USER MANUAL

MC-02A4

4-AXES MOTION CONTROLLER
FOR BIPOLAR AND UNIPOLAR STEP MOTORS

INTERINAR ELECTRONICS
http://www.interinar.com

rev.1.00

Copyright Information

Disclaimer of Liability

Interinar Electronics is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Interinar products.
Interinar Electronics is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. Customer takes full responsibility for any application where Interinar products are implemented.

Interinar Electronics Technical Support
Email: support@interinar.com
Website: http://www.interinar.com

1 OVERVIEW

1.1.1 MC-02A4 Overview

This manual describes the MC-02A4 4-Axes Motion Controller only. For MC-02A2 and MC-02A1, please refer to their designated documents.

Naming convention:

The word	CONTROLLER	- refers to the MC-02A4.
The word	DRIVER	- refers to the Step Motor Driver.
?starton		– color red, all software commands sent from PC, replace ? with axis label (x,y,z,u)
?STA		– color blue, all electrical signals connected to the controller (example: limits switch, push button, relay, etc.), replace ? with axis label (x,y,z,u)

The CONTROLLER is designed to work with most of the DRIVERS with Step/Direction interface.
Host computer communicates with CONTROLLER through USB port.
The CONTROLLER may also run as a standalone controller (custom implementation – not available in this version).

1.1.2 SAFETY SUMMARY

Do not plug/unplug any wires and connectors to/from the CONTROLLER while the power in on.
Do not make any modification to the board or components.
Always remove power and wait 10 seconds to discharge the circuit before touching it.
Various electronic components are ESD sensitive devices, therefore the CONTROLLER must be handled in accordance to safety procedures specified for ESD devices.

1.1.3 TECHNICAL SUPPORT

Interinar Electronics will be happy to respond to any question or concern regarding the CONTROLLER or any other product it manufactures or sells. Contact Technical Support Staff by sending email to support@interinar.com

2 Hardware Reference

2.1 HARDWARE DESCRIPTION

The CONTROLLER controls Unipolar and Bipolar Step Motors via various drivers manufactured by Interinar Electronics. Any other DRIVER with Step/Direction interface and inputs with TTL-level (5V) can also be used.
The PC interface communicates through USB port with the CONTROLLER, which in turn generates motion profile for the motor.

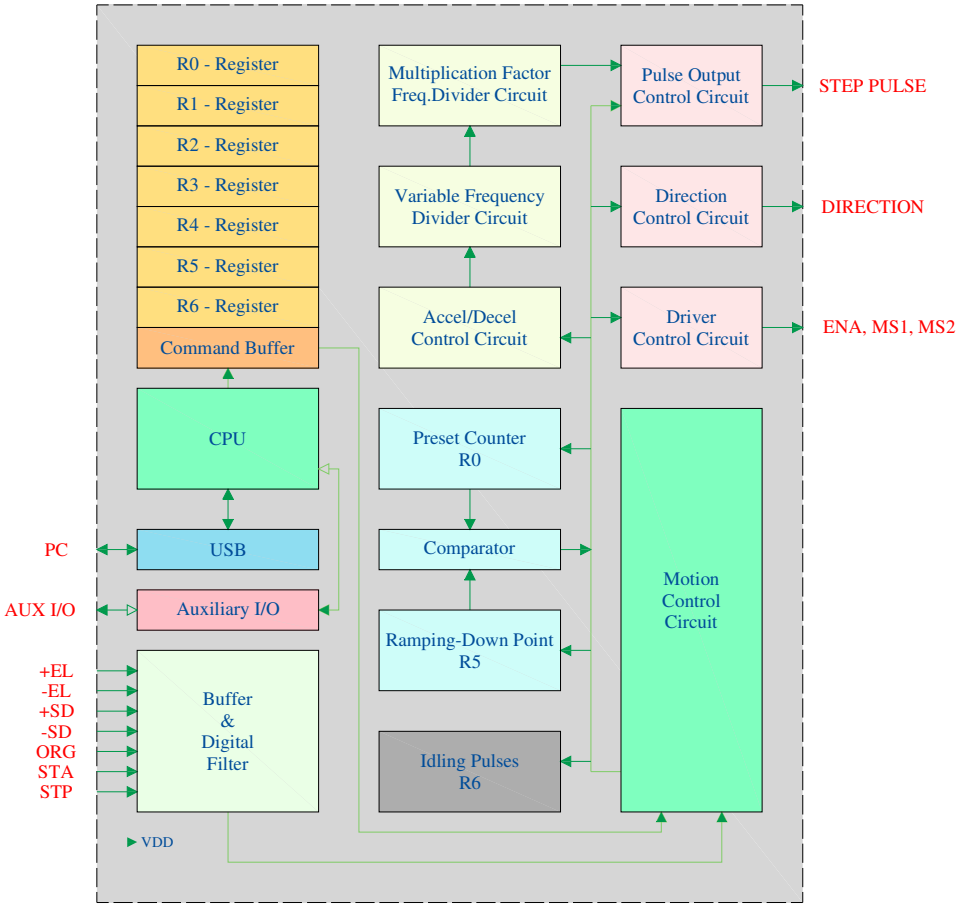


Figure 1. Simplified Block Diagram for single axis.

2.1.1 FEATURES

- Constant-Speed Operation at FL or FH rate from Start to Stop
- Varied-Speed Operation at FH rate with acceleration from FL rate at the Start and deceleration to FL rate before the Stop
- Linear or S-Curve acceleration/deceleration (ramping up/down)
- Continuous Operation until the Stop signal or command

- Preset Operation Stop after preset number of pulses
- Origin Return Operation until the ORG trigger
- Timer Function with no output pulse
- Idling Pulses from 0 to 7 pulses
- Pulse Rate variable on the way
- Suspension of acceleration/deceleration on the way
- Immediate Stop
- Deceleration Stop
- External Start and Stop control
- External signals for End Limits, Start Deceleration and Origin Return
- Monitoring operation status
- Selectable number of Positioning pulses from 1 to 16,777,215
- Speed Pulse settings separate for FL and FH rates
- Pulse rate Multiplication factor for pulse output up to 400kpps with typical values of (1x) 1to 8191pps, and (2x) 2 to 16382pps
- Acceleration and Deceleration rate settings from 2 to 1023

2.2 PORT DESCRIPTION

The CONTROLLER features the following ports:

- POWER - JS1.
- DRIVER - J7, J8, J9, J10
- LIMIT SWITCHES - J3, J4, J5, J6
- AUX - J1
- USB - CN1.
- PROG - J2 – FOR MANUFACTURING TEST ONLY

2.2.1 POWER port – JS1

Connection to external power supply.
There is no need for an external power supply if the CONTROLLER is connected to the USB port with at least **200mA** of Total Power Available.

Please check your PC (MS Windows):
Control Panel –> Device Manager –> USB Controllers –> USB Hub –> Properties –> Power tab.
It should say: "Hub Information – The hub is self-powered. Total power available: xxx mA per port."

In case no USB port with at least 200mA power is available, customer must provide external power to **JS1** Terminal Block **and remove jumper JP2**.
Recommended power supply is 500mA with DC voltage from 12V to 40V.

2.2.2 DRIVER port - J7, J8, J9, J10

Most of the BSD-series DRIVERs can be connected directly to ports J7, J8, J9, J10 (X,Y,Z,U) using 10-conductor ribbon cable (available from Interinar Electronics , P/N **FC10-14I**). No other connection to the DRIVER is required. This connection will also allow for controlling the Step Mode and Enable inputs of the BSD DRIVER. The jumpers on the BSD DRIVER need to be set as follows:

- **BSD-071**
  MS2 – OFF
  MS1 – OFF

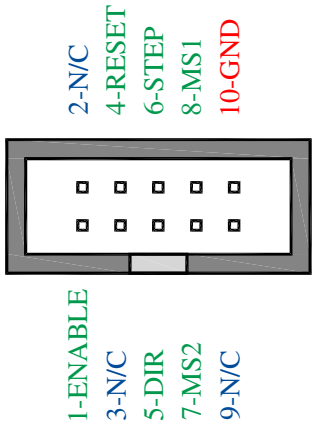
- **BSD-02 and BSD-02LH**
  JP5 – OFF
  JP6 – OFF

The other DRIVERS (not manufactured by Interinar) can be connected to the same ports after matching their signals to the corresponding pins. In addition, the inputs of such DRIVERS must accept TTL-level signals. These DRIVERS may or may not have Enable and Step Mode selection easily available.

