PIC-SERVO SC (v.10)

Servo Motion Control I.C.

- Servo controller for D.C. motors (brush or brushless) with incremental encoder feedback
- Serial interface connects to RS232, RS485 or RS422 communications ports
- Firmware supports multi-drop RS485 network for multi-axis systems
- Position control, velocity control, trapezoidal profiling, plus Step & Direction inputs
- Path control mode supports CNC and other coordinated motion control applications
- Programmable P.I.D. control filter with optional current, output, and error limiting
- 32 bit position, velocity, acceleration; 16 bit P.I.D. gain values
- EEPROM configuration allows automatic start-up in stand-alone Step & Direction mode
- Single chip solution based on the PIC18F2331 series microcontroller

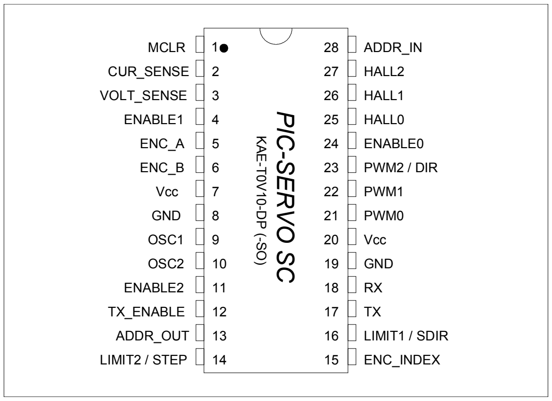

Figure 1 - ***PIC-SERVO SC***
28-pin, 0.3" DIP or SOIC

What's New in the PIC-SERVO SC (v. 10)

The primary difference between ***PIC-SERVO SC*** and previous versions of the ***PIC-SERVO*** is that the ***PIC-SERVO SC*** incorporates encoder counting within the ***PIC-SERVO SC*** chip itself, eliminating the need for the separate ***PIC-ENC*** chip. While the hardware pin-out is no longer compatible with earlier versions, most existing software applications will work with the ***PIC-SERVO SC*** without modification. In addition, the ***PIC-SERVO SC*** incorporates a host of new features:

- 2.5 MHz quadrature encoder counting rate.
- Position, velocity and acceleration can be modified on-the-fly.
- Smooth deceleration to goal points without any end-of-motion "creep" or jerk.
- Amplifier drive options include PWM & Direction, Antiphase PWM, and 3-Phase commutation. 3-Phase mode can also be used for driving two independent half-bridges.
- Optional Step & Direction inputs can be driven from stepper-style indexing systems.
- Improved limit switch options.
- Operating parameters may be saved in EEPROM for automatic servo on power-up.

Please see Section 5.7 for information regarding migration from earlier versions of the ***PIC-SERVO***.

••• CAUTION •••

The ***PIC-SERVO SC*** motion control I.C. is not warranted as a fail-safe device, and it should not be used in life support systems or in other devices where its failure or possible erratic operation could cause property damage, bodily injury or loss of life.

1.0 Overview

The ***PIC-SERVO SC*** is a single chip solution for implementing servo control of D.C. motors with incremental encoder feedback. The ***PIC-SERVO SC*** is a PIC18F2331 microcontroller programmed with a PID servo control filter, trapezoidal and velocity profiling and a serial command interface. It includes integral quadrature encoder counting for interfacing to your motor's encoder. Operation of the PID servo and of the encoder counting is described in Sections 4.2 and 4.3.

Three Output Modes

The ***PIC-SERVO SC*** has three different output modes for connecting to common types of motor amplifier circuits including: PWM and Direction, Antiphase PWM, and 3-Phase PWM output modes. The 3-Phase output mode works in conjunction with three hall effect sensor inputs and includes commutation logic (120 degree) for 3-Phase brushless motors. The 3-Phase output mode can also be used for driving two independent half-bridges for a conventional brush-type motor. Output modes are described in Section 4.5.

Five Operating Modes

This ***PIC-SERVO SC*** has 5 different operating modes to cover a wide variety of servo control applications including: raw PWM output mode, Velocity Profile mode, Trapezoidal Profile mode, Coordinated Motion Control (CMC) mode, and Step & Direction mode. These modes are described in Section 4.4.

Serial Communications Interface

Most applications will use the ***PIC-SERVO SC****'s* serial interface to send motion control commands. The serial interface, compatible with standard UART's, can be connected to an RS232 port (through the appropriate driver chip), or it also supports connection to a multi-drop RS485 network for controlling multiple motors over a single RS485 or RS422 port. The simple binary packet protocol maximizes the command data rate while ensuring reliable transmission of commands and status data.

With the full-duplex communications architecture, all commands are sent over a dedicated command line, but multiple ***PIC-SERVO*** chips can respond over a separate shared response line. Unique device addresses are dynamically assigned, eliminating the need for setting dip-switches. Serial communications and address initialization are described in Section 4.1.

EEPROM Parameter Storage

The ***PIC-SERVO SC*** also has non-volatile (EEPROM) data storage for retaining servo gains and other operating parameters. This enables the chip to be used in a stand-alone mode (no serial interface) where on power-up, it is ready to accept Step and Direction input signals. This makes it an ideal servo controller upgrade for systems designed for stepper motors. EEPROM parameter storage is described in Section 4.6.

2.0 Pin Description and Packaging

The ***PIC-SERVO SC*** comes in a 28 pin, 0.3" DIP or an SOIC package. The device operates from a +5v supply and is compatible with TTL and CMOS logic. Please refer to the PIC18F2331 data sheet from Microchip (*www.microchip.com*) for complete electrical and physical specifications.

Pin	Symbol	Description
1	MCLR	Reset pin, active low. Connects directly to Vcc for automatic reset on power-up. When the ***PIC-SERVO SC*** is operated in stand-alone mode for Step & Direction systems, this pin can be used to enable/disable the servo system.
2	CUR_SENSE	Analog input for current sensing or for use as a general analog input. (0 - +5v). (See Section 4.7.)
3	VOLT_SENSE	Analog input for sensing the motor supply voltage (connected through a voltage divider resistor network). It is used to detect undervoltage and overvoltage conditions. The input voltage must be in the range of 0.9v to 4.5v for normal operation. (See Section 4.7)
4	ENABLE1	Active HIGH output for enabling half-bridge #1 when in 3-Phase mode
5	ENC_A	Input pin connects to Channel A of your encoder. (Input has digital noise filter)
6	ENC_B	Input pin connects to Channel B of your encoder. (Input has digital noise filter)
7	Vcc	Supply voltage - connect to +5v DC.
8	GND	Ground connection.
9	OSC1	Connects directly to a 10 MHz clock source or to one side of a 10 MHz crystal. An internal phase-lock loop boosts the actual operating frequency to 40 MHz. See Microchip documentation for details of crystal connections.
10	OSC2	Connects to the other side of a 10 MHz crystal. N.C. if a clock source is used.
11	ENABLE2	Active HIGH output for enabling half-bridge #2 when in 3-Phase mode
12	TX_ENABLE	This output can be connected to the enable pin of an RS485 driver to enable output over a shared response line. Not used if a point-to-point serial connection (like RS232) is used.
13	ADDR_OUT	Output pin which powers up in the HIGH state, and is lowered when the ***PIC-SERVO***'s address is programmed. Normally connected to ADDR_IN of an adjacent controller. When in Step & Direction mode, this pin is raised on a servo fault condition and can be used as a fault detection flag.
14	LIMIT2 / STEP	REV limit switch input, except when in Step & Direction mode, it is the Step pulse input.
15	ENC_INDEX	Input pin connects to the Index pulse output of your encoder.
16	LIMIT1 / SDIR	FWD limit switch input, except when in Step & Direction mode, it is the Direction input. (0 = FWD, 1 = REV).
17	TX	Serial transmit output. Connects to the transmit input of an RS485 or an RS232 driver chip.
18	RX	Serial receive input. Connects to the receive output of an RS485 or an RS232 driver chip.
19	GND	Ground connection.
20	Vcc	Supply voltage - connect to +5v D.C.
21	PWM0	PWM output pin for PWM & Direction and antiphase PWM modes, and also used for driving half-bridge #0 when in 3-phase mode.
22	PWM1	PWM output for driving half-bridge #1 when in 3-Phase mode.
23	PWM2 / DIR	PWM output for driving half-bridge #2 when in 3-Phase mode. In PWM & Direction mode, it is used as the Direction bit (0 = FWD, 1 = REV)
24	ENABLE0	Active HIGH output for enabling the amplifier when in PWM & Direction and antiphase PWM modes, and enables half-bridge #0 when in 3-Phase mode
25,26, 27	HALL0, HALL1, HALL2	Hall effect sensor input pins used for commutation when in 3-Phase mode. These pins have internal 20K pull-up resistors.
28	ADDR_IN	This pin must be pulled low to enable communications. Normally tied to the ADDR_OUT pin of the previous ***PIC-SERVO*** on the same RS485 network. This pin has an internal 20K pull-up resistor.

Ordering Information

Part Number	***Description***
KAE-T0V10-DP	***PIC-SERVO SC*** I.C., version 10, 0.3" wide DIP package
KAE-T0V10-SO	***PIC-SERVO SC*** I.C. , version 10, SOIC package

3.0 Electrical & Timing Specifications

All timing specifications are based on using a 10 MHz clock source or crystal for the ***PIC-SERVO SC*** (boosted internally to 40 MHz). Although other clock frequencies may be used, we do not recommend this because there are some subtle internal timing considerations which have not been tested at other clock frequencies.

The ***PIC-SERVO SC*** is a 5v device with CMOS inputs compatible with either CMOS or TTL logic levels. The CMOS outputs can drive or sink up to 20 ma each. Please refer to the PIC18F2331 data sheet from Microchip (*www.microchip.com*) for complete electrical specifications.

The following timing rates are of interest:

Servo rate:	1953.125 Hz
Serial baud rate:	9,600 - 230,400 baud
	(faster rates may be possible but are untested)
PWM frequency:	19,531.25 Hz (fixed)
PWM resolution:	10 bit
Encoder counting rate:	2.5 MHz (max.)
Max step pulse rate:	100 KHz (see Section 4.4.6)
Max Command rate:	1000 Hz max. (approx.)

The rate at which commands can be sent to a ***PIC-SERVO*** controller is dependent on the number of bytes in the command and on the number of bytes returned as status data, both of which are variable. Please see Sections 4.1 and 5.1 for information on calculating communication times for specific commands

4.0 Theory of Operation

4.1 Communications & Initialization

NMC Communication Protocol

The ***PIC-SERVO*** uses the same *full-duplex*, RS232 or RS485 (4-wire) based NMC (*Networked Modular Control*) communication protocol as used by the ***PIC-STEP*** and the ***PIC-I/O*** controllers. It is a strict master/slave protocol, where command packets are sent to a controller module by the host computer, and a status packet is returned by the module. The default baud rate is 19,200, but it may be changed at any time to up to 230,400. The communication protocol uses 8 data bits, 1 start bit, 1 stop bit and no parity.

Command packets are transmitted by the host over a dedicated command line. Status packets are received over a *separate* status line which is shared by all of the modules on the network. Because the host does not have to share the command line, the host communications port can be a standard RS232 port with a simple RS232 to RS485 (or RS422) signal level converter[*]. The slave ports, however, must be able to disable their transmitters to prevent data collisions over the shared status line. Therefore, all NMC compatible controllers provide an TX_ENABLE output used for enabling or disabling an RS485 transmitter. Please refer to the sample schematic in Section 7 and to Figure 2 below.

[*] If only a single ***PIC-SERVO*** controller is used, the ***PIC-SERVO***'s communications port can be operated as an RS232 port, with TX and RX connected to an RS232 transceiver rather than to an RS485 transceiver. In this case, the TX_ENABLE output is not used and may be left open.

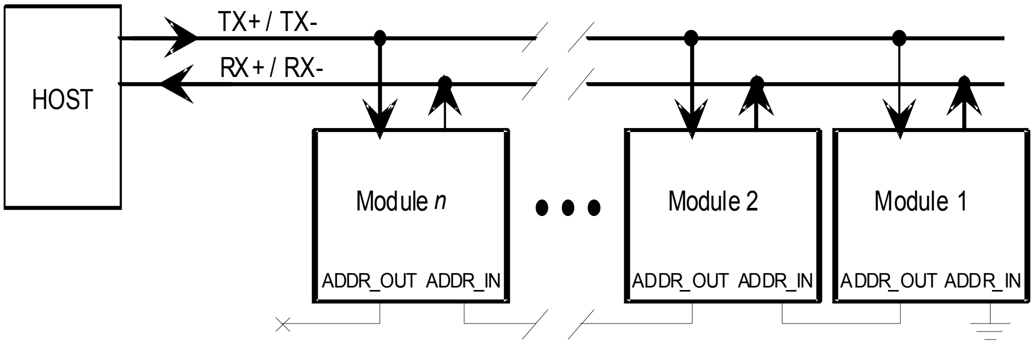

Figure 2 - Connecting Multiple Controller Modules

The command packets have the following structure:

Header byte (always 0xAA)
Module Address byte (0 - 255)
Command byte
Additional Data bytes (0 - 15 bytes)
Checksum byte (8-bit sum of the Module Address byte through the last additional data byte)

The Header byte is used to signal the beginning of a command packet. When waiting for a new command, each module will ignore any incoming data until it sees a Header byte.

The Module Address byte is the address of the target module. The address can be an individual address, or the *group* address for the module. (See *Group Commands* below.)

The Command byte is broken up into an upper nibble (4 bits) and lower nibble (4 bits). The lower nibble contains the command value (0 - 15), and the upper nibble contains the number of additional data bytes required for that command (0 - 15). It is up to the host to insure that the upper nibble matches the number of additional data bytes actually sent.

The Additional Data bytes contain the specific data which may be required for a particular command. Many commands have a “control” or “mode” byte in addition to other parameters required for the command. Some commands require no additional data. It is up to the host to make sure that the proper number of additional data bytes is sent for a particular command, and that the upper nibble of the command byte is equal to this number.

