Overview

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.

Mint also provides an extensive range of specialized functions to interface to the hardware of Baldor 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 an overwrapping machine such as that illustrated by the animation at www.baldormotion.com/solutions.

Overwrapper Animation

The Overwrapper animation has the following features:

- A NextMove e100 Motion Controller
- Servo controlled film feed, horizontal wrap and vertical wrap axes (Brushless AC Servo Motors)
- Three MotiFlex e100 AC servo drives connected over EPL (Ethernet PowerLink)
- Three VS1ST Microdrive’s to control the conveyors’ speeds
- Auxiliary encoder connected to an encoder input on the NextMove e100
- Registration sensor connected to a fast latch input on the NextMove e100
- Input sensor to detect when to turn heaters on connected to an input on the NextMove e100

The sample code shows how the overwrapping process can be synchronized to a batch of products (bottles) on a conveyor. We use the fast latch functionality to capture the position of the bottles by reading a value from an encoder attached to the conveyor. This fast latch is triggered when the bottles pass a registration sensor. From this measured position we can set a position trigger value at which point our axes will start moving to wrap the batch of bottles. The move types used to achieve this are known as 'flying shears'. Once the bottles are wrapped they will move past another sensor which will turn the heater on and shrink the wrap around the bottles.

Axis Configuration

The example application uses the following axes...
(a) Product Conveyors (used to feed the products through the machine). These axes are controlled by VS series inverters. The central conveyor has an auxiliary encoder connected to the NextMove e100 to allow the Nextmove to determine the position of the conveyor.  
(b) Horizontal wrap (a MotiFlex e100 AC servo drive controls the axis that moves parallel to the products movement)  
(c) Vertical wrap (a MotiFlex e100 AC servo drive controls the axis that moves the plastic wrap over the products to wrap the batch of bottles)  
(d) Film feed (a MotiFlex e100 Ac servo drive controls the feed of the plastic wrap, ensuring it is fed at a sufficient speed to match the wrapping process)  
(e) Virtual axis (a virtual axis is used to enable us to cut the film at the correct position)

The MotiFlex e100 and virtual axes are 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 at a later date some constants are used to define certain aspects of the application.

The wrap axes, film feed axis, conveyor encoder channel, input and output numbers, conveyor length and conveyor height 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.

```vbnet
Const _axHorizWrap As Integer = 0 'Horizontal wrap axis  
Const _axVertWrap As Integer = 1 'Vertical wrap axis  
Const _axFeed As Integer = 2 'Film feed axis  
Const _axVirtual As Integer = 3 'Virtual axis used for cutting film  
Const _axEncChannel As Integer = 0 'Encoder input channel  
Const _axProdReg As Integer = 0 'Product registration input  
Const _axOpHeater As Integer = 0 'Heater output  
Const _axOpCutter As Integer = 1 'Trigger to Cut wrap  
Const _fConvLength As Float = 1000 'Length of wrap conveyor  
Const _fConvHeight As Float = 200 'Height of wrap conveyor
```

**Program Variables**

Variables are declared for any value that may change during program execution. Although most of the variables declared in the program are actually constant values, in an actual application they are likely to be values entered on an HMI and hence are declared as variables here. Variables are declared for values such as product sizes, master distances and fly calculations.
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 Horizontal wrap axis mechanics are such that there are 200 encoder edges for a movement of 1 millimetre (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 relationships between the master (Conveyor) and slave (horizontal wrap, vertical wrap and film feed) axes. This is done in the startup block of the mint program. The following example shows how to set up the relationship for the horizontal wrap axis.

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

This code sets the master which the slave axis will synchronize with. In this case the horizontal wrap axis will be synchronized with the encoder on channel 0. This is the encoder measuring the position of the conveyor which is connected to the NextMove e100.

Trigger Configuration

By using triggering we can cause moves to be performed without issuing a GO command. The following code sets up triggering on the horizontal wrap axis and is included in the startup block of the mint program.

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

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

We have also set ENCODERWRAP(_EncChannel) = 5000 in the Mint startup block. 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 4999 (the ENCODERWRAP
can be any value larger than the distance along the conveyor from the sensor to the point at which the last axis is triggered). Setting a TRIGGERVALUE of -9999 (a value the encoder can never reach) effectively temporarily disables triggering of motion from the encoder.

To enable us to cut the film wrap at the correct position we are using a virtual axis that triggers its moves based on the position of the film feed axis. The following code sets up the triggering on the virtual axis and is included in the startup block of the mint program.

```
TRIGGERMODE(_axVirtual)    = _trFWD_MOTION
TRIGGERSOURCE(_axVirtual)      = _tsPOS
TRIGGERCHANNEL(_axVirtual)    = _axFeed
TRIGGERVALUE(_axVirtual)       = fCutPos
```

This code sets axis 3 to trigger motion when the position of axis 2 (_axFeed) passes 1000 user units (fCutPos) in a forward (positive) direction.