Each port has a shrouded, 10-pin header with pin-out matching pin-to-pin connector on the BSD-series DRIVER.

Pin# – Function

- 1 - ENABLE OUTPUT
- 2 - NOT CONNECTED
- 3 - NOT CONNECTED
- 4 - DRIVER RESET OUTPUT
- 5 - DIRECTION OUTPUT
- 6 - STEP OUTPUT
- 7 - MS2 OUTPUT
- 8 - MS1 OUTPUT
- 9 - NOT CONNECTED
- 10 - GND



2.2.3 LIMIT port - J3, J4, J5, J6

The CONTROLLER **WILL NOT** output any signal if one or more of the following terminals is High or disconnected:

- STP
- EL+
- EL-

All switches and sensors connected to LIMITS port must be NC (Normal Closed) type.
Each input on this port features 4.7kohm pull-up resistor, the Schottky buffer and internal digital filter for increased noise immunity.

Note: IF NO LIMIT SWITCH IS CONNECTED, THE APPROPRIATE TERMINAL MUST BE SHORTED TO GND USING A WIRE JUMPER.

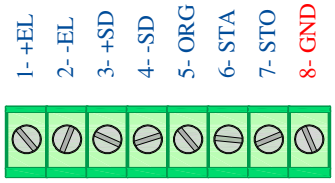


Figure 2. Limit Port

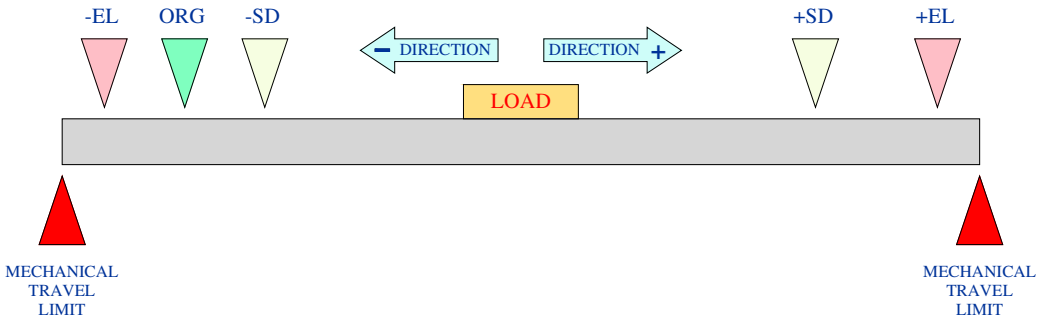


Figure 3. Typical location of Limit Switches for single axis linear motion

2.2.3.1 +EL - Input for End Limit signal in Positive direction.

If this signal becomes HIGH while moving in Positive direction, then the CONTROLLER immediately stops.
If this signal recovers LOW, the CONTROLLER will maintain the Stop condition.
If a new Start Command (`?starton`) in Positive direction is issued, the CONTROLLER will maintain the Stop condition. While in this condition, if the direction is changed to Negative or +EL signal recovers LOW, then the CONTROLLER starts immediately. To prevent such undesired behavior we recommend writing the Stop Command (`?stopon`) in advance before changing direction to Negative.

2.2.3.2 –EL - Input for End Limit signal in Negative direction.

If this signal becomes HIGH while moving in Negative direction, then the CONTROLLER immediately stops. If this signal recovers LOW, the CONTROLLER will maintain the Stop condition.
If a new Start Command (`?starton`) in Negative direction is issued, the CONTROLLER will maintain the Stop condition. While in this condition, if the direction is changed to Positive or -EL signal recovers LOW, then the CONTROLLER starts immediately. To prevent such undesired behavior we recommend writing the Stop Command (`?stopon`) in advance before changing direction to Positive.

2.2.3.3 +SD – Input for Start Deceleration (Ramping-Down) signal in Positive direction.

Valid only if SD Command (`?sdsigon`) is written in advance. Has no effect otherwise.
Valid only in Varied-Speed Operation mode.
If this signal becomes HIGH while moving in Positive direction, then the CONTROLLER ramps down the pulse from FH to FL rate according to Acceleration/Deceleration (R3) setting.
When this signal recovers LOW, the CONTROLLER accelerates back to FH rate according to Acceleration/Deceleration (R3) setting.
If new Start command (`?starton`) is issued in Positive direction while this signal is High, then the CONTROLLER will not accelerate but instead will continue output pulses at FL rate.
This signal has no effect if becomes High during deceleration caused by software ramping-down set in register R5.

2.2.3.4 –SD – Input for Start Deceleration (Ramping-Down) signal in Negative direction.

Valid only if SD Command (`?sdsigon`) is written in advance. Has no effect otherwise.
Valid only in Varied-Speed Operation mode.
If this signal becomes High while moving in Negative direction, then the CONTROLLER ramps down the pulse from FH to FL rate according to Acceleration/Deceleration (R3) setting.
When this signal recovers Low, the CONTROLLER accelerates back to FH rate according to Acceleration/Deceleration (R3) setting.
If new Start command (`?starton`) is issued in Negative direction while this signal is High, then the CONTROLLER will not accelerate but instead will continue output pulses at FL rate.
This signal has no effect if becomes High during deceleration caused by software ramping-down set in register R5.

2.2.3.5 ORG – Input for Origin Return signal.

Valid only if ORG Enable Command (`?orgon`) is written in advance. Has no effect otherwise.
When this signal becomes High the CONTROLLER will stop immediately. If this signal recovers Low, the CONTROLLER will maintain the stop condition.
If the Start Command (`?starton`) is issued while this signal is High, the CONTROLLER will not output any pulses.
If the ORG signal is disabled (`?orgoff`) while High, then the CONTROLLER will start output pulses immediately. For that reason, the Stop Command (`?stopon`) must be issued in advance.

2.2.3.6 STA – Input for External Start signal.

Valid only if External Start Valid Command (`?extstarton`) is written in advance. Has no effect otherwise.
When this signal is Low the CONTROLLER will not output pulses even if Start command is issued. Once this

input becomes High, then the CONTROLLER starts immediately. Used when operator assisted process is required or suspension of start depends on finishing other tasks.

2.2.3.7 STP – Input for External Forced Stop signal.

Regardless of moving direction, if this signal becomes High the CONTROLLER will stop immediately.
When this signal recovers Low, the CONTROLLER will maintain Stop condition.
When new Start Command (`?starton`) is issued while this signal is High, the CONTROLLER will maintain Stop condition. The Reset Command (`reset`) is required to clear the Stop condition before new Start command is issued.
This signal is useful to stop CONTROLLER in an emergency.

2.2.3.8 GND – Common GND for all signals on the same port.

This GND should not be used for any other GND connection but only as a Common GND for signals connected to the same terminal block.

2.2.4 AUX port - J1

Provides access to signals from/to external devices or used as flags.
If not used, this port may be left disconnected.

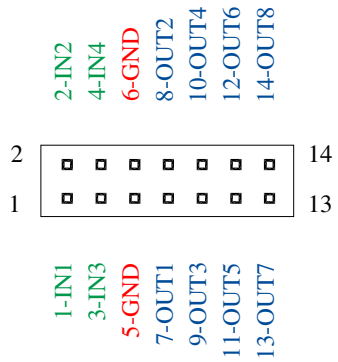
Inputs – 4 inputs, TTL-level (5V) interface with 4.7kohm pull-up resistors and Schottky buffers.

Outputs - 8 outputs, TTL-level (5V) interface.
Each of these outputs can be used as a flags or to trigger external process.
Should not be used to drive directly relays or any load exceeding 20mA per output.

