

**HiCON – pn7751**

**Ethernet Motion Controller  
Data Acquisition System**

# **HiCON Basic User Guide**

**Document Revision 1.5**

(Updated February 13, 2013)

**© 2012 Vital Systems Inc  
Phoenix, AZ USA**

For more information please visit the product web page: [www.vitalsystem.com/hicon](http://www.vitalsystem.com/hicon)



**Extremely Important Reminder**

When operating machines, take extreme precautions. The machines can have enormous power even with a small motor. Never come inside a machine path while powered. Operating machines without necessary precautions can result in lost of limbs or even death.



**WARNING:** Machines in motion can be extremely dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the system. VITAL Systems shall not be liable or responsible for any incidental or consequential damages. By Using the HiCON motion controller, you agree to the license agreement.

## Contents

I. License Agreement .....	4
II. HiCON Basic Introduction .....	5
1. What is a HiCON Basic Program? .....	5
2. HiCON Basic Loader .....	5
3. Starting HiCON Basic from Mach3 .....	6
4. Mach3 DROs and LEDs.....	6
III. HiCON Basic Language Syntax .....	7
1. HiCON Basic Language Operators.....	7
Mathematical Operators .....	7
Logical Operators .....	7
2. HiCON Basic Language Commands.....	7
PRINT [NONEWLINE] .....	7
IF/ THEN/ ELSE.....	8
GOTO .....	9
FOR/TO/DOWNTO /NEXT/ STEP.....	9
GOSUB/ RETURN .....	9
WHILE/ WEND .....	10
REM .....	10
LET .....	10
DIM.....	10
RANDOMIZE .....	11
END.....	11
IV. HiCON Basic Functions .....	12
1. Math Functions.....	12
float SQR(float number) .....	12
float LOG(float number).....	12
float EXP(float number).....	12
float POW(float base, float power) .....	12
float ABS(float number) .....	12
float SGN(float number).....	13
PI.....	13
float SIN(float radians) .....	13
float COS(float radians) .....	13
float TAN(float radians).....	13
float ATN(float ratio) .....	14
float ASIN(float radians) .....	14

<i>float</i> ACOS( <i>float</i> radians).....	14
<i>float</i> SINH( <i>float</i> radians).....	14
<i>float</i> COSH( <i>float</i> radians).....	14
<i>float</i> TANH( <i>float</i> radians) .....	14
<i>float</i> LOG10( <i>float</i> number).....	14
<i>int</i> FLOOR( <i>float</i> number).....	15
<i>int</i> CIEL( <i>float</i> number) .....	15
<i>int</i> FMOD( <i>float</i> x, <i>float</i> y).....	15
<i>float</i> FLOAT( <i>int</i> value).....	15
<i>int</i> INT( <i>float</i> value) .....	15
2. HiCON Specific Functions.....	15
<i>int</i> GetLED( <i>int</i> LedID).....	15
SetLED( <i>int</i> LedID, <i>int</i> Val).....	15
<i>float</i> GetDRO( <i>int</i> DRO_ID).....	16
SetDRO( <i>int</i> DRO_ID, <i>float</i> Val).....	16
<i>int</i> GetPin ( <i>int</i> Port, <i>int</i> Pin ) .....	16
<i>int</i> SetPin ( <i>int</i> Port, <i>int</i> Pin, <i>int</i> Value ).....	16
<i>int</i> GetHiCON( <i>int</i> x).....	16
SetHiCON( <i>int</i> x, <i>int</i> value).....	17
3. Timer Functions .....	17
Sleep( <i>int</i> milliseconds) .....	17
4. Motion Functions .....	17
DoMotionAxis( <i>int</i> axisID, <i>float</i> finalPosition, <i>float</i> acceleration, <i>float</i> velocity, <i>int</i> mode ) .....	18
DoMotionXYZ( <i>float</i> positionX, <i>float</i> positionY, <i>float</i> positionZ, <i>float</i> acceleration, <i>float</i> velocity, <i>int</i> mode) .....	18
<i>int</i> DoHoming( <i>int</i> axis, <i>float</i> vel, <i>float</i> accel, <i>float</i> HomePosValue, <i>int</i> DirReverse, <i>int</i> UseIndexPulse, <i>int</i> UseHomeSensor ) .....	18
CancelMove( <i>int</i> AxisID) .....	19
V. Standalone Operation with HiCON Basic.....	20
VI. Programming Hints.....	21

## I. License Agreement

Before using the HiCON and accompanying software, please take a moment to go thru this License agreement. Any use of this hardware and software indicate your acceptance to this agreement.