More about how this cut process works is included later in the application note.

**Fast Latch Configuration**

To set up fast latching on e100 products a selection of LATCH...... keywords are used. When using a NextMove e100 up to 32 latch channels are available. We will use latch channel 0 in this application. The fast latch inputs available on a NextMove e100 are digital inputs 0-3, we will use digital input 0.

Using the code below in the startup block of the Mint program we can configure latch channel 0 to capture the position of the conveyor axis on a rising edge of digital input 0.

```
LATCHTRIGGERCHANNEL(0)         = _ipProdReg
LATCHTRIGGERMODE(0)            = _ltmINPUT
LATCHTRIGGEREDGE(0)            = _ltePOSITIVE_EDGE
LATCHSOURCE(0)                 = _lsENCODER
LATCHSOURCECHANNEL(0)          = _EncChannel
LATCHMODE(0)                   = _lmAUTO_ENABLE
LATCHENABLE(0)                 = _true
```

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

You can view and edit the settings of the latch channels in the parameter viewer in Mint Workbench. 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 buffers and then the program waits for the moves to be triggered. This is achieved with the following code.

' Main program loop
TRIGGERVALUE(_axHorizWrap,_axVertWrap,_axFeed)) = -9999;   'Disable triggering

' Load flys into movebuffers
doLoadFeed
doLoadVert
doLoadHoriz
MOVEPULSEOOUTX(_axVirtual,_opCutter) = tmCutTime

' Wait until moves have been triggered or axis is idle
Pause ((MOVESTATUS(_axHorizWrap) & 256) = 256) Or (IDLE(_axHorizWrap))

' Wait until Film feed axis has finished moving then set its POS to zero
Pause IDLE(_axFeed)
POS(_axFeed) = 0
We start by setting TRIGGERVALUE to -9999 on all but the virtual axis which disables triggering (see previous section on trigger configuration for the reason why). We then call the sub routines which load the flying shear moves into the move buffer for each axis. We will look at one of these subroutines further later on. A command to turn on an output to cut the wrap is also loaded using the MOVEPULSEOUTX keyword. The program then waits for the moves to be triggered on the horizontal wrap axis (this is the last axis to be triggered). When a batch of bottles pass the registration sensor an event will occur which sets the TRIGGERVALUE’s for all the axes except the virtual axis which has a constant TRIGGERVALUE. Once the position of the auxiliary encoder reaches these trigger values the axes will start the flying shears or moves loaded in the move buffers. This allows the program to move on to the second Pause statement which waits for the moves on the film feed axis to finish before resetting the position of the axis back to zero. This allows the TRIGGERVALUE for the virtual axis to be constant and always active. The whole process is then repeated for the next batch of bottles.

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 horizontal wrap 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.

Sub routines – Loading the Flying Shear Segments

The doLoadHoriz sub routine loads the flying shear moves for the Horizontal axis into the move buffer ready to be triggered when required.

- MASTERDISTANCE(_axHorizWrap) = fHorizMSD1
- FLY(_axHorizWrap) = fHorizFly1
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD2
- FLY(_axHorizWrap) = fHorizFly2
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD3
- FLY(_axHorizWrap) = fHorizFly3
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD4
- FLY(_axHorizWrap) = 0
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD1
- FLY(_axHorizWrap) = -fHorizFly1
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD2
- FLY(_axHorizWrap) = -fHorizFly2
- MASTERDISTANCE(_axHorizWrap) = fHorizMSD3
- FLY(_axHorizWrap) = -fHorizFly3

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 horizontal wrap) will travel for a segment. The value of FLY is the distance the slave axis (the
horizontal wrap) will travel whilst the master axis (the conveyor) is travelling the distance set by MASTERDISTANCE. There is more information on the FLY calculations in the next section.

There are three subroutines in total, one to load the moves for each of the three axes. The other two sub routines are called doLoadVert and doLoadFeed. For our virtual axis we only load one move into the move buffer this is a MOVEPULSEOUTX. This command doesn’t actually move an axis but allows us to load into the move buffer a command to turn on an output for a set duration once the previous move segment has completed or the moves in the move buffer are triggered. In this program we turn on output 1 for 50ms. In an actual application the cut process is likely to be achieved using a rotary knife but for simplicity we are using an output in this application note.

To ensure that all the moves for a cycle can be loaded into the move buffer we must set the size of the buffer using MOVEBUFFERSIZE. The sizes of the move buffers are declared in the startup block of the mint program. We set the MOVEBUFFERSIZE to the number of moves the axis performs in a cycle. So the horizontal and vertical wrap axes are 7 and the film feed axis is 5. The virtual axis only requires 1 move to be loaded but is set to 2 which is the minimum value of MOVEBUFFERSIZE.

