**Tech Note #33**

**Introduction**

MotionBASIC® provides several Boolean variables you can use to determine where an axis is in the execution of a motion. These variables include:

- IN.MOT@
- AT.REST@
- IN.POS@
- IN.ACL@
- IN.DCL@
- AT.SPD@
- DSP.DONE@

The names of these variables are deceptively simple and often mislead programmers into ignoring certain subtleties regarding their use. This technote explains these subtleties and provides some recommendations on the use of these variables.

**MOVE Variables**

The variables in the following list apply only to motions that are commanded by a MOVE statement, they provide no information about the status of motions that result from GEAR statements:

- IN.MOT@
- IN.ACL@
- AT.SPD@
- IN.DCL@
- AT.REST@

**IN.MOT@**

This variable allows you to tell if an axis is being *commanded* to MOVE it does not tell you whether it is *actually* moving. For example, if the axis is not being commanded to move, but due to external forces, the axis is actually in motion, IN.MOT@ will remain FALSE.

**AT.REST@**

This variable is always the inverse of IN.MOT@ and exactly the same considerations apply to its use.

**IN.ACL@**

This variable is TRUE while the DSP is process in the acceleration portion of a MOVE. As with most of the other axis status variables that change state at the start of a motion, you should allow sufficient time after execution of a MOVE statement for it to be properly set before testing it.
If you execute a MOVE with an AFTER clause, IN.ACL@ becomes true once the AFTER clause is satisfied. If your acceleration time is very short, your program may miss IN.ACL@ becoming true. For example:

```
MOVE axis FOR distance AFTER ASEN@
WHILE NOT DSP.DONE@(axis)
   IF IN.ACL@(axis) THEN DIO@(accel)=20
   .
   .
   .
WEND
```

If the WHILE/WEND loop takes longer to execute than the acceleration, your program may never see IN.ACL@ become TRUE. If detecting the start of the motion is essential for your program to work correctly you should use the following alternative:

```
ASEN@(axis)=FALSE
MOVE axis FOR distance AFTER ASEN@
WHILE NOT DSP.DONE@(axis)
   IF ASEN@(axis) THEN DIO@(accel)=20 :ASEN@(axis)=FALSE
   .
   .
   .
WEND
```

For this alternative to work, ASEN@ must be configured as a latched input using SENS.MODE@.

**AT.SPD@**

This variable is set true when the motion commanded by a MOVE reaches its commanded velocity. It is reset to FALSE once the deceleration begins.

If the commanded motion is a triangular profile, AT.SPD@ will not be set TRUE. If you are commanding MOVEs by setting SPD.MAX@ and not explicitly specifying a speed in the MOVE statement, you will not know ahead of time whether the motion will have a constant speed segment. Also, if the constant speed portion is very brief, AT.SPD@ may not be true long enough for your program to detect it. To avoid either of these situations, test for IN.ACL@=FALSE.

```
MOVE axis TO position
WAIT UNTIL IN.ACL@(axis)=FALSE
PRINT "Axis has done accelerating"
WAIT UNTIL AT.SPD@(axis)=FALSE
PRINT "Axis must have started to decelerate"
WAIT UNTIL AT.REST@(axis)
PRINT "MOVE is complete"
```
IN.DCL@

This variable is set TRUE as soon as the axis begins decelerating to the final position and is set FALSE again when the motion is completed. If the deceleration time is very brief, IN.DCL@ may not be true long enough for your program to detect it. To avoid this problem, you can test for IN.ACL@ and AT.SPD@ to both be FALSE.

```
MOVE axis FOR distance AT speed
WAIT UNTIL NOT (IN.ACL @(axis) OR AT.SPD @(axis))
```

If you are changing the speed of a MOVE AT using another MOVE AT, the speed change is always considered an acceleration regardless of whether the new speed is higher or lower than the original speed. For example:

```
MOVE axis AT high.speed IN 500
WAIT UNTIL AT.SPD @(axis)
MOVE axis AT lower speed IN 500
WHILE IN.MOT @(axis)
    PRINT IN.ACL @(axis);AT.SPD @(axis);IN.DCL @(axis)
WEND
```

This will start by printing "-1 0 0 " and after 500 milliseconds will switch to printing " 0 -1 0 ".

IN.POS@

This variable indicates the position error is less than the "In Position Error" value (PERR.INPOS@) set by the user in the Range Variables Screen in the Configuration Menus. The position error is the current actual position minus the current commanded position.

Since while an axis is being commanded to move, the current commanded position is constantly changing, IN.POS@ may change from true to false and back many times. *It cannot be used to determine whether a motion has been completed or not.* Its intended function is to provide a way to check whether the axis is within an acceptable distance from the current instantaneous commanded position while an axis is either moving or stopped.

DSP.DONE@

This variable indicates whether or not the DSP command queue has any pending instructions to process. If a commanded motion has a predefined ending, i.e. a MOVE TO, MOVE FOR or GEAR FOR, or if it includes an UNTIL clause, DSP.DONE@ will be set TRUE after the DSP processor has completed processing the last deceleration segment.

In the case of a commanded motion with no predefined end, i.e. a MOVE AT or GEAR AT without an UNTIL clause, DSP.DONE@ is set TRUE when the acceleration segment has been completed.
Since some motion commands will cause faults if issued while the DSP processor is busy, this variable is useful for checking whether or not it is safe to issue another command. An example of this is:

```
MOVE axis AT speed
WAIT UNTIL DIO@(2) = OFF
HALT axis
WAIT UNTIL DSP.DONE@(axis)
POS.ACT@(axis) = 0
```

DSP.DONE@ is a better choice than AT.REST@ or NOT IN.MOT@ since it really does indicate the axis processor is done with the motion and is ready to accept another instruction. DSP.DONE@ is set TRUE after the DSP has finished the last deceleration segment. *This does not necessarily mean the axis has stopped moving.* Often times there may be one or more DSP tick's worth of motion to complete before the axis actually stops, this is especially true if you are using SCURVE@ or CAM/PROFILE. Some statements (setting POS.ACT@ for example) may cause a fault if they are executed while an axis is moving. To avoid this you should wait a suitable time after DSP.DONE@ becomes true. This has the added advantage that it also gives the axis time to settle out any remaining motion.

```
MOVE axis FOR distance IN time
WAIT 300 AFTER DSP.DONE@(axis)
POS.ACT@(axis) = 0
```