It is the nature of all machine tools that they are dangerous devices. In order to be permitted to use HiCON on any machine you must agree to the following license:

I agree that no-one other than the owner of this machine, will, under any circumstances be responsible, for the operation, safety, and use of this machine. I agree there is no situation under which I would consider Vital Systems, or any of its distributors to be responsible for any losses, damages, or other misfortunes suffered through the use of the HiCON board and its software. I understand that the HiCON board is very complex, and though the engineers make every effort to achieve a bug free environment, that I will hold no-one other than myself responsible for mistakes, errors, material loss, personal damages, secondary damages, faults or errors of any kind, caused by any circumstance, any bugs, or any undesired response by the board and its software while running my machine or device.

I fully accept all responsibility for the operation of this machine while under the control of HiCON, and for its operation by others who may use the machine. It is my responsibility to warn any others who may operate any device under the control of HiCON board of the limitations so imposed.

I fully accept the above statements, and I will comply at all times with standard operating procedures and safety requirements pertinent to my area or country, and will endeavor to ensure the safety of all operators, as well as anyone near or in the area of my machine.

## II. HiCON Basic Introduction

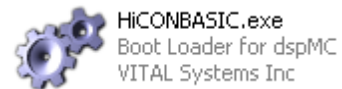
### 1. What is a HiCON Basic Program?

A HiCON Basic Program is a program coded in BASIC language (.bas file) which can be downloaded and executed directly inside the HiCON controller. The program allows the user to manipulate high speed I/O and launch motion commands, as well as, communicate with PC programs, such as, Mach3 over Ethernet.

A HiCON Basic Program is executed directly on the HiCON controller, which effectively remove the latency factor associated with data transmission from a computer to the HiCON via Ethernet. One of the main purposes of using HiCON Basic is to create a fixed sequence of operation for the HiCON to execute as opposed to having the operator command it himself. In this sense, using HiCON Basic makes the HiCON controller more “intelligent” since it allows it to act with less operator supervision.

