Motion Control Products

Application note
Indexing Conveyor

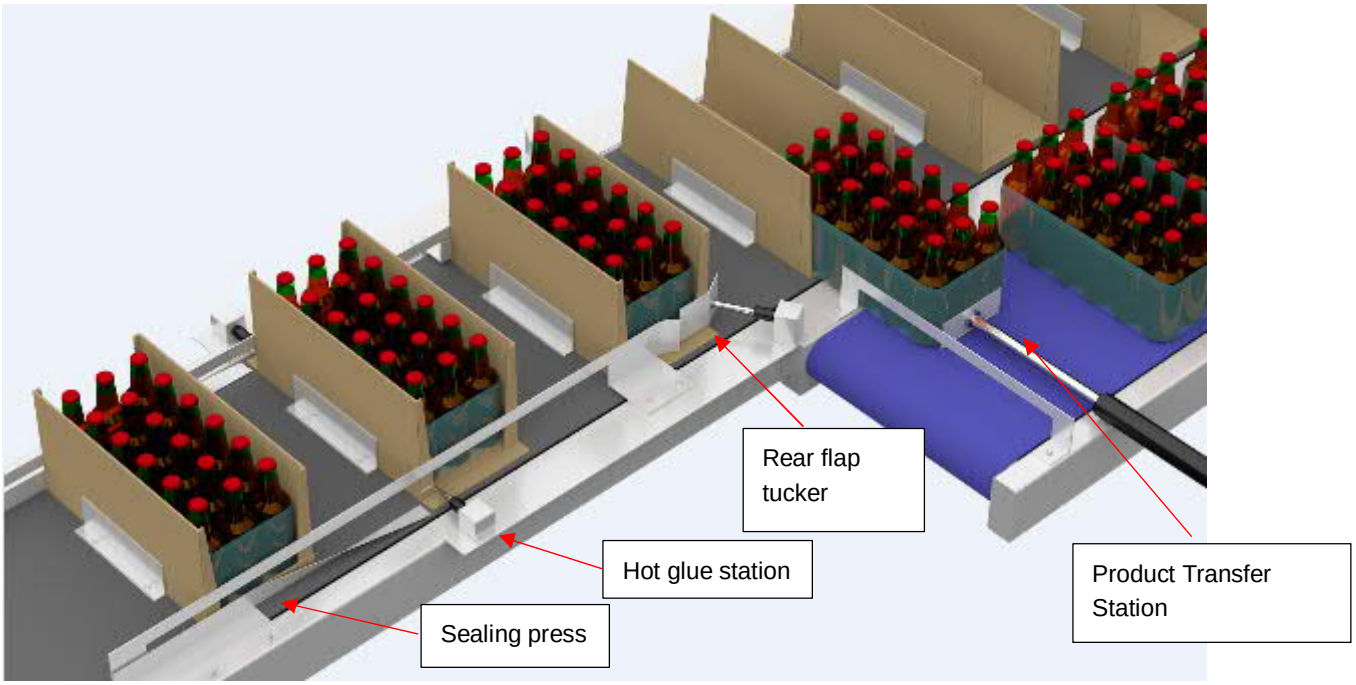
AN00230

Rev C (EN)

ABB motion products use a feature rich high-level programming language called Mint. The event driven programming capabilities of Mint make it ideal for indexing conveyor applicatons

Introduction

The Mint motion language provides an extensive range of specialized functions to interface to the hardware of ABB controllers and includes a variety of keywords to vastly simplify the implementation of complex motion control solutions.
Mint is a sequential language (i.e. a particular line of code is not executed until the previous line has completed execution) that also provides the user with the ability for parallel processing (via multi-tasking) and event/interrupt driven operation (e.g. via digital input events) making the language as a whole incredibly flexible and particularly suited to machine and motion control applications.
In this application note we are going to discuss some of the methods that could be used in the control of a simple indexing conveyor. This application note comes with a sample program that will run on both ABB MotiFlex e180 and MicroFlex e190 demo kits and a HMI project for an ABB CP600 HMI.



Indexing conveyor example

For this indexing conveyor example, a flighted conveyor similar in operation to the one shown above, feeds pre-folded cardboard display cases to a product transfer station. At this point the products are transferred into a waiting display case before the cases pass by a hot glue application station. The cases are then sealed using a series of guide rails and pressure plates.

