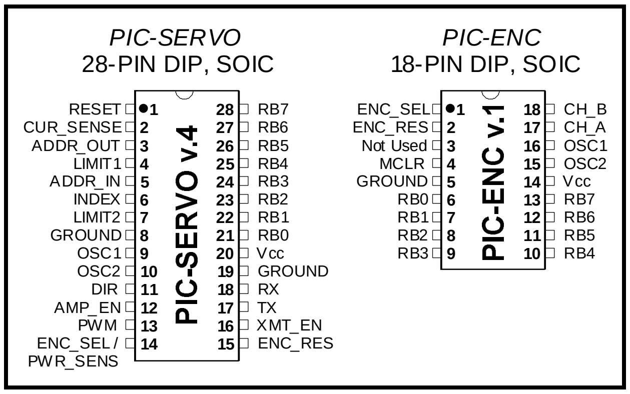
J R KERR
AUTOMATION
ENGINEERING

PIC-SERVO / PIC-ENC

Servo Motion Control Chipset

- D.C. Motor servo control for motors with incremental encoder feedback
- Serial interface connects to RS232 or RS485 communications ports
- Firmware supports multi-drop RS485 network for multi-axis systems
- Position control, velocity control, trapezoidal profiling
- 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
- Two-chip set based on the PIC16Cxx series microcontrollers



1.0 Overview

The ***PIC-SERVO*** and the ***PIC-ENC*** form a two-chip set for the servo control of D.C. motors with incremental encoder feedback. The ***PIC-SERVO*** is a PIC16C73 microcontroller programmed with a PID servo control filter, trapezoidal and velocity profiling and a serial command interface. It interfaces with the ***PIC-ENC***, a PIC16C54 microcontroller programmed as a 16 bit incremental encoder counter. With the ***PIC-ENC*** connected to the incremental encoder mounted on a D.C. servo motor, the ***PIC-SERVO*** will produce PWM and Direction outputs which can be fed to a servo amplifier to drive the motor. The position or velocity of the motor will be controlled according to the servo and motion control parameters programmed through the serial interface.

The serial interface, compatible with most 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 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 single shared response line. Unique device addresses are dynamically assigned, eliminating the need for setting dip-switches.

• • • CAUTION • • •
The ***PIC-SERVO*** and ***PIC-ENC*** are not warranted as fail-safe devices. As such, they should not be used in life support systems or in other devices where their failure or possible erratic operation could cause bodily injury or loss of life.

2.0 Pin Description and Packaging

2.1 PIC-SERVO

The ***PIC-SERVO*** comes in a 28 pin, 0.3" DIP or an SOIC package. Generally, the device operates from a +5v supply and is compatible with TTL and CMOS logic. Please refer to the PIC16C73 data sheet from Microchip (*see* Section 7.0) for complete electrical and physical specifications.

Pin	Symbol	Description
1	MCLR	Reset pin, active low. Connects directly to Vcc for automatic reset on powerup.
2	CUR_SENSE	Analog input for current sensing or for use as a general analog input. (0 - +5v)
3	ADDR_OUT	This output is high on powerup and goes low when the address of the chip has been programmed to a unique value. It is used in conjunction with the ADDR_IN input of the next ***PIC-SERVO*** to dynamically program unique device addresses.
4	LIMIT1	General purpose I/O bit, normally used as a limit switch input.
5	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.
6	INDEX	Normally connects to the INDEX output of the incremental encoder and is used for homing purposes. It can also be used as a general purpose input.
7	LIMIT2	General purpose I/O bit, normally used as a limit switch input.
8	GND	Ground connection.
9	OSC1	Connects directly to a 20 MHz clock source or to one side of a 20 MHz crystal. See Microchip documentation for details of crystal connections.
10	OSC2	Connects to the other side of a 20 MHz crystal. N.C. if a clock source is used.
11	DIR	Direction control output bit. Used with the PWM output to control a servo amp.
12	AMP_EN	Amplifier enable output, set high (or low as needed) to enable a servo amplifier. Can also be used as a general purpose output bit.
13	PWM	20 KHz (approx.) square wave of varying percentage duty-cycle, used with DIR for controlling a servo amplifier.
14	ENC_SEL / PWR_SENS	This output is connected to the ENC_SEL on the ***PIC-ENC*** to select the high byte or the low byte of the 16 bit counter. Logic 1 selects the high byte, logic 0 selects the low byte. This pin is also monitored as an input to detect the presence of motor power by connecting it to the motor supply through a high impedance voltage divider. Otherwise, it should be tied to +5v through a 10k resistor.
15	ENC_RES	Connects to the ENC_RES pin of the ***PIC-ENC***. Raising this output resets the count to zero on the ***PIC-ENC***. Lowering this output allows normal counting to resume.
16	XMT_EN	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.
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 pin, connects to +5v D.C.
21-28	RB0-7	Encoder data inputs. Connects directly to the ***PIC-ENC*** which outputs the high or low byte of its 16 bit counter.

2.2 PIC-ENC

The ***PIC-ENC*** comes in an 18 pin, 0.3" DIP or an SOIC package. Generally, the device operates from a +5v supply and is compatible with TTL and CMOS logic. Please refer to the PIC16C54 data sheet from Microchip (*see* Section 7.0) for complete electrical and physical specifications.