THIS VERSION HAS NO VALID IMPLEMENTATION FOR INPUTS ON THIS PORT (reserved for future use).

Pin – Function

- 1 – INPUT 1
- 2 – INPUT 2
- 3 – INPUT 3
- 4 – INPUT 4
- 5 – GND
- 6 – GND
- 7 – OUTPUT 1
- 8 – OUTPUT 2
- 9 – OUTPUT 3
- 10 – OUTPUT 4
- 11 – OUTPUT 5
- 12 – OUTPUT 6
- 13 – OUTPUT 7
- 14 – OUTPUT 8



2.2.5 USB port – CN1

USB TYPE B connector. USB-OTG functionality is not implemented. The CONTROLLER can be connected to any PC with USB port.
The cable required for this connection must be terminated with USB Type B connector on the CONTROLLER side (available in retail stores).

2.3 REGISTERS

2.3.1 OVERVIEW

The CONTROLLER stores motion profile parameters in seven specialized registers. There are 4 sets of 7 registers each, one set for each axis. All of them need to be written in advance before Start command is issued. The function `setRegister` with corresponding parameters writes each register separately.

Table 1 – List of Registers

REGISTER	DESCRIPTION	MIN VALUE	MAX VALUE	COMMAND
R0	Preset Counter (step pulse counter)	0	16777215	`set?r0`
R1	FL (low speed) Pulse Rate	1	8191	`set?r1`
R2	FH (high speed) Pulse Rate	1	8191	`set?r2`
R3	Acceleration/Deceleration Rate	2	1023	`set?r3`
R4	Multiplication Factor	2	1023	`set?r4`
R5	Ramping Down Point	0	65535	`set?r5`
R6	Idling Pulses Number	0	7	`set?r6`

2.3.2 R0 – PRESET PULSE COUNTER (PC)

This is a special counter, which counts down number of pulses from the preset value. In the Preset Operation Mode enter the number of positioning pulses and write the Start Command. The Preset Counter will count down with each pulse and the CONTROLLER will stop once the Preset Counter reaches 0 (zero).
The R0 register accepts values from 0 to 16,777,215. If this register is 0 (zero), then the CONTROLLER will not output any pulses once Start Command is issued and will maintain Stop condition.
If during Preset Operation, the CONTROLLER is stopped with Stop Command or External Forced Stop signal, the Preset Counter (R0) retains the remaining number of pulses. Therefore, entering new Start command allows to output remaining number of pulses.
Once 0 (zero), the R0 register needs to be written again with new number of pulses.

2.3.3 R1 – FL PULSE RATE

This register stores the FL (Frequency Low) Pulse Rate that defines Low Speed Value. There are several ways this value can be used.
If FL speed rate is selected in Constant Speed Mode, then the CONTROLLER will output at this rate.
It is important to differentiate this register from R2, which stores FH Pulse Rate. The CONTROLLER will use FH and FL rates in the Motion Profile algorithm, and if FL is higher that FH the CONTROLLER will not perform any of Varied Speed operations properly.
In the Varied Speed Mode, the CONTROLLER will output pulses while accelerating from FL to dominant FH rate. While at FH rate it is possible to write Deceleration Stop command to decelerate to FL rate before cessation.
The R1 accepts values from 1 to 8191.
The actual FL pulse rate is the product of the R1 and m (Multiplication Rate based on values of R4 register) and it is calculated as follows:

$$\mathbf{FL\ (pps) = R1} \times \mathbf{m}$$

2.3.4 R2 – FH PULSE RATE

This register stores the FH (Frequency High) Pulse Rate that defines High Speed Value. There are several ways this value is used.
If FH speed rate is selected in Constant Speed Mode, then the CONTROLLER will output at this rate.

It is important to differentiate this register from R1, which stores FL Pulse Rate. The CONTROLLER will use FH and FL rates in the Motion Profile algorithm, and if FL is higher that FH the CONTROLLER will not perform any of Varied Speed operations properly.
In the Varied Speed Mode, the CONTROLLER will output pulses while accelerating from FL to dominant FH rate. While at FH rate it is possible to write Deceleration Stop command to decelerate to FL rate before cessation.
The R2 accepts values from 1 to 8191.
The actual FH pulse rate is the product of the R2 and m (Multiplication Rate based on values of R4 register) and it is calculated as follows:

FH (pps) = R2 x **m**

2.3.5 R3 – ACCELERATION/DECELERATION RATE

This register stores parameter defining acceleration and deceleration rate.
In the Varied Speed Mode, the CONTROLLER will accelerate from FL to dominant FH rate for the time related to R3 value.
The CONTROLLER will decelerate the output pulses from FH to FL rate only when the Deceleration Stop Command is issued, Ramping Down Point is reached by Preset Counter or one of SD signals (direction dependent) becomes High.
The R3 register accepts values from 2 to 1023.
The acceleration/deceleration time, in seconds, can be calculated as follows:

Accel/Decel time (sec) = ((R2 – **R1)** x **R3) / 4915200**

Both R1 and R2 are just values stored in registers and not the actual FL and FH rates derived from calculation involving the multiplication factor R4. In other words, for the above formula use exactly the same values you entered for R1 and R2, which are unitless.

2.3.6 R4 – MULTIPLICATION FACTOR

This register stores the Multiplication Factor used together with R1 and R2 registers to obtain the output pulse rate for FL and FH speed. The lower the value of R4 the higher the frequency of the output pulse becomes.
The R4 register accepts values from 2 to 1023.
The actual Multiplication Rate **m** is calculated as follows:

m = 4915200 / (R4 x 8192)

This formula may be used to fine tune Multiplication Rate to a specific application. However, in most cases values from the following table may be used.

Table 2 – Typical Multiplication Rate m

R4	MULTIPLICATION RATE **m**	MINIMUM OUTPUT PULSE RATE **FL**	MAXIMUM OUTPUT PULSE RATE **FH**
-	-	pps	pps
1023	0.58651	0.58651	4804.106
1000	0.6	0.6	4914.6
800	0.75	0.75	6143.25
750	0.8	0.8	6552.8
625	0.96	0.96	7863.36
600	**1**	**1**	**8191**
500	1.2	1.2	9829.2
400	1.5	1.5	12286.5
375	1.6	1.6	13105.6
320	1.875	1.875	15358.13
300	2	2	16382
240	2.5	2.5	20477.5
200	3	3	24573
150	4	4	32764
120	5	5	40955
100	6	6	49146
75	8	8	65528
60	10	10	81910
50	12	12	98282
40	15	15	122865
30	20	20	163820
24	25	25	204775
20	30	30	245730
15	40	40	327640
12	50	50	409550
10	60	60	491460
2	300	300	2457300

2.3.7 R5 – RAMPING DOWN POINT

This register stores value of the starting point for deceleration called Deceleration Ramping-Down Point. The Deceleration Ramping-Down Point is defined as the number of pulses to be output before the final stop. It is important to understand the meaning of this register.
In the Preset Varied Speed Mode, the CONTROLLER continues comparing R5 value with current value of R0 (Preset Counter). When the Preset Counter, counting always down, reaches the value entered in R5, the CONTROLLER will start decelerating the output pulses from FH rate to FL rate.
In addition, if the Start Command is issued while R5 value is greater or equal than R0 value, then the CONTROLLER does not accelerate to FH rate but maintains output pulses at FL rate.
Always determine FL, FH and Acceleration/Deceleration rates before setting the R5 register. If you define improper Ramping-Down Point, the CONTROLLER may terminate output pulses during deceleration or unnecessarily output large number of pulses at FL rate after completion of deceleration.
The exact ramping-down point can be calculated as follows:

$$\mathbf{R5 = ((R2^2 - R1^2) \times R3) / (R4 \times 16384)}$$

The R5 register accepts values from 0 to 65535.

