SYNAPTRONμ MOTION CONTROLLER

User Datasheet - Revision 1.1

hardware made easy
Solutions Cubed
designservices@solutions-cubed.com
phone – 530.891.8045
256 East First Street
Chico, CA 95928

Contact Solutions Cubed, LLC for your custom designs:
Solutions Cubed is an innovative electronic design firm. We have created successful designs for a myriad of industries including mass produced consumer products, deep-sea robotic components, and encrypted encoders for the banking industry. We love meeting new customers and are interested in hearing about your design needs.

Revision	Date	Firmware Revisions	Notes
1.1	06/2012	1	original implementation

Always check our web site for most up-to-date documents

Index **Page**

User Datasheet:

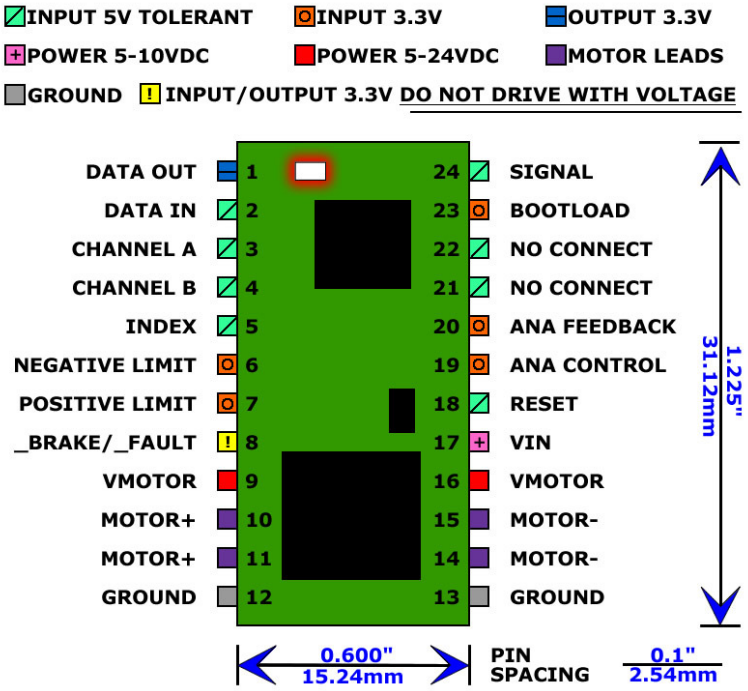
The User Datasheet shows basic connections and provides an overview of functionality, settings, mechanical, and electrical specifications of the module.

Documentation Conventions: Italicized text refers to register names. If a period is used the text following the period is the name of the bit being referenced. For example, *Function* refers to the *Function* register and *Function.ClosedLoopEB* refers to the closed loop error band bit in the *Function* register. Hardware pins are written in all capitals. For example, VMOTOR references pins 9 and 16 on this controller.

1.0 Electrical / System Specifications

Characteristic	Min	Typ	Max	Unit	Notes
Temperature					
Operating temperature	-40		+85	°C	Entire module
Operating temperature	-40		+125	°C	H-bridge IC
Thermal shutdown	+165		+185	°C	H-bridge IC
Voltages					
Motor Voltage	5		24	V	VMOTOR pins 9,16
Controller Voltage	5		10	V	VIN pin 17
Voltage on 3.3V Input	0		3.6	V	Pins 6,7,19,20,23
Voltage on 5V Input	0		5.6	V	Pins 2,3,4,5,18,24
Input Logic Low			0.8	V	
Input Logic High	2.4V				
Input Logic High SIGNAL	3.3V				SIGNAL pin 24
H-bridge under-voltage lockout	4.15		5	V	
Under-voltage hysteresis	0.15	0.20	0.30	V	
Current					
Motor Current Peak			5	A	See Fault Conditions section 3.0
Motor Current Continuous		1.5		A	No cooling, 90% duty cycle, resistive load
Module Operating Current	55	65	75	mA	Current into VIN pin
Short Circuit Detection	9	12	16	A	
Current regulation	5.2	6.5	8	A	H-bridge below 165°C
Current regulation		4.2		A	H-bridge above 165°C
Analog Conversion					
Resolution		3300	4096	bits	Firmware configurable for 12-bit resolution or 1mV per bit
Gain Error	2	11	20	LSb	At full 12-bit setting
Offset Error	2	4	10	LSb	At full 12-bit setting
Non-Linearity	-3		+3	LSb	At full 12-bit setting
System Timing					
PWM Frequency	3		20	kHz	Configured through serial interface
QE Low or High Period	325			nS	Quadrature encoder Cha, CHB, pins 3,4
QE Frequency			1.5	MHz	Quadrature encoder Cha, CHB, pins 3,4
Serial Baud Rate	1200		115,200	bps	Configured through serial interface
Serial Response Time		1	2	mS	
Control Loop Timing	0.25	1	32,767	mS	Configured through serial interface

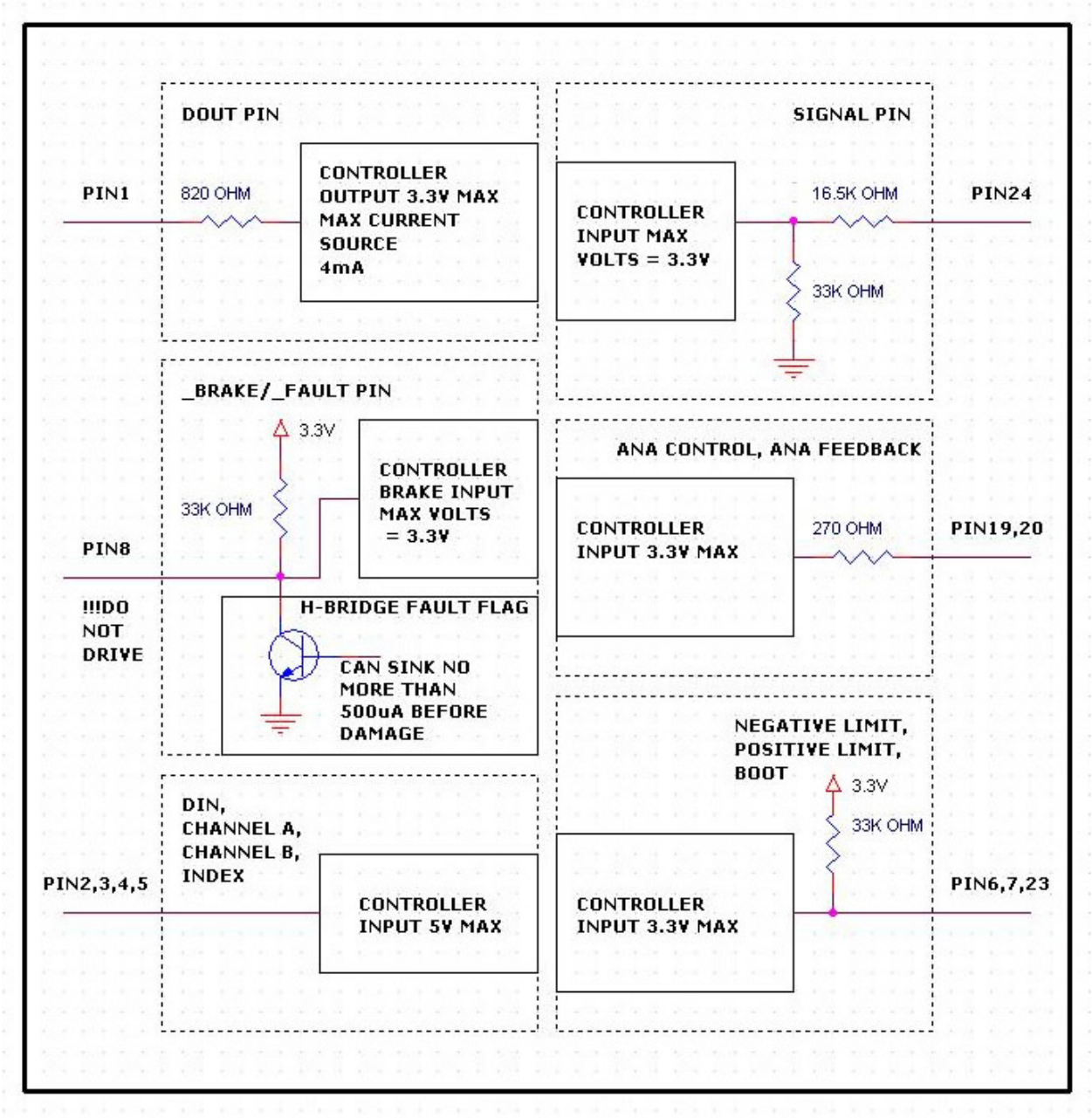
2.0 Connections/Footprint – The module fits in a 0.6" wide 24 pin DIP footprint.