The example is based on the following hardware implementation:

- A flighted conveyor with a 12 inch flight pitch, driven by a timing belt connected to a pulley with 30 teeth at a 10mm pitch (300mm circumference)
- Between the final drive pulley and the AC servo motor is a 3:1 speed reduction gearbox
- An ABB brushless AC servo motor with encoder resolution of 131072 counts per revolution
- An ABB MotiFlex e180 (or MicroFlex e190) servo drive
- An ABB CP600 series HMI
- Two hot melt glue application heads – controlled directly by the MotiFlex e180 (or MicroFlex e190)
- A proximity sensor for detecting conveyor flights and latching their position via a fast digital input on the drive

In a real application an ABB AC500 PLC may act as the main supervisory controller for this machine and ensure that the cardboard blank is in position and that the products have been successfully accumulated before transferring them to a waiting display case. However it is beyond the scope of this document to go into the detail of the PLC application. For the purposes of this exercise we will simulate an input signal to the servo drive, signaling that products have been transferred and the index should start via digital input 2.
Our system also has a sensor to detect the flights as they pass by connected to digital input 1 (not pictured in the above diagram). This sensor utilizes the drive's fast latch functionality to capture the actual position of the flights as the index takes place and allows the drive to calculate a new target position "on the fly". This ensures the conveyor always stops a known distance past the sensor and ensures there is no drift resulting from cumulative floating point inaccuracies and, to a certain extent, for stretch and wear of the mechanical setup.

As the glue is to be applied to the case during motion the drive will control two hot melt glue heads utilizing Mint sentinels to control the digital outputs that trigger them. The drive provides a further digital output, digital output 1 (which will be on when motion is in progress), to serve as a 'motion in progress' interlock with the PLC.

We will now discuss the main points of the sample code.

Scale factor

The scale factor allows an axis to be scaled into engineering units for ease of use and can take into account any mechanical linkages such as gearing. The scale factor is applied to all motion variables for an axis (speed, acceleration, move distance, etc.). The default setting for an axis scale factor is 1, meaning that all motion is recorded in encoder counts.
For this example we wish to scale our axis into linear mm of travel.
The scale factor is set by the SCALEFACTOR keyword….

```
SCALEFACTOR (axis) = value
```

So for the example mechanical setup of our indexing conveyor our scale factor can be worked out as follows:
We are driving our 300mm pulley with a 3:1 speed reduction gearbox.

- Three revolutions of the motor will produce one revolution of the pulley
- One motor revolution = 131072 encoder counts
- So 3 x 131072 encoder counts = 1 pulley rev = 300mm effective linear travel of our conveyor belt
- Counts per mm can therefore be found from 393216 / 300 = 1310.72

Therefore our SCALEFACTOR for the axis can be set to 1310.72 to represent linear mm of travel. As indexing conveyors are often constructed using chain drives with 0.25" or 0.5" links are default index will be specified by the user (via the HMI) in inches (and the program will convert this to mm of linear travel).

Indexing the conveyor

Mint events are interrupt routines that take priority over other code modules such as tasks. These set events have a hierarchy (see the Mint help file for details) and no task processing will occur while an event is executing. Because of this we don't use an event to execute our index move, instead we use an input event to initiate the running of a task. This means that we can be sure that when we get the input signal we register it immediately, process the input event as quickly as possible and therefore allow multi-tasking of the main program tasks to resume with minimum disruption.

```
POS(_axIndex) = 0
bFlightSeen = _false
INCA(_axIndex) = nfINITIAL_TARGET_POS * 25.4
GO(_axIndex)
OUTX(_opIndexInProgress) = _on
Pause(IDLE(_axIndex))
OUTX(_opIndexInProgress) = _off
```

Firstly we set our value of axis position (POS) to zero. This gives us the same starting point each time we perform an index, this proves useful for our glue application and simplifies the new target position calculation that we will discuss later. We then issue an INCA move to the initial target position (which is set to the default flight pitch). We use an incremental move type for this as it allows the target position to be modified whilst motion is in progress making this ideal for our application where we will need to change the target position when we detect a flight.
An output is switched on at this point and serves two purposes. Firstly it acts as an interlock with the PLC to signal that motion is in progress, preventing product transfer or blank insertion and secondly as an interlock with the latch event that we will discuss next. You may notice that this section of code contains a program flag/variable "bFlightSeen". This flag is set to false before we execute the initial index move. The flag is set to true in a latch event which is called when the flight sensor sees a flight. This forms the basis for some logic to monitor for correct operation of the flight sensor.

```
If (nMissedCount < _nMissedLimit) Then
  If bFlightSeen = _false Then nMissedCount = nMissedCount + 1
Else
  Run (ErrorHandler)
```

If the flight sensor doesn't see a flight a counter is incremented by one, eventually generating an error when the allowable limit is reached. If a flight sensor is detected the missed flight counter is reset to zero (so the error only occurs if a number of successive flights are missed). This ensures high machine availability without compromising on reliability.

