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 a Vertical Form, Fill and Seal machine such as that illustrated by the animation at www.baldormotion.com/solutions.

Related Applications or Terminology

- **Cut to registration**
- **Flying Shears**

Relevant products:

The application described here is implemented using two Motiflex e100 drives to control the Belt axis and Sealer Carriage. An example Mint program accompanies this application note and is described here. The program is designed to run using virtual axes to allow the concepts described in this application note to be tested easily. The program can be modified to run on a Mint Option Card installed on a MotiFlex e100 and networked to other drives. The set-up procedure for an e100 system is beyond the scope of this application note. It is described in “AN00187 - Getting started with e100”.

The Mint Option Card also provides additional IO which is used in this application. If further IO is required this can be expanded on with an additional Digital or Analog IO card. The IO in the 2nd Motiflex e100 used on the Sealer Carriage can be mapped to the Mint Option Card. This is done when setting up the .dcf file in the system configuration wizard.
Application description:

This application note is based on an animated Form Fill and Seal machine on the Baldor website. The sealer in the animation uses a rotating cam which gives a smoother operation from a mechanical point of view. The movement profile of this sealer is quite complicated. If it were to be described here, much of the application note would be spent explaining the machine geometry rather than the Mint language. The Mint language features that would be used are explained in application note: AN00122-002 “Using flying shears for embossing, rotary knife, sheet feeder and labeling applications”. In this application note a reciprocating action is explained which has similar language features.

The Form, Fill and Seal machine described in this application note is a continuous process, where the material continues to run while sealing is in progress. It consists of 2 main axes and an encoder mounted to make contact with the material to compensate for slippage.

Axis #1 Vacuum Belt Drives
In a real machine design two Servo motors may be used to drive individual belts together to pull the material off the reel. In this case both axes would follow a virtual axis to keep them in synchronization. In this application note only one axis is shown for the belt. It is
assumed mechanical couplings are used to drive both belts simultaneously. The bag material is pulled off a reel through a former. The former shapes the bag.

The material may slip relative to the belt so an additional encoder is used to monitor the position of the material. The belt motor encoder provides the position of the belt. The belt motor encoder is used as the feedback reference for the speed loop in the Motiflex e100 and the material encoder is used as the position loop feedback source.

**Axis #2 Sealer Carriage**
The Seal/Crimp/Cutting Jaws are moved in synchronization with the bag in a Flying Shear type motion.

**Additional I/O**
A registration mark sensor is wired into one of the fast latching inputs on the Motiflex e100 for the belt. This sensor detects a mark printed on the material which allows a snapshot of the materials position to be taken. This can be used to align the material pattern with the position it should be in when the sealer jaws close on it.

The Sealer is moved into contact with the material when a digital output activates. The Sealer heating element is turned on when a digital output activates. The clinching jaws are closed when a digital output activates. The knife cuts the sealed bag off the material roll when a digital output activates.

Additional axes can be added for other machine features e.g. a servo motor to drive an auger. These are not described in this application note.

**Constants and definitions:**
The Code snippets in this application note refer to axes and variables with labels rather than hard coded numbers. This is done to make the code more readable. These variables are defined at the head of the program as follows:

```plaintext
'Define constants for axes as labels
Const _axBelt = 0
Const _axCarriage = 1
Const _axVirtual = 2

'Define constants for the outputs to make them more readable as labels
Const _opSealJaws = 32
Const _opSealHeat = 33
Const _opCrimper = 34
Const _opKnife = 35
Const _opAlarm = 36
```
'Define constant for the latch channels as labels
Const _chRegMark = 1

'Define constant for the latch channels as labels
Const _MaxNumBags = 6

'Define variables for the sealer time delays, speeds and distances
Dim fRegToSealDist As Float 'Distance of seal position from registration sensor
Dim fSealOnPos As Float 'Distance after carriage is in sync with belt before turning on sealer
Dim fSealDist As Integer 'Distance the sealer is in Sync with the belt
Dim fSealToCrimpDist As Float 'The relative distance the crimper turns on
Dim fCrimpToKnifeDist As Float 'The relative distance that the knife turns on
Dim fKnifeOnDist As Float 'The relative distance that the knife is on
Dim fKnifeOffDist As Float 'The relative distance that the knife takes to turn off
Dim fFwdAccelDist As Float 'Distance when ramping to/from Synchronous speed
Dim fRtnAccelDist As Float 'Distance travelled when carriage ramps back to start
Dim fRegMarkPos As Float 'Distance of the registration mark from the top of the bag
Dim fTargetBagSealPos As Float 'Position to start moving the sealer carriage

'Define array and pointers for storing the position of registration marks
Dim fBagPositions(0 To 49) As Float = {0;}
Dim nBagWritePtr As Integer = 0
Dim nBagReadPtr As Integer = 0
Dim nBagCount As Integer = 0

Positioning of the material on the belts

In this application note only one axis is shown moving the belt. If a second belt drive is setup then both drives can be kept in synchronization if they follow a virtual axis.

