SYNAPTRONμ MOTION CONTROLLER

Serial Communication Protocol - 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**

Communication Protocol:

This document describes the communication protocol used to program and communicate with the Synaptron motion controller. Functionality in the Synaptron is adjusted by modifying registers internal to the device. The registers can be written to, or read from, via a serial communication interface. Register values that are modified from the default settings may be stored in non-volatile memory. The Synaptron has an efficient binary communication protocol and an easy to use ASCII communication protocol.

Binary communication is generally faster and better suited for use with external microcontrollers. The ASCII protocol can be used with simple terminal software programs and is well suited for quick program changes and testing.

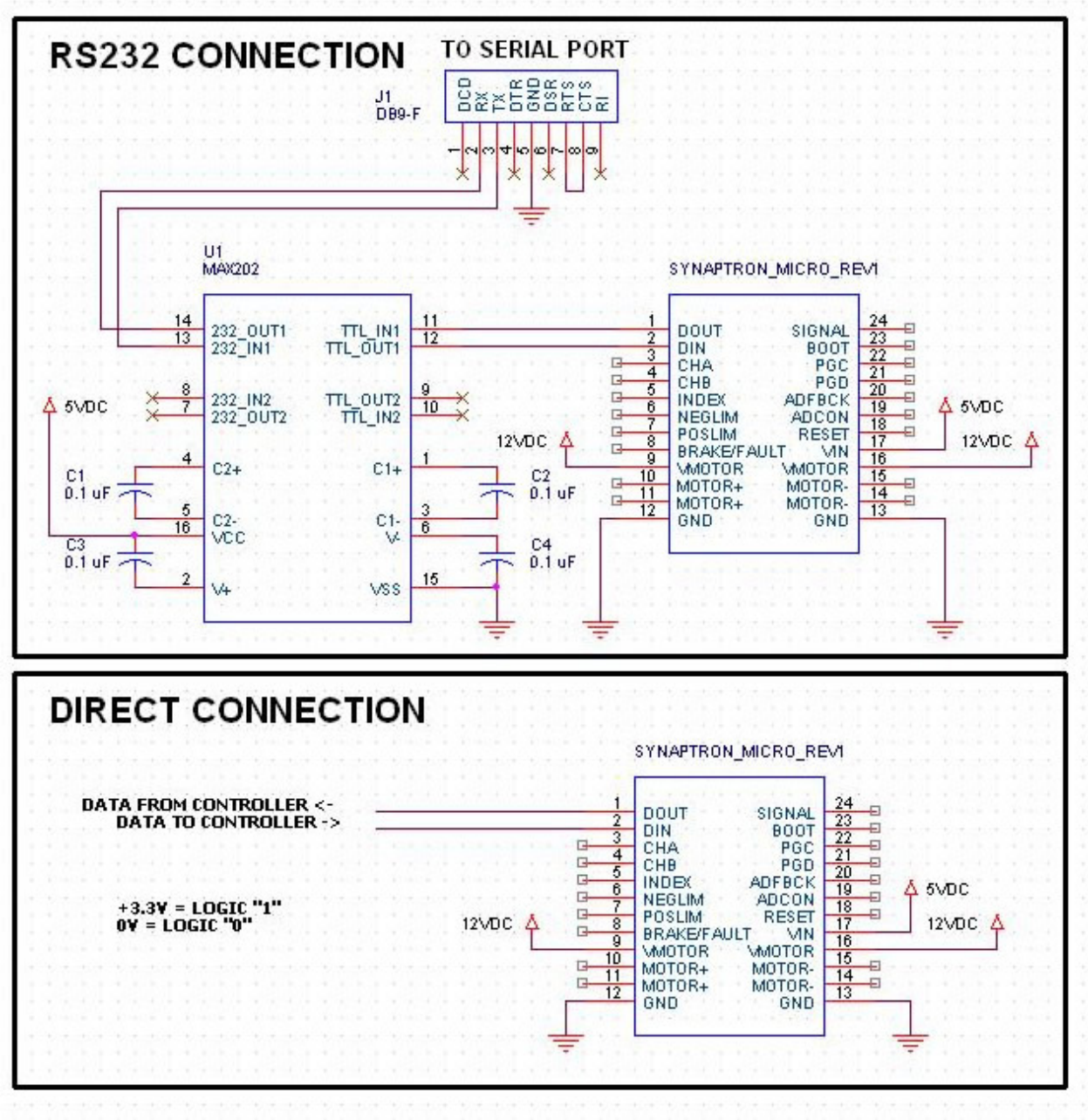
Test software for programming the Synaptron is available at **www.solutions-cubed.com**.

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.

1.0 Basics:

1.1 Electrical Connections: The DOUT pin is used by the Synaptron to send data, the DIN pin receives data. Both pins are 5V tolerant but make use of a 3.3V system regulator. 3.3V out/in is logic "1". 0V out/in is logic "0". An 820Ω current resistor is in series with the DOUT pin. All grounds should be shared between the Synaptron and connected electronics.

Schematic:



1.2 Communication Settings: The Synaptron communication protocol defaults to 9600 bits per second (baud rate) and an 8N1 format (8 data bits, no parity check, 1 stop bit). 5 baud rates are supported from 1200bps to 115200bps. The Synaptron has a default internal address of 54, which is used during communication.