Once a module receives a complete packet, and the Address byte matches its address, it will verify the checksum and immediately (within 0.51 milliseconds) begin to process the command. If there is a checksum error in the command packet or any other sort of communications error (framing, overrun), the command will not be executed, but a status packet will still be returned. If there are no errors, the command will then be executed and a status packet returned. (Note that motion commands will initiate the motion and return a status packet immediately without waiting for the motion to finish.)

The status packets have the following structure:

Status byte
Additional Status Data bytes (programmable)
Checksum byte (8-bit sum of all the bytes above)

The Status byte contains basic information about the state of the module, including whether or not the previous command had a checksum error. The specific bit definitions for the Status byte are in Section 5.3 below.

The number Additional Status Data bytes is programmable, and may contain information such as motor position, A/D values, or the module type and version numbers. Exactly which data is included in these Additional Status Bytes can be programmed using the Define Status or Read Status commands. On power-up or reset, each NMC module defaults to sending only the Status byte and Checksum byte, with no additional status data.

A command sent to a ***PIC-SERVO*** controller is stored in an internal buffer until the end of the current servo cycle (0.51 millisec. max.), when it is then executed and a status packet is returned. Therefore there will be a maximum delay of 0.51 millisec. (0.25 millisec. avg.) between when the last byte of the command is received and when the first byte of the status packet is sent. No new command should be sent until the status packet is returned to prevent overwriting the command data buffer and to prevent collisions on the status line. If, however, the host does send *any* data before a status packet is received, all slaves on the network will disable any status data transmission in progress and listen to the new command from the host. This insures that the host can always command the attention of all slaves on the network.

The Command Reference Section 5.1 below describes the data contained in the command packets and status packets.

Addressing

When multiple modules are connected to the same NMC network, they must be assigned unique addresses. This is done through the use of the ADDR_IN and ADDR_OUT signals on each NMC compatible controller. The ADDR_OUT signal from one controller is daisy-chained to the ADDR_IN signal of the adjacent controller on the network. Customarily, the ADDR_IN pin of the controller furthest from the host is tied to GND, and the ADDR_OUT signal of the controller closest to the host is left open. (See the Figure 2 above).

Unique addresses are assigned using the following procedure:

1. On power-up, all modules assume a default address of 0x00, and each will set its ADDR_OUT signal HIGH. Furthermore, a module's communications will be disabled completely until its ADDR_IN signal goes LOW. If the ADDR_OUT and ADDR_IN signals are daisy-chained as described above, all modules will be disabled except for the module furthest from the host.
2. The host starts by sending a Set Address command to module 0, changing its address to a value of 1. A side affect of the Set Address command is that the module will lower the its ADDR_OUT signal.
3. At this point, the next module in line is enabled with an address of 0. The host then sends a command to module 0 to change its address to a value of 2.
4. This process is continued until all modules have been assigned unique addresses.

Initialization of the addresses is performed by the host each time the NMC network is powered up or reset. The host can also use this mechanism to verify that the proper number of modules are present, and that their types match those expected for a particular application.

Once addresses are set, all other operations can be executed.

Group Commands

Each NMC controller module actually has two addresses: an individual address and a group address. On power-up or reset, the individual address defaults to 0x00 and the group address defaults to 0xFF. Both the individual address and the group address are set with the same Set Address Command. Individual addresses can have any value between 0 and 255, but group addresses are restricted to values between 128 and 255.

The purpose of the group address it to be able to send a single command (such as Start Motion) to a several controllers at the same time. While the individual addresses of all controllers must be unique, a group of controllers can share a common group address. When a command packet is sent over the NMC network to a group address, all modules with a matching group address will execute the command.

The issue of which module will send a status packet in response to a group command is resolved with the distinction between group *members* and group *leaders*. When the group address for a module is set, the Set Address command will also specify if the module is to be the *leader* or a *member* of that group. If a module is a member of its group and it receives a group command (*i.e.*, a command sent to its group address), it will execute the command but *not* send back a status packet. If a module is the leader of its group and it receives group command, it *will* send back a status packet in addition to executing the command. (The status packet is just the same as one sent in response to an individually addressed command.)

For any group of modules sharing the same group address, only one module should be declared the group leader.

In certain instances (as when changing the Baud rate for all modules on the network), is necessary to send a command to a group without a group leader. In this case, no status will be coming back from any controllers, and the host should wait for at least 0.51 milliseconds before sending another command to keep from overwriting the previous command. If you need to change the baud rate, it is best, when initially setting the addresses, to leave the group address for all modules at 0xFF with no group leaders. After changing the baud rate, you can then re-define individual and group addresses as needed.

Universal Reset Address

For most applications, group commands are not needed and the group address for all modules is left at 0xFF. If, however, the modules are split up into several groups with different group addresses, sending a single Reset command to reset all controllers at once becomes problematic. To address this issue, the ***PIC-SERVO SC*** will always execute a Hard Reset[*] command sent to the address 0xFF, independent of the value of the module's group address. Note that when a Reset command is sent, no status packet will be returned.

Network Initialization

The previous subsections have hinted at various operations required for network initialization. Here is a specific list of the actions which should be taken on power-up, or after a network-wide reset :

1. Set the host baud communications port to 19,200 Baud, 1 start bit, 1 stop bit, no parity.
2. Send out a string of 20 null bytes (0x00) to fill up any partially filled command buffers. Wait for at least 1 millisecond, and then flush any incoming bytes from the host's receive buffer.

[*] There are two versions of the Hard Reset command - a simple Reset and one where data is saved in EEPROM prior to reset. The universal reset address of 0xFF only works with a simple Reset command.

3. Use the Set Address command, as described in Section 5.2, to assign unique individual addresses to each module. At this point, set all group addresses to 0xFF, and do not declare any group leaders. (If you are not going to change the Baud rate from the default 19,200, you can set both the individual and group addresses at this time.)
4. Verify that the number of modules found matches the number expected.
5. Different NMC controller modules will have different type numbers and different version numbers (***PIC-SERVO*** = type 0). Use the Read Status command to read the type and version numbers for each module and verify that they match the types and versions expected.
6. Send a Set Baud command to the group address 0xFF to change the baud rate to the desired value. No status will be returned. (Only required if using other than 19,200 Baud.)
7. Change the host's Baud rate to match the rate just specified.
8. Poll each of the individual modules (using a No Op command) to verify that all modules are operating properly at the new Baud rate.
9. Use the Set Address command to assign any group addresses as needed.

At this point you are ready to send any module specific initialization commands to the individual modules and begin operation. Note that at any time, you may use the Set Address command to re-assign individual or group addresses.

4.2 Incremental Encoder Counting

A typical two-channel incremental or *quadrature* encoder puts out two 50% duty cycle square waves either +90 degrees or -90 degrees out of phase, depending on which direction the motor is rotating, as shown in Figure 3. Each channel produces one square-wave pulse per encoder line. Therefore, two channel encoders have a fundamental resolution of four times the number of lines on the encoder because each edge of each square wave provides position information.

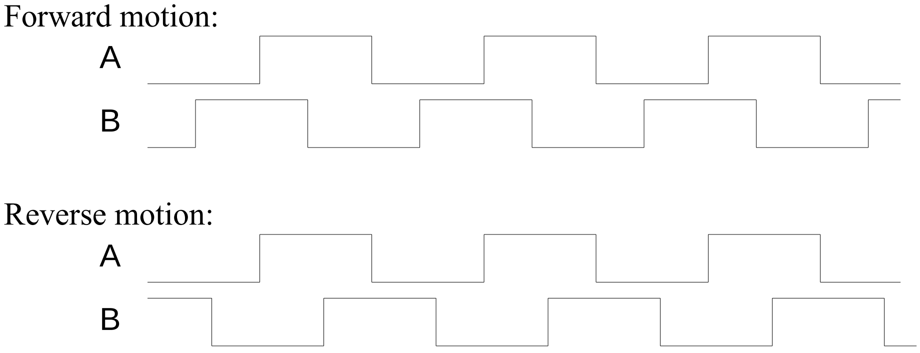

Figure 3 - Encoder Signals

For example, a 500 line encoder will produce 4 signal edges per line (2 on channel A, 2 on channel B), for a total of 2000 edges per revolution. The ***PIC-SERVO SC***'s internal circuitry takes care of decoding the direction information and counting all four edges per encoder line. The encoder counter produces a 32 bit position value which is used for the servo control operation. This position can also be read as part of the status packet returned with every command.

4.3 PID Servo Control

In general, when in position or velocity mode, the motor is controlled by a servo loop which once every servo tick (1953.125 times/sec) looks at the current position of the motor, compares it to where the motor should be, and then uses a “control filter” to calculate an output which will cause the difference in positions, or the “position error” to become smaller. Two sets of parameters will govern the motion of the motor: the desired trajectory parameters (goal position, velocity, acceleration) which are described in Section 4.4, and the control filter parameters discussed here.

The control filter used by the ***PIC-SERVO*** is a “proportional-integral-derivative”, or PID filter. The output to the motor amplifier is the sum of three components: one proportional to the position error providing most of the error correction, one proportional the *change* in the position error which provides a stabilizing damping effect, and one proportional to the accumulated position error which helps to cancel out any long-term error, or “steady state” error.

The PID control filter, operating on the command position and the actual position each servo tick, produces an output calculated as follows:

$$\mathit{output} = \mathit{Kp} \times \mathit{pos\_error} \; - \; \mathit{Kd} \times (\mathit{pos\_error} - \mathit{prev\_pos\_error}) \; + \; \mathit{Ki} \times \mathit{integral\_error}$$

The term *pos_error* is simply the current command position minus the actual position. The *prev_pos_error* is the position error from the previous servo tick. *Kp*, *Ki* and *Kd* are the 16 bit servo gains which will be programmed to optimize performance for your particular motor.

The *integral_error* is the running sum of *pos_error* divided by 256. To keep from growing a potentially huge *integral_error*, the running sum is bounded by a 16 bit user specified integration limit, IL. (Note that by temporarily setting the integration limit to 0, the user can zero out the accumulated running sum.)

The actual PWM output value (0-255) and direction bit are given by:

$$\mathit{PWM} = \min[\, \mathrm{abs}(\mathit{output}/256) + \mathit{dead\_band} \;, \mathit{output\_limit} \,] - \mathit{current\_limit\_adjustment}$$

$$\mathit{DIR} = 0 \text{ if output} > 0, \qquad \mathit{DIR} = 1 \text{ if output} < 0$$

The parameter *dead_band* is an 8-bit offset used to compensate for static friction or a dead band region in the amplifier.

First note that the scaled *PWM* output is limited by an 8-bit user defined *output_limit*. For example, if you are using a 12v motor powered by 24v, you would want to set the *output_limit* to 255/2, or 127. Also note that the final PWM value is reduced by a *current_limit_adjustment*. This value is explained in Section 4.7 below.

The *PWM* signal is a 19.53 KHz square wave of varying duty cycle with a *PWM* value of 255 corresponding to 100% and a value of 0 corresponding to 0%. As explained here, the *PWM* signal is derived from an 8-bit value. The internal calculations, however, actually provide and additional 2 bits of resolution for a final 10 bit *PWM* resolution.

An additional control parameter is the user specified 16 bit *position error limit*. If abs(*pos_error*) becomes larger than this limit, the position servo will be disabled. This is useful for disabling the servo automatically upon a collision or stall condition. Also, the POS_ERROR bit in the status byte will be set on a position error condition. (This condition can also be used for homing the motor by intentionally running it up against a limit stop.)

Selection of the optimal PID control parameters can be done analytically, but more typically, they are chosen through experimentation. As a first cut, the following procedure may be used:

1. First set the position gain, *Kp*, and the integral gain, *Ki*, to 0. Keep increasing the derivative gain, *Kd*, until the motor starts to hum, and then back off a little bit. The motor shaft should feel more sluggish as the value for *Kd* is increased.
2. With *Kd* set at this maximal value, start increasing *Kp* and commanding test motions until the motor starts to overshoot the goal, then back off a little. Test motions should be small motions with very large acceleration and velocity. This will cause the trapezoidal profiling to jump to goal position in a single tick, giving the true step response of the motor.
3. Depending on the dynamics of your system, the motor may have a steady state error with *Kp* and *Kd* set as above. If this is the case, first set a value for the integration limit *IL* of 16000 and then start increasing the value of *Ki* until the steady state error is reduced to an acceptable level within an acceptable time. Increasing *Ki* will typically introduce some overshoot in the position. The best value for *Kp* will be some compromise between overshoot and settling time.
4. Finally, reduce the value of *IL* to the minimum value which will still cancel out any steady state error.

There is one final servo filter parameter to discuss. The servo calculations above are executed at a rate of 1.953 KHz. For systems with a combination of a large inertia, little inherent damping and limited encoder resolution, it may be difficult to get sufficient damping at low speeds because the digitization noise with very large values of *Kd* will cause the servo to hum or vibrate. To decrease the digitization noise, the *servo rate divisor* (*SRD*) parameter is used to calculate how many servo cycles elapse between calculating the *pos_error* and the *prev_pos_error* used in the damping term of the PID filter. By default, *SRD* = 1, and *prev_pos_error* is equal to the *pos_error* of <u>one</u> cycle earlier. If, however, we set *SRD* = 3, *prev_pos_error* would equal the *pos_error* of <u>three</u> cycles earlier. Increasing the time difference between these values effectively averages the derivative error term and reduces the digitization noise.

In summary, we have a total of eight control filter parameters: Position Gain (*Kp*), Derivative Gain (*Kd*), Integral Gain (*Ki*), Integration Limit (*IL*), Output Limit (*OL*), Current Limit (*CL*), Position Error Limit (*EL*) and the Servo Rate Divisor (*SRD*). The details of programming these values appear in the Section 5.2 under the description of the Set Gain command.