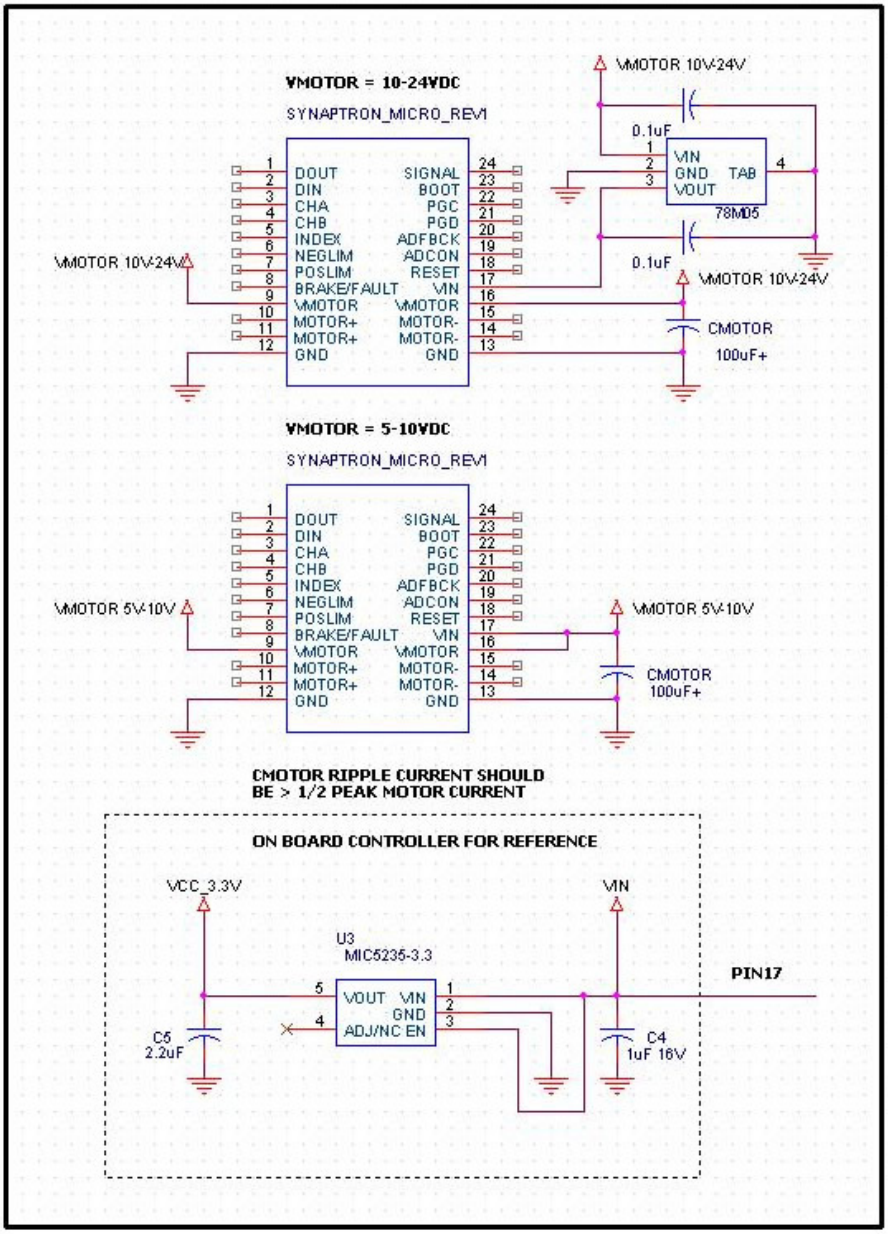
Pin Names and Notes

#	Name	Maximum Voltage	Notes
1	DATA OUT	OUTPUT	3.3V = logic 1, 0V = logic 0. 820Ω series resistor on board.
2	DATA IN	5.5V	Serial data input pin.
3	CHANNEL A	5.5V	CHA connection for quadrature encoders.
4	CHANNEL B	5.5V	CHB connection for quadrature encoders.
5	INDEX	5.5V	INDEX connection for quadrature encoders.
6	NEGATIVE LIMIT	3.6V	Should be connected to dry switch. Pulled to 3.3V through a 33KΩ resistor.
7	POSITIVE LIMIT	3.6V	Should be connected to dry switch. Pulled to 3.3V through a 33KΩ resistor.
8	_BRAKE/FAULT	DO NOT DRIVE	This pin is shared with the fault condition flag of the on-board H-bridge. It should be connected to a dry switch or open-collector circuit if the _BRAKE function is to be used. Pulled to 3.3V through a 33KΩ resistor.
9	VMOTOR	28V	Motor voltage.
10	MOTOR+	OUPUT	Connects to positive lead of the motor.
11	MOTOR+	OUPUT	Connects to positive lead of the motor.
12	GND	0V	Ground return.
13	GND	0V	Ground return.
14	MOTOR-	OUPUT	Connects to negative lead of the motor.
15	MOTOR-	OUPUT	Connects to negative lead of the motor.
16	VMOTOR	28V	Motor voltage.
17	VIN	24V	Powers the logic on board the controller, should be in the 5-10V range
18	RESET	5.5V	Pulled to 3.3V through a 33KΩ resistor. 0V resets processor.
19	ANA CONTROL	3.6V	AD input. 270Ω series resistor on board.
20	ANA FEEDBACK	3.6V	AD input. 270Ω series resistor on board.
21	NO CONNECT	N/A	Leave floating.
22	NO CONNECT	N/A	Leave floating.
23	BOOTLOAD	3.3V	Pulled to 3.3V through a 33KΩ resistor. 0V on power-up enters bootload mode, 0 after power-up restores default register settings.
24	SIGNAL	5.5V	50KΩ voltage divider on board divides signal down to 3.3V or less.

2.1 Pin Schematics – The following schematics indicate some of the components internal to the motion control module.



2.2 Power Supply Connections - We recommend connecting a capacitor to the VMOTOR pin to provide current to the motor in an instant. The ripple current rating for this capacitor should exceed ½ the peak motor current. A 100uF 500mA aluminum electrolytic part will work. It is also a good idea to provide a linear regulator between the VMOTOR voltage and the VIN pin. Systems with motor voltages greater than 10V should always step the voltage down to prevent overheating of the module's on-board 3.3V regulator.



All grounds between power supplies and any master unit should be common.

If a fuse is to be included in your design it should cut all power to the module upon opening.

The H-bridge driver IC will not function below 5V. It is rated for 40V transient voltages, but surrounding components are not. Care should be taken to ensure that VMOTOR does not exceed 28V to fully protect all components on the module.