2.3.8 R6 – IDLING PULSES

This register stores the number of idling pulses.
The CONTROLLER can generate several Idling Pulses to enable the stepper motor to start at the pulse rate near the self-starting frequency. The number of pulses entered in R6 is a subset of total number of pulses entered in R0, so the position accuracy is not affected.
If number of Idling Pulses is entered to R6 as 0, then CONTROLLER will accelerate immediately after receiving Start command or Start signal.
The R6 register accepts values from 0 to 7.

2.4 BASIC OPERATION

2.4.1 OVERVIEW

The CONTROLLER can work in three different modes:

- Continuous Mode
- Preset Mode
- Origin Return Mode

Regardless of the mode, the CONTROLLER will always stop when EL signal in moving direction or the External Forced Stop signal becomes High.
In mode with acceleration/deceleration, when the SD signals are set valid, the CONTROLLER ramps down to FL rate when SD signal in moving direction becomes High.

Reminder: Naming convention for the remaining chapters
?starton – color red, all software commands sent from PC, replace ? with axis label (x,y,z,u)
?STA – color blue, all electrical signals connected to the controller (example: limits switch, push button, relay, etc.), replace ? with axis label (x,y,z,u)

2.4.2 CONTINUOUS MODE

The CONTROLLER output pulses upon receiving Start command and continues until detects Stop command, EL signal (in moving direction) or External Forced Stop signal. The R0 (Preset Counter) value is ignored.
While in this mode, CONTROLLER can work at:

- Constant Speed
- Varied Speed

2.4.2.1 CONTINUOUS CONSTANT SPEED

The CONTROLLER will output pulses at either FL or FH rate once the Start command is received (1). The R0 value is ignored and CONTROLLER maintains output pulses until, Start is revoked, Stop command is issued, EL signal (in moving direction) or External Forced Stop signal is received (2) (see Table 3). If neither of the stop conditions occurs, the CONTROLLER will continue output pulses indefinitely.
While in this mode, it is possible to change the speed rate without stopping (see Table 4 and 5).

Table 3 – Continuous Mode Constant Speed

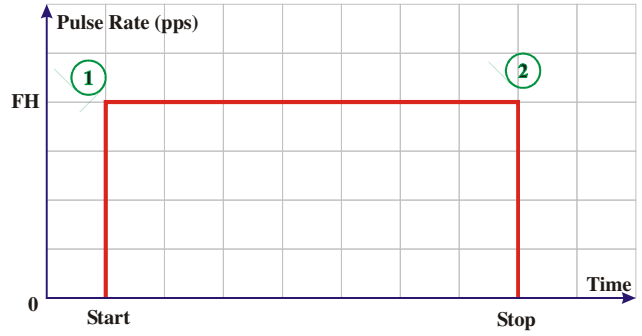
Defining commands: `?speedconstant, ?continuous`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if valid) - `?STA`
	2 OR	Start command revoked - `?startoff` Stop command – `?stopon` External Forced Stop signal – `?STO` End Limit signal – `?EL` (**+**or**-** direction dependent)

Table 4 – Continuous Mode Speed Change from High to Low

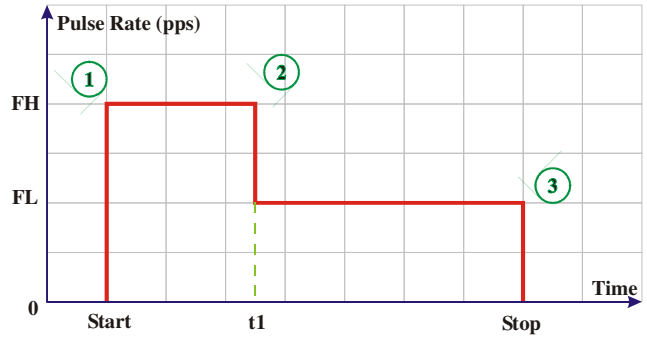
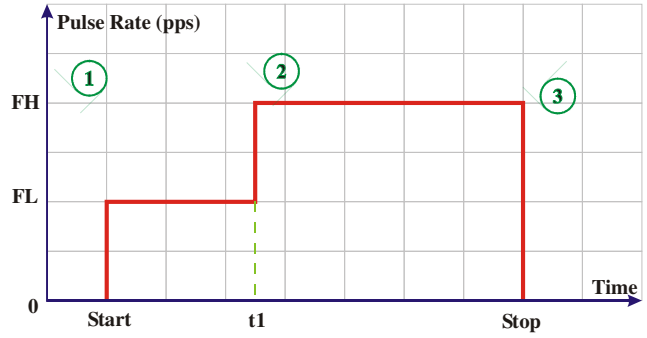
Defining commands: `?speedconstant, ?continuous`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if valid) - `?STA`
	2	Speed change command – `?flspeed`
	3 OR	Start command revoked - `?startoff` Stop command – `?stopon` External Forced Stop signal – `?STO` End Limit signal – `?EL` (+or- direction dependent)

Table 5 – Continuous Mode Speed Change from Low to High

Defining commands: `?speedconstant, ?continuous`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if valid) - `?STA`
	2	Speed change command – `?fhspeed`
	3 OR	Start command revoked - `?startoff` Stop command – `?stopon` External Forced Stop signal – `?STO` End Limit signal – `?EL` (+or- direction dependent)

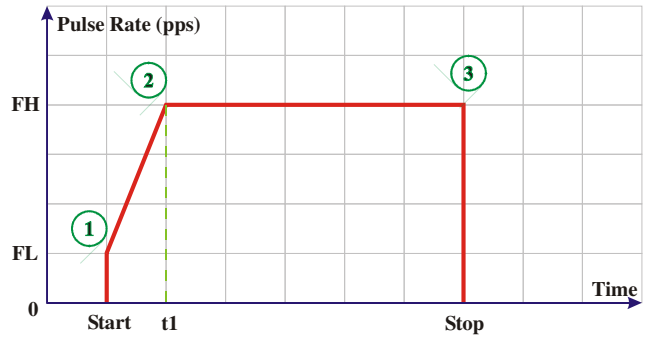
2.4.2.2 CONTINUOUS VARIED SPEED

In this mode the acceleration/deceleration can be implemented only if FL value is smaller than FH value. The CONTROLLER will start at FL rate (1) and immediately accelerate to dominant FH rate (2) in time set by R3 register. This acceleration time is represented as **ta=t1-Start**.
Once at FH rate, the CONTROLLER continues output pulses until detecting one of the conditions (Table 6):
- Start command revoked
- External Forced Stop signal
- EL signal in moving direction

At that point, it will cease operation instantly without deceleration.

Table 6 – Continuous Mode Variable-Speed with Instant Stop

Defining commands: `?variedspeed, ?continuous`

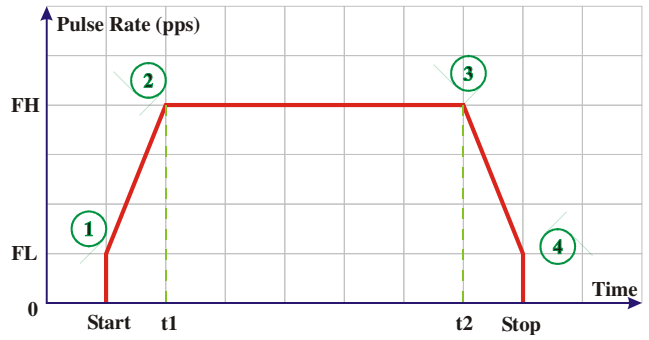


Point	Source of trigger
1	Start command – `?starton` or Start signal (if valid) - `?STA`
2	End of acceleration to FH
3 OR	Start command revoked - `?startoff` External Forced Stop signal – `?STO` End Limit signal – `?EL` (+or- direction dependent)

If the Stop command is issued, then the CONTROLLER will decelerate to FL and then stop (Table 7). This deceleration time is represented as **td=Stop-t2**.