4.4 Operating Modes

The ***PIC-SERVO SC*** has several layers of control and allows you to operate at any of these layers as required by your application. Generally, each layer of control sends commands to layer of control below it to create the desired motion.

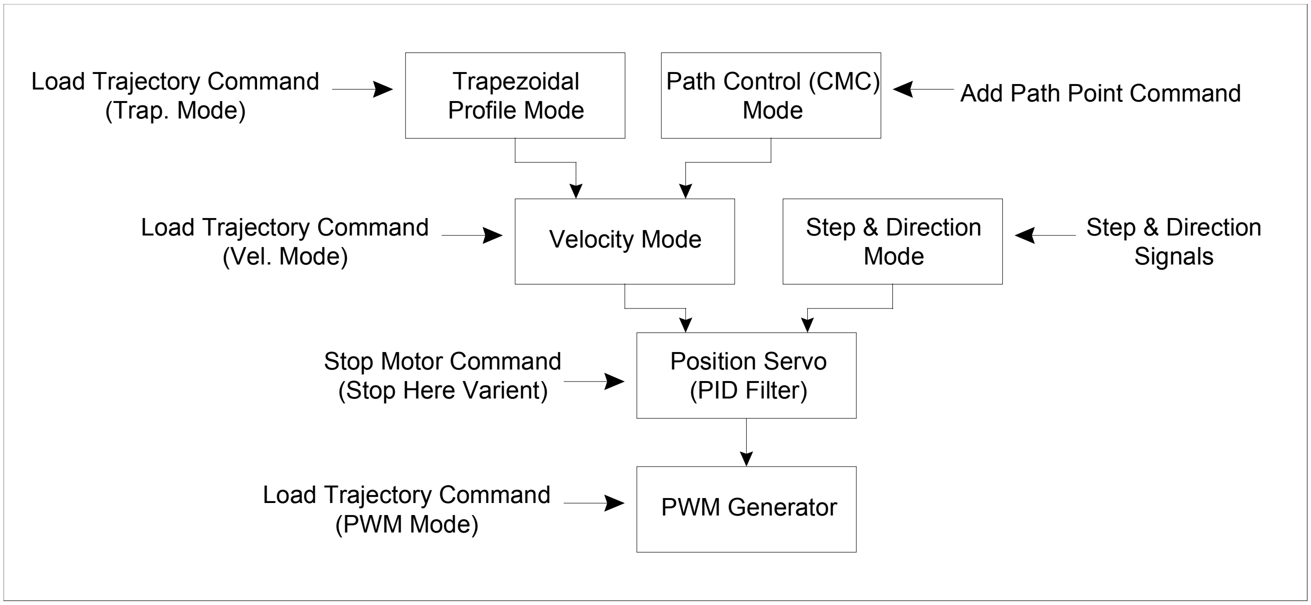

Figure 4 - Levels of control within the ***PIC-SERVO SC***

4.4.1 PWM Mode

The lowest layer of control is raw PWM mode, where the user can specify the PWM output signal sent directly to the amplifier. When the ***PIC-SERVO SC*** normally powers up, it enters PWM mode with the PWM value set to zero, and the servo and profiling modes are turned off. PWM mode is also entered when the position servo is terminated automatically by a loss of power, excess position error, or turned off by the Stop Motor command.

During experimentation or startup, it may be useful to send a non-zero output directly to the amplifier using the Load Trajectory command. Note that in specifying a PWM value directly, the current limiting of an external amplifier is still be performed, but the PWM output limit is ignored.

When in PWM mode, even though the position servo is disabled, the current command position is updated continually to match the actual position of the motor. Thus, when position mode is entered, there will be no abrupt jump in the motor's position. Also while the position servo is disabled, the command velocity is continually updated to match the actual velocity of motor. Thus, when velocity mode is entered, there will be no discontinuity in the motor's velocity.

4.4.2 Position Servo Mode

The next layer of control is the position servo mode where the PID servo calculates PWM values to actively drive the motor to the *current command position*. When the motor is stopped, the current command position is a constant value. When a trapezoidal position command terminates, when a Stop Motor Abrupt command is issued, or a Stop Abrupt condition occurs while homing, all motion profiling is terminated and the PID control filter servos to the current command position.

The ***PIC-SERVO SC*** gives the user direct access to the current command position through the "Stop Here" variant of the Stop Motor command. In other words, the user can specify a current command position and the servo will drive the motor to that position with no motion profiling to smooth the trajectory of the motion.

4.4.3 Velocity Mode
Velocity mode allows the user to drive the motor at a constant velocity and also to transition smoothly from one velocity to another with a specific rate of acceleration. The velocity mode operates by incrementing the current command position by some amount every servo cycle (1.953 KHz). When traveling at a constant velocity, the current command position is simply incremented by the *current velocity value* every servo cycle. The underlying position servo then drives the motor to this constantly changing current command position.

When the goal velocity value is changed, rather than abruptly jerking to the new velocity, the current velocity value is incremented (or decremented) by some amount once per servo cycle until the goal velocity is reached. The amount incremented (or decremented) is the *acceleration value*.

The goal velocity and the acceleration value are set using the Load Trajectory command. Goal velocities and accelerations can be changed at any time, independent of the current mode of operation.

4.4.4 Trapezoidal Position Mode
Trapezoidal position mode allows the user to specify a goal position, a maximum slew velocity, and an acceleration value. When the motion starts, the motor will accelerate up to the maximum velocity, slew at that constant velocity until it nears the goal position, and then decelerated to a stop exactly at the specified goal position. A plot of *velocity v. time* will produce a trapezoidal shaped profile. For very short motions, or if a very low acceleration is specified, the motor may never reach the maximum velocity before beginning to decelerate, thus producing a triangular shaped profile instead. The trapezoidal position mode operates by using the velocity mode layer of control beneath it. When a motion starts, it simply issues a velocity command to accelerate up to the maximum velocity. Once the motion starts, however, it constantly calculates the point at which the motor must begin decelerating in order to exactly stop at the goal position. When the motor reaches this point, it sets the goal velocity to zero and the motor decelerated to a stop.

The trapezoidal position mode is relatively sophisticated in that allows the user to change the goal position, the maximum velocity or the acceleration at any time. For instance, if the maximum velocity is increased while slewing, the motor will speed up to the new velocity, and then begin decelerating at the proper time to end up stopped at the goal position.

If the goal position, velocity or acceleration are changed in the middle of a motion, it is possible that the motor will no longer be able to stop at the goal position in time. In this case, the motor will overshoot the goal position, smoothly reverse direction, and then decelerate to the goal position. The motion will still adhere to the specified velocity and acceleration limits.

Trapezoidal position motions are also specified using the Load Trajectory command. You can issue a command to enter trapezoidal position mode at any time and while in any other operating mode.

4.4.5 Path Control Mode
Path control mode, or Coordinated Motion Control (CMC) mode, is a special mode which allows the host computer to easily coordinate the motion of several ***PIC-SERVO SC***'s. The fundamental feature of this mode is a *path point buffer* which stores a series of closely spaced motor goal points. Coordination of multiple axes happens as follows:

1. The host (PC, etc.) calculates a series of closely spaced path points for one or more motors. The idea is that each motor will move from one path point position to the next at a fixed time interval. The multi-axis path is followed as all motors move from one path point to then next with the exact same timing.
2. The host then downloads the sets path points to the individual ***PIC-SERVO SC*** controllers.
3. With a single *group* command, the host then starts all axes moving at the same time. Because of the accurate crystal controlled timing of the ***PIC-SERVO SC***, each motor will move synchronously from one path point position to the next.

There are several important features to note about the path control mode. First, the path point buffer can be continuously reloaded with new points while the motor is moving. The path point buffer can hold up to about 4 seconds worth of path data, but if a desired motion is longer than that, new points can be added to the buffer *while moving* to create continuous motions of unlimited length. For some applications, it is desirable to only keep a small amount of data in the path point buffers at any time to allow the host to change the path on-the-fly with a minimal delay.

The second important feature is that when the ***PIC-SERVO SC*** moves from path point to path point, it doesn't simply jerk from one point to the next - it creates a series of intermediate path points so that it moves to the next path point with a constant velocity. Path points are calculated as the positions where the motor should be at either 30, 60 or 120 Hz intervals. If, for example, the host created a set of path points at 60 Hz intervals, the ***PIC-SERVO SC*** would calculate intermediate path points at 1953.125 Hz intervals. The effect of this intermediate smoothing with a multi-axis system is that all of the axes will move from one path point to the next along a straight-line segment. This internal smoothing allows path points calculated by the host to be spaced much more widely apart, and requires much less data to be sent to the ***PIC-SERVO SC***.

The obvious question in specifying path points at a relatively low frequency is: "how accurately will the motors be able to follow my ideal path". In fact, for typical applications, this error is very small. For example, let's say we have an X-Y table with two motors and we wish to move along a circular path. If we were to move along a 1.0" diameter circle at a speed of 1.0" per second, and we approximated the circle by a series of straight line segments spaced at 30 Hz intervals, the maximum error between the straight lines and a true circle would be 0.00028". If we switched to 60 Hz intervals, the maximum error would drop to 0.000069". Slowing down the motion or increasing the radius of curvature would further increase the accuracy.

Path points are downloaded are downloaded to the ***PIC-SERVO SC*** using the Add Path Points command. With a *single* Add Path Points command, up to 7 path points can be downloaded to a controller. Once all of the controllers have been loaded with their path point data, the Add Path Point command, this time with no path point data, is issued to the entire group of controllers using a group command. This causes all controllers to start executing the path at the same time.

While the path is running, it is useful to poll at least one of the controllers and have the number of points remaining in the path point buffer returned in the status packet. The host can then add more path points as needed, and also avoid overflowing the path point buffer, which can hold a maximum of 128 path points.

When a path point is downloaded into the path point buffer, the position data also includes a bit which specifies whether that path point should be reached in 33 milliseconds (30 Hz) or in 16.7 milliseconds (60 Hz). This allows the host, in the middle of a move, to use more closely spaced when moving along a tightly radiused curve. While normally paths are run at 30 or 60 Hz, there is also a fast path mode bit (which can be set using the I/O Control command) which allows the path to run at 60 or 120 Hz instead.

The number of axes of motion which can be coordinated is limited by the speed of the serial port connection and by the path point rate. At a Baud rate of 115,200, it is possible to coordinate the motions of up to about 16 axes when using a 30 Hz path point rate. At 60 or 120 Hz, the number of axes drops to 8 or 4 respectively.

4.4.6 Step and Direction Input Mode

Step and direction input mode allows step pulses and a direction signal from a stepper-style indexing system to be used to modify the current command position. The STEP input generates an interrupt in the ***PIC-SERVO SC***, which then looks at the direction (SDIR) signal and increments (SDIR = 0) or decrements (SDIR = 1) the step count accordingly. Every servo cycle, the step count is multiplied by a *step multiplier* parameter (specified with the Set Gains command) and then added to the current command position. Thus, if no motion profile is being executed, the motor simply tracks the command position as it is modified by the step and direction inputs.

Step and Direction mode is enabled using the I/O Control command. Note that because the pins used for step and direction inputs are also used for limit switch inputs, if you enable Step and Direction mode, you should not attempt to use any other limit switch functions such as homing or enabling automatic stopping when a limit switch is hit.

You should note that all other operations of the ***PIC-SERVO SC***, such as polling the controllers for the current motor position, are still possible while in step and direction mode. In fact, motion profiles can be executed while in step and direction mode and the step input adjustments are simply superimposed upon the profiled motion. We do not, however, recommend executing motion profiles in step and direction mode because it will significantly reduce the step rate possible without encountering servo overrun errors.

To be able to use a servo motor and the ***PIC-SERVO SC*** as a replacement for a stepper motor and driver, it is useful to have the controller power-up in a state ready to receive step and direction signals without requiring a host to initialize gains and other operating parameters. Therefore, the ***PIC-SERVO SC*** utilizes its internal EEPROM to hold startup information. For stand-alone step and direction operation, the EEPROM should be configured to power-up with the position servo enabled and in Step and Direction mode. (See Section 4.6 below.)

Lastly, when operated as a stand-alone step and direction controller, the ***PIC-SERVO SC*** needs to be enabled or disabled by the indexing system, and it also needs to report any servo fault conditions to the indexing system. To enable or disable the ***PIC-SERVO SC***, the MCLR pin can be used - pulled low to disable or set high to enable. Also when in Step and Direction mode (and only in Step and Direction mode), the ADDR_OUT pin is raised whenever there is a servo fault condition and the position servo is disabled. If the ***PIC-SERVO SC***'s EEPROM is set to power-up in stand-alone mode, the ADDR_OUT pin will be lowered automatically when it is ready to receive step and direction inputs.

Step and Direction Signal Timing

The step pulse should be a rising pulse with a duration of at least 0.2 microsecond. The direction signal should be stable for at least 6 microseconds from the rising edge of the step pulse.

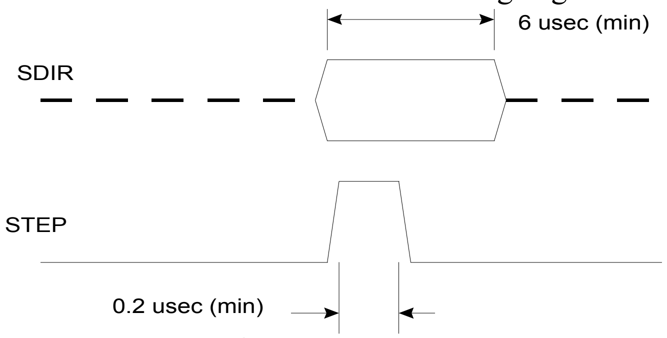

Figure 5 - Step and Direction Timing

If you have an indexing system that does not hold the SDIR signal stable for at least 6 microseconds, you can fix it by running the direction signal into the D input of a D-style flip-flop and clocking the flip-flop with the rising edge of the step pulse as shown below.