Pin	Symbol	Description
1	ENC_SEL	When this input is low, the lower 8 bits of the 16 bit count will be output on RB0-7. When high, the upper 16 bits will be output.
2	ENC_RES	When this input is high, the internal count will be set to zero, and a zero value will be output on RB0-7. Lowering this input allows normal counting to resume.
3	Not Used	Tie this pin to Vcc.
4	MCLR	Reset pin, active low. Connected directly to Vcc for automatic reset on powerup.
5	GND	Ground connection.
6-13	RB0-7	Encoder count output data. Either the high byte or the low byte of the 16 bit encoder counter. The data is not latched and should be read twice to ensure its validity.
14	Vcc	Supply pin, connects to +5v D.C.
15	OSC2	Connects to the other side of a 20 MHz crystal from OSC1. Left open if an external clock source is used.
16	OSC1	Connects directly to a 20 MHz clock source or to one side of a 20 MHz crystal. See Microchip documentation for details of crystal connections.
17	CH_A	Encoder channel A input.
18	CH_B	Encoder channel B input.

2.3 Ordering Information

Part Number	***Description***
KAE-T0V4-DPS	***PIC-SERVO / PIC-ENC*** chipset, version 4, 0.3” wide DIP package
KAE-T0V4-SOS	***PIC-SERVO / PIC-ENC*** chipset, version 4, SOIC package
KAE-T0V4-DP	***PIC-SERVO*** chip only, version 4, 0.3” wide DIP package
KAE-T0V4-SO	***PIC-SERVO*** chip only, version 4, SOIC package
KAE-PEV1-DP	***PIC-ENC*** chip only, 0.3” wide DIP package
KAE-PEV1-SO	***PIC-ENC*** chip only, SOIC package

3.0 Electrical & Timing Specifications

All timing specifications are based on using a 20 MHz clock source or crystal for the ***PIC-SERVO*** and the ***PIC-ENC***. In fact, any clock between 10 MHz and 20 MHz may be used, with the timing specifications adjusted proportionally. Alternate clock frequencies may be desirable to for getting a better match with a particular communications baud rate. Please refer to the "set baud" command in Section 5.2 and the Microchip PIC16C7x data sheet for further information.

3.1 PIC-SERVO

The ***PIC-SERVO*** is a 5v device with CMOS inputs compatible with either CMOS or TTL logic levels. The CMOS outputs can drive up to 20 ma each. Please refer to the PIC16C7x data sheet from Microchip (see Section 7.0) for complete electrical specifications.

The following timing rates are of interest:

Servo rate:	1953.12 Hz (max.)
Serial baud rate:	9600 - 115200 baud (faster rates are possible at lower servo rates)
PWM frequency:	19531.2 Hz (fixed)
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 programmable. The minimum number of bytes in a command packet is 4 and the minimum number of bytes in a status packet is 2. Furthermore, a response packet will not be sent until the end of a servo cycle, introducing as much as a 0.51 millisec delay between the receipt of a command packet and the transmission of a status packet.

3.2 PIC-ENC

The ***PIC-ENC*** is a 5v device with CMOS inputs compatible with either CMOS or TTL logic levels. The CMOS outputs can drive up to 20 ma each. Please refer to the PIC16C5x data sheet from Microchip (see Section 7.0) for complete electrical specifications.

The ***PIC-ENC*** accepts two square wave inputs, CH_A and CH_B from an incremental encoder. Ideally, these square waves are 50% duty cycle and exactly +/-90 degrees out of phase. In any case, the time between encoder state transitions should be no less than 2 usec. With ideally formed encoder pulses, this would correspond to a 500 line encoder (2000 counts/rev) rotating at 15,000 RPM.

The other critical timing constraint is the delay between changing ENC_SEL and when the new data byte is valid. With the ***PIC-ENC*** clocked at 20 MHz, the count data should not be read until 2.8 usec after ENC_SEL is changed.

When resetting the count by raising ENC_RES, the signal should be held high for a minimum of 2.8 usec. After returning ENC_RES low, counting will not resume for another 2.8 usec.

Encoder transition rate:	500 KHz (max.)
ENC_SEL delay:	2.8 usec (max.)
ENC_RES pulse width:	2.8 usec (min.)
Data valid from reset:	2.8 usec (max.)

4.0 Theory of Operation

4.1 Communications & Initialization

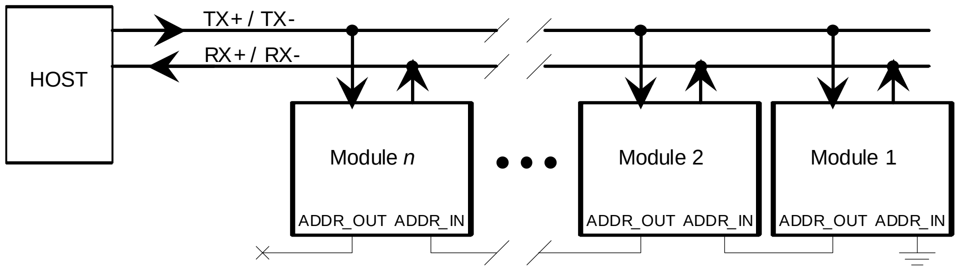
NMC Communication Protocol

The ***PIC-SERVO*** uses the same *full-duplex*, RS232 or RS485 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 115,200. The communication protocol uses 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 signal level converter*. The slave ports, however, must be able to

* 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 XMT_EN output is not used and may be left open.

disable their transmitters to prevent data collisions over the shared status line. Therefore, all NMC compatible controllers provide an XMT_EN output used for enabling or disabling an RS485 transmitter. Please refer to the sample schematic in Section 7 and to the figure below.



