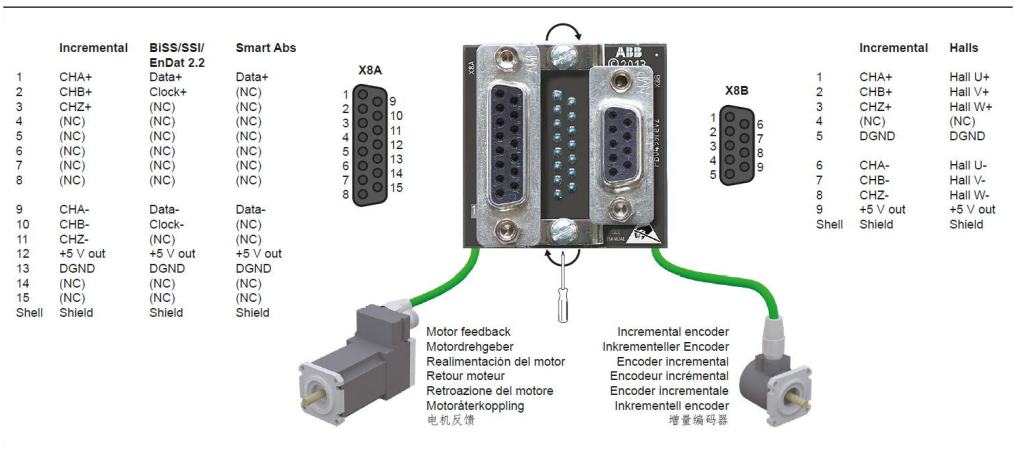
Drive wiring

When using EtherCAT, the cabling order of the EtherCAT slaves defines the slave addresses they will be given, with the first device being given the address 1001, the second 1002 and so on. It is important therefore that the drives are wired in the physical order they are defined in the Automation Builder device tree (i.e. the first drive in the EtherCAT network should be the Primary belt axis and so on).

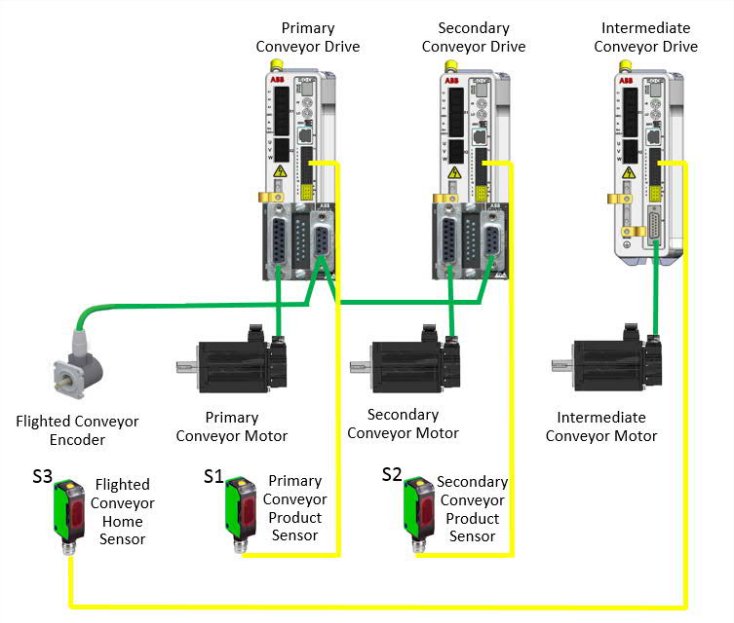
OPT-MF-200 wiring and configuration

For us to use the OPT-MF-200 there is no pre-configuration required. The function blocks we will use to configure the latches using the PDO objects we have mapped across.

The OPT-MF-200 should be wired as shown below, on the left (X8A) is the motor feedback device. The below diagram shows which encoder types are compatible (in addition to the types shown below also includes resolver with the use of the OPT-MF-201 resolver adapter). On the right (X8B) is shown the different options for the master (or auxiliary) encoder – of these the incremental variant is the most common and cost effective.