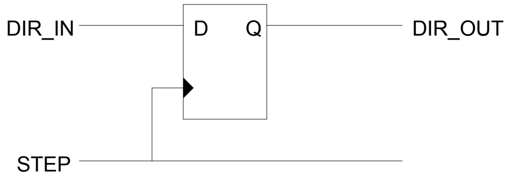

Figure 6 - Cleaning up Messy Direction Signals

4.4.7 Specifying Positions, Velocities and Accelerations

The position, velocity and acceleration are programmed as 32 bit quantities in units of encoder counts and servo ticks. For example, a velocity of one revolution per second of a motor with a 500 line encoder (2000 counts/rev) at a servo tick time of 0.512 msec. would correspond to a velocity of 1.0240 counts/tick. Velocities and accelerations use the lower 16 bits as a fractional component so that the actual programmed velocity would be 1.024 x 2^{16} or 67,109. An acceleration of 4 rev/sec/sec (which would bring us up to the desired speed in 1/4 sec) would be 0.0021 counts/tick/tick; with the lower 16 bits the fractional component, this would be programmed as 0.0021 x 2^{16} or 137. Position is programmed as a signed 32 bit quantity with no fractional component.

4.5 Output Modes for Driving Amplifiers

The ***PIC-SERVO SC*** can be configured to drive several different types of amplifiers. The I/O Control command is used to specify which mode is used. By default, the ***PIC-SERVO SC*** powers up or resets to PWM and Direction mode. If your amplifier uses something other than the default PWM & Direction mode, you should make sure to set the output mode *before* you enable the amplifier. Otherwise, you may get unexpected motion of your motor.

4.5.1 PWM and Direction Mode (default)

In this default output mode, the PWM0 pin (21) outputs a 19.53 KHz square wave with a duty cycle (active High) proportional to the magnitude of the PID filter output. This PWM signal has 10 bit resolution, which are the 8 bits calculated in Section 4.3 plus two lower order bits to increase the resolution. The DIR signal on pin 23 tells the amplifier whether the output should be positive (DIR = 0) or negative (DIR = 1). The ENABLE0 signal (active High) on pin 24 is used to enable or disable the amplifier.

4.5.2 Antiphase PWM Mode

Antiphase PWM mode also uses the PWM0 pin (21) similar to the PWM and Direction mode above, except that it instead puts out a 50% duty cycle when no current it driven to the motor. A 0% duty cycle corresponds to a full negative output, and a 100% duty cycle produces a full positive output. The PWM signal has a +/- 10 bit resolution.

Amplifiers using this type of output are less efficient, but they are more linear near zero output levels. This type of output signal is also easily filtered to produce a +/- 10v drive signal required by many servo amplifier modules. (See example in Section 6.2)

Even though the DIR signal (pin 23) may not be needed by the amplifier, it is still set to reflect the direction of the output (0 = FWD, 1 = REV).

Because the ***PIC-SERVO SC*** will power-up using the default PWM output of 0% duty cycle (full negative output!), you must make sure that the ENABLE0 signal is used to disable your amplifier on power-up. This will allow your software to set the Antiphase PWM mode prior to enabling the amplifier. (Note that you can use the EEPROM to specify the Antiphase mode be restored automatically on any type of power-up or reset.)

Once Antiphase PWM mode has been set, a calculated or specified PWM output of 0 will be translated to a 50% duty cycle output at the pin PWM0.

4.5.3 Three-Phase Output Mode

The 3-Phase output mode is used for the commutation of three-phase brushless motors with hall-effect (or similar) commutation sensors (120 degree configuration). Nine pins are used for the 3-Phase brushless mode: HALL0, HALL1, and HALL2 hall sensor inputs; PWM0, PWM1 and PWM2 output pins for chopping each of three half-bridge drivers; and ENABLE0, ENABLE1 and ENABLE2 output pins for enabling or disabling each of the three half-bridge drivers. The three HALL inputs have internal 20K pull-up resistors, but in practice, additional 4.7K pull-up resistors should be added for noise immunity if your motor's hall sensors have open collector outputs.

The state of the three hall sensor inputs will determine the which of the PWM and ENABLE outputs are active according to the tables below:

Forward Commutation Sequence	1	2	3	4	5	6		
HALL0 HALL1 HALL2	H L L	H H L	L H L	L H H	L L H	H L H	H H H	L L L
PWM0 PWM1 PWM2	C L L	L C L	L C L	L L C	L L C	C L L	C L L	C L L
EN0 EN1 EN2	H L H	L H H	H H L	H L H	L H H	H H L	H H L	H H L

Reverse Commutation Sequence	1	2	3	4	5	6		
HALL0 HALL1 HALL2	H L H	L L H	L H H	L H L	H H L	H L L	H H H	L L L
PWM0 PWM1 PWM2	L C L	L C L	C L L	C L L	L L C	L L C	L C L	L C L
EN0 EN1 EN2	H H L	L H H	H L H	H H L	L H H	H L H	H H L	H H L

H = High output, L = Low output, C = Chopped output (PWM signal active)

Note that the last two columns in the table represent invalid hall sensor combinations (for 120 degree commutation). The ***PIC-SERVO SC*** makes use of these invalid codes to allow brush-commutated motors to be used with brushless motor drives. If the ***PIC-SERVO SC*** detects that one of the invalid hall sensor combinations is being used (all High or all Low), it activates only the first two half bridges and switches them in accordance with driving a brush commutated motor. In fact, this is the easiest method for implementing a simple H-Bridge from two independent half-bridge drivers.

The combination of PWM outputs of Chopped or Low is designed to interface directly with common half-bridge drivers as follows:

1. At any given time, only two half bridges are enabled.
2. The half-bridge attached to the chopped, or active PWM output will alternate between having the upper transistor turned on (PWM High) and the lower transistor turned on (PWM Low).
3. The other half-bridge will always have the low transistor turned on.

This will result in the windings being actively driven when the PWM signal is High, and it will allow the winding current to recirculate through the lower transistors when the PWM signal is Low.

4.6 EEPROM and Standalone Operation

The ***PIC-SERVO SC*** has an internal EEPROM which can be used to save programmed operating parameters such as the servo gains and the motion parameters. The EEPROM can also be configured to make the ***PIC-SERVO SC*** power-up in a stand-alone mode with the servo enabled and ready to accept step and direction signals. This is particularly useful when the ***PIC-SERVO SC*** is used as a replacement for a stepper motor driver. Note the even when the ***PIC-SERVO SC*** powers up in stand-alone mode, it will still respond to commands sent over the serial port.

Users wishing to use the ***PIC-SERVO SC*** in standalone mode for use with step and direction inputs can use the Windows based NMCTEST.EXE utility program to set all of the operating parameters and then save them in EEPROM.

In general, we do not recommend using the EEPROM to save operating parameters when there is a host sending motion commands over the serial port interface[+] . This is because it is very easy to have the wrong parameters saved without knowing it, especially when one ***PIC-SERVO SC*** controller is replaced with another in the field. It is much more reliable, from a system engineering point of view, to have all of the operating parameters stored in the host processor and downloaded each time the network of ***PIC-SERVO SC*** controllers is initialized.

Motion parameters and start-up data are saved in the EEPROM using a version of the Reset command. Normally, the Hard Reset command is sent with no additional data bytes, and it simply resets the controller to its power-up state. However, if the Hard Reset command is sent with an additional control byte, the bits of the control byte can be set or cleared to save data and configure the start-up mode. See the command reference in Section 5.2 for details on configuring the EEPROM.

Lastly, the EEPROM data is only reloaded into the ***PIC-SERVO SC***'s operating registers if the controller is reset as a result of a *hardware* event such as power-up, brown-out, or a reset via the MCLR pin. If a reset command is sent by a host through the serial port, the ***PIC-SERVO SC*** will always power-up to its normal default condition and expect to be explicitly initialized by the host

[+] The only exception to this is that you may want to have Antiphase or 3-Phase output options stored in EEPROM if required by your amplifier. This will ensure that the ***PIC-SERVO SC*** starts up with the proper output signals.

using serial port commands. This insures that a host processor can always reset the ***PIC-SERVO SC*** to a known state. For example, even if the ***PIC-SERVO SC*** is configured to start up in stand-alone mode with an unknown pre-programmed address, a host can send a reset command to the universal reset address of 0xFF and the ***PIC-SERVO SC*** will reset itself to a known state.

The exception to *not* reloading the EEPROM data on a software reset is that options to select either Antiphase or 3-Phase output modes will always be restored on *any* kind of reset. This enables the controller to remain in the proper output mode, even on a software reset. These output options can always be cleared by your software and erased in EEPROM if desired.

4.7 Protection Features

Homing Control and Limit Switch Inputs

The ***PIC-SERVO SC*** has two limit switch inputs which can be used for both homing and for overtravel protection. When the limit switch overtravel protection is enabled[*] using the I/O Control command, the LIMIT1 pin is used as the forward direction limit switch input, and the LIMIT2 pin is used as the reverse direction input. You can choose to either have the motor stop abruptly or have the servo turned off altogether when a limit switch is hit. The action of the overtravel protection is summarized in the following table:

	LIMIT1 = 0
	LIMIT2 = 0
		LIMIT1 = 1
		LIMIT2 = 0
			LIMIT1 = 0
			LIMIT2 = 1
				LIMIT1 = 1
				LIMIT2 = 1

Position Servo Enabled	Motion is not inhibited	Forward motion is stopped abruptly, or the servo is turned off. Any motion command requiring forward velocity is ignored.	Reverse motion is stopped abruptly, or the servo is turned off. Any motion command requiring a reverse velocity is ignored.	All motions are either stopped abruptly or the servo is turned off.
Position Servo Disabled (PWM Mode)	Any PWM value may be set	If PWM value is positive, the output will be set to zero. Commands setting a positive PWM value are ignored.	If PWM value is negative, the output will be set to zero. Commands setting a negative PWM value are ignored.	The PWM output is set to zero.

Limit switch behavior when limit inputs are set to stop abruptly or turn the motor off.

Homing

The ***PIC-SERVO SC***'s homing control function can also use the limit switch inputs to find a home position for your motor. Using the Set Homing command, you can "arm" the homing function to look for any *change* in the state of LIMIT1, LIMIT2 or the ENC_INDEX pin. Homing can also be triggered when ever the POS_ERROR or OVERCURRENT flag is set. Homing will be triggered when any one of the homing conditions occurs. You can also choose to automatically stop smoothly, stop abruptly, or turn the motor off when one of the homing conditions is triggered.

[*] Earlier versions of the ***PIC-SERVO*** did not have a dedicated overtravel protection function. Instead the Set Homing command was used to implement overtravel protection.

Once one of the selected homing conditions is triggered, the ***PIC-SERVO SC*** will take the current position of the motor and store it in the Home Position register, and the motor will stop if one of the autostop modes is selected. The Home Position can be read by the host using the Read Status command. The Set Homing function must be re-issued if you wish to re-arm the homing triggers.

The current data stored in the Home Position register can also used by the Reset Position command. If the Reset Relative to Home option is selected for the Reset Position command, the home position will be defined as zero, and all motor positions will be reported or commanded relative to home. Note that if the Home Position data is subsequently changed, the motor positions will not be affected unless another Reset command is issued.

Note that the Set Homing command does not initiate any motion of the motor - it simply arms the homing function to look for the specified homing conditions. In practice, you should issue the Set Homing command and then issue a motion command to move the motor towards one of your limit switches or towards the encoder's index.

Lastly, the homing function only looks at the motor positions once per servo cycle. To get the maximum accuracy in homing, the motor should be moving at less than one encoder count per servo cycle. In practice, you may want to execute homing with two separate homing sequences - one fast to get you within the vicinity of your homing switch, followed by a short, slow move for accuracy.

Overcurrent Protection

The ***PIC-SERVO SC*** has a programmable current limit which is used in conjunction with the analog input CUR_SENSE pin. The current limit is a value between 0 and 255 set with the Set Gains command. If the motor current exceeds the programmed current limit, the calculated output from the PID filter will be decremented by a value of 2 each servo cycle. Once the current level drops back below the limit, the output will be incremented by a value of 2 each servo cycle until it is again equal to the calculated output. This non-linear adjustment is designed to work with a linear current sense signal or with a binary overcurrent warning signal to keep the drive current just below the shutdown threshold of your amplifier.

Please note that because the output is reduced smoothly over time, the overcurrent protection is not suitable for actually protecting your amplifier from overcurrent damage resulting from short circuits or other current spikes.

The OVERCURRENT bit of the status byte is used to alert the host to an overcurrent condition. This is a latched bit such that it is set and stays set when an overcurrent condition first occurs. It must be cleared by the host using the Clear Bits command.

Motor Power Undervoltage and Overvoltage Protection

Valid motor power voltage can be detected by connecting the motor power supply to the VOLT_SENSE pin through a voltage divider (see examples in Section 6.0). If the voltage on this pin is between 0.9v and 4.5v, the POWER_ON bit of the status byte will be HIGH.

The ***PIC-SERVO SC*** automatically monitors the voltage on the VOLT_SENSE pin. If the voltage is less than 0.9v, indicating that the motor power supply is off or that the voltage is too low, it will automatically disable the servo and temporarily lower the ENABLE outputs to the amplifier. If the voltage on the VOLT_SENSE pin rises back above 0.9v, the ENABLE pins will be reset to their former state, but the host will have to issue a Stop Motor command or a motion command to re-start the servo.

If the voltage on the VOLT_SENSE pin rises above 4.5v (as can happen when a decelerating motor acts as a generator), the ENABLE pins will temporarily be lowered until the voltage is back within range. Note that on overvoltage, the servo is not necessarily disabled, giving the voltage a chance to drop back down into the allowable range without aborting the motion in progress. (You should also note that if multiple motors are connected to the same supply, overvoltage generated by one motor will affect all other controllers as well!)

The valid VOLT_SENSE input range of 0.9v to 4.5v allows a power supply voltage variation of 5:1 when connected through a simple voltage divider. However, this range can be increased or decreased by using a series zener diode and resistor in parallel with either the upper low lower half of the voltage divider.