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 thru 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. 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.)

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, input bit 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. No new command should be sent until the status packet is received 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 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 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 Move) 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 modules 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 (one 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 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.

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 16 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.
3. Use the Set Address command, as described in Section 3.2, to assign unique individual addresses to each module. At this point, set all group addresses to 0xFF, and do not declare any group leaders.
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.

[†] For most basic applications which do not use group commands or faster baud rates, only steps 1 and 3 are really required.

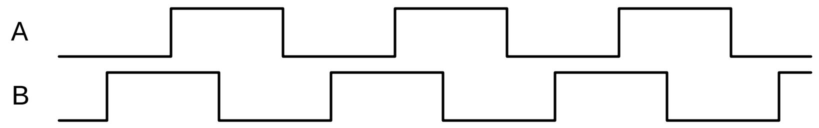
6. Send a Set Baud command to the group address 0xFF to change the baud rate to the desired value. No status will be returned.
7. Change the host's Baud rate to match the rate just specified.
8. Poll each of the individual modules (using a No Op command) to verify that all modules are operating properly 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 group addresses.

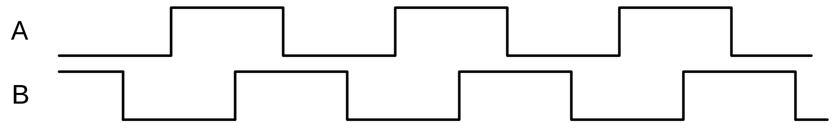
4.2 Incremental Encoder Counting

A typical two-channel incremental 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.

Forward motion:



Reverse motion:



A 500 line encoder, for example, 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-ENC*** performs the time critical task of detecting edge transitions and incrementing (or decrementing if reversing) its internal 16 bit counter. The ENC_SEL input is used to specify whether the high or low byte of the counter appears on output pins RB0-7.

The ***PIC-ENC*** was designed specifically for use with ***PIC-SERVO***, but may be used by itself. However, the ***PIC-ENC*** is an extremely stupid device which does no latching of the data, and therefore, pins RB0-7 must be read twice to make sure that the pins have not been read while in transition, or while the low byte is rolling over to the high byte. In general, the procedure for reading the ***PIC-ENC*** is as follows:

1. Set ENC_SEL to read the high byte
2. Wait 2.8 usec for the data to change
3. Read the high byte
4. Lower ENC_SEL to read the low byte
5. Wait 2.8 usec for the data to change
6. Read the low byte
7. Read the low byte again and compare
8. If low byte values are different, go to 6
9. Set ENC_SEL to read the high byte again
10. Wait 2.8 usec for the data to change
11. Read the high byte again and compare

12. If high byte values are different, go to 3

Steps 6 and 7 should be performed in less than 2 usec, the maximum encoder rate. The entire procedure should be performed in less than 500 usec, the time it would take the low byte to roll over at the maximum encoder rate. In addition, the 16 bit counter will automatically wrap around on overflow or underflow, and therefore must be read often enough to detect these conditions.

4.3 PID Servo Control

In general, in position or velocity mode, the motor is controlled by a servo loop which once every servo tick (1953 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:

output = Kp(pos_error) - Kd(pos_error - prev_pos_error) + Ki(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 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 user specified integration limit. (Note that some other controllers will bound the value of the integral_error, but leave the actual running sum to grow unbounded, causing greater integral error windup.) 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:

PWM = min(abs(output/256), output_limit)) - current_limit_adjustment
Dir = 0 if output>0, Dir = 1 if output < 0

First note that the scaled PWM output is limited by a 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. Under normal operation, current_limit_adustment = 0. If the motor current, as indicated by the A/D value, exceeds a user specified limit, current_limit_adjustment is incremented by 1 each servo tick, up to a maximum value of *min(abs(*output/256*)*, output_limit*)*. If the motor current is below the specified limit, current_limit_adjustment is decremented by 1, down to a minimum value of zero. This incremental adjustment is used rather than a proportional adjustment due to the non-linearity of many current sensing schemes, and in fact can be used with external amplifiers which provide only a binary current threshold value.

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%.

One last control parameter is the user specified 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. (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 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.

The default (and maximum) servo rate is approximately 2 KHz (1.953 KHz, to be more exact). 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. Fortunately, such systems typically have a rather slow response and the servo rate can be decreased considerably. For example, switching from 2 KHz to 200 Hz will allow you to achieve the same level of damping with a value of Kd/10. The minimum possible servo rate is 7.6 Hz.

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 (SR). The details of programming these values appear in the Section 5.2 under the description of the "Set Gain" command.

4.4 Trapezoidal and Velocity Profiling

The trapezoidal and velocity profiling functions are used to automatically generate a smooth trajectory for the motor, always limiting the motor acceleration and deceleration to the acceptable programmed value. Both the position and velocity profiling modes operate by calculating where the motor should be at each servo tick, forming the current *command position* for the motor. The PID control filter then operates on this command position to produce the appropriate PWM output value.