The machine will use varying bag lengths. It is likely that the length of a bag will not be a whole number of encoder counts but may instead be an irrational number. There is also a possibility that the encoder wheel may slip slightly relative to the material. Both of these problems would combine to cause the material position to drift out of alignment in time, unless our program is written to compensate for this.

The material needs to be positioned so that the seal matches up with the pattern on the material. A registration sensor is used to achieve this. A snapshot of the position can be taken as the registration mark goes past. This can be used to adjust the distance the material is being moved by the belts. A latch channel is set up to latch the position of the registration input as follows:
LATCHSOURCE (_chRegMark) = _IsAXIS_POSITION  'The scaled belt position is being latched
LATCHSOURCECHANNEL (_chRegMark) = axBelt  'The latch channel is the Belt axis
LATCHTRIGGERMODE (_chRegMark) = _ItmINPUT  'Latch belt position on an input
LATCHTRIGGERCHANNEL (_chRegMark) = 1  'Latch on input 1
LATCHTRIGGEREDGE (_chRegMark) = _ItmNEGATIVE_EDGE  'Trigger on a negative edge of the Reg mark

LATCHMODE (_chRegMark) = _ImMANUAL_ENABLE  'The latch channel must be re-enabled manually
LATCHINHIBITVALUE (_chRegMark) = 5  'Introduces a 5mm de-bounce to filter out rogue signals
LATCHENABLE (_chRegMark) = _on

The material is moved with a speed of 1100 mm / min with a Jog command as shown:

JOG (_axBelt) = 1100 / 60

The registration mark will be detected triggering a Latch event. The latch event interrupts the program execution so it is kept as short as possible. Here it just starts a task to record the registration mark.

Event LATCH0
  If TaskStatus (RecordRegMark) = _tskTerminated Then Run RecordRegMark
End Event

In the task “RecordRegMark” record the position of the registration mark in an array:

fRegMarkPos = LATCHVALUE (_chRegMark)
fBagPositions (nBagWritePtr) = fRegMarkPos

Increment the array write position pointer by one.

nBagWritePtr = (nBagWritePtr + 1) % (UBound(fBagPositions) + 1)

Keep track of the number of bags being loaded on the machine between the registration mark and the sealer. This information will be used to detect when the first bag reaches the sealing position.

nBagCount = nBagCount + 1

Re-enable the latch channel to capture the next registration mark.
LATCHENABLE (_chRegMark) = _on

It is a good idea to have a monitoring routine to make sure that the registration mark is still being detected or that the bag sealer is still moving. This would help avoid the situation where the material runs out. Another possibility is that a mark is detected sooner than it should be which would indicate a kink has occurred in the material. This would result in a loss of alignment of the pattern with respect to the sealing position.

Later in the program, the Sealer Carriage will be triggered to move once the bag material passes a certain position. When this move has completed the bag count will be decremented. This is described in more detail in the section on the Sealer Carriage movement.

In this routine, if the current belt position has moved beyond the target seal trigger position then a bag has been missed. This situation can be detected by checking that the number of bags between the registration mark and sealer does not exceed a threshold. If the bag count drops below 0 then the registration sensor is not detecting bags. Raise an alarm output and stop the belt axis.

If (nBagCount > _MaxNumBags) Or nBagCount < 0 Then
    STOP (_axBelt)
    OUTX (_opAlarm) = _on
End If

**Moving the Sealer Carriage as a Flying Shear**

To seal as efficiently as possible the material can be run from the belt continuously. The sealer is on a moving axis which will accelerate up to the speed of the belt. When the sealer is synchronized with the belt it closes its jaws and seals, crimps and cuts the material. It then opens the jaws. It now decelerates and reverses back to the starting position. The reverse move takes the form of an acceleration and a deceleration. The move profile is illustrated below.
For each segment in the profile above, the area under the curve corresponds to the distance travelled by the sealer carriage axis. This area can be specified using the FLY keyword. The distance travelled by the material for each segment can be specified in Mint by using the MASTERDISTANCE keyword.