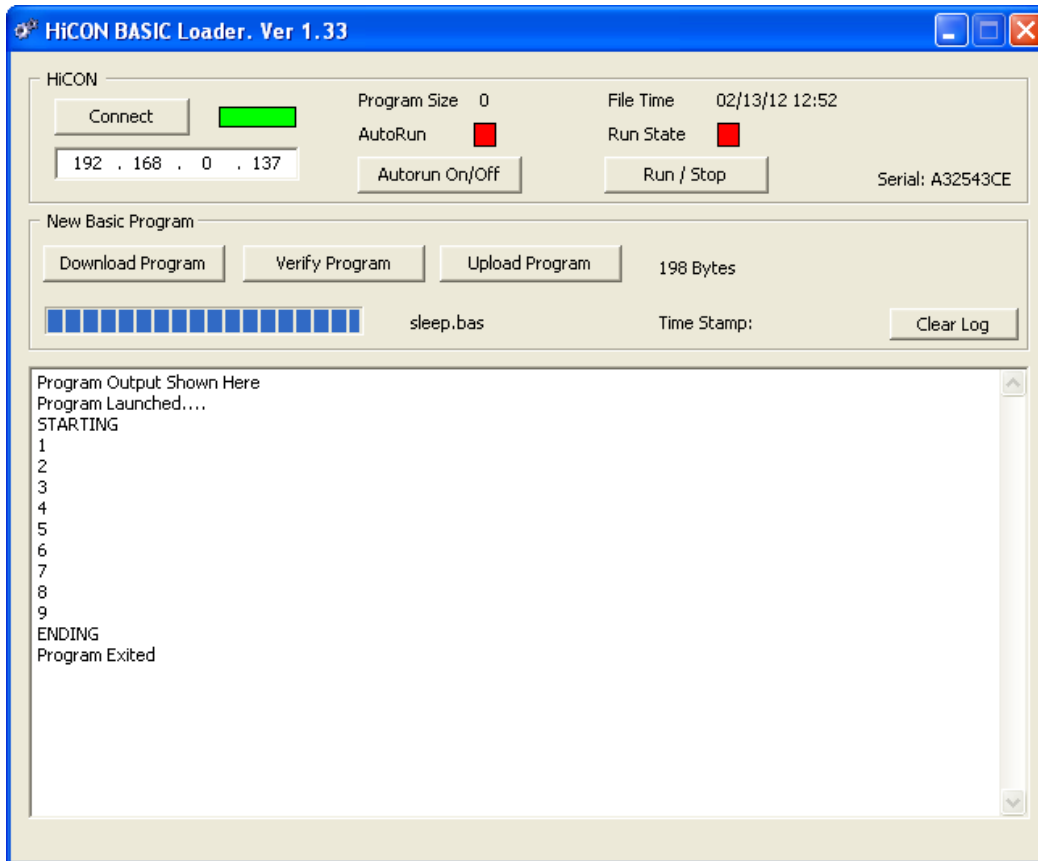
### 2. HiCON Basic Loader

HiCON Basic Loader is an application that is used to install and debug the HiCON Basic Program on the HiCON controller. The user can select the HiCON Basic file (.bas file) and download it to the controller. After launching the HiCON Basic program, the user can see the print statement outputs on the output window.



To obtain a copy of HiCON Basic loader, please visit [www.vitalsystem.com/hicon](http://www.vitalsystem.com/hicon).

Interface:



To run a HiCON Basic program on the HiCON:

1. Run the HiCON Basic Loader application.
2. Click the “Connect” button to connect to your HiCON. The IP address in the text box below the button should be the IP address of your HiCON.
3. If a connection was successfully established, click the “Download Program” button to download a HiCON Basic program to the device and select which program (.bas file) you would like to run on the HiCON.
4. If the download was successful, click the “Run/ Stop” button to run the application.
5. Use the output window to get feedback and check for errors when running the program.

**NOTE:** HiCON Basic programs may also be allowed to execute automatically on HiCON power-up using the “Autorun On/ Off” button, however this is only recommended when the HiCON Basic code is fully debugged and error-free.

### 3. Starting HiCON Basic from Mach3

Turning on LED 2035 from Mach3, will also launch the HiCON Basic program. The plugin will clear this LED after launching the program.

### 4. Mach3 DROs and LEDs

There are certain ranges of Mach3 DROs (Floating Point values) and LEDs (Binary 0 or 1) that are shared between Mach3 and HiCON Basic.

<b>Type</b>	<b>Range</b>	<b>Description</b>	<b>HiCONBasic Function</b>
LED	2000...2031	LEDs written by Mach3. Read-only inside HiCON Basic	GetLED
LED	2050...2081	LEDs written by HiCON Basic. Read-only inside Mach3	GetLED, SetLED
DRO	2000...2009	DROs written by Mach3. Read-only inside HiCON Basic	GetDRO
DRO	2050...2059	DROs written by HiCON Basic. Read-only inside Mach3	GetDRO, SetDRO

In addition, LED 2035 is used to launch the HiCON Basic program from within Mach3.

## III. HiCON Basic Language Syntax

### 1. HiCON Basic Language Operators

#### **Mathematical Operators**

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
%	Modulus Operator

#### **Logical Operators**

&&	Logical AND	
	Logical OR	
=	Logical equal	eg if ( input = 1 && output = 5 ) ...
<>	not equal	eg if ( input <> 1 ) ...
<=	Less-than or equal	
>=	Greater-than or equal	
<	Less-than	
>	Greater-than	
NOT()	Logical Not	eg if not( input ==1    input == 2 ) ...
	Format: <i>NOT(expression)</i>	

### 2. HiCON Basic Language Commands

#### **PRINT [NONEWLINE]**

The command PRINT is used to display information on the output window. The general syntax for PRINT is:

*PRINT list of constants and/or expressions*

Parameter "list of constants and/or expressions" contains the data that you wish to output to the screen. Numeric constants, numeric expressions, string constants and string expressions may all be contained within this list. If more than one value is to be output to the screen using a single PRINT statement, these values must be separated by some type of delimiter.

When separating output data contained in a PRINT statement, you may use either a comma or a semicolon. A semicolon will cause the next piece of information to be written directly after the preceding data. A comma will cause the next piece of information to be written at the next available "tab" position in the screen. Tab positions will be denoted by 8 byte blocks, starting from the last occurrence of a carriage return.

PRINT "Output a is: "; a; ", Counter X = "; X > Output a is 1, Counter X = 5  
 PRINT "Output a is: "; a, "Counter X = "; X > Output a is 1      Counter X = 5

The keyword NONEWLINE can be used at the end of a PRINT statement, to avoid the interpreter to put the carriage return at the end of the printed line.