In position mode, the motor trajectory follows what is known as a trapezoidal profile. When a motion is started, the motor will accelerate up to the programmed peak velocity at a constant acceleration which is also programmable. It will then slew at the maximum velocity until it nears the destination and begins to decelerate at a constant acceleration. When the motor reaches its goal position, the motor controller will continue to servo the motor to the specified goal position. When commanding a trapezoidal profile motion, the motor should *always* start at zero velocity and the move will end at zero velocity. While in position mode, changing the velocity, acceleration, or goal position before the motor has reached the original goal position will cause erratic and potentially damaging motion of the motor or connected mechanism.

For short motions or motions with very low accelerations, the motor may never reach its peak velocity before it needs to begin decelerating. In this case, the velocity vs. time profile will be triangular instead of trapezoidal.

If the velocities or accelerations of the motor needs to be changed dynamically, the controller should be operated in velocity mode. In this mode, the command position is incremented each servo tick by the velocity profile generator. If the motor is going too slowly, the velocity will be increased at the acceleration rate specified until the goal velocity is reached, or if it is going to fast, the velocity will be decreased at the acceleration rate until the goal velocity is reached. The goal velocity and acceleration may be changed at any time. (In fact, velocity mode may be entered while in the *middle* of a trapezoidal move.) Note that because the *command position* is constantly being changed to effect the desired velocity, the average velocity will have virtually no error and the motor's position will always be equal to the exact integral of the command velocity, give or take the normal position error.

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 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 straight 32 bit quantity with no fractional component.

Note that if the servo rate divisor is modified, the time dependent velocity and acceleration parameters will also have to be modified.

4.5 Odds and Ends

PWM Mode Operation

If the position servo is disabled, the motor is operated in a raw PWM output mode and no trapezoidal or velocity profiling is performed. In this mode, a user changeable PWM value is output directly to the amplifier. Normally, this value will simply be set to zero. If the position servo is terminated automatically by a loss of power, excess position error, or turned off by the “stop motor” command, the PWM value will default to zero. During experimentation or startup, however, it may be useful to send a non-zero output directly to the amplifier. Note that in specifying a PWM value directly, the current limiting of an external amplifier may still be performed, but the PWM output limit is ignored.

While the position servo is disabled, the command position is continually updated to match the actual position of the motor. Thus, when position or velocity modes are entered, there can 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. (Trapezoidal profile motions, however, will still force the motor to begin at zero velocity.)

Motor Power Monitoring

The state of the motor power (on or off) can be detected by connecting the motor supply to the ENC_SEL pin through a high impedance voltage divider (see Section 6.0). The resistor values should be selected so that with the maximum motor voltage applied, the voltage at the pin will not exceed 5v. The resistors should also be large enough so that when the ***PIC-SERVO*** is driving ENC_SEL as an output (0 - 5v), it will never have to source or sink more than 10 ma. If motor power sensing is not desired, simply tie ENC_SEL to +5v through a 10k resistor.

The ***PIC-SERVO*** will automatically monitor the state of ENC_SEL as an input (when it is not reading the ***PIC-ENC***) and disable the position servo should the motor power ever go out. This prevents the motor from lurching should the motor power be re-enabled without the host knowing about it. The motor power bit of the status byte indicates the state of the motor power, and the pos_error bit will also be set to indicate that the servo has been terminated. The host should be sure to monitor these status bits before inadvertently issuing additional move commands in the event motor power is unknowingly re-enabled after having been shut off.

Powerup and Reset Conditions

On powerup or reset, the following state is established:

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 (1.953 KHz servo rate)
The PWM value is set to zero
The controller is placed in PWM mode
AMP_EN is set low
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 disables 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.

5.0 Command Specification

5.1 List of Commands

Command	CMD Code	# Data bytes	Description	While Moving?
reset position	0x0	0 or 1	Sets position counter to zero.	no
set address	0x1	2	Sets the individual and group addresses	yes
define status	0x2	1	Defines which data should be sent in every status packet	yes
read status	0x3	1	Causes particular status data to be returned just once	yes
load trajectory	0x4	1-14	Loads motion trajectory parameters	maybe[1]
start motion	0x5	0	Executes the previously loaded trajectory	maybe[2]
set gain	0x6	14	Sets the PID gains and operating limits	yes
stop motor	0x7	1	Stops the motor in one of three manners	yes
I/O control	0x8	1	Sets the direction and values of the LIMIT pins	yes
set home mode	0x9	1	Sets conditions for capturing the home position	yes
set baud rate	0xA	1	Sets the baud rate (group command only)	yes
clear bits	0xB	0	Clears the sticky status bits	yes
save as home	0xC	0	Saves the current position in the home position register	yes
---	0xD	---	Reserved for future use	---
nop	0xE	0	Simply causes the defined status data to be returned	yes
hard reset	0xF	0	Resets the controller to its powerup state.	yes

5.2 PIC-SERVO Command Description

Reset Position

Command value:	0x0
Number of data bytes:	0
Command byte:	**0x00**

Description:

Resets the 32 bit encoder counter to 0. Also resets the internal command position to 0 to prevent the motor from jumping abruptly if the position servo is enabled. Do *not* issue this command while executing a trapezoidal profile motion

Set Address

Command value:	0x1
Number of data bytes:	2
Command byte:	**0x21**

Data bytes:

1. Individual address: 0-0xFF (initial value 0x00)
2. Group Address: (initial value 0xFF)

[1]Only allowed while moving if the "start motion now" bit of the trajectory control word is not set or if the "profile mode" bit is set for velocity mode.