3.0 Faults and Limits – The controller employs a variety of protective schemes, some of which are user configurable.

3.1 Fault Conditions – There are two forms of fault conditions on the motion control module. Those generated by the H-bridge IC and those generated by the microcontroller. Please note that faults 3.1.2 and 3.1.3 require a reset via the serial interface to clear the fault. Alternatively the *Function.F_EnableOnBrake* bit may be set to automatically reset the H-bridge upon a latched fault.

3.1.1 H-bridge Under Voltage Fault – By design the under-voltage lockout occurs when the voltage falls below 0.415V. It is re-enabled when the voltage exceeds 5V. In practice, care should be taken to ensure VMOTOR always remains above 5V. The *Status.S_BrakeFault* bit will be set when this occurs. Pin 8 will also be pulled low during this fault.

3.1.2 H-bridge Short Circuit Fault – When a short circuit is detected (9-16A depending on temperature) the H-bridge is latched off and the BRAKE/FAULT pin (pin 8) is latched low. The *Status.S_BrakeFault* bit will be set when this occurs. The H-bridge must be reset by sending "65" to the *Command* register. If the *Function.F_EnableOnBrake* bit is set the controller will automatically reset the H-bridge on a fault.

3.1.3 H-bridge Over Temperature Fault – When the H-bridge IC exceeds 175°C the H-bridge is latched off and the BRAKE/FAULT pin (pin 8) is latched low. The *Status.S_BrakeFault* bit will be set when this occurs. The H-bridge must be reset by sending "65" to the *Command* register. If the *Function.F_EnableOnBrake* bit is set the controller will automatically reset the H-bridge on a fault.

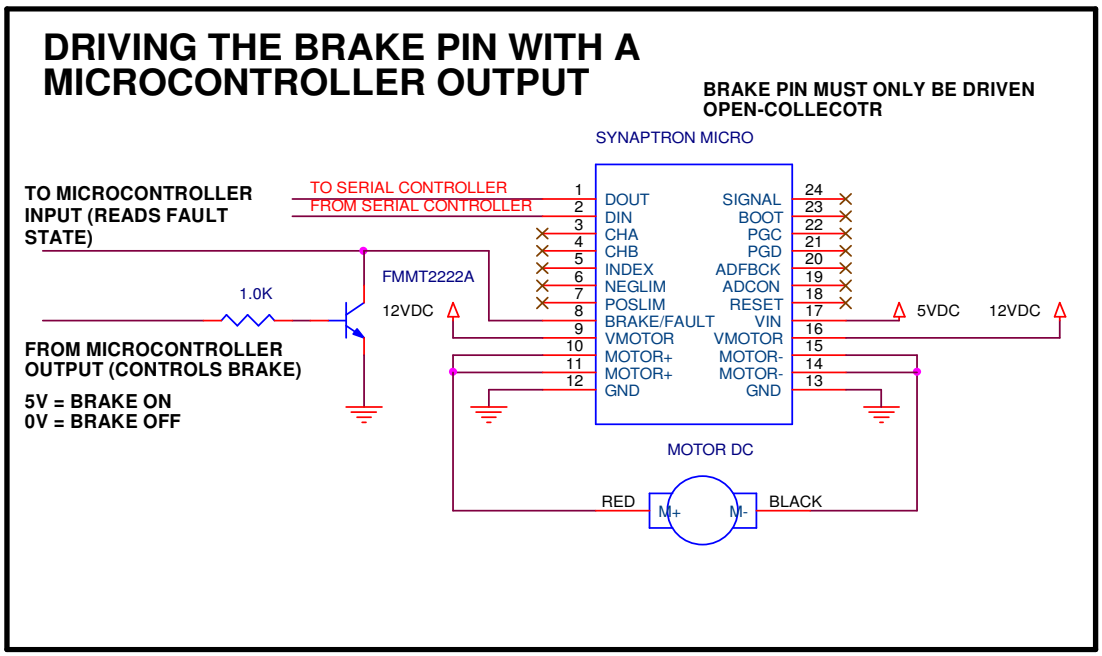
3.1.4 Internal Current Limit – If the microcontroller reads a current value greater than 5A +/-20% the drive signal to the H-bridge is set to 0% duty-cycle. When this occurs the *Status.S_InternalCurrentLimit* bit will be set. When the current reading drops below 5A the duty-cycle is allowed to exceed 0%.

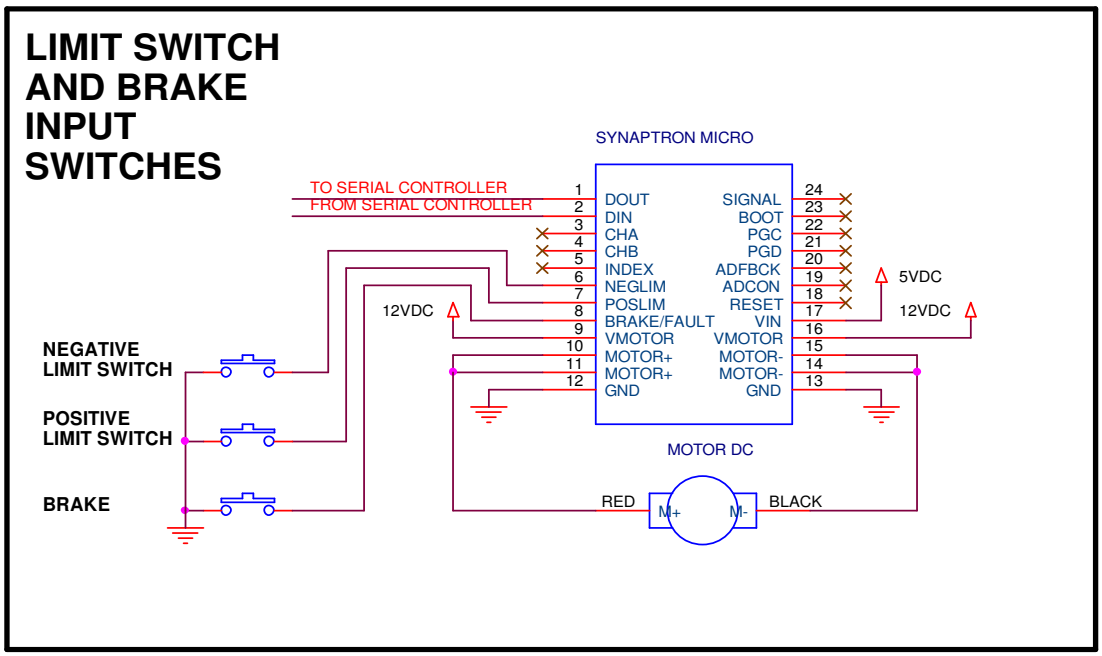
3.1.5 User Current Limit – If the microcontroller reads a current value greater than *PositiveCurrentLimit* register or less than the *Negative CurrentLimit* register the current limit becomes active. When this occurs PWM signal will be limited in an attempt to keep the current measurement below the user set limits. The *Status.S_UserCurrentLimit* bit is set during this event.

3.2 BRAKE/FAULT Indicator Pin – Pin 8 is an input/output pin which is open collector.

You should never apply a voltage directly to this pin.

When an H-bridge fault occurs (sections 3.1.1 through 3.1.3) this pin will be pulled low. This pin may also be used to implement a brake which will stop the motor. A dry switch may be used, or a microcontroller can be collected to the BRAKE/FAULT pin through an open collector style connection.





3.3 Limit Switch Settings – The controller employs both hardware limits and software limits. The limits, when enabled and asserted limit commanded movements in specific directions.