If we examine a particular segment we can see that the segment area can be derived from:

\[ \text{FLY} \text{ (i.e. Area)} = \text{MASTERDISTANCE} \times (\text{Initial Speed Ratio} + \text{Final Speed Ratio}) / 2 \]

This generic formula can be used to calculate FLY if only MASTERDISTANCE is known and vice versa.

To return the shear axis to the start position the sum of the two return areas must equal exactly the distance travelled in the forward direction.
Triggering the Sealer Carriage to move in sync with the belt

Firstly set the carriage axis up to use the belt as its master reference for the flying shear:

- `MASTERSOURCE (_axCarriage) = _msPOS` 'Follow master position'
- `MASTERCHANNEL (_axCarriage) = _axBelt` 'Axis belt is master'
- `MOVEBUFFERSIZE (_axCarriage) = 100` 'Make enough room to load Fly segments'

Wait until at least one registration mark has been detected before triggering the sealer carriage to move on a target belt position being reached.

`PAUSE( nBagCount > 0 )`

When calculating the position of the next bag’s seal position, the registration mark array is used to read the position of this bags registration mark when it was lined up with the sensor. Since that position was recorded the material has been advanced X bag lengths plus an offset. In the meantime the encoder may have slipped relative to the material so we use the bag length and the registration mark recorded position to calculate the target position where the sealer carriage should start accelerating.

\[
f_{\text{TargetBagSealPos}} = f_{\text{BagPositions}}(n_{\text{BagReadPtr}}) + (f_{\text{RegToSealDist}})
\]

If the belt axis scaling was in counts then the position counter would eventually roll over from 8388607 to -8388607. Therefore the target position needs to be adjusted if it is on the other-side of this transition. In our example we would have a different scale factor. For example: If the belt roller diameter was 50mm and there were 10,000 encoder counts per motor revolution, our scale factor would be:

\[
\text{SCALEFACTOR (}_ax\text{Belt)} = \frac{10000}{(50 \times 3.14)}
\]

We can use a constant to represent the max or min position before the position counter wraps would occur at:

\[
\text{Const } _\text{PosMax} = 8388607 \times 50 \times 3.14 / 10000
\]

There is a Wrap function in Mint which does this.

\[
f_{\text{TargetBagSealPos}} = \text{Wrap}( f_{\text{TargetBagSealPos}}, -_\text{PosMax}, \text{PosMax})
\]

Now trigger the carriage to ramp up to Synchronous speed with the Belt at the seal position

'Use an axis position as the trigger reference.

- `TRIGGERSOURCE (_axCarriage) = _tsPOS`
'Specify the belt to be the axis used for triggering
TRIGGERCHANNEL (_axCarriage) = _axBelt

'Specify the position to start motion
TRIGGERVALUE (_axCarriage) = fTargetBagSealPos

'Trigger motion when specific axis position is passed
TRIGGERMODE (_axCarriage) = _trFWD_MOTION

Now increment the array read pointer for the bags position

nBagReadPtr = (nBagReadPtr + 1) % (UBound(fBagPositions) + 1)

Now load the move buffer with all the flying shear segments required. It is best to call a subroutine to do this as the action will be repeated.

'Ramp up to the belt speed
MASTERDISTANCE (_axCarriage) = fFwdAccelDist
FLY (_axCarriage) = fFwdAccelDist / 2

' Run at Synchronous speed
MASTERDISTANCE (_axCarriage) = fSealDist
FLY (_axCarriage) = fSealDist

'When sealing is complete this routine moves the sealer carriage back to the beginning.
' Now decelerate to zero
MASTERDISTANCE (_axCarriage) = fRtnAccelDist
FLY (_axCarriage) = -(fFwdAccelDist + fSealDist) / 2

'Return segment acceleration
MASTERDISTANCE (_axCarriage) = fRtnAccelDist
FLY (_axCarriage) = -(fFwdAccelDist + fSealDist) / 2

'Return segment deceleration
MASTERDISTANCE (_axCarriage) = fRtnAccelDist
FLY (_axCarriage) = -(fFwdAccelDist + fSealDist) / 2

Now the fly segments are loaded. When the belt reaches to just before the first seal position the sealer carriage will move through the 5 flying shear segments shown above. Wait till the move buffer is empty, i.e. its status is 8 before loading flying shear segments for the next seal position.
The machine has so far been setup to load material and to keep track of the material position for each bag by way of the registration mark sensor. One sealer cycle will be executed when the material reaches the first seal trigger position. It is necessary now to repeatedly re-load the fly segments as each new bag comes into position. This is done by repeating some of the previously described actions in a continuous loop.

