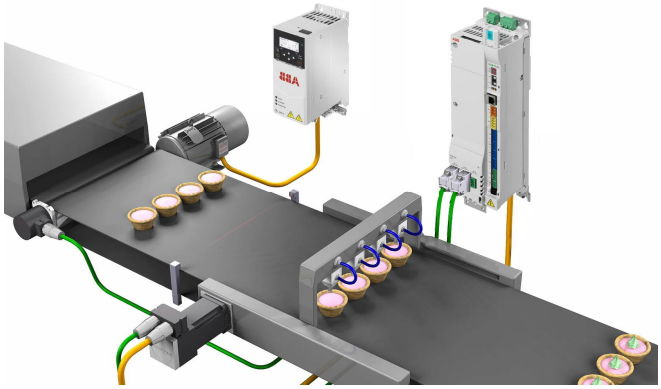
Motion Control Products

Application note
Synchronizing a process to a product

AN00180

Rev C (EN)

Mint is an easy to use high-level programming language rich in features, including facilities to write modular, block-structured programs.
The Mint language includes subroutines, functions, tasks, structures (user-defined data types), conditional statements, looping statements, etc



Introduction

Mint also 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.
This application note provides an insight into some of the techniques which can be used to easily implement a machine that requires a process to be synchronized to a passing product such as that illustrated by the animation at https://new.abb.com/drives/low-voltage-ac/motion.

Product synchronization animation

The product synchronization animation has the following features:

- Servo controlled dispenser axis (brushless AC servo motor)
- MotiFlex e180 AC servo drive
- Fitted in the MotiFlex e180 is a Mint enabled memory module (+N8020). This allows the axis to be programmed using Mint using the drives incremental encoder input and digital I/O
- ACS micro drive to control the conveyor speed
- Registration sensor connected to a fast latch input on the drive
- Auxiliary encoder connected to the auxiliary encoder input on the drive

The sample code shows how an axis can be synchronized to a point on another axis, in our case the batch of cakes. We use the fast latch functionality to capture the position of the cakes by reading a value from an encoder attached to the conveyor. This fast latch is triggered when the cakes pass a registration sensor. From this measured position we can set a position trigger value at which point our axis will start moving to synchronize itself with the batch of cakes. Once our axis is synchronized with the cakes icing can be dispensed. The move types used to achieve this are known as flying shears.

Axis configuration

The example application uses the following axes…

(a) Product conveyor (used to feed the products through the machine, this axis is controlled by an inverter. The conveyor has an auxiliary encoder connected to the MotiFlex e180's Mint enabled memory module)
(b) Icing dispenser (the MotiFlex e180 controls the axis that carries the icing dispensers, the icing is dispensed when a digital output on the MotiFlex e180 is turned on)

The Icing dispenser axis is configured in the Mint Startup block.

Mint program

Program constants

To make the code easier to read and to allow the program to be easily updated later some constants are used to define certain aspects of the application. The icing dispenser axis, conveyor encoder channel, input and output numbers, icing duration and a dwell distance are all declared as constants. By utilizing these constants, instead of 'hard coding' numerical data later in the program, any changes made to the application parameters can be easily updated by changing the values in these declarations.

```
Const _axDisp As Integer                = 0     'Axis with icing dispensers
Const _EncChannel As Integer    = 1     'Encoder input channel
Const _ipProdReg As Integer     = 2     'Product registration input
Const _opIcing As Integer       = 0     'Icing dispenser output
Const _IcingTime As Integer     = 500   'Duration Icing should be dispensed in ms
'Distance from registration sensor to position when axis must begin move
Const _DwellDistance As Float   = 200
```

Variable declarations

The various master distances used in the flying shears are declared as variables, calculated as fractions of the minimum distances between cakes.

```
Dim fMinDist As Float   = 1000              'mm   Min pitch between cakes
'Master Distances
Dim fMSD1 As Float      = fMinDist * 0.15   '150mm
Dim fMSD2 As Float      = fMinDist * 0.35   '350mm
Dim fMSD3 As Float      = fMinDist * 0.15   '150mm
```

Scale factors

The scale factor allows each axis to be scaled into engineering units, for ease of use. The scale factor is a division factor that is applied to all motion variables for an axis (speed, acceleration, move distances, etc.) and is set using the Mint SCALEFACTOR keyword.