Flying shear calculations

The value of FLY can be calculated using the following formula:

\[
FLY = \left( \frac{\text{Initial Ratio} + \text{Final Ratio}}{2} \right) \times \text{MASTERDISTANCE}
\]

The initial and final ratios refer to the speed ratio of the master to slave. This formula is coded in to the program (in the variable declarations) so that if any values are changed (e.g. the MASTERDISTANCE) the program will calculate the correct FLY value.

We will look at the Horizontal axis for an example of how this works. The graph below shows the speed time graph of the horizontal axis over one cycle. Highlighted are the areas of the graph that each FLY segment refers to.
Using the above formula for the Horizontal axis; the initial ratio is 0 as the axis is not moving, we do not know the final ratio but can calculate this, and we also know the master distance as this is declared in the program. We can work out the final ratio from the FLY values (using the previous formula) and the fact that we know that:

\[ FLY_1 + FLY_2 + FLY_3 = f_{HorizMove} \]

This tells us that the total value of FLY\(_1\), FLY\(_2\) and FLY\(_3\) will be equal to the value of \(f_{HorizMove}\). After doing some simultaneous equations we find a final ratio of:

\[
\frac{f_{HorizMove}}{(f_{HorizMSD1}/2) + f_{HorizMSD2} + (f_{HorizMSD3}/2)}
\]

Using this value we can calculate the Horizontal axis FLY values as:

\[
\begin{align*}
\text{Dim} \ f_{HorizFly1} \ \text{As} \ \text{Float} & = 0.5 \times (0 + f_{HorizFinalSpeed}) \times f_{HorizMSD1} \\
\text{Dim} \ f_{HorizFly2} \ \text{As} \ \text{Float} & = 0.5 \times (f_{HorizFinalSpeed} + f_{HorizFinalSpeed}) \times f_{HorizMSD2} \\
\text{Dim} \ f_{HorizFly3} \ \text{As} \ \text{Float} & = 0.5 \times (f_{HorizFinalSpeed} + 0) \times f_{HorizMSD3}
\end{align*}
\]

These three FLY values are used twice, once in moving the horizontal axis along the conveyor and then again with a negative value to return the axis to its start position. There is a fourth FLY value and MASTERDISTANCE which is a dwell where the horizontal axis does not move allowing the vertical axis time to move down at the end of the conveyor.
Graph of one cycle

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 are the states of the registration input and wrap cut output.

The negative areas displayed on the graph indicate that the axes have changed direction and are returning to their start position to await the next batch of bottles. The area enclosed by the graph and the x axis should be the same above and below the x axis for a particular axis. This indicates the distance travelled forward is the same as the distance travelled backwards. The feed axis does not return to its starting position as it just feeds out a length of film material.

The complete cycle (initial registration of product to last axis stopping) takes place over approximately 1.5m of conveyor travel, however the minimum distance between registrations of batches of bottles can be a little less than 1.1m. This gives us a pitch between batches of approximately 800mm. This is because the Horizontal axis (the last axis to finish moving in a cycle) does not begin moving immediately after registration so can still be returning to its start position as the batch of bottles passes the registration sensor.
**Input event**

The event IN1 will occur when input 1 on the NextMove e100 becomes high. The default setting for the input is level triggered, active high. To avoid calling the event all the time the input is on we need to configure the input to be triggered only on a rising edge of the input. We achieve this using the INPUTMODE and INPUTPOSTRIGGER keywords. This code is placed in the Mint Startup block:

```
INPUTMODE(0) = 000000000000000000010    'Input 1 is edge triggered
INPUTACTIVELEVEL(0) = 011111111111111111111 'Inputs are active high
INPUTPOSTRIGGER(0) = 000000000000000000010    'Input 1 triggered on the rising edge
INPUTNEGTRIGGER(0) = 0    'Input does not occur on the falling edge
```

The heater sensor is connected to input 1 on the NextMove e100, when the batch of bottles passes this sensor this event will occur. Within the event the following code is executed.

```
PULSEOUTX(_opHeater) = tmHeatTime
```

The PULSEOUTX command is similar to the MOVEPULSEOUTX command used previously. The difference is that the PULSEOUTX command will execute immediately and is not loaded into the move buffer. The variable tmHeatTime is currently just a value that is assigned by the user. In an application this time period would be calculated based on the speed the conveyor.

**Latch event**

When the latch event is called the following lines of code are executed.

```
nConvPos = LATCHVALUE(0)
TRIGGERVALUE(_axHorizWrap) = ((nConvPos + fHorizDwellDist) % ENCODERWRAP(_EncChannel))
TRIGGERVALUE(_axVertWrap) = ((nConvPos + fVertDwellDist) % ENCODERWRAP(_EncChannel))
TRIGGERVALUE(_axFeed) = ((nConvPos + fFeedDwellDist) % ENCODERWRAP(_EncChannel))
```

The latch event is called when the next batch of bottles 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 to define when the axis must begin its move. We have used variable values to define these offset distances as very often they would be adjustable (e.g. via an HMI) to allow the operator to adjust the timing of the system. 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.