Table 7 – Continuous Mode Variable-Speed with Decelerated Stop

Defining commands: `?variedspeed, ?continuous`



Point	Source of trigger
1	Start command – `?starton` or Start signal (if valid) - `?STA`
2	End of acceleration to FH
3	Decelerated Stop– `?stopon`
4	End of deceleration to FL and stop

2.4.3 PRESET MODE

The Preset Mode allows accurate positioning by designating the number of output pulses in specified direction. The number of pulses must be written to R0 (PC - Preset Counter) in advance. The CONTROLLER will output pulses as long as R0 value is greater than 0 and none of the stop conditions occur. Once R0 reaches 0 the CONTROLLER stops. The R0 needs to be written with new number of pulses every time next

position is required. The Preset Counter (R0) always counts down regardless of direction.
While in this mode, CONTROLLER can work at:

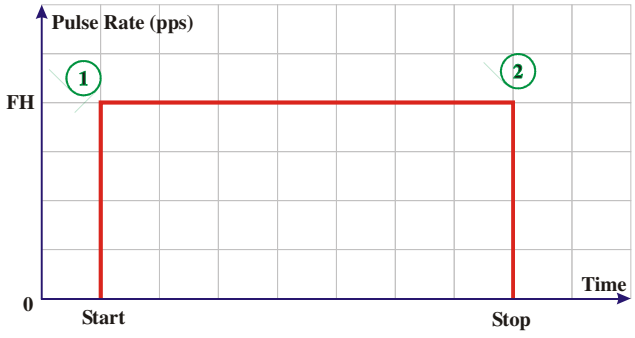
- Constant Speed
- Varied Speed

2.4.3.1 PRESET CONSTANT SPEED

The CONTROLLER starts at either FL or FH rate and only if R0 value is greater than 0. The CONTROLLER maintains output pulses until R0 reaches 0 or one of the stop events occur (Table 8):
- Start command revoked
- External Forced Stop signal
- EL signal in moving direction

At that point, it will cease operation instantly without deceleration.

Table 8 – Preset Mode Constant-Speed

<table>
<tr><td colspan="3">Defining commands: <code>?speedconstant, ?preset, set?r0</code></td></tr>
<tr><td rowspan="3"> 
</td><td>Point</td><td>Source of trigger</td></tr>
<tr><td>1</td><td><code>?starton</code> or
Start signal (if valid) - <code>?STA</code></td></tr>
<tr><td>2

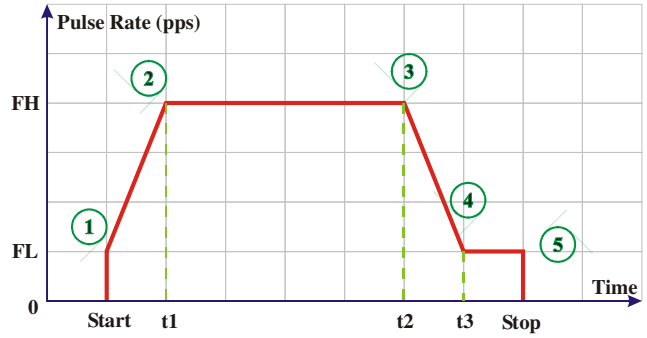
OR</td><td><code>?PC = 0</code> or
Start command revoked - <code>?startoff</code>
Stop command – <code>?stopon</code>
External Forced Stop signal – <code>?STO</code>
End Limit signal – <code>?EL</code> (+or- direction dependent)</td></tr>
</table>

2.4.3.2 PRESET VARIED SPEED

This mode includes acceleration and deceleration only if the FL value is set lower than FH value.
The CONTROLLER, upon receiving Start command (or signal), starts output pulses at FL rate (1) and immediately accelerates to dominant FH rate (2) in time set by R3 register. This acceleration time is represented as **ta=t1-Start**.

Once FH rate is reached, the CONTROLLER continues output pulses until one of the following events occur:
1- the Preset Counter R0 value becomes equal to R5 value (3), then the CONTROLLER will start deceleration from FH rate to FL rate (4) in time set by R3 value. The deceleration time is represented as **td=t3-t2**.
Once it reaches the FL rate, the CONTROLLER output pulses until R0 counts down to 0. It is important to set R5 value small enough to prevent pulses at FL rate for prolonged time **tn=Stop-t3** (see Table 9).

Table 9 – Preset Mode Varied-Speed with High value of R5

Defining commands: `?variedspeed, ?preset, set?r0`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration to FH
	3 OR	`?PC = ?R5` or Stop command – `?stopon`
	4	End of deceleration to FL
	5	`?PC=0` then stop

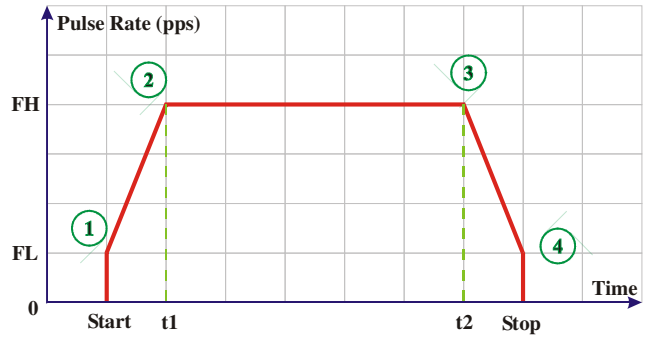
2 - the Preset Counter R0 value becomes equal to R5 value (3) and R5 value was calculated according to formula:

$$\mathbf{R5 = ((R2^2 - R1^2) \times R3) / (R4 \times 16384)}$$

In such case deceleration stops exactly when R0 will become 0(see Table 10).

3 - the Decelerated Stop command is received – the CONTROLLER will decelerate from FH rate to FL rate and once FL rate is reached the CONTROLLER will stop output pulses. The R0 may still contain remaining pulses to execute (see Table 10).

Table 10 – Preset Mode Varied-Speed with Exact value of R5

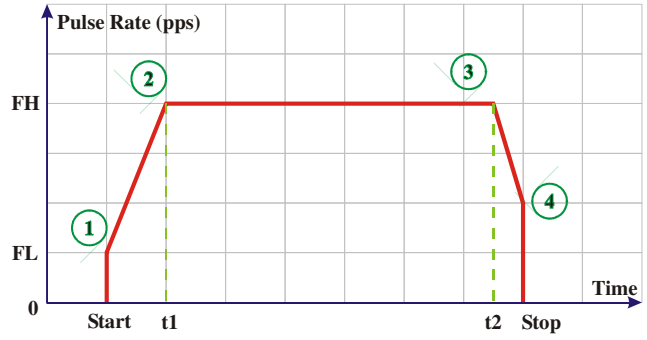
Defining commands: `?variedspeed, ?preset, set?r0`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration to FH
	3 OR	`?PC = ?R5` (?R5 exactly calculated) or Stop command – `?stopon`
	4	End of deceleration to FL and Stop

4 - the Preset Counter R0 value becomes equal to R5 value (3) and R5 was value is smaller than result of the formula:

$$\mathbf{R5 = ((R2^2 - R1^2) \times R3) / (R4 \times 16384)}$$

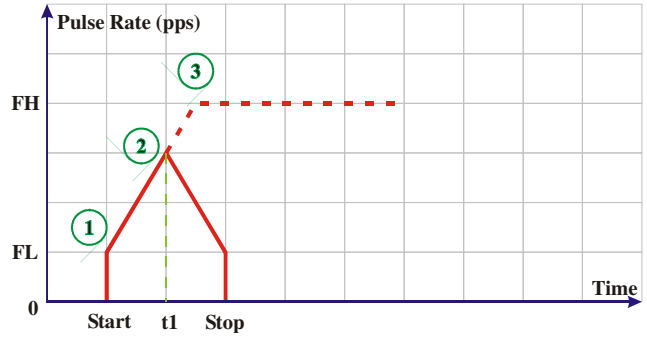
In this case, deceleration will end at the speed higher than FL because there was not enough pulses left in R0 to complete deceleration to FL. (see Table 11)

Table 11 – Preset Mode Varied-Speed with Low value of R5

Defining commands: `?variedspeed, ?preset, set?r0`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration to FH
	3	`?PC = ?R5` (?R5 < formula result)
	4	End of deceleration to FL and Stop

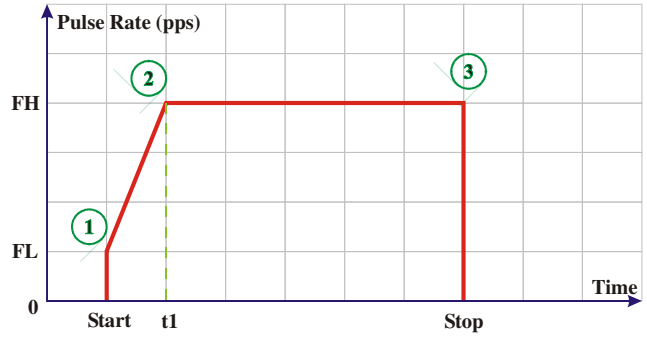
5 - the Preset Counter R0 value becomes equal to R5 value (3) during acceleration
Excessive R5 value causes deceleration before acceleration ends. Motor will not reach FH rate before deceleration starts (see Table 12).