The number of edges generated by the feedback device for movement of one user unit (uu) defines the SCALEFACTOR. By default, the SCALEFACTOR is 1 which means all axis motion parameters are defined in encoder quadrature counts.

The Icing dispenser axis mechanics are such that there are 200 encoder edges for a movement of 1 millimeter (mm) in a linear direction. Therefore, setting the SCALEFACTOR to 200 allows moves to be demanded in linear mm.

The conveyor encoder can also be scaled within the mint program using the ENCODERSCALE keyword. The mechanics of the conveyor are such that there are 400 encoder edges per linear mm of conveyor travel so setting ENCODERSCALE(_EncChannel) to 400 will allow keywords relating to distance traveled by the conveyor (the master axis) to be programmed in linear mm.

Master slave configuration

To allow us to use flying shears we must set up the relationship between the master (Conveyor) and slave (Icing dispenser) axes. This is done in the startup block of the mint program with the following code.

```
MASTERSOURCE(_axDisp)       = _msENCODER
MASTERCHANNEL(_axDisp)      = _EncChannel
```

This code sets the master which the slave axis will synchronize with. In this case the icing dispenser axis will be synchronized with the encoder on channel 1. This is the encoder measuring the position of the conveyor which is connected to the Mint enabled memory module.

Trigger configuration

By using triggering, we can cause moves to be performed without issuing a GO command. The following code is included in the startup block of the mint program.

```
TRIGGERMODE(_axDisp)        = _trFWD_MOTION
TRIGGERSOURCE(_axDisp)      = _tsENCODER
TRIGGERCHANNEL(_axDisp)      = _EncChannel
TRIGGERVALUE(_axDisp)       = -9999                 'Initially disable triggering
```

This code sets axis 0 to trigger motion when the value of encoder channel 2 passes -9999 user units in a forward (positive) direction.

In the Mint startup block we have also set;

```
ENCODERWRAP(_EncChannel) = 1000
```

This is the maximum value the encoder will reach before returning (wrapping back) to zero. This ensures that operating values for this encoder are only ever in the range 0 to 999. As a result, setting a TRIGGERVALUE of -9999 (a value the encoder can never reach) effectively temporarily disables triggering of motion from the encoder.

Fast latch configuration

To set up fast latching on e180 products a selection of LATCH…… keywords are used. When using a MotiFlex e180, latch channels 0-3 are used to latch hardware relating to the drive (e.g. the base encoder or axis position).

When latching the value of the encoder on the Mint enabled memory module the input or output used to trigger the latch must also be on the Mint enabled memory module. Inputs 1 and 2 are fast inputs on the e180 and outputs 0 and 1 are the fast outputs.

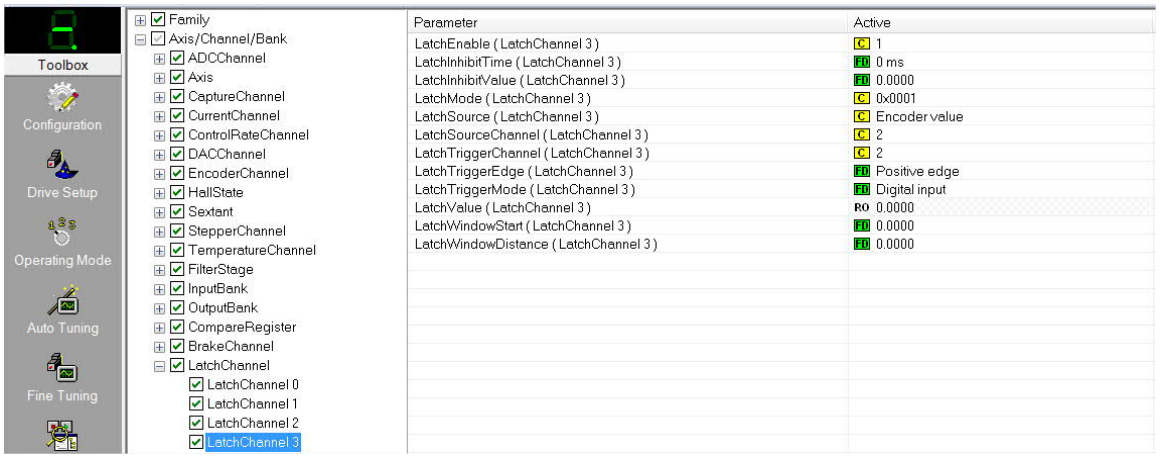
Using the code below in the startup block of the Mint program we can configure latch channel 3 to capture the position of the conveyor axis on a rising edge of digital input 2.

```
LATCHTRIGGERCHANNEL(3)      = _ipProdReg
LATCHTRIGGERMODE(3)         = _ltmINPUT
LATCHTRIGGEREDGE(3)         = _ltePOSITIVE_EDGE
LATCHSOURCE(3)              = _lsENCODER
LATCHSOURCECHANNEL(3)       = _EncChannel
LATCHMODE(3)                = _lmAUTO_ENABLE
LATCHENABLE(3)              = _true
```