Care should be taken so that in the worst case scenario, the VOLT_SENSE pin should not rise above 5.0v. The input pin does have protection diodes but which can handle only a limited amount of current. Therefore the upper resistor of the voltage sensing divider network should be sized to limit the current to a few milliamps if the sense voltage does exceed 5.0v.

To disable the voltage sensing feature, simply connect the VOLT_SENSE pin to Vcc through a voltage divider resistor network (*eg*, two 4.7K resistors in series to GND) so that the pin voltage is 2.5v.

Lastly, the voltage sensing only occurs once per servo cycle (1953 Hz) and therefore cannot be used to detect fast voltage spikes in your system. You should provide other forms of protection if this is a potential problem.

Powerup and Reset Conditions

If the EEPROM is not used to store operating parameters, on power-up or reset, the ***PIC-SERVO SC*** is initialized to the following state:

Motor position is reset to zero
Velocity and acceleration values are set to zero
All gain parameters and limit values are set to zero
The servo rate divisor is set to 1
The step multiplier is set to 1
The PWM value is set to zero
The PID servo filter is disabled
ENABLE outputs are set low
Step and Direction mode is disabled
The amplifer output mode is set for PWM and Direction
The default status data is the status byte only
The individual address is set to 0x00 and the group address to 0xFF (group leader not set)
Communications are disabled pending a low value of ADDR_IN
The baud rate is set to 19.2 KBaud
In the status byte, the move_done and pos_error flags will be set and the overcurrent and home_in_progress flags will be clear.
In the auxiliary status byte, the pos_wrap, servo_on, accel_done, slew_done and servo_overrun flags will be clear.

The table below describes the actions taken on power-up or reset in different circumstances:

	Hardware Reset	Software Command to Reset
Parameters saved in EEPROM	All parameters stored in EEPROM are restored	***PIC-SERVO SC*** is initialized as described above, except for that the selected output mode (3-Phase, Antiphase or default PWM & DIR) is restored
No parameters saved	***PIC-SERVO SC*** is initialized as described above	***PIC-SERVO SC*** is initialized as described above

5.0 Command Specification

Data format

All data is sent as 8-bit data bytes with one start bit and one stop bit. Multi-byte data (16 bit and 32 bit integers) is always sent with the least significant byte first. If integer data is signed, a 2's compliment format is used for negative numbers.

5.1 Command Summary

Command	CMD Code	# Data bytes	Description
Reset Position	0x0	0, 1 or 5	Set or clear 32 bit position counter
Set Address	0x1	2	Sets the individual and group addresses
Define Status	0x2	1	Defines which data should be sent in every status packet
Read Status	0x3	1	Causes particular status data to be returned just once
Load Trajectory	0x4	1-14	Loads motion trajectory parameters for velocity and trapezoidal profile modes
Start Motion	0x5	0	Executes the previously loaded trajectory
Set Gain	0x6	15	Sets the PID gains and operating limits
Stop Motor	0x7	1 or 5	Stops the motor in one of four modes
I/O Control	0x8	1	Sets various I/O control options
Set Homing	0x9	1	Sets conditions for capturing the home position
Set Baud Rate	0xA	1	Sets the baud rate (group command only)
Clear Bits	0xB	0	Clears any latched status bits
Save as Home	0xC	0	Saves the current position in the home position register
Add Path Points	0xD	0 - 14	Add points to path point buffer for path control mode
NoOp	0xE	0	Simply causes the defined status data to be returned
Hard Reset	0xF	0 or 1	Resets the controller to its power-up state with the option to save data in EEPROM just prior to reset.

5.2 Command Descriptions

The following pages have descriptions of each of the commands, followed by examples of literal binary command strings (written in hexadecimal) for the use of each command. Note that the command strings all start with the header byte (0xAA) and end with the checksum byte.

Reset Position	CMD Byte = 0x00 or 0x10 or 0x50
Reset encoder position counter	

Number of data bytes: 0, 1 or 5

Data Byte 1 - Control Byte

Bit 0 - Reset relative to the Home Position (use CMD Byte 0x10 if set)

Bit 1 - Set position counter to a specific value (use CMD Byte 0x50 if set)

Bits 2 - 7 - unused, set to 0

Data Bytes 2 - 5

32 bit position data used for setting position counter

(signed 32 bit integer: -2,147,483,648 to +2,147,483,647)

Description

Reset Position is used to set or clear the value of the ***PIC-SERVO SC*** ’s 32 bit position counter. There are three different version of the command.

Simple Reset - Using a CMD Byte value of 0x00 and no additional data bytes will simply reset the position counter to a value of zero. The current command position used by the PID position servo will also be set to zero to prevent the motor from jumping.

Reset Relative to Home Position - If you use a CMD Byte value of 0x10 and set Bit 0 of the control byte, the 32 bit position register will be set so that the Home Position (as determined using the Set Homing command) is now the zero position and all motor positions will be commanded or reported relative to the home position.

Set Position to a Specific Value - If you use a CMD Byte value of 0x50 and set Bit 1 of the following control byte, you can send 4 additional bytes of data specifying the 32 bit position value to be loaded into the ***PIC-SERVO SC*** ’s position register.

Please note that only one bit (Bit 0 or Bit 1) of the control byte should be set.

Examples

To clear the position of module address 1 to zero, use the following command string:

0xAA	0x01	0x00	0x01
header	address	cmd	checksum

To reset the position of module address 1 relative to the current Home Position:

0xAA	0x01	0x10	0x01	0x12
header	address	cmd	control	checksum

Set the position of module address 1 to a value of 0x015432A2:

0xAA	0x01	0x50	0x02	0xA2	0x32	0x54	0x01	0x7C
header	address	cmd	control	position data				checksum

Set Address	CMD Byte = 0x21
Set the individual and group address for a module	

Number of data bytes: 2

Data Byte 1 - Individual Address (0 - 0xFF)
Data Byte 2 - Group Address (0x80 - 0xFF)

Description

Set Address is used to set both the individual and group address for a module. On normal power-up or reset, all modules have an address of 0x00, but only one module should have its communications enabled by its ADDR_IN pin. Sending a set address command to module 0 will set its address and then raise its ADDR_OUT pin, enabling communications for the next module. On initialization, each module should be assigned a unique individual address.

Group addresses for modules are restricted to the range 0x80 to 0xFF. If you wish to specify that a module be the group leader for its group address, you should clear Bit 7 of the group address. The ***PIC-SERVO SC*** will check this bit to see if the module is to be a group leader, and then internally set Bit 7 again to 1.

Although normally just set once during initialization, individual and group addresses can be reset to different values at any time.

The Set Address command is common to all NMC compatible controller modules.

Examples

To set the individual address of module 0 to 1, and set the group address to 0x81 (not a group leader):

0xAA	0x00	0x21	0x01	0x81	0xA3
header	address	cmd	addr	group addr	checksum

To leave the individual address of module 1 at 1, but set the group address to 0x81 and have it be the group leader:

0xAA	0x01	0x21	0x01	0x01	0x24
header	address	cmd	addr	group addr	checksum

Define Status	CMD Byte = 0x12
Define which status data is returned after each command packet sent	

Number of data bytes: 1

Data Byte 1 - Control Byte

- Bit 0 - Send position data (4 bytes - signed 32 bit integer)
- Bit 1 - Send A/D value of voltage on CUR_SENSE pin (1 byte, 0 - 255)
- Bit 2 - Send actual velocity in encoder counts per servo cycle - the actual velocity has no integer component (2 bytes - signed 16 bit integer)
- Bit 3 - Send auxiliary status byte (1 byte)
- Bit 4 - Send Home Position (4 bytes - signed 32 bit integer)
- Bit 5 - Send device type and device version number (2 bytes) (PIC-SERVO = type 0, PIC-I/O = type 2, PIC-STEP = type 3)
- Bit 6 - Send servo position error (2 bytes - signed 16 bit integer)
- Bit 7 - Send number of path points left in path point buffer (1 byte)

Description

Define Status defines what status data will be included in the status packet returned after a command packet is sent. The specific data selected with the individual bits of the control byte will be sent in the order listed and will follow the default status byte which is always sent with each packet.

For efficient communications, you should just select the data which you will always need access to. For data that only needs to be read periodically, use the Read Status command instead. You can use the Define Status command at any time to change what status data is returned.

This command is common to all NMC compatible control modules, although the interpretation of the control byte will be different for each module type.

Example

To have module address 1 send back position data and position error data with each status packet send the command string:

0xAA	0x01	0x12	0x41	0x54
header	address	cmd	control	checksum

Read Status	CMD Byte = 0x13
Read specified status data one time	

Number of data bytes: 1

Data Byte 1 - Control Byte

- Bit 0 - Send position data (4 bytes - signed 32 bit integer)
- Bit 1 - Send A/D value of voltage on CUR_SENSE pin (1 byte, 0 -255)
- Bit 2 - Send actual velocity in encoder counts per servo cycle - the actual velocity has no integer component (2 bytes - signed 16 bit integer)
- Bit 3 - Send auxiliary status byte (1 byte)
- Bit 4 - Send Home Position (4 bytes - signed 32 bit integer)
- Bit 5 - Send device type and device version number (2 bytes) (PIC-SERVO = type 0, PIC-I/O = type 2, PIC-STEP = type 3)
- Bit 6 - Send servo position error (2 bytes - signed 16 bit integer)
- Bit 7 - Send number of path points left in path point buffer (1 byte)

Description

Read Status is used to read specific status data from the ***PIC-SERVO SC*** just one time; that is, the status packet returned in response to a Read Status command will include the data specified in the Control Byte, but the status packet returned with any subsequent commands will be include the status data specified with the most recent Define Status command. For example, the Read Status command is useful for retrieving device type and version number data which only needs to be read once. The specific data selected with the individual bits of the control byte will be sent in the order listed and will follow the default status byte always sent with each packet.

This command is common to all NMC compatible control modules, although the interpretation of the control byte will be different for each module type.

Example

To read from module address 1 the device type and version number just once:

0xAA	0x01	0x13	0x20	0x34
header	address	cmd	control	checksum

Load Trajectory	CMD Byte = 0x*n*4, n = 1 - 14
Send motion trajectory command data for PWM, velocity, and trapezoidal profile modes	

Number of data bytes: 1 - 14

Data Byte 1 - Control Byte
Bit 0 - Load position data (add 4 additional data bytes)
Bit 1 - Load velocity data (add 4 additional data bytes)
Bit 2 - Load acceleration data (add 4 additional data bytes)
Bit 3 - Load PWM data (add one data byte)
Bit 4 - Enable PID servo
Bit 5 - Profile mode: 0 = trapezoidal profile, 1 = velocity profile
Bit 6 - Direction / Relative select bit
in velocity mode or PWM mode: 0 = Forward, 1 = Reverse
in trapezoidal profile mode: 0 = absolute position, 1 = relative position
Bit 7 - Start motion now

Data Bytes 2 - 5
Position data if Bit 0 of the control byte is set
(signed 32 bit integer: -2,147,483,648 to +2,147,483,647)

Data Bytes 6 - 9
Velocity data if Bit 1 of the control byte is set
(positive 32 bit integer: 0 to +83,886,080)

Data Bytes 10 - 13
Acceleration data if Bit 2 of the control byte is set
(positive 32 bit integer: 0 to +2,147,483,647)

Data Byte 14
PWM data if Bit 3 of the control byte is set
(positive 8 bit integer: 0 to +255)

Description

The Load Trajectory command is used to send motion trajectory and PWM information. It is flexible in that it permits you to only send the data needed for a particular motion. For example, suppose you have already loaded acceleration and velocity parameters, and you only need to send commands with updated position data. In this case, you would set Bit 0 of the control byte (along with bits 4-7 as needed), and then only send additional data bytes 2 - 5. Or if you only needed to send updated velocity information, you would set Bit 1 of the control byte and then next 4 bytes sent would be the velocity data (position data bytes would be eliminated).

You should note, as described in Section 4.4.7, that the lower 16 bits of the velocity and acceleration parameters represent a fractional component. Please refer to this Section for details on how to specify these parameters.

Bits 4, 5 and 6 of the control byte govern the mode of operation. Bit 4 should be set to enable the PID servo - this is the normal mode of operation. If bit 4 is not set, you will default to PWM mode operation and the PWM value provided will be used. (If no PWM value is sent, the current output value will be used.)

(Load Trajectory continued…)

Bit 5 is used to select trapezoidal profile mode (Bit 5 = 0) or velocity profile mode (Bit 5 = 1).

Velocity, acceleration and PWM parameters should all be positive. If you need to specify a reverse velocity or PWM value, you should set Bit 6 of the control byte.

In trapezoidal position mode, however, the position data may be positive or negative, and a reverse direction bit is not needed. Therefore, if you select trapezoidal profile mode, Bit 6 is used to specify if the position data is an absolute position (Bit 6 = 0) or if the position data is relative to the current command position (Bit 6 = 1).

Lastly, Bit 7 of the control byte is used to specify if you want the command data to take effect immediately, or if you want to wait for a Start Motion command to be sent. For individual axis control, it is usually easiest to set Bit 7 and eliminate the need for a separate Start Motion command.

However, if you need to start several controllers moving at exactly the same time, you can send them individual Load Trajectory commands without Bit 7 set. This will cause the data to simply sit in a temporary buffer. You can then issue a Start Motion command to the group address containing several controllers, thereby starting all motions at the same time.

You should note that if Bit 7 is not set, the motion parameters will be ignored until a Start Motion command is issued. If you send a new Load Trajectory before sending a Start Motion command, the temporary buffer will be over-written, erasing your previous command data.

There are three status bits associated with velocity and trapezoidal profiling: MOVE_DONE (in the status byte), SLEW and ACCEL (in the aux. status byte). In velocity mode, the MOVE_DONE bit is set when the goal velocity is reached. In trapezoidal mode, the MOVE_DONE bit is set when the servo command position is equal to the final goal position. In both modes, the SLEW bit is set whenever the motor is moving at a constant velocity, and the ACCEL bit is set when accelerating or cleared when deceleration.