[2]Only allowed while moving if the previously loaded trajectory has the "profile mode" bit set for velocity mode.

Description:

Sets the individual address and group address. Group addresses are always interpreted as being between 0x80 and 0xFF. If a ***PIC-SERVO*** is to be a group *leader*, clear bit 7 of the desired group address in the second data byte; the ***PIC-SERVO*** will automatically set bit 7 internally after flagging the ***PIC-SERVO*** as a group leader. (If bit 7 of the second data byte is set, the module will default to being a group *member*.) The first time this command is issued after power-up or reset, it will also enable communications for the next module in the network chain by lowering the its ADDR_OUT signal.

Define Status

Command value:	0x2
Number of data bytes:	1
Command byte:	**0x12**

Data bytes:

1. Status items: (default: 0x00)
   - Bit 0: send position (4 bytes)
   - 1: send A/D value (1 byte)
   - 2: send actual velocity (2 bytes - no fractional component)
     The signed 16 bit integer returned is the negative of the velocity in units of counts per servo tick
   - 3: send auxiliary status byte (1 byte)
   - 4: send home position (4 bytes)
   - 5: send device ID, version number (2 bytes)
     (***PIC-SERVO*** device ID = 0)
   - 6, 7: not used - clear to zero

Description:

Defines what additional data will be sent in the status packet along with the status byte. Setting bits in the first data byte will cause the corresponding additional data bytes to be sent after the status byte. The status data will always be sent in the order listed. For example if bits 0 and 3 are set, the status packet will consist of the status byte followed by four bytes of position data, followed by the aux. status byte, followed by the checksum. The status packet returned in response to *this* command will include the additional data bytes specified. On power-up or reset, the default status packet will include only the status byte. (See section 5.3 for a definition of the status byte and the auxiliary status byte.)

Read Status

Command value:	0x3
Number of data bytes:	1
Command byte:	**0x13**

Data bytes:

1. Status items:

Bit 0:	send position (4 bytes)
1:	send A/D value (1 byte)
2:	send actual velocity (2 bytes - no fractional component) *The signed 16 bit integer returned is the negative of the velocity in units of counts per servo tick*
3:	send auxiliary status byte (1 byte)
4:	send home position (4 bytes)
5:	send device ID, version number (2 bytes) (***PIC-SERVO*** device ID = 0)
6, 7:	not used - clear to zero

Description:

This is a non-permanent version of the "define status" command. The status packet returned in response to *this* command will incorporate the data bytes specified, but subsequent status packets will include only the data bytes previously specified with the “define status” command.

Load Trajectory

Command value:	0x4
Number of data bytes:	*n* = 1-14
Command byte:	**0x*n*4**

Data bytes:

1. Control byte:

Bit 0:	load position data (*n* += 4 bytes)
1:	load velocity data (*n* += 4 bytes)
2:	load acceleration data (*n* += 4 bytes)
3:	load PWM value (*n* += 1 bytes)
4:	servo mode - 0 = PWM mode, 1 = position servo
5:	profile mode - 0 = trapezoidal profile, 1 = velocity profile
6:	in velocity & PWM modes: 0 = FWD direction, 1 = REV direction
7:	start motion now

Description:

All motion parameters are set with this command. Setting one of the first four bits in the control byte will require additional data bytes to be sent (as indicated) in the order listed. The position data (range[*] +/- 0x7FFFFFFF) is only used as the goal position in trapezoidal profile mode. The velocity data (range 0x00000000 to 0x7FFFFFFF) is used as the goal velocity in velocity profile mode or as the maximum velocity in trapezoidal profile mode. The acceleration data (range 0x00000000 to 0x7FFFFFFF) is used in both trapezoidal and

[*] While the position may range from -0x7FFFFFFF to +0x7FFFFFFF, the goal position should not differ from the current position by more then 0x7FFFFFFF.

velocity profile mode. The PWM value (range 0-0xFF), used only when the position servo is not operating, sends a raw PWM values directly to the amplifier. The PWM value is reset to 0 internally on any condition which automatically disables the position servo. Bit 4 of the control byte specifies whether the position servo should be used or if the PWM mode should be entered. Bit 5 specifies whether a trapezoidal profile motion should be initiated or if the velocity profiler is used. Trapezoidal profile motions should only be initialized when the motor velocity is 0. (Bit 0 of the status byte indicates when a trapezoidal profile motion is complete, or in velocity mode, when the command velocity has been reached.) Bit 6 indicates the velocity or PWM direction when velocity or PWM modes are selected. If bit 7 is set, the command will be executed immediately. If bit 7 is clear, the command data will be buffered and it will be executed when the "start motion" command is issued. *Example: To load only new position data and acceleration data but not start the motion yet, the command byte would be 0x94, the control byte would be 0x15, followed by 4 bytes of position data (least significant byte first), followed by 4 bytes of acceleration data.*

Start Motion

Command value:	0x5
Number of data bytes:	0
Command byte:	**0x05**

Description:

Causes the trajectory information loaded with the most recent Load Trajectory command to execute. This is useful for loading several ***PIC-SERVO*** chips with trajectory information and then starting them simultaneously with a group command.