Table 12 – Preset Mode Varied-Speed with value of R5 in acceleration range

Defining commands: `?variedspeed, ?preset, set?r0`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration before FH `?PC = ?R5` Deceleration Start
	3	End of deceleration to FL and Stop

6 - the Preset Counter R0 reaches 0 value while R5 value was 0 (no ramping-down point set) – the CONTROLLER will stop without deceleration (see Table 13)
7 - the Stop command is received – the CONTROLLER will cease operation instantly without deceleration (see Table 13)
8 - the External Forced Stop signal is received – the CONTROLLER will cease operation instantly without deceleration (see Table 13)
9 - the EL signal in moving direction is received – the CONTROLLER will cease operation instantly without deceleration (see Table 13)

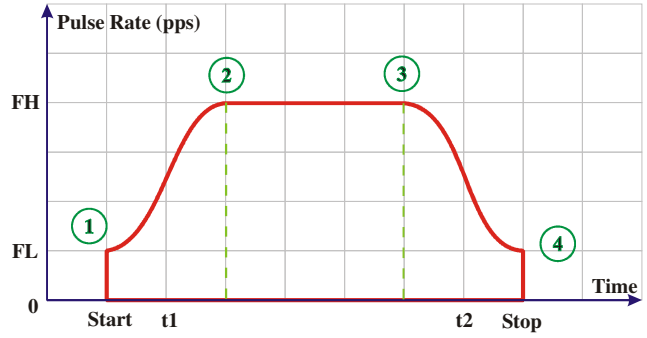
Table 13 – Preset Mode Varied-Speed with R5=0

Defining commands: `?variedspeed, ?preset, set?r0`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration to FH
	3 OR	`?PC = 0` or Start command revoked - `?startoff` Stop command – `?stopon` External Forced Stop signal – `?STO` End Limit signal – `?EL` (+or- direction dependent)

2.4.4 S-CURVE

By sending `?scurveon` the CONTROLLER will use S-curve for acceleration and deceleration (see Table 14). To restore Linear acceleration and deceleration the CONTROLLER must receive `?scurveoff` command. S-curve is recommended for high inertia loads.

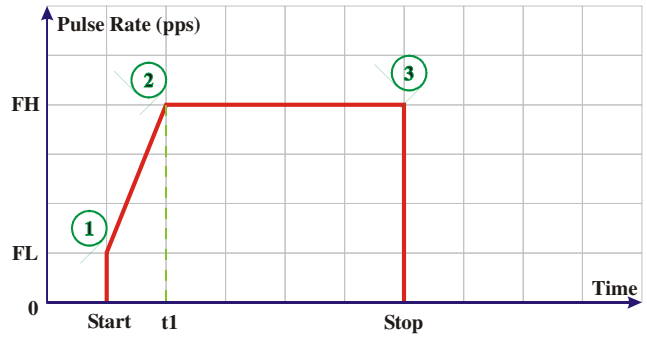
Table 14 – S-Curve Acceleration and Deceleration

Defining commands: `?scurveon`		
	Point	Source of trigger
	1	Start command – `?starton` or Start signal (if Valid) - `?STA`
	2	End of acceleration to FH
	3 OR	`?PC = ?R5` or Start command revoked - `?startoff` Stop command – `?stopon` External Forced Stop signal – `?STO` End Limit signal – `?EL` (+or- direction dependent)

2.4.5 ORIGIN RETURN MODE

This mode is available by making ORG signal valid. If ORG is valid, then CONTROLLER, regardless of selected mode will always stop pulses when ORG signal becomes HIGH.

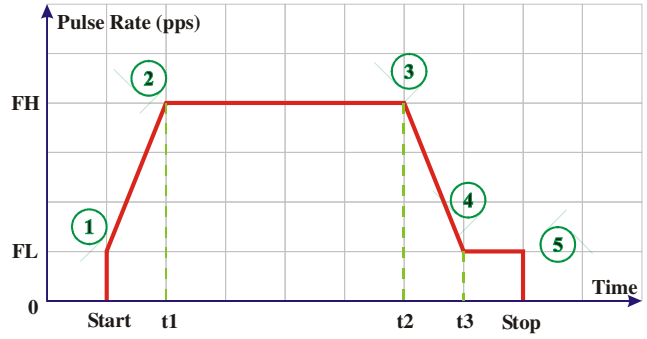
Table 15 – Origin Return Mode with valid ORG signal



Point	Source of trigger
1	Start command – `?starton` or Start signal - `?STA`
2	End of acceleration to FH
3	Origin signal - `?ORG`

However, the smooth origin return operation can be achieved making SD signal valid and placing SD sensor in close proximity of Origin Point. In this case, the CONTROLLER will start deceleration to FL rate right before Origin Point and finish rest of the travel at FL rate until ORG signal becomes HIGH.

Table 16– Origin Return Mode with valid ORG signal and deceleration before ORG



Point	Source of trigger
1	Start command – `?starton` or Start signal - `?STA`
2	End of acceleration to FH
3	Start Deceleration Point - `?SD`
4	End of deceleration to FL and Stop
5	Origin signal - `?ORG`

3 MC-02 LANGUAGE REFERENCE

The custom application can be created with help of two libraries:

- MC02HID - USB communication with the CONTROLLER
- MC02ENCOD - access functions in MCEncoder class

Both libraries **must be referenced** in customer's code.

The CONTROLLER comes also with an additional program allowing interactive development, code editing, testing and monitoring.

3.1 Package MC02HID

3.1.1 Classes

- class **MCDevice**

This is the only class available in MC02HID Package. Its role is to establish communication with MC-02 series motion controllers.

3.1.2 MC02HID.MCDevice Class Documentation

3.1.2.1 Static Public Member Functions

- static Boolean **FindTheHid** ()

  Search for a HID-class device by its Vendor ID and Product ID.
  Vendor Id is set to 0x03EB
  Product Id is set to 0x2013

Declaration:

```
static Boolean MC02HID.MCDevice.FindTheHid()
```

Returns: True - device detected
False - device not detected

This is the only function available in MCDevice class.
Other functions are not available to the end user and will not be documented here.

If the user desires to create his own language instead of using MCEncoder class, then the direct implementation of this class is necessary. In such case, please request document "MC02HID Library Reference", where two additional functions are documented.

3.2 Package MC02ENCOD

3.2.1 Classes

- class **MCEncoder**

This is the only class available in MC02ENCOD Package. It gives access to internal registers of MC-02 series motion controllers.

3.2.2 MC02ENCOD.MCEncoder Class Documentation

3.2.2.1 Static Public Member Functions

- static void **setRegister** (string code, Int32 regVal)
- static void **setCommand** (string code)
- static Byte[] **getData** (string code)

3.2.3 FUNCTION setRegister

This function writes data to registers R0 to R6.