(Examples on following page.)

(Load Trajectory continued…)

Examples

Move module address 1 to an absolute position of -1024 (0xFFFFFC00), velocity of 100,000 (0x186A0), acceleration of 100 (0x64) in trapezoidal profile mode, starting now:

0xAA	0x01	0xD4	0x97	0x00 0xFC 0xFF 0xFF	0xA0 0x86 0x01 0x00
header	address	cmd	control	position data	velocity data

0x64 0x00 0x00 0x00	0xF1
acceleration data	checksum

Move module address 1 with a velocity of -100,000 in velocity profile mode starting now (assume the acceleration parameter has already been loaded):

0xAA	0x01	0x54	0xF2	0xA0 0x86 0x01 0x00	0x6E
header	address	cmd	control	velocity data	checksum

Load velocity (100,000) and acceleration (100) data for subsequent motions, and leave the ***PIC-SERVO SC*** in PWM mode with an PWM value of 0:

0xAA	0x01	0xA4	0x8E	0xA0 0x86 0x01 0x00	0x64 0x00 0x00 0x00
header	address	cmd	control	velocity data	acceleration data

0x00	0xBE
PWM	checksum

Start Motion	CMD Byte = 0x05
Start the motion specified with the previous Load Trajectory command	

Number of data bytes: 0

Description

The Start Motion command is intended to be sent to a group of controllers which have been preloaded (using Load Trajectory commands) with motion profile data. When sent to the group address for these controllers, it will cause all controllers to start their motions simultaneously. Note that the data loaded into each controller with the Load Trajectory command merely sits unused in a buffer until a Start Motion command is received. If another Load Trajectory command is received before the Start Motion command, it will overwrite all of the previous trajectory data.

This command is common to all NMC compatible controllers, although the action taken in response to this command will be different for each type of controller.

Example

To start the previously loaded motion for module address 1, use the command string:

0xAA	0x01	0x05	0x06
header	address	cmd	checksum

Set Gain	CMD Byte = 0xF6
Set PID servo filter operating parameters	

Number of data bytes: 15

Data Bytes 1, 2 - Position gain Kp (positive 16 bit integer: 0 - +32,767)
Data Bytes 3, 4 - Derivative gain Kd (positive 16 bit integer: 0 - +32,767)
Data Bytes 5, 6 - Integral gain Ki (positive 16 bit integer: 0 - +32,767)
Data Bytes 7, 8 - Integration limit IL (positive 16 bit integer: 0 - +32,767)
Data Byte 9 - Output limit OL (0 - 255)
Data Byte 10 - Current limit CL (0 - 255)
odd values: CUR_SENSE proportional to motor current
even values: CUR_SENSE inversely proportional to motor current
Data Bytes 11,12 - Position error limit EL (positive 16 bit integer: 0 - +32,767)
Data Byte 13 - Servo rate divisor SR (1 - 255)
Data Byte 14 - Amplifier deadband compensation DB (0 - 255)
Data Byte 15 - Step rate multiplier SM (1 - 255)

Description

The Set Gain is used to set most of the non-motion profile related operating parameters of the ***PIC-SERVO SC***. The PID gain value, the integration limit, output limit, position error limit, servo rate divisor and amplifier deadband are all described in detail in Section 4.3. The step rate multiplier is described in Section 4.4.6

The current limit value CL is used to set the allowable current level as described in Section 4.7. The parameter itself has a different meaning based on whether the value is odd or even (*i.e.*, bit 0 is set or cleared). If the CL value used is odd, the ***PIC-SERVO SC*** assumes that the voltage on the CUR_SENSE pin is directly proportional to the current driven through the amplifier. (A voltage of 0.0v at this pin will produce an analog reading of 0 and a voltage of 5.0v will produce an analog reading of 255.) If the analog reading at the CUR_SENSE pin is greater than CL, an overcurrent condition will be triggered.

Some amplifiers, however, may produce a voltage signal inversely proportional to the motor current, or may only have a digital signal which is lowered when some current threshold is reached. (*i.e.*, the voltage on CUR_SENSE goes down as the current goes up.) Therefore, the ***PIC-SERVO SC*** will interpret an *even* values of CL such that if the analog reading at the CUR_SENSE pin is *less than* CL, and overcurrent condition will be triggered.

Note that a CL value of 0 (the minimum even value) or a CL value of 255 (the maximum odd value) will effectively disable the current limiting feature.

(Set Gain continued…)

Example

To set the gains of module address 1 to the following values:

Kp = 100, Kd = 1000, Ki = 50, IL = 200,
OL = 255, CL = 53 (directly proportional), EL = 4000
SR = 1, DB = 0, SM = 5

use the command string:

0xAA 0x01 0xF6
header address cmd
0x64 0x00 0xE8 0x03 0x32 0x00 0xC8 0x00
Kp Kd Ki IL
0xFF 0x35 0xA0 0x0F
OL CL EL
0x01 0x00 0x05
SR DB SM
0x29
checksum

Stop Motor	CMD Byte = 0x17 or 0x57
Stop the motor and enable or disable the amplifier	

Number of data bytes: 1 or 5

Data Byte 1 - Control Byte

- Bit 0 - Enable amplifier
- Bit 1 - Turn motor off
- Bit 2 - Stop motor abruptly
- Bit 3 - Stop motor smoothly
- Bit 4 - Stop at specified position (Stop Here)
- Bits 5 - 7 - unused, set to 0

Data Bytes 2 - 5

- Unprofiled command position (signed 32 bit integer)

Description

The Stop Motor command is used to both stop the motor in one of four ways, and also to control whether the amplifier is enabled or not.

If Bit 0 of the control byte is set, the amplifier will be enabled if and when the motor supply voltage is within the proper range. If in PWM & Direction or Antiphase PWM modes, this will simply result in raising the ENABLE0 output. If in 3-Phase commutation mode, this will enable the commutation logic to raise the proper combination of ENABLE pins. If Bit 0 is cleared to 0, all ENABLE pins will be lowered.

If Bit 1 of the control byte is set, the motor will be turned off by disabling the servo and setting the PWM output to 0. The amplifier enable bit may be set or cleared with this stopping mode, depending on the desired amplifier behavior. This stopping mode will cause the MOVE_DONE bit of the status byte will be set.

If Bit 2 of the control byte is set, the motor will immediately attempt to servo to its current position, causing the motor to stop abruptly. If this mode is selected, the amplifier enable bit should be set as well. This stopping mode will cause the MOVE_DONE bit of the status byte will be set.

If Bit 3 is set, the motor will decelerate to a stop smoothly at the current programmed acceleration value. If this mode is selected, the amplifier enable bit should be set as well. This stopping mode will cause the MOVE_DONE bit of the status byte to be cleared while decelerating and then set again once the motor has stopped.

If Bit 4 of the control byte is set, the motor will be servoed immediately to the position specified with data bytes 2 - 5, and it will stop abruptly at that position. There is no profiling to smooth the motion. If using this mode, the distance between the specified goal position and the current motor position should be less than the position error limit specified with the Set Gain command, otherwise, a position error will be generated. If this mode is selected, the amplifier enable bit should also be set. This stopping mode will cause the MOVE_DONE bit of the status byte to be set.

(Stop Motor continued…)

If one of the first three stopping modes is selected, you should use a command byte of 0x17 and send only one additional data byte, the control byte. If the Stop Here mode is selected, you should use the command byte of 0x57 and send the position data as well.

Lastly, only one of the stopping mode bits 1 - 4 should be selected. If none of these bit are set, the amplifier will be enabled or disabled as specified, but no other action will be taken.

Examples

For module address 1, to turn off the motor and disable the amplifier, use the following command string:

0xAA	0x01	0x17	0x02	0x1A
header	address	cmd	control	checksum

For module address 1, smoothly decelerate to a stop:

0xAA	0x01	0x17	0x09	0x21
header	address	cmd	control	checksum

For module address 1, stop the motor at a position of 100:

0xAA	0x01	0x57	0x11	0x64	0x00	0x00	0x00	0xCD
header	address	cmd	control	position data				checksum

I/O Control	CMD Byte = 0x18
Set amplifier output mode and other I/O options	

Number of data bytes: 1

Data Byte 1 - Control Byte

Bit 0 - Not used (clear to 0 for future compatibility)
Bit 1 - Not used (clear to 0 for future compatibility)
Bit 2 - Enable limit switch protection, turn motor off when limit is hit
Bit 3 - Enable limit switch protection, stop abruptly when limit is hit
Bit 4 - Enable 3-Phase commutation output mode
Bit 5 - Enable Antiphase PWM output mode
Bit 6 - Set fast path option for path control mode
Bit 7 - Enable Step & Direction input mode

Description

CAUTION: Use extreme care in setting the parameters for this command - incorrect settings could damage your amplifier or the *PIC-SERVO SC* chip.

I/O control is used to set a number of operating options for the ***PIC-SERVO SC***. An I/O control command should be issued prior to enabling the amplifier to make sure that the output options are set to be compatible with the amplifier type.

Bits 0 and 1 are not used[*] . You should clear these bits to a value of 0 for future compatibility.

Bits 2 and 3 are used to enable the limit switch protection described in Section 4.7, automatically stopping the motor abruptly or turning the motor off when a limit switch is hit. Only one of these bits should be set. If Step and Direction mode is enabled, neither of these bit should be set.

Bit 4 is used to enable 3-Phase commutation mode (Section 4.5.3) and Bit 5 is used to enable Antiphase PWM mode (4.5.2). Only one of these bits should be set. If you want to use PWM & Direction mode (the default), neither bit should be set.

Bit 6 is used to set the fast path option for path control mode described in Section 4.4.5.

Bit 7 is used to enable the Step & Direction input mode described in Section 4.4.6. If Step & Direction mode is selected, Bits 2 and 3 should both be clear.

Note that each time an I/O control command is issued, every one of the mode options will be enabled or disabled according the corresponding bit of the control byte.

Example

For module address 1, disable the limit switch protection, enable 3-phase commutation, and enable Step & Direction input mode:

0xAA	0x01	0x18	0x90	0xA9
header	address	cmd	control	checksum

[*] Earlier versions of the ***PIC-SERVO*** used the I/O control command to optionally define the LIMIT pins as outputs. This feature is no longer available in the ***PIC-SERVO SC***, and care should be taken in setting the correct control byte bits for the each version of the ***PIC-SERVO***.

Set Homing	CMD Byte = 0x19
Set homing mode parameters for capturing the home position	

Number of data bytes: 1

Data Byte 1 - Control Byte

- Bit 0 - Capture home position on *change* of LIMIT1
- Bit 1 - Capture home position on *change* of LIMIT2
- Bit 2 - Turn motor off when home position is captured
- Bit 3 - Capture home position on *change* of ENC_INDEX
- Bit 4 - Stop motor abruptly when home position is captured
- Bit 5 - Stop motor smoothly when home position is captured
- Bit 6 - Capture home position when a position error occurs
- Bit 7 - Capture home position when current limiting occurs

Description

Set Homing is used for specifying conditions for capturing the home position of a motor, and also for specifying any desired automatic stopping mode for when the home position is found.

Bits 0, 1, 3, 6 and 7specify the conditions for capturing the home position. The home position will be captured when *any* of the specified conditions occurs. Bits 0, 1 and 3 specify that the homing function look for *changes* in the states of the limit and index input pins from when the homing command is issued. It does not matter if the pin voltages start off high or low.

Bits 6 and 7 allow you to also capture home on the occurrence of a position error or on current limiting. Note that you should use the Clear Bits command to clear the value of these status bits before issuing the Set Homing command.

Bits 2, 4 and 5 are used to specify an automatic stopping mode for the motor once the home position has been captured. Only one (or none) of these bits should be set.

When the Set Homing command is issued, the HOME_IN_PROG bit of the status byte will be set. You should then issue a motion command to move towards one of the triggers used for homing. The HOME_IN_PROGR bit will then be cleared when any of the selected homing conditions occur, and the current motor position is stored in the home position register. The home position register can be read using the Read Status command.

Once the homing process is complete, you can re-issue the Set Homing command if desired to capture a different home position. Sending a Set Homing command with the control byte equal to zero will cancel any homing in progress and clear the HOME_IN_PROG bit.

Example

To have module address 1 capture the home position on a change of LIMIT1 or LIMIT2 and then stop abruptly:

0xAA	0x01	0x19	0x13	0x2D
header	address	cmd	control	checksum

Set Baud	CMD Byte = 0x1A
Set the Baud rate for serial communications	

Number of data bytes: 1

Data Byte 1 - Baud rate specifier

127 = 9,600 bits / sec
64 = 19,200 bits / sec
21 = 57,600 bits / sec
10 = 115,200 bits / sec
5 = 230,400 bits / sec (valid for ***PIC-SERVO SC*** only)

Description

The Set Baud command sets the serial communications bit rate. By default, the ***PIC-SERVO SC*** powers up communicating at 19,200 bits per second. Using one of the listed Baud rate specifier values will cause the ***PIC-SERVO SC*** will adjust its internal UART parameters to the proper value.

All NMC controllers connected through the same serial port should have their baud rates changed simultaneously. Therefore this command should be issued to a *group* of all controllers connected to the serial port using the group address. Because a status packet coming back from this command would be at the new baud rate, it is usually easiest to send this command when no group leader has been declared. This will give the host time to change its Baud rate to the new value.

This command and the baud rate specifier values are common to all NMC compatible controllers.

Example

Change the Baud rate of all controllers with a group address of 0xFF to 115,200 bps:

<u>0xAA</u>	<u>0xFF</u>	<u>0x1A</u>	<u>0x0A</u>	<u>0x23</u>
header	address	cmd	rate	checksum

Clear Bits CMD Byte = 0x0B
Clear the latched status bits

Number of data bytes: 0

Description

The OVERCURRENT and POS_ERROR bits in the status byte and the POS_WRAP and SERVO_OVERRUN in the auxiliary status byte are latched flags which remain set until explicitly cleared with the Clear Bits command. All of these latched bits are cleared with a single command.