The system overview will look like this;



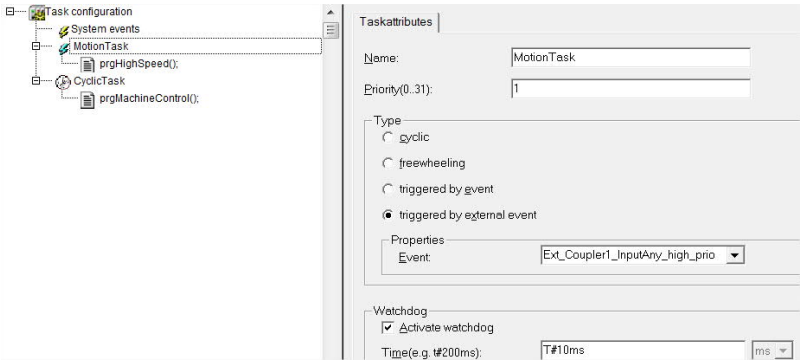
It is this second auxiliary encoder position which will be matched by the sensors mentioned earlier so this will be where the flighted conveyor encoder will be fitted.

Note: In the e190 it is also possible to use the built in additional X7 Encoder in / out rather than using the OPR-MF-201 (if configured correctly in the drive).

General application principles

In any AC500 PLCopen motion application it is normal to have an event driven task for processing motion related programs (POUs) and another slower speed cyclic task (or tasks) for any non-synchronous operations. Depending on your application requirements it's also common to require a task that occurs once only when the application starts.

The event driven (motion) task should be configured to be triggered by an external event (where this event is usually InputAny_high_priority for the slot in which the EtherCAT coupler is fitted);



Using InputAny synchronises the task with reception of the EtherCAT telegram which offers a slight performance advantage over Input2Any which synchronises the task with transmission of the EtherCAT telegram.

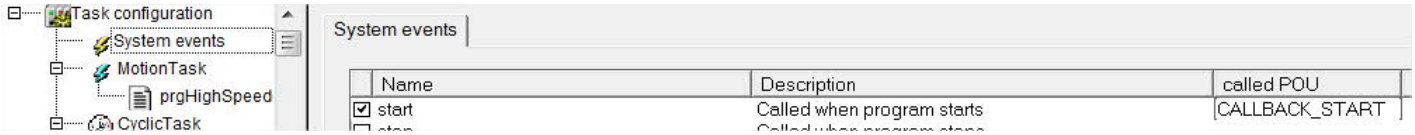
In our example the event driven task (MotionTask) calls a program named prgHighSpeed which in turn calls all the motion related POUs…

```
0001 PROGRAM prgHighSpeed
0002 VAR
0003
0004 END_VAR
```

```
0001 (* Add axes programs here *)
0002 prgFlightedConv();
0003 prgPrimPhaseConv();
0004 prgIntermediateConv();
0005 prgSecPhaseConv();
0006 prgPrimPhaseConvTP();
0007 prgSecPhaseConvTP();
0008 prgErrorCheck();
```

Our example also includes a 10ms cyclic task (named CyclicTask) that calls a program named prgMachineControl. This program is used for any logic that isn't time critical.

The 'start' system event is typically used to call a POU (program) that may be used to calculate or set initial values when the program first starts (i.e. for the first scan of the code only). Note that is it mandatory that this POU is named 'CALLBACK_START'.



In our example application we use this POU to initialise the starting values for the gear ratios for each belt.

General motion principles

The table below summarises the motion requirements for our example application axes:

Description	Flighted conveyor encoder	Primary phasing belt	Intermediate belt	Secondary phasing belt
Axis type	Virtual	e190	e190	e190
Address	N/A	1001	1003	1002
Master/Slave	Master	Slave	Slave	Slave
Axis name	**axFlightedConv**	**axPrimPhaseConv**	**axIntermediateConv**	**axSecondayPhaseConv**
Key motion	Position of this axis used as master reference for all slave axes	MC_GearIn: Belt travels one product length for every pitch travelled by the flighted conveyor MC_SetPositionContinuous: Axis position adjusted over known master travel, triggered by primary product sensor	MC_GearIn: Belt travels one flighted conveyor pitch for every pitch travelled by the flighted conveyor (i.e. 1:1)	MC_GearIn: Belt travels one flighted conveyor pitch for every pitch travelled by the flighted conveyor (i.e. 1:1) MC_SetPositionContinuous: Axis position adjusted over known master travel, triggered by secondary product sensor
Counts Per Rev (PPR x 4)	10000 (cpr encoder)	131072 (cpr encoder)	10000 (cpr encoder)	10000 (cpr encoder)
Enable Modulo	TRUE	TRUE	TRUE	TRUE
Modulo Range	'**lFlightedPPR**' (20000)	2^30	2^30	2^30
Scaled in	**counts**	**mm**	**mm**	**mm**
Mechanics	1 encoder rev = 1 pitch = 20000 counts	1 motor rev = Product length	1 motor rev = 200mm of belt travel	1 motor rev = 200mm of belt travel