PRINT "Output a is: "; a; NONEWLINE



```
PRINT ", Output b is: "; b  
➤ Output a is 1, Output b is 2
```

### **IF/ THEN/ ELSE**

These commands are used to define conditionals in the program code. It will execute a command depending on specified conditions. There are two different possibilities to the IF/THEN/ELSE statements, for example:

```
IF a > 10 THEN GOTO 2000
```

In the example above, the program to were the line belongs, will change its flow to the label 2000 if the value stored in the variable 'a' is bigger than 10.

```
IF a > 10 THEN GOTO 2000 ELSE GOTO 3000
```

In the line above, the program flow will change to the label 2000 if the variable 'a' is bigger than 10 or to the label 3000 if this condition is not true.

You can also evaluate a combination of logical statements. For example:

```
IF (10 > 5 && 6 == 6) THEN GOTO 4000
```

In the above statement, 10 > 5 -> TRUE, (&& logical AND), 6 == 6 -> TRUE. So the equation will evaluate to TRUE (since TRUE && TRUE -> TRUE) and the code will move to label 4000.

```
if not(a == 5 || b == 6) then goto 4000
```

In the above statement, program will jump to line 4000 if 'a' is not equal to 5 and 'b' is not equal to 6.

The IF/THEN conditional statement only allows statements on the same line as THEN to be evaluated for execution by the conditional statement. For example, if you want variables 'a' and 'b' to be made 0 only if they are both non-zero values, the following code is WRONG.

```
IF (a != 0 && b !=0 6) THEN a = 0  
b = 0
```

In the above code, regardless if 'a' and 'b' are both non-zero values, 'b' will be assigned a value of zero.

```
IF (a != 0 && b !=0 6) THEN a = 0 b = 0
```

The above code is the correct way. Variables 'a' and 'b' will be made zero only if both a and b are non-zero values.

To allow multiple lines to be evaluated by the conditional statement, you may also use GOSUB/RETURN or GOTO. For example:

```
IF (a != 0 && b !=0 6) THEN GOSUB 4000  
...  
...  
4000 a = 0  
    b = 0  
    PRINT a  
    PRINT b  
RETURN
```

...

In the above code the lines `a = 0`, `b = 0`, `PRINT a`, and `PRINT b` are only executed if the `IF/THEN` statement evaluates to true.

### **GOTO**

The `GOTO` statement branches to a specified label. For example:

```
GOTO 1000
```

The code above will redirect the program flow to the label 1000.

### **FOR/TO/DOWNTO /NEXT/ STEP**

These commands are used to repeat a block of statements for a certain number of times. For example:

```
FOR I = 1 TO 10  
  PRINT "I="; I  
NEXT
```

The code will repeat the `PRINT` statement 10 times. The variable 'I' is the loop variable which holds the repetition counter.

```
FOR I = 10 DOWNTO 1  
  PRINT "I="; I  
NEXT
```

Same as above, except that the counter will decrement.

```
FOR I = 1 TO 9 STEP 2  
  PRINT "I="; I  
NEXT
```

In this example, the loop variable will be incremented according to the expression used after the `STEP` statement.

**NOTE: When using FOR loop and STEP, only integer variables are allowed.**

### **GOSUB/ RETURN**

This command branches to and return from a subroutine.

```
GOSUB 1000  
PRINT "Back from subroutine"  
...  
...  
1000 PRINT "Starting subroutine"  
  calcvar = a+b*(c/10)  
  ...  
  ...  
RETURN
```

In the example above, the command flow is redirected to the label 1000 through a GOSUB call. All the lines inside the subroutine will be executed and when a RETURN command is found, the program flow is redirected to the next sentence just after the GOSUB call.

### **WHILE/ WEND**

Execute a series of statements as long as a specified 'condition' is true.

```
WHILE condition
...
...
...
WEND
```

The *condition* could be any logical expression, for example:

```
a = 10
WHILE a > 0
  PRINT "a="; a
  a = a-1
WEND
```

All statements between the block WHILE/WEND will be executed until the 'a' variable is greater than 0.

### **REM**

The statement REM allows explanatory remarks (comments) to be inserted in a program.

```
REM any text
PRINT "The line above is a remark"
PRINT "next command is also a remark" :REM comments
```

The line above is just ignored by the interpreter and does not occupy memory space when the program is executed.

#### **IMPORTANT NOTE:**

Do not label the lines where you define your comments or the interpreter will allocate memory space also for the REM statements.

### **LET**

The LET statement assigns the value of an expression to a variable.

```
LET variable = expression
```

The use of the optional LET keyword is optional. The *variable* = *expression* assignment statement performs the same action with or without LET.