The latch is set to automatically re-enable itself and initially enabled ready for use.

You can view the settings of the latch channels in the parameter viewer in Mint Workbench. Initially only latch channels 0-3 are displayed as these are the channels associated with the drive. If you have set up an additional latch channel and you cannot see it in the parameter viewer you will need to disconnect and reconnect the drive from workbench to cause the parameter viewer to update. To do this, click this icon  . It is located in the top left-hand corner of Mint Workbench, pressing it once will disconnect and pressing it again will reconnect.

Once set up the parameter viewer will display the following settings for the latch channel, see image below.



Main program

Within the main program loop triggering is disabled, the flying shear moves are loaded into the move buffer and then the program waits for the moves to be triggered. This is achieved with the following code.

```
TRIGGERVALUE(_axDisp) = -9999                    'Disable triggering
doLoadFlys                                       'Load fly's into movebuffer
'Wait until moves have been triggered or axis is idle
Pause ((MOVESTATUS(_axDisp) & 256) = 256) Or (IDLE(_axDisp))
```

We start by setting TRIGGERVALUE to -9999 which disables triggering (see previous section on trigger configuration for the reason why). We then call the sub routine doLoadFlys which loads the flying shear moves into the move buffer along with a command to turn on an output to dispense the icing. We will look at this subroutine further later on. The program then waits for the moves to be triggered. When a batch of cakes pass the registration sensor an event will occur which sets a TRIGGERVALUE that will cause the icing dispenser axis to start the flying shears when the conveyor reaches a certain position past the sensor. The whole process is then repeated for the next line of cakes.
To improve the speed of the program (and hence overall machine speed) we can start to load the next set of flying shears as soon as we detect the current set of moves has been triggered (MOVESTATUS of 256).

If we detect the icing dispenser axis has become IDLE we also start to load the next set of flying shears. For IDLE to be true there must be no moves loaded in the move buffer, the axis must be stationary, and it must be in position.