Please refer to Application Note AN00205 for further information about the configuration of EtherCAT axes and Application Notes AN00241 for further information about the configuration and use of a master axis derived from an encoder input.

It is common for flighted conveyors to be physically constructed using imperial pitch chain drives (with a ¼" or ½" chain link). As a result the flighted pitch is very often either 4.5", 6" or 12" in size. For our example we have assumed a 12" pitch, but all other units are in mm, so our code includes logic to convert our 12" pitch into an equivalent distance in mm. To allow the code to be easily converted to work with any flighted conveyor pitch a global variable is included (lFlightedPitch) that defines the pitch in use…

lFlightedPitch: LREAL := 12; (*12inch pitch*) Global_Var

Use of MC_GearIn

All three profiled axes in our application are geared to the flighted conveyor encoder using MC_GearIn (as shown by the previous table).

As the primary phasing conveyor must travel one product length for every pitch of the flighted conveyor, and this product length may vary from product to product, our application includes a retain persistent variable (IProdLength) that the user will set to suit whatever product is being run through the system (this may be entered via an HMI or may come from an upstream system via an appropriate fieldbus).

For all the axes the gear ratio to be used by MC_GearIn is defined by…

Ratio = 'Distance travelled by slave' / 'Distance travelled by master'

From this we can calculate the gear ratios for all axes as shown below;

lPrimaryRatio := lProductLength / lFlightedPPR; (*Calc Primary belt ratio*)

lIntermediateRatio := (IFlightedPitch * 25.4) / lFlightedPPR; (*Calc Intermediate belt ratio*)

lSecondaryRatio := (IFlightedPitch * 25.4) / lFlightedPPR;(*Calc secondary belt ratio*)

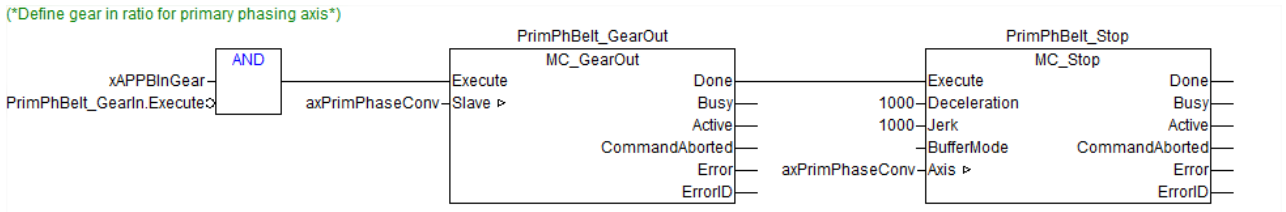
Note Our constant for the pitch of the flighted conveyor has been multiplied by 25.4 to convert the 12" belt travel into mm

Note In the example program the ratio calculations are carried out at the first start of the CPU in the **CALLBACK_START POU**. If these need to be changed during normal running then this code will need to be moved into a cyclically executed POU.

Our overall ratios are all fractional (LREAL's) but MC_GearIn requires that the required ratio is defined via two INTEGER inputs (RatioNumerator and RatioDenominator)). As these are integer values the maximum value for either of these is 32767 so this must be considered when configuring the MC_GearIn code. The easiest way to solve this is to employ code as shown below…

RatioDenominator should either be a value of 1, 10, 100, 1000 or 10000…..select the highest possible value that still doesn't result in RatioNumerator exceeding 32767. This will ensure the most number of decimal places are used for the overall ratio.