### **DIM**

DIM declares a variable that holds an array of data. Different from many BASIC implementations, HiCON Basic arrays are not typed, being able to handle both numeric and string data.

```
DIM arr(5)
arr(1) = 10
arr(2) = 20
arr(3) = 30.85
REM The line below will produce '10 -20 - 30.85'
PRINT arr(1);" - ";arr(2);" - ";arr(3)
```

### **RANDOMIZE**

RANDOMIZE initializes the random-number generator. Internally, the interpreter engine has a seed that is used to generate random numbers, see the function "*rnd()*" to know more about random generated numbers.

```
REM The line below starts the random number seed based in the
REM clock time
RANDOMIZE TIMER
```

### **END**

The END statement ends a program immediately.

## **IV. HiCON Basic Functions**

### **1. Math Functions**

#### ***float* SQR(*float* number)**

Returns the square root of a numeric expression.

*SQR(numeric-expression)*

*numeric-expression*: A value greater than or equal to zero.

Example:

```
PRINT SQR(25), SQR(2) :REM Output is: 5 1.41421
```

#### ***float* LOG(*float* number)**

LOG returns the natural logarithm of a numeric expression.

*LOG(numeric-expression)*

*numeric-expression*: Any positive numeric expression.

Example:

```
PRINT EXP(0), EXP(1) :rem Output is: 1 2.71828  
PRINT LOG(1), LOG(EXP(1)) :rem Output is: 0 1
```

#### ***float* EXP(*float* number)**

EXP returns e raised to a specified power, where e is the base of natural logarithms.

*EXP(numeric-expression)*

*numeric-expression*: A number less than or equal to 88.02969.

Example:

```
PRINT EXP(0), EXP(1) :rem Output is: 1 2.71828  
PRINT LOG(1), LOG(EXP(1)) :rem Output is: 0 1
```

#### ***float* POW(*float* base, *float* power)**

Returns x raised to the y power.

*POW(x, y)*

x ,y: Any numeric expression.

NOTE: See also the ^ operator.

#### ***float* ABS(*float* number)**

ABS returns the absolute value of a number.

*ABS(numeric-expression)*

*numeric-expression*: Any numeric expression.

Example:

```
PRINT ABS(45.5 - 100!) : REM Output is: 54.5  
PRINT SGN(1), SGN(-1), SGN(0) : REM Output is: 1 -1 0
```

### **float SGN(float number)**

SGN returns a value indicating the sign of a numeric expression (1 if the expression is positive, 0 if it is zero, or -1 if it is negative).

*SGN(numeric-expression)*

*numeric-expression*: Any numeric expression. Example:

```
PRINT ABS(45.5 - 100!) : REM Output is: 54.5  
PRINT SGN(1), SGN(-1), SGN(0) : REM Output is: 1 -1 0
```

### **PI**

Returns the value of pi with five decimal digits.

```
PRINT PI
```

Prints 3.14159

### **float SIN(float radians)**

Returns the sine of a specified angle passed as an expression.

*SIN(angle)*

*angle*: An angle expressed in **radians**.

### **float COS(float radians)**

Returns the cosine of a specified angle passed as an expression.

*COS(angle)*

*angle*: An angle expressed in radians.

### **float TAN(float radians)**

Returns the tangent of a specified angle passed as an expression.

*TAN(angle)*

*angle*: An angle expressed in radians.

***float* ATN(*float* ratio)**

ATN returns the arctangent of a specified numeric expression.

*ATN(numeric-expression)*

*numeric-expression*: The ratio between the sides of a right triangle.

***float* ASIN(*float* radians)**

Returns the arc sine, in radians, of a specified angle passed as an expression.

*ASIN(angle)*

*angle*: An angle expressed in radians.

***float* ACOS(*float* radians)**

Returns the arc cosine, in radians, of a specified angle passed as an expression.

*ACOS(angle)*

*angle*: An angle expressed in radians.

***float* SINH(*float* radians)**

Returns the hyperbolic sine of a specified angle passed as an expression.

*SINH(angle)*

*angle*: An angle expressed in radians.

***float* COSH(*float* radians)**

Returns the hyperbolic cosine of a specified angle passed as an expression.

*COSH(angle)*

*angle*: An angle expressed in radians.

***float* TANH(*float* radians)**

Returns the hyperbolic tangent of a specified angle passed as an expression.

*TANH(angle)*

*angle*: An angle expressed in radians.

***float* LOG10(*float* number)**

Returns the logarithm base 10 of an expression.

*LOG10(expression)*

*expression*: Any numeric expression

***int* FLOOR(*float* number)**

Returns the largest integer value less than or equal to a numeric expression.