Set Gain

Command value:	0x6
Number of data bytes:	14 (for *version* 4 and higher - only 13 bytes for *versions* 1,2 & 3)
Command byte:	**0xE6** (for *version* 4 and higher - use 0xD6 for *versions* 1,2 & 3)

Data bytes:

1,2. Position gain Kp (0 - 0x7FFF)
3,4. Velocity gain Kd (0 - 0x7FFF)
5,6. Position gain Ki (0 - 0x7FFF)
7,8. Integration limit IL (0 - 0x7FFF)
9. Output limit OL(0 - 0xFF)
10. Current limit CL(0 - 0xFF)
    odd values: CUR_SENSE proportional to motor current
    even values: CUR_SENSE negatively proportional to motor current
11,12. Position error limit EL (0 - 0x3FFF)
13. Servo rate divisor SR (1 - 0xFF)
14. Amplifier deadband compensation (0 - 0xFF) (*version* 4 and higher only)

Description:

Sets all parameters and limits governing the behavior of the position servo. The use of the PID gain parameters (Kp, Kd, Ki, IL, OL) are described in section 4.3. The 16 bit value used to specify IL is multiplied by 256 internally to get the actual IL value used by the PID algorithm. The current limit, CL (used if the CUR_SENSE input is connected to a current sense output of the amplifier) is actually a seven bit value with bit 0 used to indicate if CL is to be used as an upper bound or a lower bound on the CUR_SENSE value. Odd values of CL are used if CUR_SENSE is directly proportional to motor current (0v = 0 amp); even values assume CUR_SENSE is negatively proportional to motor current (5v = 0 amp). Setting CL to 0 effectively disables current limiting. The position error limit will cause the position servo to be disabled should the position error grow beyond the limit. The servo rate divisor sets the servo tick time to be a multiple of 0.51 msec (1.953 KHz). For example SR=3 gives a servo rate of 651 Hz. The servo tick rate is also used as the profiling timebase, although command processing and current limiting are always performed at the maximum tick rate.

Version 4 of the ***PIC-SERVO*** has a 14th data byte which is used to compensate for the deadband region around zero PWM output exhibited by some amplifier/motor combinations. The deadband compensation value will be added to the magnitude of the PWM output to force the amplifier into its active region. If the command 0xD6 is used (and only 13 data bytes are sent, *version* 4 will use a default value of zero for the deadband compensation. (Also note that *versions* 1,2 & 3 will simply ignore the 14th data byte if it is sent.)

Stop Motor

Command value:	0x7
Number of data bytes:	1 or 5
Command byte:	**0x17 or 0x57**

Data bytes:

1. Stop control byte
   - Bit 0: Amplifier enable
   - 1: Turn motor off
   - 2: Stop abruptly
   - 3: Stop smoothly
   - 4: Stop here (not available on *version* 1)
   - 5,6,7: Clear to zero

2-5 Stopping position (only required if bit 4 above is set)

Description:

Stops the motor in the specified manner. If bit 0 of the Stop Control Byte is set, AMP_EN will be set; if bit 0 is cleared, AMP_EN will be cleared, regardless of the state of the other bits. If bit 1 is set, the position servo will be disabled, the PWM output value will be set to 0, and bits 2, 3 and 4 are ignored. If bit 2 is set, the current command velocity and the goal velocity will be set to zero, the position servo will be enabled, and velocity mode will be entered. If the velocity servo was previously disabled, the motor will simply start servoing to its current position. If the motor was previously moving in one of the profiling modes, it will stop moving abruptly and servo to its current position. This stopping mode should only be used as an emergency stop where the motor position needs to be maintained. A more graceful stop mode is entered by setting bit 3 - this sets the goal velocity to 0 and enters velocity mode, causing the motor to decelerate to a stop at the current acceleration rate. *If bit 4 is set, the motor will move to the specified stopping position abruptly with no profiling. This mode can be used to cause the motor to track a continuous string of command positions. Note that if the stopping position is too far from the current position, a position error will be generated.* Only one of the bits 1, 2, 3 or 4 should be set at the same time. Note: the Stop Motor command must be issued initially to set AMP_EN (if used) before other motion commands are issued.

I/O Control

Command value:	0x8
Number of data bytes:	1
Command byte:	**0x18**

Data bytes:

1. I/O control byte
   - Bit 0: Output value of Limit1
   - 1: Output value of Limit2
   - 2: Direction of Limit1 (0 = output, 1 = input)
   - 3: Direction of Limit2 (0 = output, 1 = input)
   - 4,5,6,7: Clear to zero

Description:

Controls whether the limit1 and 2 signals are inputs (default) or outputs and sets the output values.

Set Homing Mode

Command value:	0x9
Number of data bytes:	1
Command byte:	**0x19**

Data bytes:

1. Homing control byte
   - Bit 0: Capture home position on *change* of Limit1
   - 1: Capture home position on *change* of Limit2
   - 2: don't care (versions1 & 2)
     Turn motor off on home (version 3 or higher)
   - 3: Capture home on *change* of Index
   - 4: don't care (versions 1 & 2)
     Stop abruptly on home (version 3 or higher)
   - 5: don't care (versions1 & 2)
     Stop smoothly on home (version 3 or higher)
   - 6: Capture home position when an excess position error occurs
   - 7: Capture home position when current limiting occurs

Description:

Causes the controller to monitor the specified conditions and capture the home position when *any* of the flagged conditions occur. The home_in_progress bit in the status byte is set when this command is issued and it is then lowered the home position has been found. With firmware versions 1 & 2, the user must stop the motor explicitly, if necessary, after the home position has been found. In firmware version 3 (or higher) setting one (and only one) of bits 2, 4 or 5 will cause the motor to stop automatically in the specified manner once the home condition has been triggered. This feature can also be used as a safety shutoff.