Example

To clear the latched status bits of module address 1:

0xAA	0x01	0x0B	0x0C
header	address	cmd	checksum

Save As Home	CMD Byte = 0x0C
Load the current motor position into the Home Position register	

Number of data bytes: 0

Description

The Save as Home command is used to synchronously save the current positions of a number of motors. The Save as Home command issued to a group of controllers will cause them all store their current positions in their corresponding home position registers. The home position registers can then be read individually using a Read Status command for each controller. This allows the host to take a snapshot of the configuration of a multi-axis system.

This command is common to all NMC compatible controllers, but the exact data saved will be different for different types of controllers.

Example

To save the positions of all modules with group address 0x81, use the command string:

0xAA	0x81	0x0C	0x8D
header	address	cmd	checksum

Add Path Points	CMD Byte = 0x*n*D , n = 0, 2, 4, 6, 8, A, C, D
Add path points to the path point buffer, or start path execution	

Number of data bytes: Number of path points added x 2

Data Byte 1, 2 - 16 bit data for path point 1
Data Byte 3, 4 - 16 bit data for path point 2
Data Byte 5, 6 - 16 bit data for path point 3
Data Byte 7, 8 - 16 bit data for path point 4
Data Byte 9, 10 - 16 bit data for path point 5
Data Byte 11, 12 - 16 bit data for path point 6
Data Byte 13, 14 - 16 bit data for path point 7

Description

The Add Path Points command is used to add points to the path point buffer for path control mode. Up to 7 path points can be added with a single command. The upper nibble of the CMD Byte, *n*, should be equal to the number of data bytes sent, which is twice the number of path points added. For example, to add 4 path points, you would use a CMD Byte of 0x8D, followed by 8 bytes of path point data.

Path point data is specified differentially (*i.e.*, you specify the distance between the previous path point position and the current path point position) using the following format:

30 Hz Path:

$P_{13}\ P_{12}\ P_{11}\ P_{10}\ P_9\ P_8\ P_7\ P_6$	$P_5\ P_4\ P_3\ P_2\ P_1\ P_0$ F D
most significant byte	*least significant byte*

60 Hz Path:

$P_{12}\ P_{11}\ P_{10}\ P_9\ P_8\ P_7\ P_6\ P_5$	$P_4\ P_3\ P_2\ P_1\ P_0$ 0 F D
most significant byte	*least significant byte*

where P_0 - P_{13} are the 14 bits of differential position data, F is the path frequency bit (0 = 60 Hz, 1 = 30 Hz) and D is the direction bit (0 = forward, 1 = reverse). Note that for a 60 Hz path, the position data bits are shifted to the left, and bit 2 is always zero. As with other all other types of multi-byte data, the least significant byte is always sent first.

If “fast path” mode has been selected using the I/O Control command, bit F will be set to 0 for 120 Hz or 1 for 60 Hz, and the path point data will have the following format:

60 Hz Path (fast path mode):

$P_{12}\ P_{11}\ P_{10}\ P_9\ P_8\ P_7\ P_6\ P_5$	$P_4\ P_3\ P_2\ P_1\ P_0$ 0 F D
most significant byte	*least significant byte*

120 Hz Path (fast path mode):

$P_{11}\ P_{10}\ P_9\ P_8\ P_7\ P_6\ P_5 P_4$	$P_3\ P_2\ P_1\ P_0$ 0 0 F D
most significant byte	*least significant byte*

(Add Path Points continued…)

This compact format minimizes the amount of data sent.

Starting the Path Motion - Sending an Add Path Points command with no additional data bytes will cause the path motion to start executing, and the PATH_MODE bit in the aux. status byte will be set. Usually you will want to send this command to the entire group of controllers involved in a multi-axis motion to retain coordination. As the path motion executes, the old path points will be removed from the path point buffer.

The number of path points currently residing in the buffer can be read using the Read Status command. The buffer on the ***PIC-SERVO SC***[*] can hold a maximum of 128 path points. Even after the path mode motion has started, you can dynamically add additional path points to the buffers as they empty to create motions of any length.

When the path point buffer runs out, the motor will stop at the last specified path point. The Stop Motor command (any mode) can also be used to terminate a path mode motion. When a path mode motion is terminated, the PATH_MODE bit in the aux. status byte will be cleared.

The path points added to the path point buffer should be closely spaced and form a smooth path for the motor to follow. Note that you can get the exact initial command position of the motor by reading the motor position and the position error with the same Read Status command and them adding them together. You will then specify your path points starting from there.

Examples

For module address 1 which is currently at position 0, add four path points -100, -201, -303, -406 (the differential data for the these path points would be 100, 101, 102, 103 with a direction bit of 1) at 60 Hz:

0xAA	0x01	0x8D	0x21 0x03	0x29 0x03	0x31 0x03	0x39 0x03	0x4E
header	address	cmd	path point 1	path point 2	path point 3	path point 4	checksum

To start path motion for group address 0x81:

0xAA	0x81	0x0D	0x8E
header	address	cmd	checksum

[*] The ***PIC-SERVO CMC*** path point buffer can only hold 96 points. Therefore, you will want to limit the number of points you add to 96 if using a mix of ***PIC-SERVO SC*** and ***PIC-SERVO CMC*** controllers.

No Op	CMD Byte = 0x0E
No operation - return current status data only	

Number of data bytes: 0

Description

The No Op command is used to force the ***PIC-SERVO SC*** to send back a current status data packet without taking any other action. For example, it is useful for polling the MOVE_DONE flag in the status byte to determine when a motion has finished.

Example

To have module address 1, send back a status packet, send the command string:

0xAA	0x01	0x0E	0x0F
header	address	cmd	checksum

Hard Reset	CMD Byte = 0x0F or 0x1F
Reset the controller to its power-up state and optionally store configuration data	

Number of data bytes: 0, or 1

Data Byte 1 - Control Byte

Bit 0 - Save or erase configuration data in EEPROM (1 = save, 0 = erase)
Bit 1 - Restore individual and group addresses on power-up
Bit 2 - Enable the amplifier on power-up
Bit 3 - Enable the PID servo on power-up
Bit 4 - Enable Step & Direction mode on power-up
Bit 5 - Enable selected limit stop protection on power-up
Bit 6 - Enable 3-Phase commutation on power-up
Bit 7 - Enable Antiphase PWM on power-up

Description

Hard Reset resets the ***PIC-SERVO SC*** to its power-up state, but it does not restore any configuration data stored in EEPROM, except for the Antiphase or 3-Phase options. Only an actual power-cycle or reset via the MCLR pin will cause the rest of the EEPROM data to be restored. There are two versions of the Hard Reset command:

Simple Reset - If the CMD Byte 0x0F is used and no control byte is sent, a simple reset is executed where no data is written to or erased from the EEPROM. Normally, a simple reset is sent to all controllers on the network via the universal reset address of 0xFF

Configuration Reset - If the CMD Byte 0x1F is used and a control data byte is sent, data will be written to (Bit 0 = 1) or erased from (Bit 0 = 0) the EEPROM. If Bit 0 is set, the control byte itself will be saved in EEPROM, along with the individual and group addresses, the current velocity and acceleration values, and all of the parameters set with the Set Gain command. If Bit 0 of the control byte is cleared, a value of 0 will be stored in the EEPROM for the control byte, and all other EEPROM data will be erased. Normally, a configuration reset will be sent to an individual controller.

On a hardware reset (power-up or reset via MCLR), the ***PIC-SERVO SC*** will read the control byte and restore the saved data if Bit 0 is set. It will also look at bits 1 - 7 of the control byte to see what other operating options should be restored.

Note that if Bit 1 of the control byte is not set, the individual and group addresses will not be restored, and the address of the module will have to be initialized using the procedure described in Section 4.1. (Note: the servo and amplifier will not be enabled until the address is initialized.) This is useful for when you want to save operating parameters, but are using the ***PIC-SERVO SC*** with other NMC modules which do not have the EEPROM configuration feature. If Bit 1 is set, the addresses will be restored and the ADDR_OUT pin will be lowered just as if a Set Address command had been issued.

If you set Bit 5 of the control byte, the currently selected option for limit switch protection will be saved in EEPROM and then restored on power-up.

(Hard Reset continued…)

Examples

To do a simple reset of all modules, send the command string:

0xAA	0xFF	0x0F	0x0E
header	address	cmd	checksum

To save operating parameters for module address 1 and have the it power-up ready to receive Step & Direction signals and in 3-Phase commutation output mode:

0xAA	0x01	0x1F	0x5F	0x7F
header	address	cmd	control	checksum

To erase the EEPROM configuration data for module address 1:

0xAA	0x01	0x1F	0x00	0x20
header	address	cmd	control	checksum

5.3 Status Byte, Auxiliary Status Byte Definitions

Status Byte

Bit	Name	Definition
0	MOVE_DONE	Clear when in the middle of a trapezoidal profile move, or in velocity mode, when accelerating from one velocity to the next. This bit is set otherwise, including while the position servo is disabled
1	CKSUM_ERROR	Set if there was a checksum error in the most recently received command packet
2	OVERCURRENT	Set if current limiting occurred. Must be cleared by user with Clear Bits command.
3	POWER_ON	Set if motor power is the voltage on the VOLT_SENSE pin is between 0.9v and 4.5v. Clear otherwise.
4	POS_ERROR	Set if the position error exceeds the position error limit. It is also set whenever the position servo is disabled. Must be cleared by user with Clear Bits command.
5	LIMIT1	Value of limit switch 1 input.
6	LIMIT2	Value of limit switch 2 input.
7	HOME_IN_PROG	Set while searching for a home position. Reset to zero once the home position has been captured.

Auxiliary Status Byte

Bit	Name	Definition
0	INDEX	Encoder index input value.
1	POS_WRAP	Set if the 32 bit position counter overflows or underflows. Must be cleared with the Clear Bits command
2	SERVO_ON	Set if the position servo is enabled, clear otherwise
3	ACCEL	Set when the motor is accelerating, clear when decelerating. This bit has no meaning when stopped or at a constant velocity.
4	SLEW	Set when moving at a constant velocity or when stopped. Clear when accelerating or decelerating.
5	SERVO_OVERRUN	This bit is set only if the calculations required for one servo cycle take longer than 0.51 milliseconds. This can happen if Step inputs exceed the allowable step input rate. Cleared with the Clear Bits command.
6	PATH_MODE	This bit is set when a path mode motion is in progress. It is cleared when the path point buffer is emptied or if a Stop Motor command is issued.
7	not used	

5.4 Differences Between the PIC-SERVO SC (v.10) *and Previous Versions*

This section describes any software differences between the ***PIC-SERVO SC*** commands and command for earlier versions of the ***PIC-SERVO***. Please read the Theory of Operation (Section 4) for mode details on other hardware and behavioral differences.

Most applications written for older versions of the ***PIC-SERVO*** will not have problems running with the ***PIC-SERVO SC***. Mostly, you need to be careful if your application sets previously unused bits in various commands' control bytes. Before attempting to run your application written for older ***PIC-SERVO***'s with the ***PIC-SERVO SC***, you should read through this list and correct any potential problems. If using our Windows library NMCLIB0x.DLL, we recommend that you recompile your application using the new library NMCLIB04.DLL, even if there are no obvious conflicts.

Reset Position

Forward compatibility - No issues. A reset position command for older ***PIC-SERVO***'s will simply reset the ***PIC-SERVO SC***'s position counter to zero.

Backward compatibility - The optional additional data byte sent to a ***PIC-SERVO SC*** will be ignored by earlier versions.

Set Address

No issues. This command is identical for all versions.

Define Status and Read Status

Forward compatibility - A control byte sent to earlier ***PIC-SERVO*** versions will be properly interpreted by the ***PIC-SERVO SC***. However, if software for older controllers sets bits 6 or 7 of the control byte (these bits were previously ignored), the ***PIC-SERVO SC*** will send back additional data which the software would not be expecting.

Backward compatibility - Bits 6 and 7 of the control byte are not used by earlier versions, and they will not send back the expected data if these bits are set.

Load Trajectory

Forward compatibility - In earlier versions of the ***PIC-SERVO*** , the Direction/Relative bit (bit 6) was not used when trapezoidal mode was selected. If older software sets this bit while in trapezoidal mode, the ***PIC-SERVO SC*** will interpret the goal position data as relative rather than absolute.

Backward compatibility - The Relative bit will be ignored by earlier versions of the ***PIC-SERVO***. Also, the ***PIC-SERVO SC*** allows you to modify the goal position, acceleration and velocity in the middle of a trapezoidal motion. Software for written for the ***PIC-SERVO SC*** which does modify these parameters on the fly should not be used with earlier versions of the ***PIC-SERVO***.

Start Motion

No issues. This command is identical for all version.

Set Gain

Forward compatibility - The Servo Rate Divisor is used differently by the ***PIC-SERVO SC*** than in earlier versions. Please see Section 4.3 for details. Older Set Gain commands will not set the value Step Multiplier, and the ***PIC-SERVO SC*** will leave it at the default value of 1.

Backward compatibility - Older versions of he ***PIC-SERVO*** will ignore the Step Multiplier byte sent with the new Set Gain command.

Stop Motor

With older versions, the state of the AMP_ENABLE pin was determined solely by the value set with the Stop Motor command, and therefore, the AMP_ENABLE signal could be used as either an active high or active low signal. The ***PIC-SERVO SC*** requires that the AMP_ENABLE signal be active high because it is altered by the overvoltage protection.

Also, older versions of the ***PIC-SERVO*** would set the velocity parameter to zero whenever a Stop Motor command was executed, requiring the velocity parameter to be reloaded with the Load Trajectory command. The ***PIC-SERVO SC*** buffers the value of the velocity parameter, eliminating the need to reload the velocity parameter unless you need to change it.