The same indexing routine can be used to initially home the conveyor. In this case the program automatically prevents the application of glue during homing. A digital output could be provided as a "system homed" status for use by the supervisory PLC (this is omitted from the code, via conditional compilation, when using the MicroFlex e190 due to the limited number of digital outputs on the standard demo kit version of the drive).

The ABB MotiFlex e180 and MicroFlex e190 drives support a flexible latching system capable of latching both an encoder value and/or axis position using digital inputs, digital outputs and encoder Z pulses as a trigger. We use a latch in this application to ensure that the conveyor is in the correct position at the end of each cycle.

A latch channel is configured in the startup block of the program. Before we discuss what the event does let us briefly look at setting up the latch:

```
LATCHENABLE(1) = 0 'disables the latch channel to allow us to configure it
LATCHSOURCE(1) = _lsAXIS_POSITION 'what we are going to latch? We are going to latch position
LATCHSOURCECHANNEL(1) = 0 'axis number of the source
LATCHTRIGGERMODE(1) = _ltmINPUT 'what will trigger the latch event? In this case the flight sensor digital input
LATCHTRIGGERCHANNEL(1) = 1 'input number the flight sensor is connected to
```

```
LATCHTRIGGEREDGE(1) = _ltePOSITIVE_EDGE 'our latch event will be triggered by the rising edge of the input
LATCHMODE(1) = _lmAUTO_ENABLE 'when the latch has occurred the channel will re-enable ready for the next input
LATCHENABLE(1) = 1 'we enable the latch channel now the configuration is done
```

So now we have set up the latch let's look at what the event does.
We mentioned earlier that a digital output acts as an interlock with the latch event. We use this along with a check that we have not already seen a flight to prevent any false operation of the sensor triggering an index or a double index.

```
If (OUTX(_opIndexInProgress) = _on) AndAlso (bFlightSeen = _false) Then
```

This ensures operation of the flight sensor input has no effect on the operation of the conveyor unless an initial index was already in progress and a flight had not already been seen.
If the conditions of the "If statement" are met we can then calculate the actual target position required from the position that was latched. Because we previously set the value of POS to zero this is a simple case of adding a defined offset value to the latched value (where the offset value is adjusted to stop the card blanks in the correct position for the product transfer station.

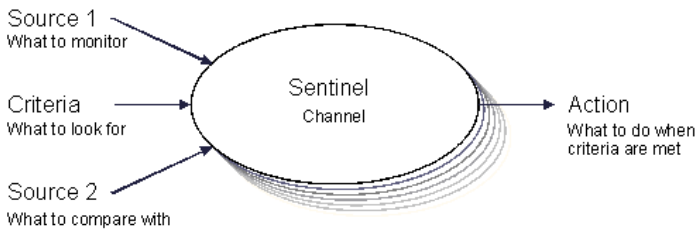
```
fActualTargetPos = LATCHVALUE(1) + niOFFSET
```

Note that the offset parameter is entered in mm (and for this example is configured for whole mm units only).

Some consideration should be given to how far away the sensor is located from the transfer station. When a new INCA command is issued the target position changes immediately. If the new target position is just in front of the previously demanded position and the axis is moving at such a rate that there is insufficient distance to decelerate and achieve the target position the axis will decelerate to a stop and then reverse to the new target position. To avoid this happening we position the sensor a greater distance away from the transfer station than the required deceleration distance.

Applying glue to the cases

As mentioned at the beginning of this example, the drive will control the two hot melt glue heads. To make the glue position on the case easily adjustable we make use of the Mint sentinel keywords to control them.
Sentinel channels monitor input sources and compare against set criteria to carry out actions when the criteria is met.



This makes them ideal for the control of our glue heads. Our system makes use of 4 sentinel channels controlling 2 glue heads (one sentinel per target glue position).