Loop

**Pause** ( MOVEBUFFERSTATUS (_axCarriage) = 8)

Now one bag has left the stack between the registration mark and the sealer so decrement the bag count by one.

\[ nBagCount = nBagCount - 1 \]

Wait until at least one registration mark has been detected before triggering the sealer carriage to move on a target belt position being reached.

**PAUSE** ( nBagCount > 0 )

Now update the Carriage and Sealer re-triggering positions for the next bag. This is done in a similar manner as last time.

\[ fTargetBagSealPos = fBagPositions( nBagReadPtr) + (fRegToSealDist) \]

Wrap the target position between _PosMax to -PosMax

\[ fTargetBagSealPos = \text{Wrap}( fTargetBagSealPos, -_PosMax, _PosMax) \]

The trigger position can be re-primed by changing the trigger value. The trigger mode has not changed and so it does not need to be setup again.

**TRIGGERVALUE** (_axCarriage) = fTargetBagSealPos

Now call the subroutine to re-load the fly segments as above.

**doLoadCarriageFlys()**

Now increment the array read position pointer by one.

\[ nBagReadPtr = (nBagReadPtr + 1) \mod (\text{UBound}(fBagPositions) + 1) \]

End Loop ‘Now repeat the loop.
Operation of the Sealer jaws, Crimper and Knife

So far the belts have been set up to dispense bags continuously and in line with the material registration mark. Also, the sealer carriage is now moving in sync with the belt over the sealing position through a portion of its cycle.

A Virtual axis is used to follow the carriage axis during sealing. This axis is used to turn on and off the Sealer outputs at target positions. It is set up to follow the Sealer Carriage with the following code:

Mastersource (axVirtual) = _msPos     'Follow master position
Masterchannel (axVirtual) = _axCarriage 'Axis carriage is the master
Followmode (axVirtual) = _fmNo_Ramp     'No ramping up to belt velocity
Movebuffersize (axVirtual) = 100        'Make enough room to load Fly segments

The sealer outputs will be turned on by moves loaded into the move buffer for the virtual axis. The move is triggered when the carriage reaches the seal position. The seal position will not change as the sealer carriage always returns to the start position.

Triggersource (axVirtual) = _tsPos     'Use an axis position as trigger reference.
Triggerchannel (axVirtual) = _axCarriage 'Specify the axis used for triggering
Triggervalue (axVirtual) = fSealPos     'Specify the position to start motion
Triggermode (axVirtual) = _trFwd_Motion 'Trigger motion when specific axis position is passed in a forward direction

Now load the move buffer with MoveOutX Commands by calling a sub routine that is called doSeal(). In this subroutine the move buffer is loaded with move commands that turn outputs on or off. The output moves are spaced by padding out the move buffer with flying shear segments spaced using MASTERDISTANCE values. These moves are performed on a virtual axis so that no plant has to move in the real world. All these moves will be triggered when the sealer carriage is synchronized with the belt.

'Move the move buffer is loaded with the move to turn on the sealer outputs
Moveoutx (axVirtual, _opSealJaws) = _on
Moveoutx (axVirtual, _opSealHeat) = _on

'Now set the distance that the sealer will move before the crimper turns on
MASTERDISTANCE (axVirtual) = fSealToCrimpDist
FLY (axVirtual) = 0

'Now turn on the crimper
Moveoutx (axVirtual, _opCrimper) = _on
'Now move to the knife position
MASTERDISTANCE(_axVirtual) = fCrimpToKnifeDist
FLY(_axVirtual) = 0

'Now turn on the Knife
MOVEOUTX(_axVirtual, _opKnife) = _on

'Now wait for the point where the knife is turned off
MASTERDISTANCE(_axVirtual) = fKnifeOnDist
FLY(_axVirtual) = 0

'Now turn off the Knife and other sealer outputs
MOVEOUTX(_axVirtual, _opKnife) = _off
MOVEOUTX(_axVirtual, _opCrimper) = _off
MOVEOUTX(_axVirtual, _opSealHeat) = _off
MOVEOUTX(_axVirtual, _opSealJaws) = _off

Note that FLY is always set to 0 for the virtual axis in the above moves. We want the master distance to be monitored to trigger the output moves but we don’t need to move the virtual axis position itself. In fact if we did move the virtual axis we would have to move it back to the start at the end. Otherwise the next time we loaded these move segments the outputs wouldn’t trigger as the virtual axis would be advanced beyond the position of the carriage axis.

Wait until at least one registration mark has been detected before triggering the sealer carriage to move on a target belt position being reached.

PAUSE(nBagCount > 0)

When the sealer carriage has completed a cycle the sealer outputs can be set up again to turn on by re-loading the move buffer. This is done by calling doSeal() again.