*FLOOR(expression)*

*expression*: Any numeric expression.

***int* CEIL(*float* number)**

Returns the smallest integer value greater than or equal to a numeric expression.

*CEIL(expression)*

*expression*: Any numeric expression

***int* FMOD(*float* x, *float* y)**

Returns the remainder of x/y.

*FMOD(x, y)*

NOTE: See also the operator %

***float* FLOAT(*int* value)**

Converts an integer to a floating point value.

*FLOAT(integer)*

***int* INT(*float* value)**

Converts a floating point value to an integer.

*INT(floatValue)*

## 2. HiCON Specific Functions

***int* GetLED(*int* LedID)**

Read the selected LED values from Mach3 or HiCON, where LED ID ranges from 2000 – 2031 (written by Mach3) and 2050 – 2081 (written by HiCON using SetLED).

**SetLED(*int* LedID, *int* Val)**

Write a bit value (0 or 1) to the LED in Mach3, where x ranges from 2050 – 2081.



**float GetDRO(int DRO\_ID)**

Read the selected DRO values from Mach3 or HiCON, where DRO\_ID ranges from 2050 – 2059 (Outputs), and 2000 – 2009 (Inputs).

**SetDRO(int DRO\_ID, float Val)**

Write a floating point value to the selected DRO in Mach3, where DRO\_ID ranges from 2050-2059.

**int GetPin (int Port, int Pin )**

Returns current state of digital input.

Port range: 11..14,

Pin Range: 0..15

**int SetPin (int Port, int Pin, int Value )**

Sets the digital output to a new value.

Port range: 11..14,

Pin Range: 0..15

Value: 0 or 1

**int GetHiCON(int x)**

where x is:

0 – 31	Digital Inputs
35	Drive Enable
36	Command Position Fifo Level
40 – 71	digital Outputs 0..31
80 – 85	Axis Feedback 0 – 5
90 – 95	Axis Command 0 – 5
100 – 107	RESERVED
110	Threading RPM
111 – 112	RESERVED
114 – 116	RESERVED
120	Motion Planner Done
125	User Data File Reached End of File
126	Error LED state
130 – 138	Encoder Counter Channel 0..8
140 – 171	Digital Input 32 – 63 (P13 and P14)
180 – 185	Instantaneous Velocity for axis 0 – 5

- 190 – 195 Homing in progress for axis 0 – 5
- 200 – 205 Home position is found for axis 0 – 5
- 210 – 215 Point-to-point Motion in progress for axis 0 – 5

### **SetHiCON(int x, int value)**

Value must be whole integer (no floating point)

where x is:

- 40 – 71 Digital Outputs 0 – 31
- 80 – 85 Clear Axis 0 – 5 position. value must be 0
- 90 Force Mach3 to sync to current axis positions
- 91 Arm drives. Value is a bitmask of which axes to arm. (e.g. '7' => 00000111, arms axes: x, y, and z). Value of 0 will disarm the drives.
- 92 Set Macro To Terminate on Disarm. 1 to activate or 0 to deactivate setting.
- 100 – 107 RESERVED
- 130 – 138 Write any value to Encoder Counter Channel 0 – 8
- 140 – 145 Change axis control input index. This allows a physical axis to receive the motion data of a logical axis (e.g. using slave axes)
- 150 – 155 Macro motion override setting. Set to 1 to allow the macro program to override all other motion data sources like Mach3. Set to 0 to disable the override and allow the axis to use motion data from other sources.

## **3. Timer Functions**

### **Sleep(int milliseconds)**

Pause the program flow for a specified number of milliseconds.

Example:

```
Sleep(1000) :rem This will pause the program flow for 1 second
```

## **4. Motion Functions**

**NOTE:** Motion functions are NON-BLOCKING. It is important to monitor whether the device is done with the current motion process or not, if you plan to move in reverse direction, otherwise the motion function on the current axis will be ignored if the device is still in the motion process. You can check whether the device is done with the motion process by calling **GetHiCON(120)**, which returns if the motion planning on all axes are done or not, as well as **GetHiCON(210 – 215)** which checks for axis 0 – 5 individually. You can also cancel an existing motion process by calling **CancelMove(int AxisID)**, which cancels the motion process on the selected AxisID.

**DoMotionAxis(int axisID, float finalPosition, float acceleration, float velocity, int mode )**

Execute point to point motion for selected axis from current position. Position in units, acceleration in unit/sec<sup>2</sup>, velocity in units/min. The sign of finalPosition (eg +1, -1, etc) selects the direction of rotation.