Lastly, the ***PIC-SERVO CMC*** used Bit 5 of the Stop Motor control byte to enable the advanced features of v.5 (it defaulted to strict compatibility mode). The ***PIC-SERVO SC*** does not have a compatibility mode, and it ignores Bit 5.

You should examine your code for each of these issues to insure forward or backward compatibility.

I/O Control

Earlier versions of the ***PIC-SERVO*** only used the I/O Control command for optionally programming the LIMIT inputs as output, and most applications did not use this feature at all. This option, however, involved the risk of the software inadvertently defining an input as an output, with the potential for damaging the hardware. The ***PIC-SERVO SC***, therefore, eliminated this feature. The ***PIC-SERVO SC*** now uses the I/O Control command to set a variety of other options. Please see the I/O Control command description in Section 5.2 to ensure forward or backward compatibility.

Set Homing

No issues. This command is identical for all version.

Set Baud

Forward compatibility - No issues. All old Baud rate values will work with the ***PIC-SERVO SC***.

Backward compatibility - The Baud rate of 230,400 is only valid for ***PIC-SERVO SC*** controllers. If your system uses a mix of controllers, the fastest Baud rate you can use is 115,200.

Clear Bits

No issues. This command is identical for all version.

Save as Home

No issues. This command is identical for all version.

Add Path Points

Forward compatibility - No Issues.

Backward compatibility - The ***PIC-SERVO SC*** has a path point buffer which can hold 128 points, whereas the ***PIC-SERVO CMC*** can only hold 96 path points. If you are using a mix of controllers, you should limit the number of path points used to 96.

No Op

No issues. This command is identical for all version.

Hard Reset

Forward compatibility - No issues.

Backward compatibility - No issues, older versions will ignore the optional control byte for the ***PIC-SERVO SC*** if sent.

Status Byte

Older versions of the ***PIC-SERVO*** would set the POWER_ON bit if the voltage on the ENC_SEL pin was above about 2.0v. The ***PIC-SERVO SC*** uses a dedicated VOLT_SENSE pin and POWER_ON bit will only be set if the voltage on this pin is between 0.9 and 4.5v.

Older versions of the ***PIC-SERVO*** would set the POS_ERROR bit whenever the servo was disabled, but would not continuously clear this bit while the servo remained disabled. The ***PIC-SERVO SC,*** however, will continue to set the POS_ERROR bit while the servo is disabled. You must first enable the servo and then use the Clear Bits command to clear the POS_ERROR bit.

Auxiliary Status Byte

The ACCEL and SLEW bits have different meanings in the ***PIC-SERVO SC***. Also the PATH_MODE bit is only defined for the ***PIC-SERVO CMC*** and ***PIC-SERVO SC***.

6.0 Example Applications

6.1 PIC-SERVO SC With RS232 Communications

The following diagram shows the ***PIC-SERVO SC*** configured with RS232 communications and an integrated amplifier circuit. The LMD18200 amplifier chip will source up to 3 amps (continuous) for driving a conventional brush-type DC motor. The current sense resistor R7 will produce a voltage signal of approximately 1.0 volts per amp. You should also note that the BRAKE input of the LMD18200 requires that amplifier enable signal to be inverted in order to enable the amplifier.

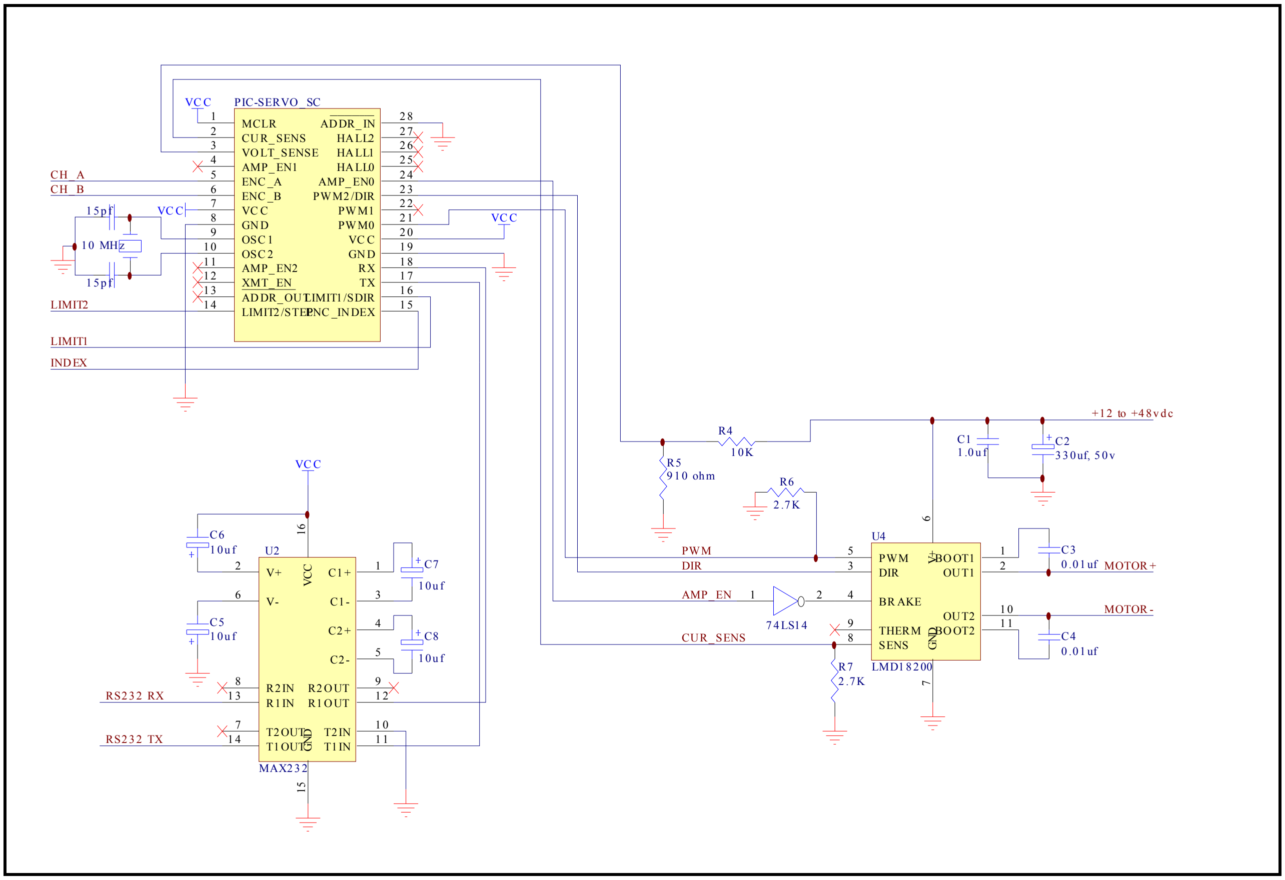

Figure 7 - ***PIC-SERVO SC*** controller with RS232 communications

6.2 Converting Antiphase PWM Output to a +/- 10V Signal

Figure 8 below is an example of how to convert the ***PIC-SERVO SC***'s Antiphase PWM signal to an analog +/-10v signal commonly used by industrial servo amplifiers. Note that the LM358 op amp is used in an inverting configuration so that a 5v input signal (100% duty cycle) will give a -10v output and 0v input (0% duty cycle) will give a +10v output. You can effectively invert the sign of the output by switching the polarity of your motor when you connect it to the servo amplifier. (If you are using a brushless motor, it may be easier to swap the Channel A and B encoder signals, as necessary, to make the position feedback signal match the polarity of the drive system.)

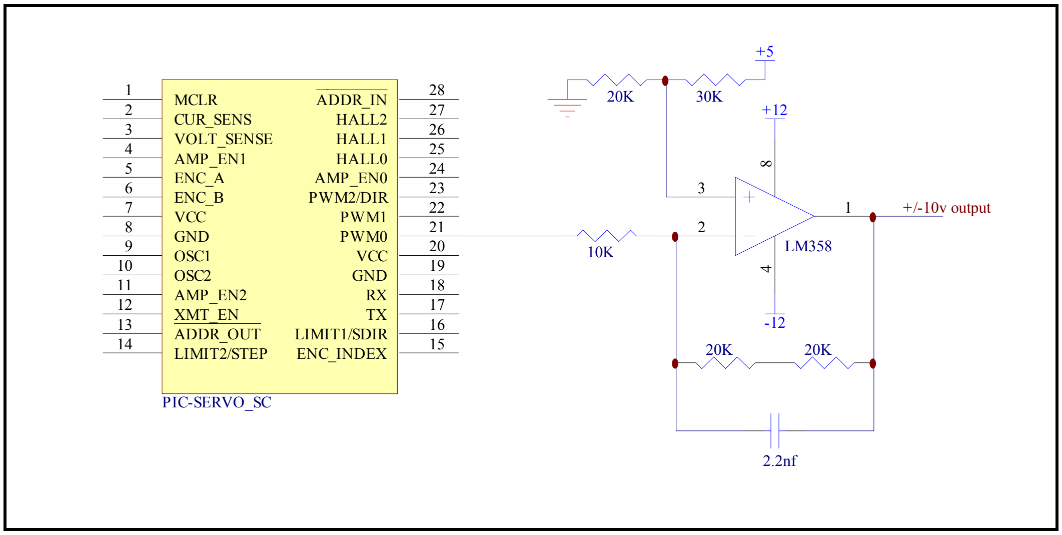

Figure 8 - Converting Antiphase PWM to +/- 10V

6.3 Simplified 3-Phase Motor Driver

Figure 9 below is a simplified 3-Phase driver for small brushless motors. The L293D is capable of driving 0.6 amps per output. The circuit below parallels two of the outputs (1 & 2 and 3 & 4) to double the drive current to 1.2 amps per phase. The corresponding inputs are also driven in parallel from the ***PIC-SERVO SC***'s PWM outputs. Note that this circuit has no current sensing. Please see the L293D data sheet from ST Microelectronics for more details on using the L293.

Note that it is also possible to use this circuit for driving brush-commutated motors. Simply leave the hall sensor inputs unconnected and eliminate the second L293D. You will still want to set the ***PIC-SERVO SC*** for 3-phase output mode to make it put out the proper PWM and ENABLE signals to the driver chips.

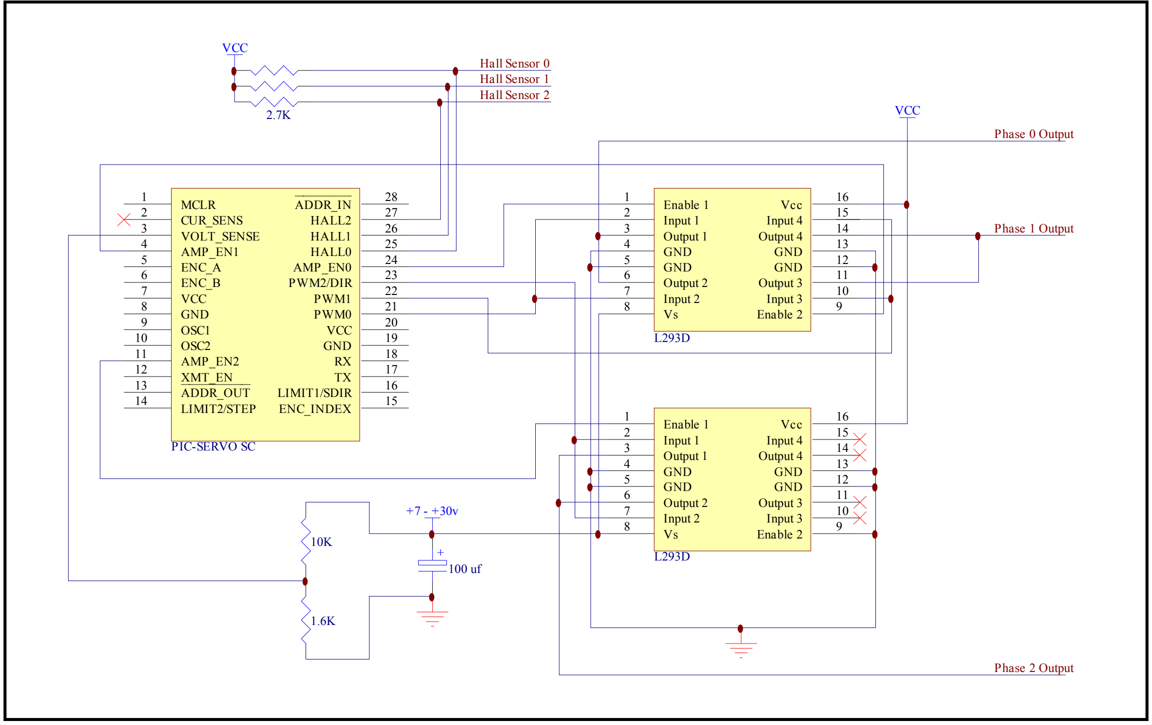

Figure 9 - Simplified 3-Phase driver

7.0 Other References

The following Companies' Web sites may provide useful information and data sheets for developing complete motor control systems using the ***PIC-SERVO SC***:

Microchip **www.microchip.com**
The ***PIC-SERVO SC*** is based on the Microchip PIC8F2331 microcontroller Please refer to the Microchip data sheets for this device for complete electrical, timing, dimensional and environmental specifications.

National Semiconductor **www.national.com**
Data sheets for the LMD18200 / LMD18201 PWM amplifiers, featured in the ***PIC-SERVO*** application notes.

Linear Technology Datasheets **www.linear.com**
Data sheets for the LTC491 RS485 transceiver as well as other interface I.C.'s.

Maxim **www.maxim-ic.com**
Data sheets for the MAX232 RS232 transceiver as well as other interface I.C.'s.

HdB Electronics **www.hdbelectronics.com**
Carries the complete line of ***PIC-SERVO*** products as well as other electronic components, accessories and tools.

JEFFREY KERR, LLC **www.jrkerr.com**
Application notes, new products, and useful links can be found at this site. Technical support is provided via e-mail with contact information on the web page: *www.jrkerr.com/contact.html*.