Once MC_GearIn is active the axis remains geared to the defined master axis until MC_GearOut is executed or an error occurs. After executing MC_GearOut the axis will continue running at its last demand velocity until MC_Stop is executed to bring it to a controlled stop. A snippet of code outlining this from our example application is shown below for reference…



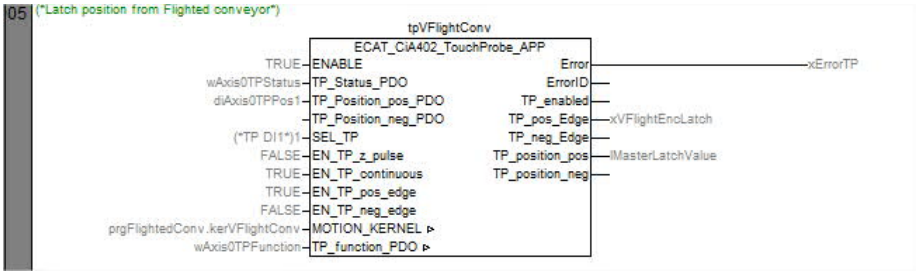
Use of Encoder splitter OPT-MF-100 and Master Encoder on board e190

Once the drive has been wired and configured to use either the OPT-MF-100 or the master encoder input on the e190 and the relevant sensor input wired to DI1 we must now how to configure the 'latch' on the drive. To do this we must add 'ECAT_CiA402_TouchProbe_APP' function blocks to our program and configure them to use the positive edge of the sensor wired to Digital input 1 (DI2 can also be used if needed). Please refer to AN00241 (*Using e190 or e180 encoder channels for master encoder input*) for mode information.

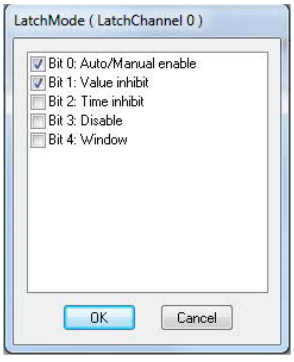
Once the ECAT_CiA402_TouchProbe_APP has been configured and used in the running program you will see that the settings in the drive for Latch Channel 0 will change as per your configuration.

Latching Positions

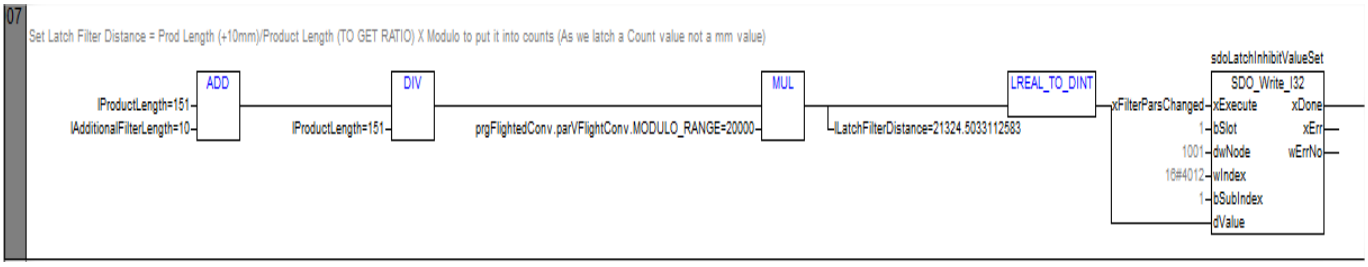
In our example application these two latches of the flighted conveyor encoder are performed using functionality built into the ECAT_CiA402_TouchProbe_APP function block that is used in combination with the e190 Encoder splitter OPT-MF-100 aux encoder input in our PLC system. The sensor for the primary phasing belt is wired to input DIN1 on the e190 (and all latch values relate to latch channel 0) whilst the sensor for the secondary phasing belt is wired to input DIN2 (and all latch values relate to latch channel 1). Please refer to application note AN00221 for further details about configuring and using the e190 fast latch functionality.



To avoid spurious latching of the encoder causing mal-operation of the phasing belt (e.g. false detection of a rising edge on the trail edge of the product, maybe due to the nature of the product packaging) the application code using the latch filtering within the e190 drive. We do this by setting the latch up to use an inhibit value (a number of encoder counts that must elapse between valid latch values. (See AN00241 for further details).