Declaration:

```
public static void MC02ENCOD.MCEncoder.setRegister(string code, Int32 value)
```

All commands using this function will have the same format:

```
setRegister(code, value)
```

Arguments:

`code` – one from the following list
`value` – value in the range valid for particular register (see Table 1 for Register Limits)

Table 17- List of valid code names

X	Y	Z	U
setxr0	setyr0	setzr0	setur0
setxr1	setyr1	setzr1	setur1
setxr2	setyr2	setzr2	setur2
setxr3	setyr3	setzr3	setur3
setxr4	setyr4	setzr4	setur4
setxr5	setyr5	setzr5	setur5
setxr6	setyr6	setzr6	setur6

3.2.3.1 Register R0 - PRESET COUNTER

Description: Writes new value to register R0 (Preset Counter).
Each axis has its own, separate register R0.

Details: see chapter 2.3.2 on pg 2-8

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr0`
`setyr0`
`setzr0`
`setur0`
`value` – must be in range of `0 <= value <= 16777215`

Modes: Preset Mode - R0 stores number of pulses to be executed
Continuous Mode - R0 value is ignored

Example: `setRegister(setxr0, 12345)` - writes 12345 to register R0 on X axis

3.2.3.2 Register R1 - FL PULSE RATE (Frequency Low)

Description: Writes new value to register R1 (FL Pulse Rate).
Each axis has its own, separate register R1.

Details: see chapter 2.3.3 on pg 2-8

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr1`
`setyr1`
`setzr1`
`setur1`
`value` – must be in range of `0 <= value <= 8191`

Example: `setRegister(setxr1, 300)` - writes 300 to register R1 on X axis

3.2.3.3 Register R2 - FH PULSE RATE (Frequency High)

Description: Writes new value to register R2 (FH Pulse Rate).
Each axis has its own, separate register R2.

Details: see chapter 2.3.4 on pg 2-9

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr2`
`setyr2`
`setzr2`
`setur2`
`value` – must be in range of `0 <= value <= 8191`

Example: `setRegister(setxr2, 2500)` - writes 2500 to register R2 on X axis

3.2.3.4 Register R3 – ACCELERATION / DECELERATION RATE

Description: Writes new value to register R3 (Acceleration/Deceleration Rate).
Each axis has its own, separate register R3.

Details: see chapter 2.3.5 on pg 2-9

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr3`
`setyr3`
`setzr3`
`setur3`
`value` – must be in range of `2 <= value <= 1023`

Example: `setRegister(setxr3, 720)` - writes 720 to register R3 on X axis

3.2.3.5 Register R4 - MULTIPLICATION FACTOR

Description: Writes new value to register R4 (Multiplication Factor).
Each axis has its own, separate register R4.

Details: see chapter 2.3.6 on pg 2-9

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr4`
`setyr4`
`setzr4`
`setur4`
`value` – must be in range of `2 <= value <= 1023`

Example: `setRegister(setxr4, 600)` - writes 600 to register R4 on X axis

3.2.3.6 Register R5 - RAMPING-DOWN POINT

Description: Writes new value to register R5 (Ramping Down Point).
Each axis has its own, separate register R5.

Details: see chapter 2.3.7 on pg 2-10

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr5`
`setyr5`
`setzr5`
`setur5`

`value` – must be in range of `0 <= value <= 65535`

Example: `setRegister(setxr5, 3200)` - writes 3200 to register R5 on X axis

3.2.3.7 Register R6 - Idling Pulses Number

Description: Writes new value to register R6 (Idling Pulses Number).
Each axis has its own, separate register R6.

Details: see chapter 2.3.8 on pg 2-11

Syntax: `setRegister(code, value)`

Arguments: `code` – one of the following:
`setxr6`
`setyr6`
`setzr6`
`setur6`
`value` – must be in range of `0 <= value <= 7`

Example: `setRegister(setxr6, 4)` - writes 4 to register R6 on X axis

3.2.4 FUNCTION setCommand

This function writes command to the controller.

Declaration:

```
public static void MC02ENCOD.MCEncoder.setCommand(string code)
```

All these commands have the same format:

```
setCommand(code)
```

Arguments:
code – one from the following list

Table 18- List of valid code names

X	Y	Z	U	AUX
xstarton	ystarton	zstarton	ustarton	auxout1on
xstartoff	ystartoff	zstartoff	ustartoff	auxout1off
xextstarton	yextstarton	zextstarton	uextstarton	auxout2on
xextstartoff	yextstartoff	zextstartoff	uextstartoff	auxout2off
xstopon	ystopon	zstopon	ustopon	auxout3on
xstopoff	ystopoff	zstopoff	ustopoff	auxout3off
xpreset	ypreset	zpreset	upreset	auxout4on
xcontinuous	ycontinuous	zcontinuous	ucontinuous	auxout4off
xspeedconstant	yspeedconstant	zspeedconstant	uspeedconstant	auxout5on
xspeedvaried	yspeedvaried	zspeedvaried	uspeedvaried	auxout5off
xspeedfl	yspeedfl	zspeedfl	uspeedfl	auxout6on

xspeedfh	yspeedfh	zspeedfh	uspeedfh	auxout6off
xdircw	ydircw	zdircw	udircw	auxout7on
xdirccw	ydirccw	zdirccw	udirccw	auxout7off
xorgon	yorgon	zorgon	uorgon	auxout8on
xorgoff	yorgoff	zorgoff	uorgoff	auxout8off
xsdsigon	ysdsigon	zsdsigon	usdsigon	reset
xsdsigoff	ysdsigoff	zsdsigoff	usdsigoff	
xscurveon	yscurveon	zscurveon	uscurveon	
xscurveoff	yscurveoff	zscurveoff	uscurveoff	
xenaon	yenaon	zenaon	uenaon	
xenaoff	yenaoff	zenaoff	uenaoff	
xstep1	ystep1	zstep1	ustep1	
xstep2	ystep2	zstep2	ustep2	
xstep4	ystep4	zstep4	ustep4	
xstep16	ystep16	zstep16	ustep16	

3.2.4.1 ORG SIGNAL CONTROL, ORIGIN RETURN MODE CONTROL

Description: Activates Origin Return mode by making ORG signal valid or invalid.
The CONTROLLER will stop output pulses if ORG is valid and ORG signal is received.
Each axis has its own, separate pair of commands.

Details: see chapter 2.2.3.5 on pg 2-6

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xorgon
xorgoff
yorgon
yorgoff
zorgon
zorgoff
uorgon
uorgoff
```

Example: `setCommand(xorgon)` - makes ORG signal for x axis valid

3.2.4.2 SD SIGNAL CONTROL

Description: Makes SD (Start Deceleration) signals valid or invalid.
The CONTROLLER will start deceleration if SD is valid and SD signal is received.
Each axis has its own, separate pair of commands.

Details: see chapter 2.2.3.3 on pg 2-6

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xsdsigon
xsdsigoff
ysdsigon
ysdsigoff
zsdsigon
zsdsigoff
usdsigon
usdsigoff
```

Example: `setCommand(xsdsigon)` - makes SD signals for x axis valid

Notes: Requires exact calculation of Ramping Down point - see chapter 2.3.7 on pg 2-10.

3.2.4.3 CONTINUOUS MODE and PRESET MODE

Description: Select between the Continuous Mode (where CONTROLLER will output pulses regardless of value of Preset Counter) and the Preset Mode (where CONTROLLER will stop output pulses when the Preset Counter counts down to 0).
Each axis has its own, separate pair of commands.

Details: for Continuous Mode see chapter 2.4.2 on pg 2-11
for Preset Mode see chapter 2.4.3 on pg 2-13

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xpreset
xcontinuous
ypreset
ycontinuous
zpreset
zcontinuous
upreset
ucontinuous
```

Example: `setCommand(xpreset)` - x axis will operate in Preset Mode

3.2.4.4 DIRECTION CONTROL

Description: Select between CW (+) and CCW (-) direction of the rotation.
Each axis has its own, separate pair of commands.

Details: -

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xdircw
xdirccw
ydircw
ydirccw
```

```
zdircw
zdirccw
udircw
udirccw
```

Example: `setCommand(xdircw)` - x axis will rotate in Clockwise CW (+) direction

Notes: This command is hardware dependent. Check your driver user manual for information about allowed direction signals. Install +EL and +SD sensors on the way of positive travel direction and –EL and –SD on the way of negative travel direction.