Subroutine – loading the flying shear segments

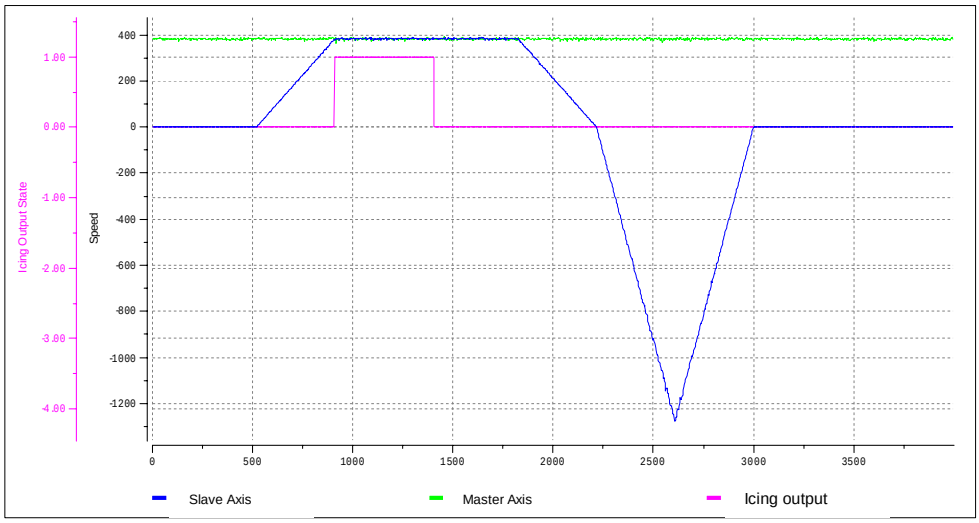
The doLoadFlys sub routine loads the flying shear moves into the move buffer ready to be triggered when required.

```
MASTERDISTANCE(_axDisp) = fMSD1                  'Ramp up to synchronous speed
FLY(_axDisp)            = fMSD1/2

MOVEPULSEOUTX(_axDisp, _opIcing) = _IcingTime    'Dispense Icing

MASTERDISTANCE(_axDisp) = fMSD2                  'Synchronized with product
FLY(_axDisp)            = fMSD2

MASTERDISTANCE(_axDisp) = fMSD1                  'Ramp down to stop
FLY(_axDisp)            = fMSD1/2

MASTERDISTANCE(_axDisp) = fMSD3                  'Return to start position
FLY(_axDisp)            = -(fMSD1+fMSD2) /2

MASTERDISTANCE(_axDisp) = fMSD3                  'Ramp down returning to start position
FLY(_axDisp)            = -(fMSD1+fMSD2) /2
```

Each move segment consists of a MASTERDISTANCE and a FLY command. The value of MASTERDISTANCE is the distance on the master axis (the conveyor) over which the slave (the icing dispenser) will travel for a segment. The value of FLY is the distance the slave axis (the icing dispenser) will travel whilst the master axis (the conveyor) is travelling the distance set by MASTERDISTANCE.

By using the FLY command, we can achieve synchronized movement of two axes, in our case the conveyor and icing dispenser axis. Once the axes are synchronized the icing is dispensed onto the cakes. The icing dispenser axis then slows to a stop before returning to the start position to await the next batch of cakes.

The graph below is taken from workbench, it shows the speed of the master and slave axes as the moves in the buffer are profiled. Also, on the graph is the state of the icing output displaying when it turns on as the axes are synchronized.



The MOVEPULSEOUTX keyword enables us to turn on an output for a set amount of time once the previous move segment has completed. The movement of the axis will not be affected by turning the output on and the next flying shear will be applied without stopping the axis. In this program we turn on output 0 for 500ms.

The negative area displayed on the graph indicates that the axis has changed direction and is returning to the start position to await the next batch of cakes. The area enclosed by the graph and the x axis should be the same above and below the x axis, indicating the distance travelled forward is the same as the distance travelled backwards.

The complete cycle takes place over the sum of all of the MASTERDISTANCE's defined in the code. This equates to 0.95 * fMinDist thereby ensuring that the cycle always completes before the next set of cakes arrives at the cycle start position.

Latch event

When the latch event is called the following line of code is executed.

TRIGGERVALUE(_axDisp) = ((LATCHVALUE(3) + _DwellDistance) % ENCODERWRAP(_EncChannel))

The latch event is called when the next batch of cakes passes the registration sensor. At this point we take the latched value from the master encoder (attached to the conveyor) and add a known offset distance (_DwellDistance) to define when the icing dispenser axis must begin its move. We have used a constant value to define this offset distance but very often this would be adjustable (e.g. via an HMI) to allow the operator to adjust the timing of the system and hence the position of the icing. This combined value is assigned to TRIGGERVALUE which will trigger the moves loaded in the move buffer when the master encoder is at the set value. To ensure the value assigned to TRIGGERVALUE is not larger than the max value returned by the master encoder (maximum value is set by ENCODERWRAP) we take the modulus (indicated by the % symbol) of the value with ENCODERWRAP. This will return the remainder of the value after ENCODERWRAP is divided into the value.

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