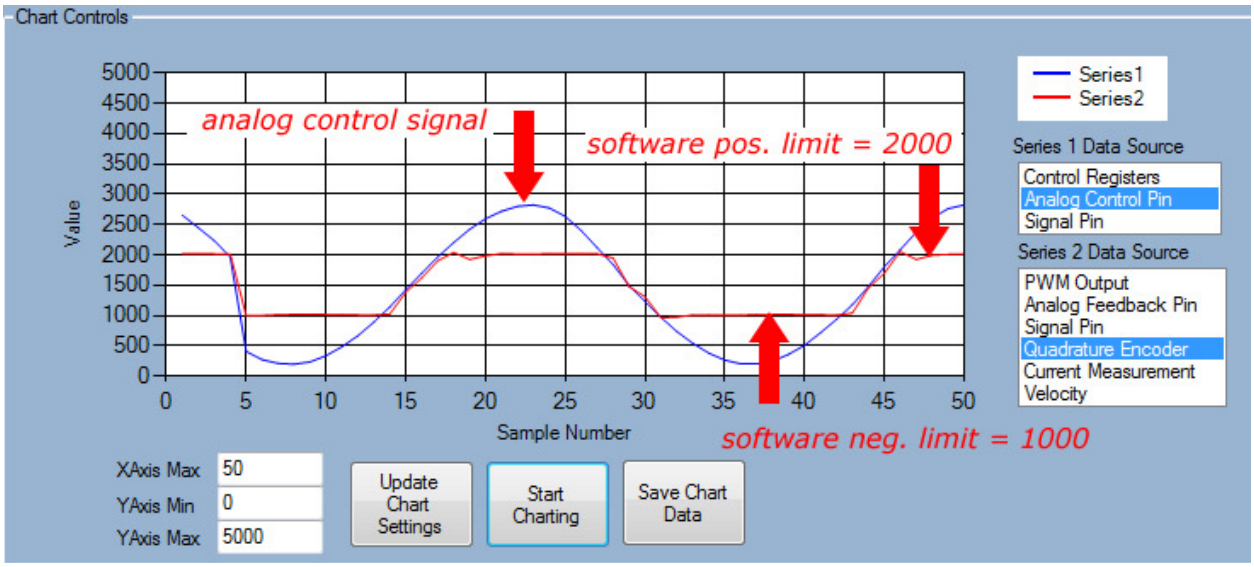
3.3.1 Hardware Negative Limit Switch – In order for hardware limits to work the *Function.F_HardLimits* bit must be set. A switch or voltage may be connected to pin 7. When this pin is grounded the motor will be restricted when moving in the negative direction. The *Status.S_HardNegLim* bit will be set when the pin 7 is grounded regardless of whether or not the hard limits are enabled through the *Function* register.

3.3.2 Hardware Positive Limit Switch – In order for hardware limits to work the *Function.F_HardLimits* bit must be set. A switch or voltage may be connected to pin 8. When this pin is grounded the motor will berestricted when moving in the positive direction. The *Status.S_HardPosLim* bit will be set when the pin 8 is grounded regardless of whether or not the hard limits are enabled through the *Function* register.

3.3.3 Software Negative Limit – In order for software limits to work the *Function.F_SoftLimits* bit must be set. Additionally, the *NegativeLimitLow* (and *NegativeLimitHigh* if using 32-bit mode) register should be loaded with the desired limit value. If the controller is commanded to move to a value lower than the contents of the *NegativeLimit* registers the limit setting will be used in lieu of the commanded value. If this limit is used in an open-loop system (where *FeedbackSource* = 0) care should be taken to only use a proportional gain of 1. If this limit is asserted the *Status.S_SoftNegLim* bit will be set.

3.3.3 Software Positive Limit – In order for software limits to work the *Function.F_SoftLimits* bit must be set. Additionally, the *PositiveLimitLow* (and *PositiveLimitHigh* if using 32-bit mode) register should be loaded with the desired limit value. If the controller is commanded to move to a value higher than the contents of the *PositiveLimit* registers the limit setting will be used in lieu of the commanded value. If this limit is used in an open-loop system (where *FeedbackSource* = 0) care should be taken to only use a proportional gain of 1. If this limit is asserted the *Status.S_SoftPosLim* bit will be set.

The image below provides a visual example of software limits. The controller is operating in analog control mode with a quadrature encoder used for feedback. The negative limit is set at 1000 and the positive limit is set at 2000. When a sine wave is used for the analog control signal the motor position follows the analog signal until it tries to exceed 2000 or go below 1000, the software limits.



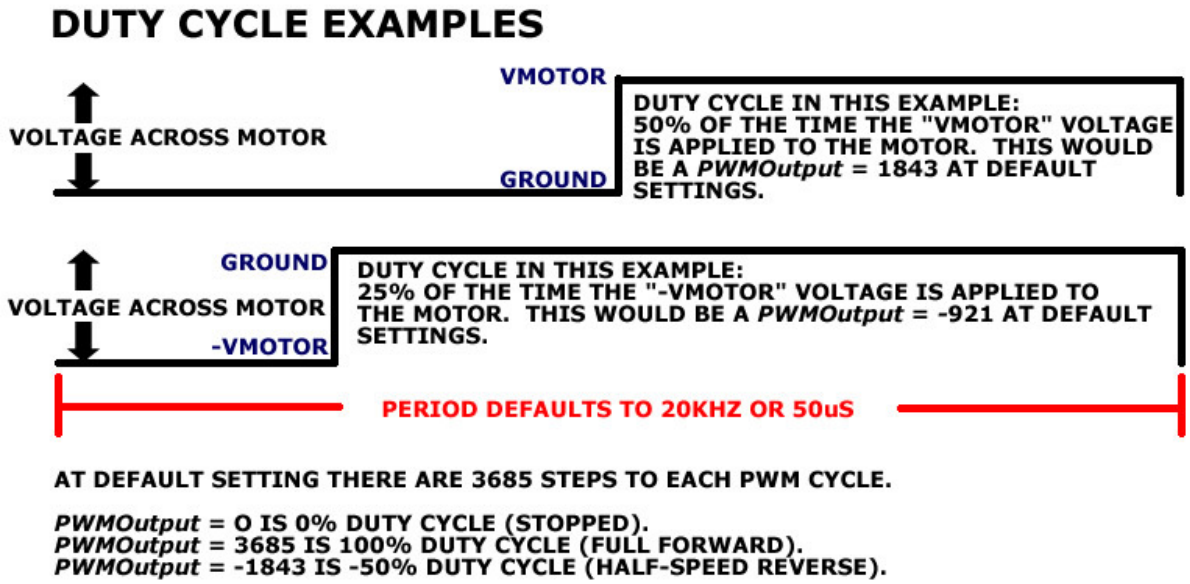
3.4 PWM Limits – In closed loop modes the PWM output signal is constantly being adjusted by the PID algorithm. You may want to limit motor speeds. The *PositivePWMLimit* register limits the duty cycle of positive motor drive signals. The *NegativePWMLimit* register does the same for negative motor drive signals.

Negative PWM values represent motors running in reverse, and positive values, forward. The duty cycle of the signal may be calculated using the equation…

$$\frac{PWM\ Value}{Maximum\ Duty\ Cycle} * 100\% = duty\ cycle\ percentage$$

Using the default settings there are 3685 steps in each PWM period. You adjust the number of steps to determine how much power is delivered to the load. For example, a PWM Value of 2000 would result in a duty cycle of 54%, or just a little more than ½ power. Higher duty cycles turn the motor faster. The sign of the PWMValue determines the direction the motor is turning, with negative values being reverse.

The ability to set the PWM limits to different values in different directions can be useful in some applications. For example, you might wish to open something quickly, but close it slowly, or vice versa.



3.5 Current Limits – In addition to various fault protections listed in previous sections the user may program current limits for forward and reverse movements. The *NegativeCurrentLimit* and *PositiveCurrentLimit* registers can be programmed to force current limits.

The register values are 1mA per bit, so a value of 2000 equals 2000mA, or 2A. The current measurement itself should be considered a first order estimate. The H-bridge IC that provides the current feedback is rated at +/- 20% between 500mA and 6A. Therefore, accurate current limits below 500mA can only be attained through trial and error, and even those above that threshold will only be in the +/-20% range. Current measurements can be adjusted via the *CurrentMultiplier* and *CurrentDivider* registers.