3.2.4.5 SPEED SELECTION

Description: Selects between two available output pulse rates FL and FH.
Each axis has its own, separate pair of commands.

Details: for FL see chapter 2.3.3 on pg 2-10
for FH see chapter 2.3.4 on pg 2-11

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xspeedfl
xspeedfh
yspeedfl
yspeedfh
zspeedfl
zspeedfh
uspeedfl
uspeedfh
```

Example: `setCommand(zspeedfl)` - z axis will output pulse at FL frequency

3.2.4.6 STA SIGNAL CONTROL, EXTERNAL START CONTROL

Description: Makes STA (External Start) signals valid or invalid.
The CONTROLLER will start if STA is valid and STA signal is received.
Each axis has its own, separate pair of command.

Details: see chapter 2.2.3.6 on pg 2-6

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xextstarton
xextstartoff
yextstarton
yextstartoff
zextstarton
zextstartoff
```

```
uextstarton
uextstartoff
```

Example: `setCommand(xextstarton)` - makes STA signal for x axis valid

Notes: If the CONTROLLER was started by the STA signal, and then stopped, it cannot be started again using STA, unless Reset command is written in advance.

3.2.4.7 CONSTANT and VARIED SPEED

Description: Selects between Constant and Varied speed.
Each axis has its own, separate pair of commands.

Details: see chapter 2.4 on pg 2-11

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xspeedconstant
xspeedvaried
yspeedconstant
yspeedvaried
zspeedconstant
zspeedvaried
uspeedconstant
uspeedvaried
```

Example: `setCommand(zspeedconstant)` - z axis will operate at constant speed

3.2.4.8 STOP or DECELERATED STOP CONTROL

Description: Immediately stop if in Constant speed mode
Decelerated stop if in Varied speed mode.
Each axis has its own, separate pair of commands.

Details: see chapter 2.4 on pg 2-11

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xstopon
xstopoff
ystopon
ystopoff
zstopon
zstopoff
ustopon
ustopoff
```

Example: `setCommand(xstopon)` - x axis will stop immediately

3.2.4.9 START CONTROL

Description: Start or suspend start until External Start received.
Each axis has its own, separate pair of commands.

Details: see chapter 2.2.3.6 on pg 2-6

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:
`xstarton`
`xstartoff`
`ystarton`
`ystartoff`
`zstarton`
`zstartoff`
`ustarton`
`ustartoff`

Example: `setCommand(xstarton)` - x axis will start or suspend start until valid STA

3.2.4.10 RESET

Description: This command will reset CONTROLLER to default.

Details: clears all registers, sets Continuous Constant Speed mode in CW direction at FL speed.

Syntax: `setCommand(reset)`

Example: `setCommand(reset)` - makes SD signals for axis x valid

3.2.4.11 LINEAR OR S-CURVE ACCELERATION/DECELERATION

Description: Select between Linear and S-curve acceleration and deceleration.
Each axis has its own, separate pair of commands.

Details: see chapter 2.4.4 on pg 2-17

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:
`xscurveon`
`xscurveoff`
`yscurveon`
`yscurveoff`
`zscurveon`
`zscurveoff`

```
uscurveon
uscurveoff
```

Example: `setCommand(xscurveon)` - x axis will use S-curve during acceleration/deceleration

3.2.4.12 DRIVER ENABLE CONTROL

Description: Enable or disable the step motor driver.
Each axis has its own, separate pair of commands.

Details: -

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
xenaon
xenaoff
yenaon
yenaoff
zenaon
zenaoff
uenaon
uenaoff
```

Example: `setCommand(xenaon)` - x axis driver will become enabled

Notes: Hardware dependent. All Interinar drivers have Enable input. Other drivers may or may not feature this control.

3.2.4.13 STEP MODE CONTROL

Description: Changes step resolution.
Each axis has its own, separate set of commands.

Details: -

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

`xstep1`	BSD driver in Full Step mode
`xstep2`	BSD driver in Half Step mode
`xstep4`	BSD driver in 1/4 Step mode
`xstep16`	BSD driver in 1/16 Step mode
`ystep1`	
`ystep2`	
`ystep4`	
`ystep16`	
`zstep1`	

```
zstep2
zstep4
zstep16
ustep1
ustep2
ustep4
ustep16
```

Example: `setCommand(xstep4)` - x axis driver in ¼ Step mode

Notes: Hardware dependent. All Interinar drivers have Step Mode inputs. Other drivers may or may not feature this control.

3.2.4.14 AUXILIARY OUTPUTS

Description: This changes the logic level of auxiliary output AUX0..
Each axis has its own, separate pair of commands.

Details: see chapter 2.2.4 on pg 2-7

Syntax: `setCommand(code)`

Arguments: `code` – one of the following:

```
auxout1on
auxout1off
auxout2on
auxout2off
auxout3on
auxout3off
auxout4on
auxout4off
auxout5on
auxout5off
auxout6on
auxout6off
auxout7on
auxout7off
auxout8on
auxout8off
```

Example: `setCommand(auxout1on)` - set output AUX1 High (1)

3.2.5 FUNCTION getData

This function reads data from the controller.

Declaration:

```
static Byte[] MC02ENCOD.MCEncoder.getData(string code)
```

All these commands have the same format:

```
getData(code)
```

Arguments:

code – one from the following list

```
gethid
getxr0
getyr0
getzr0
getur0
getstat
```

Returns: reportIN - packet of 10 bytes
Byte 0 – Report ID – unique number ID of received packet
Byte 1 – Report Type – unique number describing requested report
Byte 2 to 9 - Data

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Report ID	Report Type	Data	Data	Data	Data	Data	Data	Data	Data

3.2.5.1 get Hardware ID

Description: Returns information about controller model, revision and firmware version.

Syntax: `getData(gethdid)`

Returns: valid data in 3 bytes, invalid data in all remaining bytes.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Report ID	Report Type	Data Hdw ID	Data Hdw Rev	Data Frmw Ver	Invalid	Invalid	Invalid	Invalid	Invalid

Byte #2	hardware ID	controller model
	0x10	MC-02A1
	0x20	MC-02A2
	0x40	MC-02A4
Byte #3	hardware revision	
Byte #4	firmware version	

3.2.5.2 get register R0

Description: Returns value stored in registers R0.
Each axis has its own, separate command.

Syntax: `getData(code)`

Arguments: code – one of the following:

getxr0	get value in register R0 on X axis
getyr0	get value in register R0 on Y axis
getzr0	get value in register R0 on Z axis
getur0	get value in register R0 on U axis

Returns: valid data in 3 bytes, invalid data in all remaining bytes.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Report ID	Report Type	Data Low	Data Mid	Data High	Invalid	Invalid	Invalid	Invalid	Invalid

Byte #2 - R0 Lower Byte
Byte #3 - R0 Middle Byte
Byte #4 - R0 Higher Byte

Example: `getData(getxr0)` - x axis value of R0

3.2.5.3 get Status

Description: Returns the STATUS 0 and STATUS 1 for all axes at once.

Syntax: `getData(getstat)`

Response: returned data contains information in 8 bytes

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Report ID	Report Type	Data X S0	Data X S1	Data Y S0	Data Y S1	Data Z S0	Data Z S1	Data U S0	Data U S1

Byte #2 - x axis Status 0
Byte #3 - x axis Status 1
Byte #4 - y axis Status 0
Byte #5 - y axis Status 1
Byte #6 - z axis Status 0
Byte #7 - z axis Status 1
Byte #8 - u axis Status 0
Byte #9 - u axis Status 1

Bit	Status 0	Status 1
0	INT at Stop	-EL
1	INT at Ramping Down Point	+EL
2	INT at External Start	ORG
3	Running	STP
4	R0 Pulse Counter Zero (Empty)	STA
5	R0 Pulse Counter Smaller than R5	-SD
6	Accelerating	+SD
7	Decelerating	Excitation Zero Position