**Mode Value:**

2: Incremental Move

4: Absolute Move

8: Velocity move

24: Velocity move with exact stop in degrees (-360 < finalPosition < 360)

Example:

```
DoMotionAxis( 1, 500.25, 2.45, 10.2, 4 )
```

Move Axis1, to absolute position 500.25 inch with an acceleration of 2.45 inch/sec<sup>2</sup> and velocity of 10.2 inch/min (assuming units are set to inches in Mach3).

Example:

```
DoMotionAxis( 0, -90, 2.45, 10.2, 24 )
```

Move Axis0 in velocity mode with an acceleration of 2.45 inch/sec<sup>2</sup> and velocity of 10.2 inch/min. Rotation is in reverse direction. When CancelMove() is initiated, the axis will decelerate to stop at the angle of 90 degrees (defined by finalPosition).

NOTE:

- You may use the CancelMove(int AxisID) to stop this motion at any time.
- Velocity mode motion is continuous and will not stop unless the CancelMove(int AxisID) function is called.

**DoMotionXYZ(float positionX, float positionY, float positionZ, float acceleration, float velocity, int mode)**

Execute point to point motion for x, y, and z axes from current position. Position in units, acceleration in unit/sec<sup>2</sup>, velocity in units/min.

Mode: 2 for incremental, 4 for absolute, 8 for velocity move

Example:

```
DoMotionXYZ( 500.25, 600.25, 700.25, 2.45, 10.2, 4 )
```

Move XYZ position, to X 500.25, Y 600.25, and Z of 700.25 inches, with an acceleration of 2.45 inches/sec<sup>2</sup> and velocity of 10.2 inches/minute (assuming units are set to inches in Mach3).

NOTE:

- You may use the CancelMove(int AxisID) to stop this motion at any time.
- Velocity mode motion is continuous and will not stop unless the CancelMove(int AxisID) function is called.

**int DoHoming(int axis, float vel, float accel, float HomePosValue, int DirReverse, int UseIndexPulse, int UseHomeSensor )**

Move the specified axis to find it's home position. Returns zero if there was an error starting the homing routine or non-zero for success.

Homing Sequence for Home Sensor (no index pulse):

1. Move in one direction attempting to search for the home sensor.
2. Once the sensor is found, it reverses direction (using 20% of the specified velocity) to unblock the sensor
3. The moment the sensor is unblocked, the homing sequence is completed and the “HomePosValue” is used as the new home position.

Homing Sequence for Index Pulse Only:

1. If the index pulse is already active, the axis moves a short distance to move away from the index pulse.
2. The axis moves in one direction attempting to find the Index Pulse.
3. The moment the index pulse is found, the homing sequence is completed and the “HomePosValue” is used as the new home position.

Homing Sequence for Home Sensor with Index Pulse:

1. Move in one direction attempting to search for the home sensor.
2. Once the sensor is found, it reverses direction (using 20% of the specified velocity) to unblock the sensor
3. When the home sensor is unblocked, the axis keeps moving until the index pulse is found.
4. The moment the index pulse is found, the homing sequence is completed and the “HomePosValue” is used as the new home position.

Velocity – units/sec

Acceleration – units/sec<sup>2</sup>

HomePosValue – The axis position value (in units) to assign to the home position when found.

DirReverse – 0 for forward motion, and 1 for reverse.

UseIndexPulse – 0 to disable, 1 to enable.

UseHomeSensor – 0 to disable, 1 to enable.

Example:

Home Z axis, Velocity 2in/sec, Accel 5in/sec<sup>2</sup>, Final Home position 10.0Inch, Forward direction, Index pulse not used, Use home sensor.

```
Ret = DoHoming( 2, 2.0, 5.0, 10.0, 0, 0, 1 )
```

NOTE:

- This function will return an error if both “UseIndexPulse” and “UseHomeSensor” are set to zero.
- It is important to monitor the result of the homing routine by checking **GetHiCON(190 – 195)** to check if homing process is still ongoing for the axis. **GetHiCON(200 – 205)** is used to check if the home position has been found for the specified axis upon completion of the homing process.

**CancelMove(int AxisID)**

Cancels the motion process of the selected Axis.

```
CancelMove(0)
```

Stops whatever motion there is on Axis0.