***Current* = Raw Measurement * (*CurrentMultiplier*/*CurrentDivider*)**

These current limit settings are useful in avoiding fault conditions and can prevent thermal shutdown from occurring. When a current limit is reached the PWM signal will be restricted from increasing in magnitude.

The default current measurement multiplier and divider register values were determined with a 5Ω resistive load. Different loads, including inductive motors, are likely to require different values to fine tune accuracy.

4.0 Control Modes – Using the module's serial interface the user may program the control source and the feedback source via the *ControlSource* and *FeedbackSource* registers. Each of the registers above is accompanied by a multiplier, divider, offset, and result register. This allows for mixing and matching of the input sources for control or feedback. Adjustments to the raw signals to fine-tune control schemes are also possible. Details on these registers may be found in the communication protocol documentation. The tables below shows some control modes that can be created using the *ControlSource* and *FeedbackSource* registers.

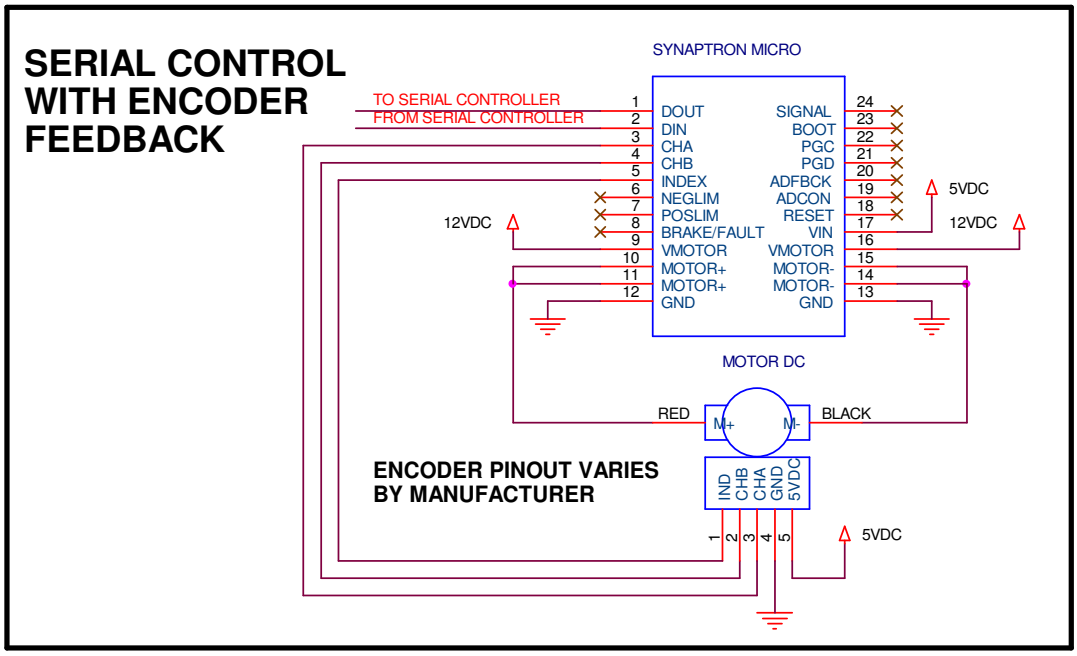
Register	Value	Description
ControlSource	0	Control source comes from the *ControlInputLow* and *ControlInputHigh* registers
ControlSource	1	Control source comes from 0-3.3V analog signal at pin 19
ControlSource	2	Control source is the signal at pin 24 typically a pulse or frequency
FeedbackSource	*0*	There is no feedback source, the control system is open-loop
FeedbackSource	1	Feedback source comes from 0-3.3V analog signal at pin 20
FeedbackSource	2	Feedback source is the signal at pin 24, typically a pulse or frequency
FeedbackSource	3	Feedback source is an incremental encoder attached to pins 3, 4.
FeedbackSource	4	Feedback source is the current measurement from the H-bridge located in the *Current* register
FeedbackSource	5	Feedback source is the *Velocity* register where the velocity is derived from a quadrature encoder attached to pins 3, 4.

Example Settings

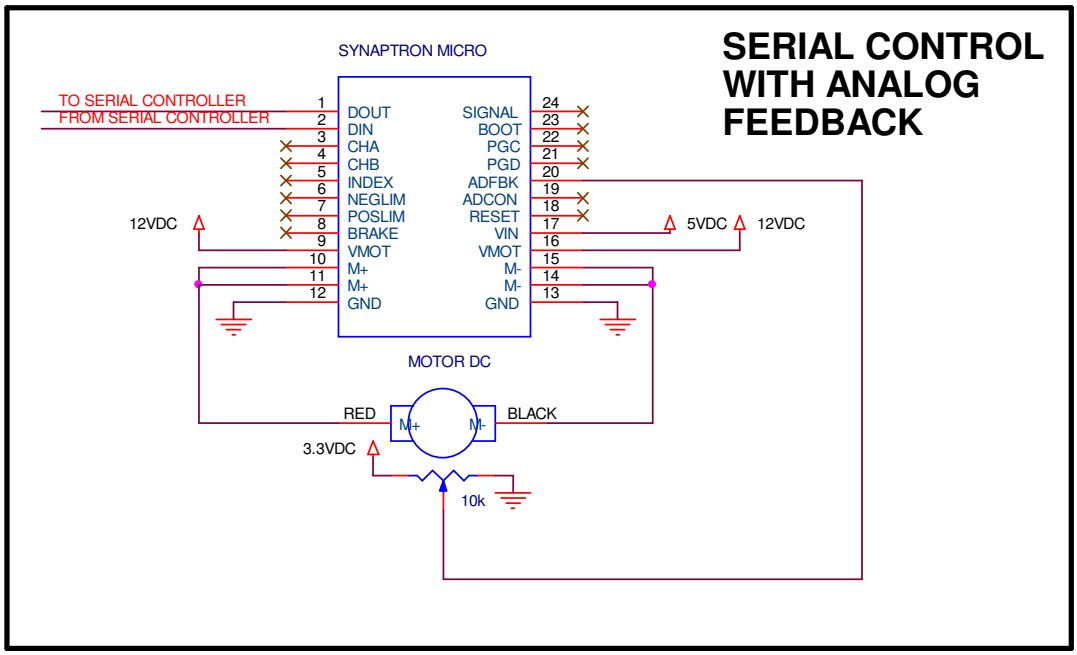
Control-Source	***Feedback-Source***	**Closed Loop**	**Control System Description**
0	0	NO	The duty-cycle is set by writing values to the *ControlInputLow* register.
1	0	NO	The duty-cycle is set by the analog measurements at pin 19 of the controller. The measurement may be modified by the multiplier, divider, and offset registers associated with the control value.
3	0	NO	A signal based controller. A pulse signal at pin 24 is used to determine motor speed. This may be used with a standard radio control servo signal, or some other pulse/frequency.
0	1	YES	An analog signal at pin 20 is used as the feedback signal and the serial interface is used to modify *ControlInputLow* to set the desired position.
1	1	YES	An analog signal at pin 20 is used as the feedback signal and an analog signal at pin 19 sets the desired position-voltage.
2	1	YES	A pulse value at pin 24 sets the desired position, while an analog signal at pin 20 is used to provide position feedback. This could be used to control the position of a linear actuator that has a feedback potentiometer with a 1-2ms RC signal.
0	2	YES	The *ControlInputLow* register determines the desired signal at pin 24. This may be used with feedback sources such as US Digital's MA3-P12 series.
0	3	YES	The *ControlInputLow* and *ControlInputHigh* registers may be modified through the serial interface to set a desired position while an incremental encoder connected to pins 3,4, and optionally 5 is used to provide position feedback. The position value is located in the *PositionLow* and *PositionHigh* registers.
1	3	YES	The analog signal at pin 19 sets a desired position while an incremental encoder connected to pins 3, 4, and optionally 5 provides position feedback. The position value is located in the *PositionLow* and *PositionHigh* registers.
1	4	YES	The analog signal at pin 19 sets the desired load current, while the *Current* register (after modification by the *CurrentMultiplier* and *CurrentDivider*) provides the feedback signal.
0	5	YES	The *ControlInputLow* and *ControlInputHigh* registers determine the desired system velocity measured from an incremental encoder attached to pins 3, 4, and optionally 5. The *Velocity* register is used for feedback.