There must also be the means of filtering out unwanted signals from the product sensors. We do this by setting a Latch Inhibit value which will set a distance the encoder value must increase by before a new latch signal is considered to be valid. The logic shown below will achieve this by writing the latch value which is scaled in counts. So from the front edge of the first product it must wait one modulo of the master encoder plus an additional filer length set in mm but converted to counts.



The SDO write block then (when executed) changes the setting for the latch channel inhibit value as per the calculations.

Parameter	Active
LatchEnable (LatchChannel 0)	C 1
LatchInhibitTime (LatchChannel 0)	FD 0 ms
LatchInhibitValue (LatchChannel 0)	C 21325.0000
LatchMode (LatchChannel 0)	C 0x0003
LatchSource (LatchChannel 0)	C Encoder value
LatchSourceChannel (LatchChannel 0)	C 2
LatchTriggerChannel (LatchChannel 0)	FD 1
LatchTriggerEdge (LatchChannel 0)	FD Positive edge
LatchTriggerMode (LatchChannel 0)	FD Digital input
LatchValue (LatchChannel 0)	RO 0.0000
LatchWindowStart (LatchChannel 0)	FD 0.0000
LatchWindowDistance (LatchChannel 0)	FD 0.0000

Use of MCA_SETPOSITIONCONTINUOUS

Both the primary and secondary phasing belts are used to shift the products relative position so that it eventually drops into the flighted conveyor correctly. This is achieved by sensors on each belt (i.e. one on each belt) detecting the lead edge of a product and at this same point latching the position of the flighted conveyor (which wraps every pitch) and comparing this to an “ideal position” the operator has configured (thereby measuring the error in flighted conveyor position which can then be translated back into required movement of the relevant phasing belt).

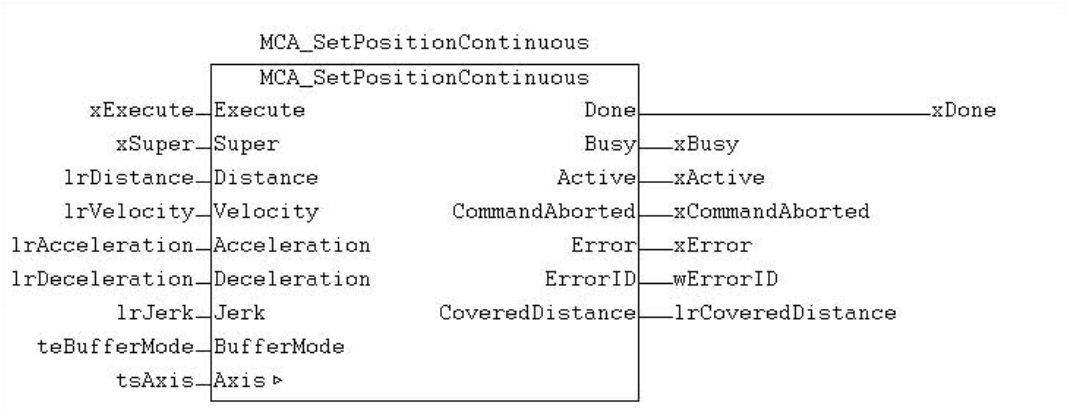
Once a valid latch value has been obtained this is then compared against the “ideal” value previously configured by the operator (i.e. where the flighted conveyor should be within one pitch when the product lead edge is detected by each belt’s sensor). The corresponding error in position of the phasing belt itself can be calculated by considering the nominal gear ratio between the phasing belt and the flighted conveyor…

*Primary phasing error = Measured flighted conveyor error * (Product Length / Flighted conveyor pitch)*
*Secondary phasing error = Measured flighted conveyor error * 1*

The code has to consider the “shortest direction” to apply the correction for these errors (i.e. it may be quicker to retard the belt a short distance rather than advance it a long distance). We will explain later why the belt travel must be calculated from the measured error in position of the flighted conveyor.