Set Baud Rate

Command value:	0xA
Number of data bytes:	1
Command byte:	**0x1A**

Data bytes:

1. Baud rate divisor, BRD

sample values:

9600	BRD = 129
19200	BRD = 63
57600	BRD = 20
115200	BRD = 10

Description:

Sets the communications baud rate. All ***PIC-SERVO*** chips on the network must have their baud rates changed at the same time, therefore this command should only be issued to a group including all of the controllers on the network. A status packet returned from this command would be at the new baud rate, so typically (unless the host's baud rate can be accurately synchronized), there should be no group leader when this command is issued. The baud rate divisor is programmed directly into the PIC16C73's SPBRG register with the bit BRGH = 1. Please refer to the PIC16C7x data sheet for details in obtaining other baud rates.

Clear Sticky Bits

Command value:	0xB
Number of data bytes:	0
Command byte:	**0x0B**

Description:

The overcurrent and position error bits in the status byte and the position wrap and servo timer overrun bits in the aux. status byte will stay set unless cleared explicitly with this command.

Save Current Position as Home

Command value:	0xC
Number of data bytes:	0
Command byte:	**0x0C**

Description:

Causes the current position to be saved as the home position. This command is typically issued to a group of controllers to cause their current positions to be stored synchronously. The stored positions can then be read individually by reading the home position

No Operation

Command value:	0xE
Number of data bytes:	0
Command byte:	**0x0E**

Description:

Does nothing except cause a status packet with the currently defined status data to be returned.

Hard Reset

Command value:	0xF
Number of data bytes:	0
Command byte:	**0x0F**

Description:

Resets the control module to its power-up state. No status will be returned. Typically, this command is issued to all the modules on the network.

5.3 Status Byte, Aux. 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 just received command packet
2	overcurrent	Set if current limiting occurred. Must be cleared by user with "clear sticky bits" command.
3	power_on	Set of motor power is greater than 12v. 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 sticky bits" command.
5	limit1	Value of limit switch input 1
6	limit2	Value of limit switch input 2
7	home_in_progress	Set while searching for a home position. Reset to zero once the home position has been captured.

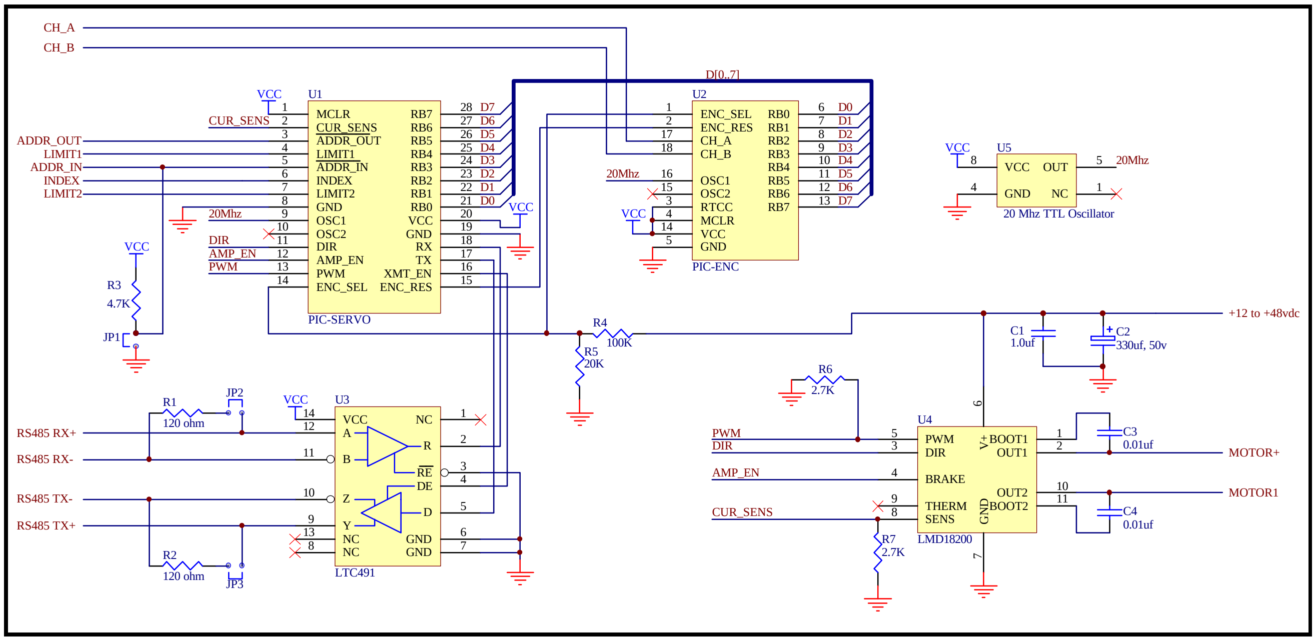
Auxiliary Status Byte

Bit	Name	Definition
0	index	Compliment of the value of the index input.
1	pos_wrap	Set if the 32 bit position counter wraps around. Must be cleared with the Clear Sticky Bits command
2	servo_on	Set if the position servo is enabled, clear otherwise
3	accel_done	Set when the initial acceleration phase of a trapezoidal profile move is complete. Cleared when the next move is started.
4	slew_done	Set when the slew portion of a trapezoidal profile move is complete. Cleared when the next move is started.
5	servo_overrun	At the highest baud rate and servo rate, certain combinations of calculations may cause the servo, profiling, and command processing to take longer than 0.51 msec, in which case, this bit will be set. This is typically not serious, only periodically introducing a small fraction of a millisecond delay to the servo tick time. Cleared with the Clear Sticky Bits command.