4.1 Control Mode Schematics – The following schematics show basic connections for different control and feedback modes. Not all connections are shown. For more detailed application information please review application notes at www.solutions-cubed.com.

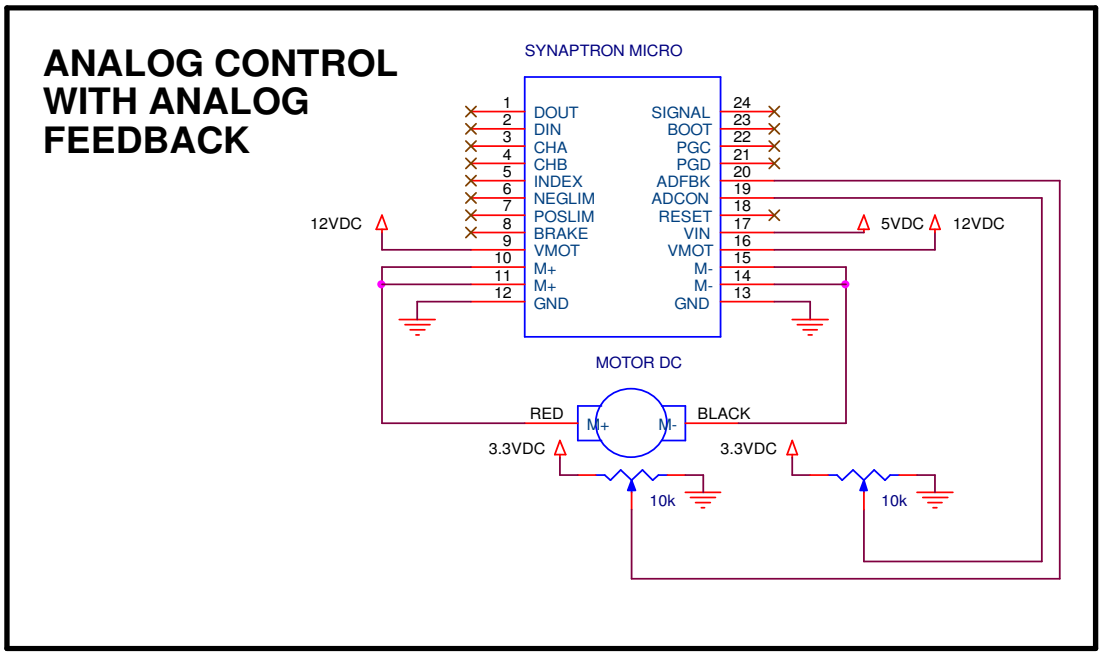
ControlSource = 0, *FeedbackSource* = 3



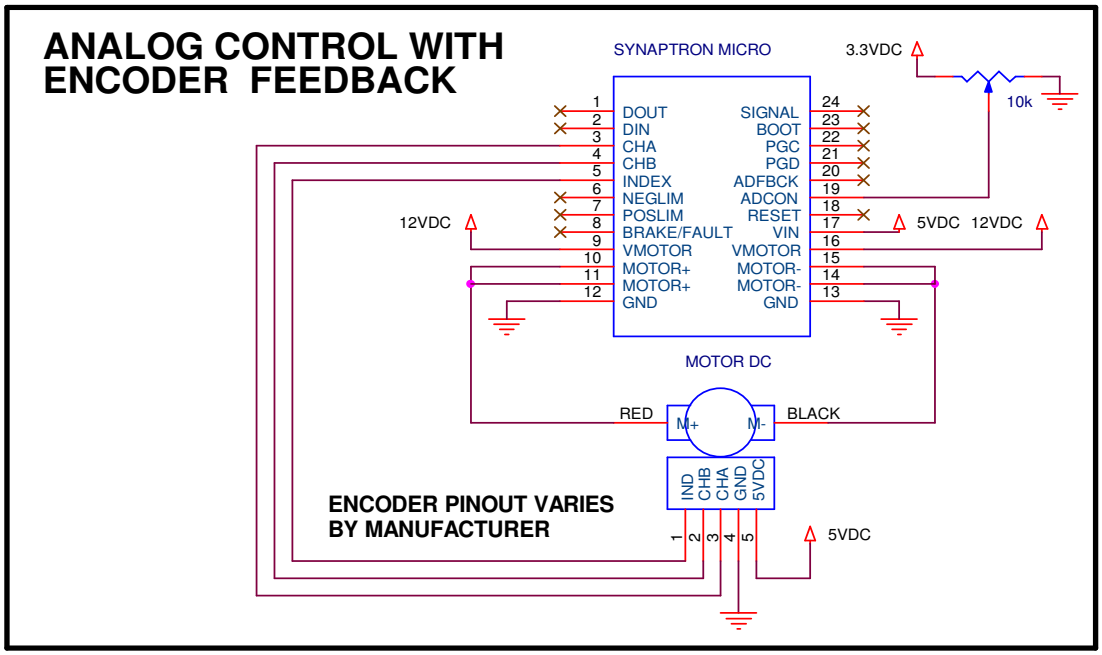
ControlSource = 0, *FeedbackSource* = 1



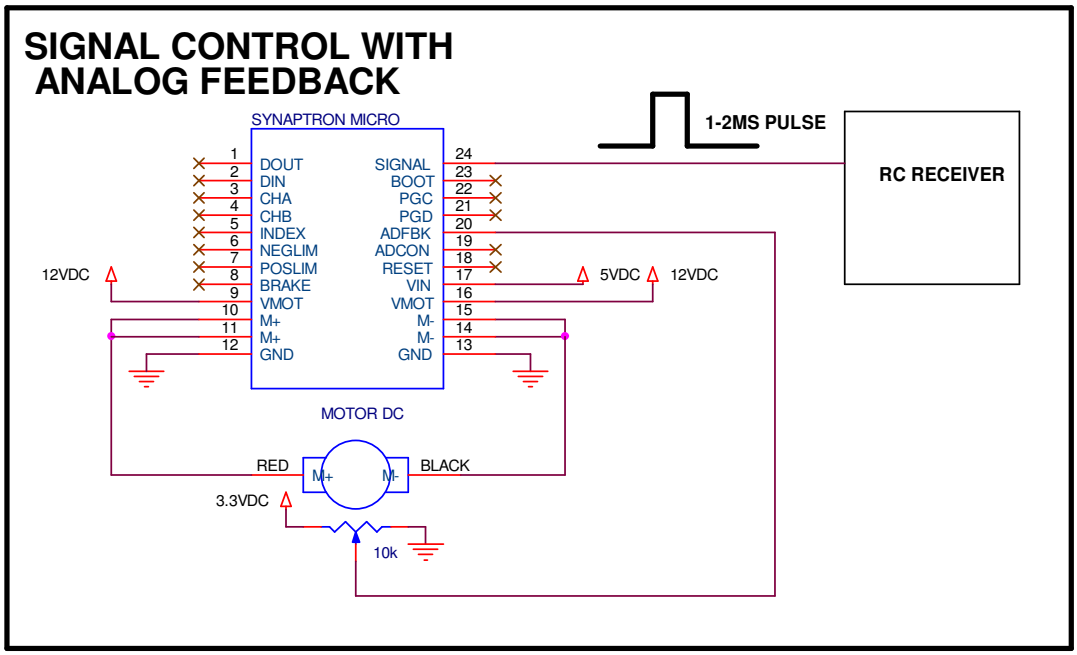
ControlSource = 1, *FeedbackSource* = 1



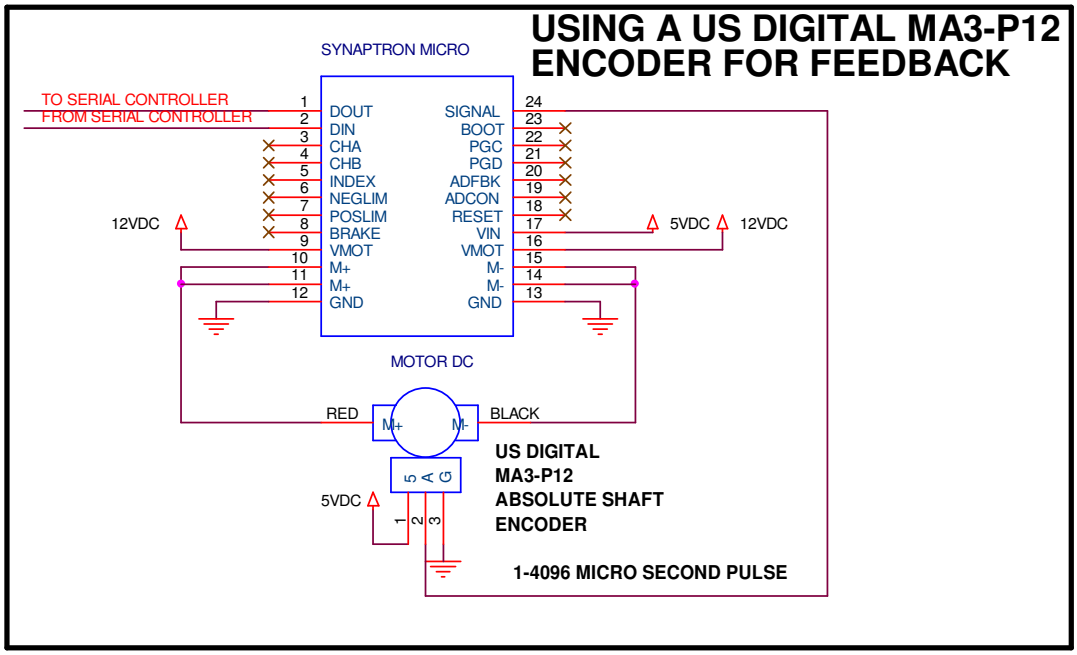
ControlSource = 1, *FeedbackSource* = 3



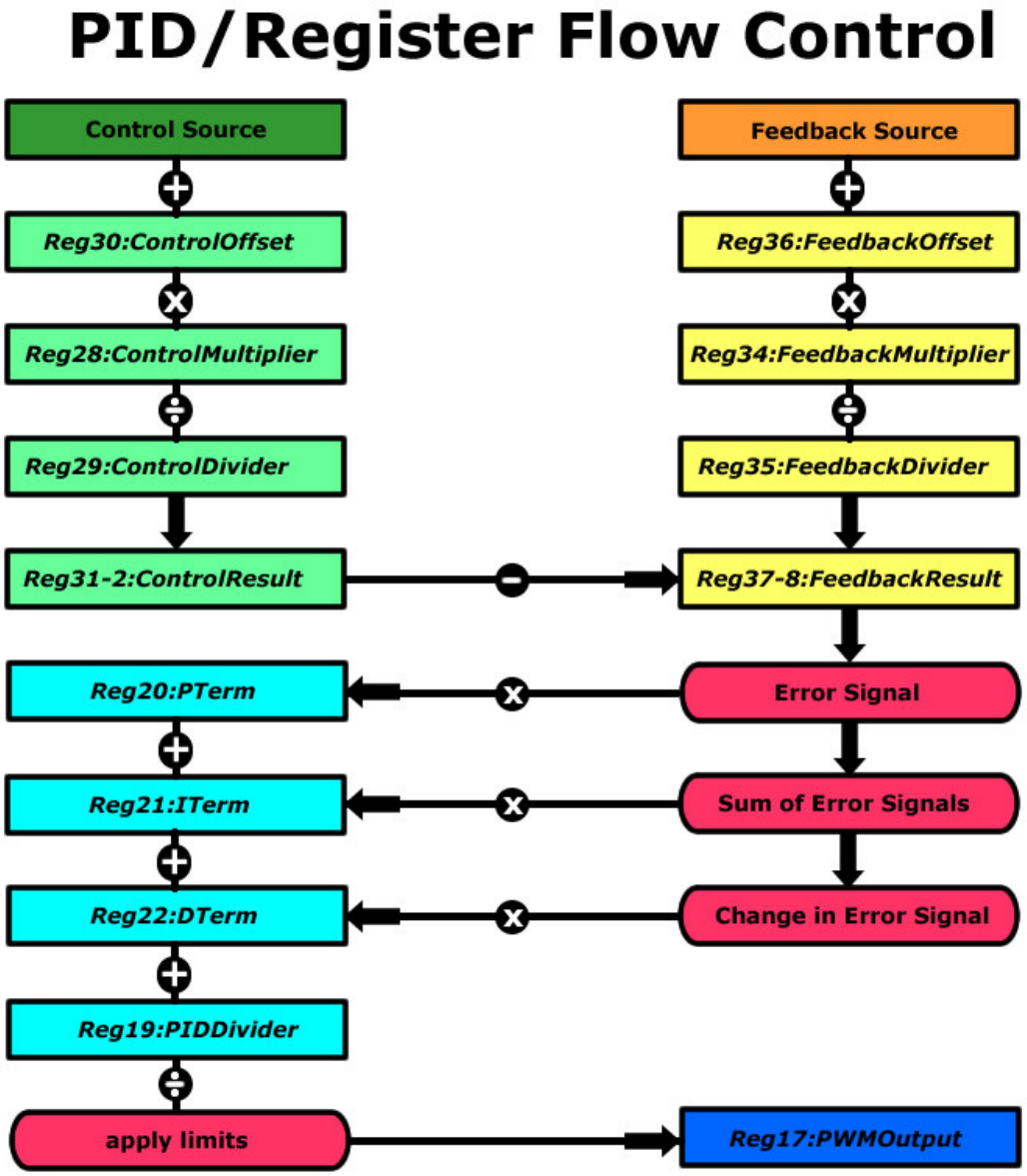
ControlSource = 2, *FeedbackSource* = 1



ControlSource = 0, *FeedbackSource* = 2



5.0 PID Settings and Functions – The Synaptron controller employs a versatile, and highly programmable, method to convert input signals to motor drive signals. The basic connections between registers and the control algorithm are shown below. A number of simple steps are required to program the registers. The serial interface and software available at Solutions Cubed, LLC, or a variety of ASCII terminal programs, may be used for programming.



Step1: Select the method of control by programming the *ControlSource* register. Also, select the feedback source by programming the *FeedbackSource* register. For example, if you wanted to control a motor with an analog input signal and have no feedback (an open loop system) you would program *CountrolSource* = 1 and *FeedbackSource* = 0.

Step 2: If either the control source or feedback source need to be modified or adjusted you can adjust the values in the appropriate offset, multiply, and divide registers. The results of these values on the control or feedback signal are stored in the associated result registers. Sticking with the previous example, let's say you want to use a 0-2.5V analog signal to drive a motor forward and reverse. You would program the *ControlOffset* register with -1250 and the *ControlMultiplier* register with 3.