Let's look at the setup of a sentinel channel (channel one) which is done in the start-up block of the program.

```
SENTINELACTIONMODE(1) = _samOFF - Sentinel action off to allow the channel to be configured.
SENTINELSOURCE(1) = _cpDEMAND_POSITION - The source to be monitored, in this case demanded position
SENTINELSOURCEPARAMETER(1) = 0 - Axis 0
SENTINELTRIGGERMODE(1) = _ctmINSIDE_BOUNDS - Sentinel action true inside the bounds of the high and low values
SENTINELTRIGGERVALUEFLOAT(1, _stvLOW) = nfGLUE_START - Lower limit or glue on in the case
SENTINELTRIGGERVALUEFLOAT(1, _stvHIGH) = nfGLUE_STOP - Upper limit or glue off in this case
SENTINELACTION(1) = _saOUTX - Sentinel action will be control of a digital output
SENTINELACTIONPARAMETER(1) = 1 - digital output one will be controlled by sentinel channel one
```

So that's the basic setup of one of our channels. This is repeated for each of them putting the appropriate variables in for the trigger values and assigning the required action parameter.
If the sentinel channel was to be active all the time we would also set SENTINELACTIONMODE(1) = _samAutomatic here. However the ABB AC500 PLC is keeping track of when glue is required (i.e. whether a card blank is present in a particular flight) so has the ability to turn the sentinel channels on and off via a digital input on the drive. Likewise the user may need to adjust the position or length of the glue on the case so our trigger values will need to be updated as and when. They are updated on the HMI using Net data events.
Input three is the input designated to control whether glue is required or not. The input is configured to be both rising and falling edge triggered meaning that the event will be called whenever the input changes state. The event calls a subroutine containing the following code for when the input state goes from low to high…

```
For s = 1 To 4
  SENTINELACTIONMODE(s) = _samAUTOMATIC
Next s
```

The loop will effectively turn on all four sentinel channels. If the input goes from high to low another For loop will run instead setting SENTINELACTIONMODE(s) = _samOFF.

To adjust the glue on and off positions we need to update the associated SENTINELTRIGGERVALUEFLOAT values. As the setup of the sentinel occurs within the startup block of code, it is only run once so in order to alter these settings we call a NETDATA event whenever the associated HMI data location is updated to store the new setting.

Other modules of code

In the sample program the 'Stop Handler' framework is taken from the Mint sample library in Workbench and adapted to suit this single axis application. Input 0 is designated as our stop input and configured to be active low.
In addition to the configured stop input, the MotiFlex e180 and MicroFlex e190 drives benefit from on board STO functionality (Safe Torque Off) making SS1 (Safe Stop category 1) easily achievable with the use of an ABB Jokab safety relay. Please refer to AN00205 (Implementing safety functions on ABB servo drives) and the MotiFlex e180 or MicroFlex e190 installation manuals for further information.

When an error occurs the ONERROR event is called. The event included in the sample code prints the error code, description, line number, and time to the terminal window. It also uses the subroutine "doPackString" to write the packed error string to a Netdata location to be displayed on the HMI. It then runs a task "ErrorHandler". The task then awaits the reset button to be pressed before clearing the error active flag.

Backup and restore of HMI data

The main program loop runs the subroutine "doBackUp". This runs constantly and uses "For loops" to write each used netfloat and netinteger value (i.e. the data from the HMI) to a corresponding non-volatile memory location. Sub "RestoreHMIData" is called at the beginning of the startup block running once and writes the non-volatile memory back to netdata in the same manner.

Sub "doLoadDefaults" contains some initial values for the HMI. If nvINITIALISED (a Mint macro to define one of the non-volatile locations) doesn't contain 123456789 - for example if it is a new drive straight out of the box with no values in non-volatile memory. The first time the program is downloaded and run it will load the pre-set default values, writing 123456789 to nvINITIALISED to prevent the default values being loaded on every power cycle.

HMI example project

The example HMI project (suitable for use with Panelbuilder v2.6.0 or later) is designed to operate on a CP620 HMI connected to the drive via Modbus TCP (i.e. the drive's standard Ethernet socket). The MicroFlex e190 and MotiFlex e180 drives use IP address 192.168.0.1 by default so the HMI must be configured to operate on this same subnet (we used 192.168.0.20).

Please refer to the Panelbuilder user manual for further details on the operation of the HMI and its programming software.

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