6.0 Example Applications

6.1 PIC-SERVO With RS485 Communications

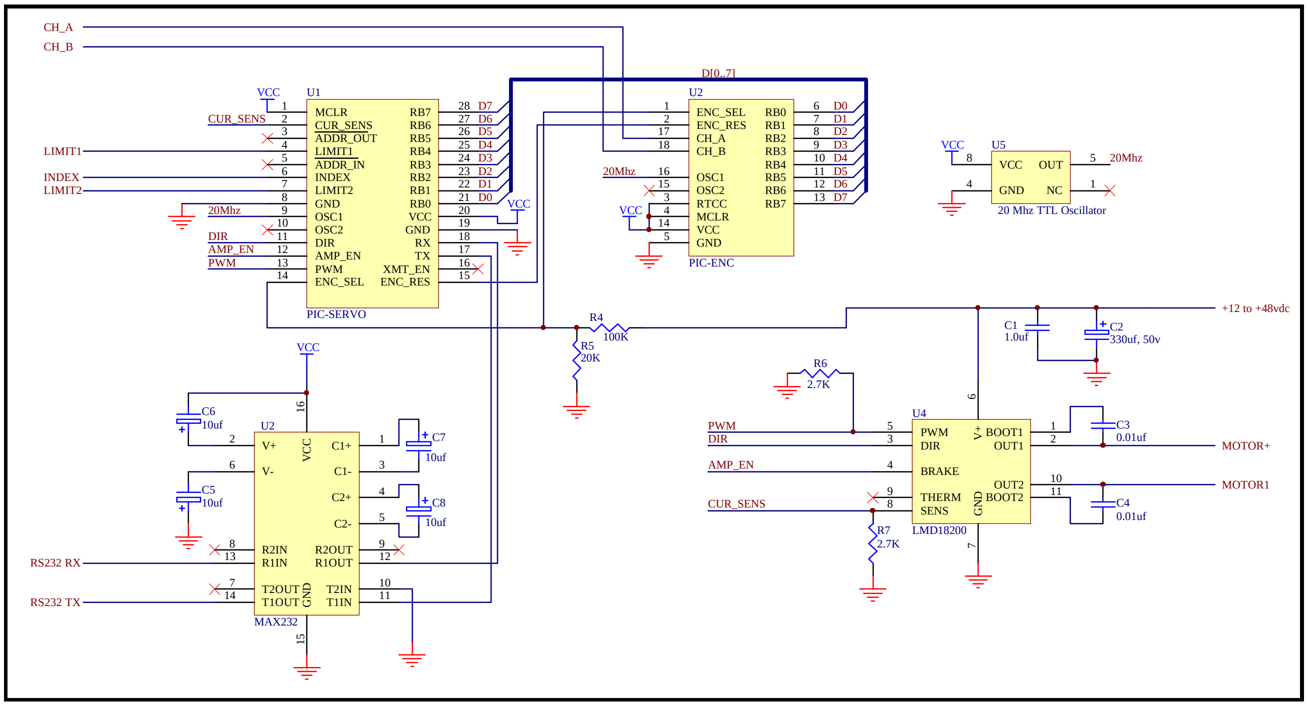
The following diagram shows the ***PIC-SERVO*** chipset configured with full-duplex RS485 communications and an integrated amplifier circuit. The LMD18200 amplifier chip will source up to 3 amps (continuous) for driving a conventional brush-type DC motor. The current sense resistor R7 will produce a voltage signal of approximately 1.0 volts per amp. Addressing jumper JP1, and termination jumpers JP2 and JP3 should be installed on the last ***PIC-SERVO*** controller module on your RS485 network. Up to 32 of the modules shown below may be interconnected on a single RS485 network.



PIC-SERVO With Full-duplex RS485 Communications

6.2 PIC-SERVO With RS232 Communications

The following diagram shows the ***PIC-SERVO*** chipset configured with full-duplex RS232 communications. The same LMD18200 amplifier is used as above. With RS232 communications, only one ***PIC-SERVO*** module can be connected to the host's communications port.



PIC-SERVO With Full-duplex RS232 Communications

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*** and ***PIC-ENC***:

Microchip **www.microchip.com**

The ***PIC-SERVO*** is based on the Microchip PIC16C73 microcontroller and the ***PIC-ENC*** is based on the PIC16C54 microcontroller. Please refer to the Microchip data sheets for these devices 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.

Jameco **www.jameco.com**
Carries the ***PIC-SERVO*** chipset and also a general electronics supplier with a wide variety of parts and excellent service.

Digikey **www.digikey.com**
General electronics supplier with a wide variety of parts and excellent service. They carry all the parts used in the sample applications.

J R Kerr Automation Engineering **www.jrkerr.com**
Application notes, new products, and useful links can be found at this site. Technical support is provided via e-mail. Send your questions to "techsupport@jrkerr.com".