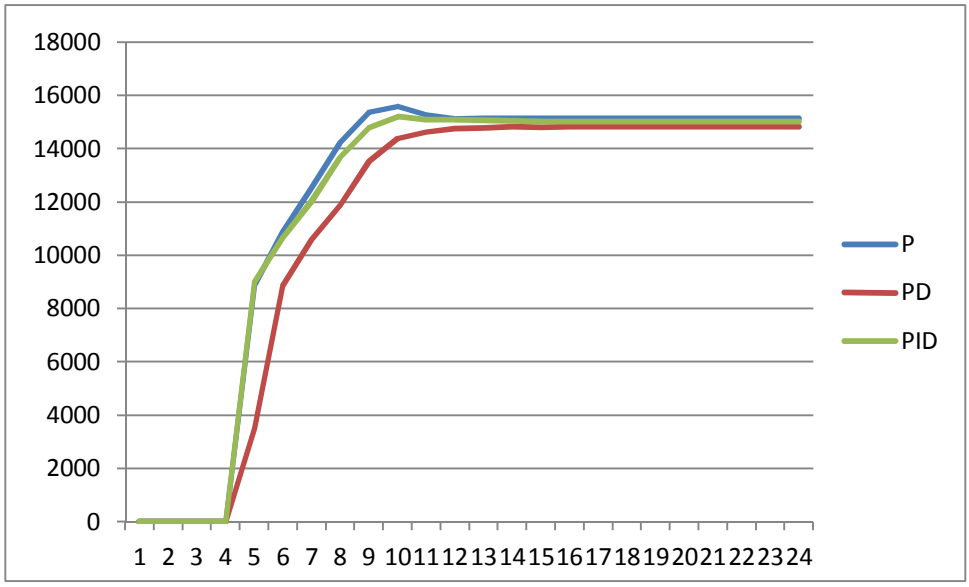
Step3: Program the PID settings. If creating an open loop system the PID registers should be left in their default states. When creating a closed loop system keep in mind the following:

Pterm – This is the proportional gain and provides most of the muscle in controlling the system. You should start tuning the PID with the *Iterm* and *Dterm* registers set to 0. You may find you only need proportional control.

Iterm – This is the integral term and nudges the system into its final position. The integral term is multiplied by the sum of past error signals. This term is normally very small and if too large will cause oscillations. This term is limited internally to prevent "wind up". Many systems will function well with just PI settings.

Dterm – This is the derivative term and acts as a damper on the system. The derivative term is multiplied by the change in error signals. A large derivative value will slow responses to commanded changes. This term may need to exceed the *Pterm* if you want the drag to be noticeable.

PIDDivider – The output of your PID algorithm may be scaled too large causing the PWM signal to oscillate between full reverse and full forward. Set your *PIDDivider* to ½ the *Pterm* for starters.



The graph above shows an example of how PID settings change closed loop control. In this example the *ControlSource* = 0 (*ControlInput* registers are the source of the control signal), *FeedbackSource* = 3 (quadrature encoder is the source of the feedback signal), *PIDDivider* = 50, *Pterm* = 100, *Iterm* = 1, and *Dterm* = 25. The controller was commanded to move the motor to position 15,000 with three different settings.

"P" Blue Line:
Pterm = 100, Iterm = 0, Dterm = 0, PIDDivider = 50
The motor overshoots the target position by 600 encoder counts, and settles to about 100 counts higher than the target.

"PD" Red Line:
Pterm = 100, Iterm = 0, Dterm = 25, PIDDivider = 50
The motor undershoots the target position by 200 encoder counts.

"PID" Green Line:
Pterm = 100, Iterm = 1, Dterm = 25, PIDDivider = 50
The motor slightly overshoots the target position and then settles to the exact commanded position.

Step 4: Program the *ErrorBand* if desired, and enable error band functionality by setting the *Function* register bits.

> *Function.ClosedLoopEB* (bit10; when set) – If the control and feedback values are within the error band setting of each other the error signal is set to 0. The *FeedbackSource* register must be set to a non-zero value for this bit to have an effect.
>
> *Function.OpenLoopEB* (bit11; when set) – If a PWM output is within +/- the error band setting of 0 then the PWM signal is set to 0. The FeedbackSource register must equal 0 for this bit to have an effect.

Step 5: Set any PWM limits, current limits, and any function bits associated with software limits or hardware limits. And then test your system under a variety of conditions. Program all settings to non-volatile memory by sending the "Store registers" command. The ASCII version of this command using the default address of 54 is "54,02,67"<cr><lf> (see the communication protocol documentation for more details on this command).

5.1 Control Loop Rates- When operating as a closed-loop controller the PID loop update rate (how often the PWM signal is modified) is determined by the *ControlLoopRate* register. Motor drive signals are updated every 250uS**ControlLoopRate.* The default setting of 4 means the motor speed is updated once per milli-second.

NOTE: Using the SIGNAL input as a control or feedback source will limit the update rate. The pulse measurement at the SIGNAL input is measured every 3 pulses. So a 5ms pulse would effectively create an update rate of 15ms regardless of the *ControlLoopRate* setting.

5.2 Velocity Control – There are a few ways to implement velocity control.

a) You may limit PWM output by writing values to the *NegativePWMLimit* and *PositivePWMLimit* registers. Reducing these registers from their default settings will limit the drive signal applied to the motor and therefore limit motor speed.

b) When using serial control (*ControlSource* = *0*) you can set the *Function.F_VelocityLimit* bit and write a value to the *VelocityLimit* register. When a value is written to the *ControlInput* registers the Synaptron will move at a limited velocity to the desired position. This mode works best with a quadrature encoder as the feedback source (*FeedbackSource* = 3).

When this functionality is used the velocity limit is applied prior to any adjustments to the control signal (*ControlMultiplier* through *ControlResult* registers). These registers should not be modified from their default settings unless the impact of the modification on the velocity limit functionality is considered.

c) You can ramp up and down your control source externally to the controller. For serial control this could be a for…next loop sending a sequence of commands, or for an analog system it could be a simple RC filter.

5.3 Alternate PID – Setting the *Function.F_AlternatePID* bit causes the PID filter's output to be added to the PWM signal. This can be useful when the feedback source is the *Velocity* register (*ControlSource* = 5).

When *Function.F_AlternatePID* = 0 PWM drive signal = PID output.

When *Function.F_AlternatePID* = 1 PWM drive signal = PWM drive signal + PID output.

6.0 Nonvolatile Memory and Restoring Default Settings-

Flash Memory:
Registers may be modified and stored in nonvolatile flash memory. This allows the user to program the unit and then place it in the field for use (eliminating the need for a serial interface after initial programming for many applications). The controller uses a portion of its own flash program memory. The minimum number of write cycles for this memory is 9,999.

You should not employ programs or hardware that cause the flash memory to be continually "stored"!

To store program settings in nonvolatile memory you must send the "Store registers" command by writing 67 to the *Command* register (see the communication protocol documentation for additional details). When this command is received the *FlashCycles* register will decrement. If *FlashCycles* reaches 0 you will have exceeded the manufacturer's specification for the number of write cycles the program memory can accept. Avoid doing this.

Writing to the internal registers does not impact flash memory. Only storing the values will decrement the *FlashCycles* register settings.

Note: Bootloading in a new firmware revision will reset the *FlashCycles* value to 9998 the default setting.

Restoring Default Settings:
There may be instances when you need to restore the default settings of the device. There are two methods to accomplish this.

> Serial Interface: Send the "Restore defaults" command by writing 66 to the *Command* register.
>
> Hardware: If you've lost serial communication you can perform a hardware restoration of the internal registers by powering the module and then pulling the BOOT pin (pin 23) to ground. Do not power the module with the BOOT pin already grounded, that will place you in bootloader mode.

7.0 Firmware Upgrades / Bootloader – Firmware upgrades are typically in the form of text files which must be "bootloaded" into the device through the serial interface. See our application note on using the bootloader for detailed information.