Now that we’ve seen how the primary and secondary phasing belts use MC_GearIn to establish the required geared motion and how product sensors on the two phasing belts are then used to calculate the amount of correction required to ensure the product remains in position to drop into the flighted conveyor correctly, we can look at the PLCopen motion command used to apply these corrections….**MC_SetPositionContinuous**.

MCA_SETPOSITIONCONTINUOUS explained

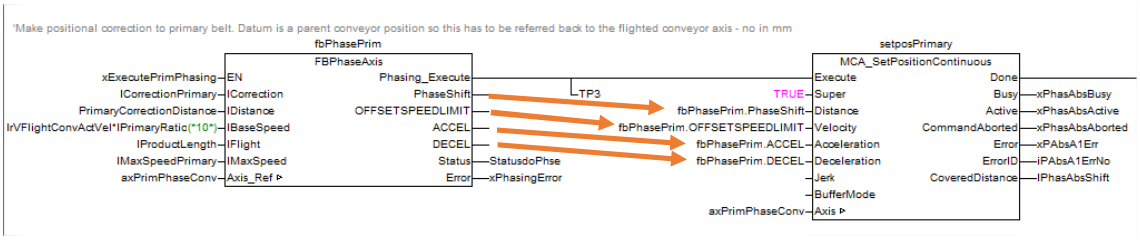


This Function Block shifts the coordinate system of an axis by manipulating set-point position of an axis (as we are using it with SUPER = TRUE). This can be used for instance for a reference situation “on the fly” where no abrupt position change is allowed, eg when a slave axis is linked to the modified axis. This Function Block can also be used during motion without changing the commanded position, which is now positioned in the shifted coordinate system. A continuous position correction will be achieved, with a defined profile.

With Super = TRUE, the axis will hold the setpoint position while an offset “phase shift” is applied to the actual position. This will result in a movement as the position control loop will keep the distance between setpoint- and actual position constant. A slave axis will not see this movement and will not follow. When the block is ready, the axis will have moved physically by - Distance but the positions in AXIS_REF will not have been changed.

Our application code for the primary and secondary phasing belts captures the flighted conveyor encoder and compares this in each case to a stored “ideal value”, so we already have the information needed to set the PhaseShift input parameter for our MC_SetPositionContinuous function block. But earlier we also calculated the equivalent travel the belts themselves (i.e. the slave axes) would make, this data is needed to allow us to calculate values for the velocity, acceleration and deceleration input parameters so that the phase shift occurs over a known travel of the master axis.

Our application calculates these profile parameters via a user defined function block named **FBPhaseAxis**. An example usage of this function block is shown below…



For lMaxSpeed I used two variable lMaxSpeedPrimary and lMaxSpeedSecondary which are both set to 2000mm/sec, but could be adjusted to suit the mechanical constraints of the machine mechanics.

The following table describes the inputs and outputs used with the FBPhaseAxis function block:

	Data type	**Description**
Inputs		
EN	BOOL	Function block is processed if this input is TRUE
lCorrection	LREAL	Distance the slave axis needs to be adjusted by
lDistance	LREAL	Master distance over which the slave axis adjustment should take place
lBaseSpeed	LREAL	The current speed of the slave axis (without any adjustment in progress) (uu/s)
lFlight	LREAL	The size of the flighted conveyor pitch
lMaxSpeed	LREAL	Maximum speed the slave axis is allowed to reach during adjustment (uu/s)
Axis_Ref	AXIS_REF	The axis that needs to be adjusted
Outputs		
Phasing_Execute	BOOL	Activates when the function block has calculated all necessary profile parameters
PhaseShift	LREAL	The distance the master axis needs to be shifted by
OffsetSpeedLimit	LREAL	The maximum change in speed to be used by the MC_SetPositionContinuous function block
Accel	LREAL	The acceleration to be used by the MC_SetPositionContinuous function block
Decel	LREAL	The deceleration to be used by the MC_SetPositionContinuous function block
Status	BYTE	Indicates type of adjustment required (Bit 0 = Triangular, Bit 1 = Trapezoidal, Bit 2 = Positive adjustment, Bit 3 = Negative adjustment

Please contact your local ABB Office for program examples and further support.

Contact us

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

new.abb.com/drives/low-voltage-ac/motion
new.abb.com/drives
new.abb.com/channel-partners
new.abb.com/plc