## V. Standalone Operation with HiCON Basic

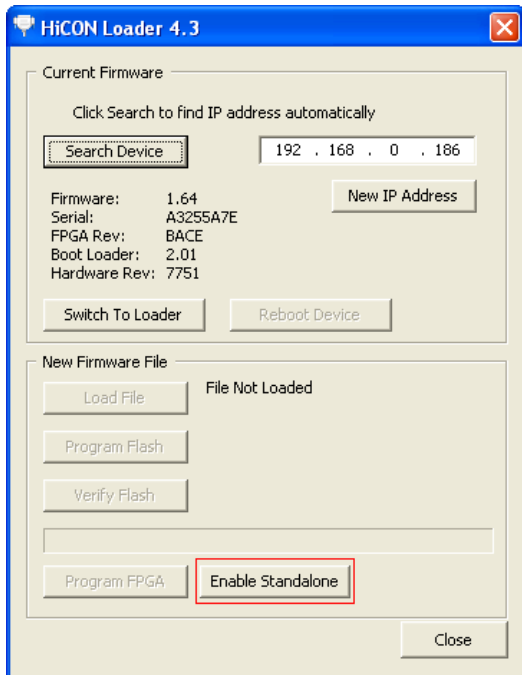
HiCON allows complete standalone operation without the need for PC software intervention. This feature is accessed with the use of HiCON Basic programs to instruct HiCON to perform sequential complex operations.

To enable Standalone Operation of the HiCON, you will need:

- A HiCON Basic program set to auto run on the HiCON. (See: HiCON Basic Loader)
- A function call to arm the drive in your program. (See: SetHiCON(91, <axis selection bits>))
- The configuration for each axis flashed on the HiCON.

To flash the Axis Configuration to the HiCON:

1. Open Mach3 and set your desired configuration for each axis.
2. Arm the HiCON and exercise the axes with mach3 jogging feature or gCode commands.
3. Open the HiCON Loader Application (downloaded from [vitalsystem website](http://vitalsystem.com))
4. Press the “Search Device” Button on the Loader Application.
5. With the HiCON still armed, click on “Enable Standalone”. This will flash the axis configuration to the HiCON.
6. Ethernet connection can now be removed from the HiCON.



## VI. Programming Hints

1. It is recommended to use the text editor Notepad++ as an alternative to Windows Notepad for easier coding. The user can set the language to VB in order to allow syntax coloring. This can be done by selecting the "Language" Menu on the menu bar, then V->VB. Notepad++ can be downloaded from here: <http://notepad-plus-plus.org/download/v5.9.8.html>.
2. For specifying floating point data, always use format like 5.00 (indicate the decimal point e.g. 10.0, 12.05 etc.).
3. When programming, do not label "if" statements the same as its destination. They must be on separate lines.

Example:

This is **NOT CORRECT** and will cause the program to lock up.

```
100 if a = 1 THEN GOTO 100
```

The **CORRECT** way:

```
100
200 if a = 1 THEN GOTO 100
```

OR

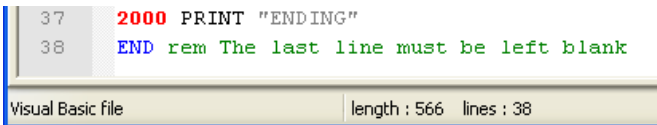
```
100
    if a = 1 then GOTO 100
```

4. Declaring integer and floating point values

```
a = 10 :rem a is an integer value
b = 10.0 :rem b is a floating point value
```

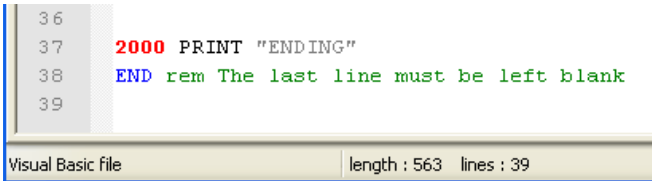
5. The last line at the end of the HiCON Basic program must be a blank line otherwise the program will return an error.

**Incorrect:**



```
37 2000 PRINT "ENDING"
38 END rem The last line must be left blank
```

**Correct:**



```
36
37 2000 PRINT "ENDING"
38 END rem The last line must be left blank
39
```

6. Use the Sleep(milliseconds) command to pause the program flow as necessary. It is recommended to use "Sleep(int)" when the task calls for putting the program flow on hold.

7. When issuing multiple motion commands on the same axis, it is necessary to check if the previous motion is complete. Otherwise, the new motion command will be ignored if the first command is still being processed
8. When using HiCON Basic for Standalone Operation, make sure the HiCON Basic program can arm the device. "SetHiCON(91, <bitmask>)". If it appears that the HiCON is not performing any motion, make sure that the ARM LED (orange LED) on the HiCON CPU is on.