1.3 Communication Timing Requirements:

1.3.1 Binary Requirements: There are no timing requirements for the first two bytes of data received. The first byte must equal 0 and the second byte must equal the unit's address, or the broadcast address. Subsequent to receiving these two bytes the Synaptron will process data it receives after the DIN line is stable high for approximately 3 byte periods (or 30 bit periods). Therefore, communication should have less than 2 byte periods (20 bit periods between) bytes. At 38400bps you must have less than 520uS between each byte in a command packet.

Command is sent			**DIN line is high**			**Synaptron processes command**	**Synaptron responds**
1st byte command	Data…	Last byte command	1st byte period	2nd byte period	3rd byte period	1-2 control loops (default is 1-2ms)	1st Byte response

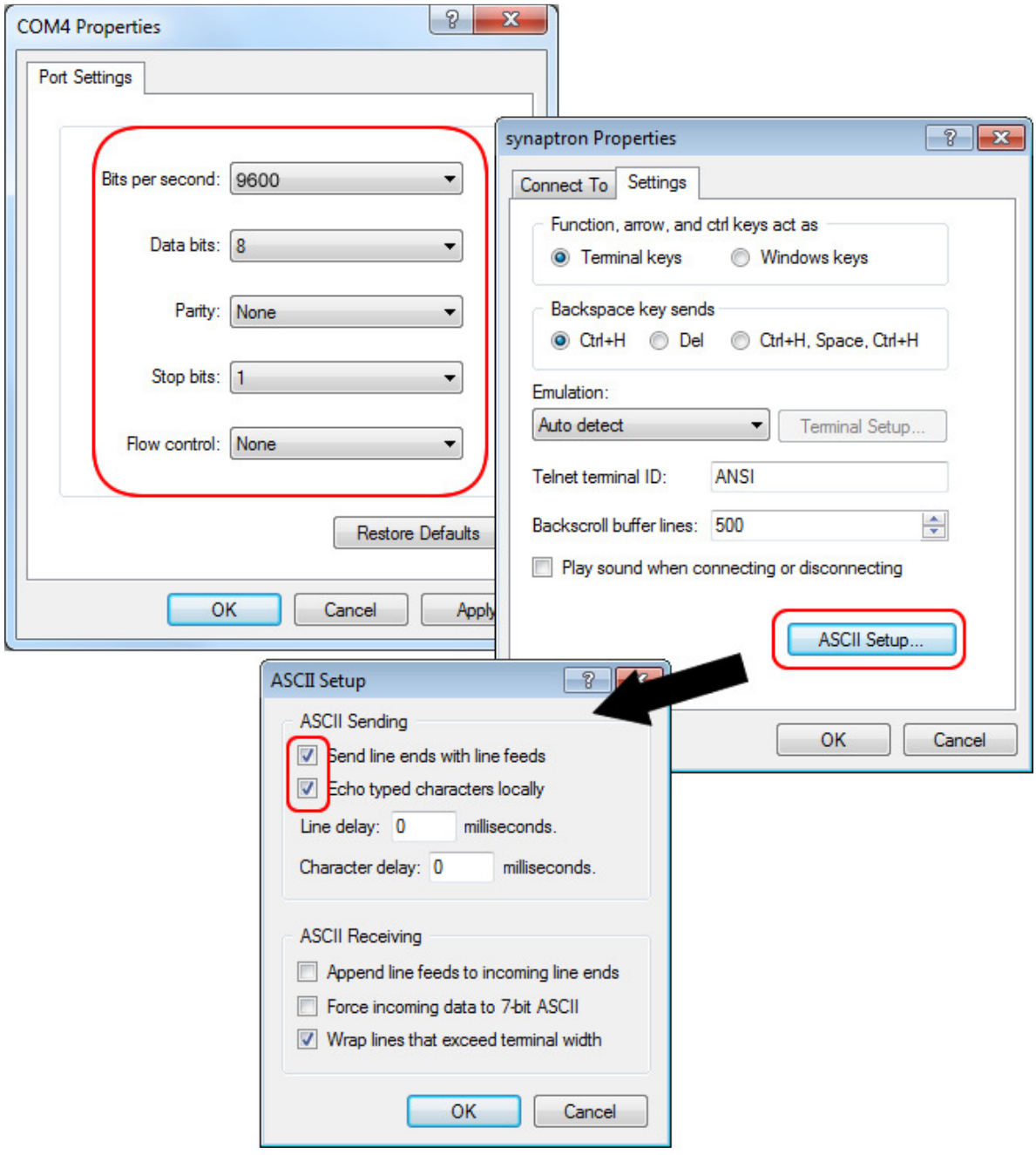
1.3.2 ASCII Requirements: When sending ASCII commands there are no timing requirements. The first two characters sent must equal the unit's address, or the broadcast address of "99". If that occurs subsequent characters will be stored until a carriage return (<cr> = 0x0d) followed by a line feed (<lf> = 0x0a) are received. When these terminating characters are received the command string will be processed.

Example: Read Command Sent "54,00,"<cr><lf>
Response "54,9998"<cr><lf>

1.4. Numeric Systems: The communication protocol transmits data as single bytes. This document may use character, decimal, or hexadecimal number representations. Characters (ASCII) are represented by quotation marks. Hexadecimal numbers are indicated by the 0x prefix. Negative hexadecimal numbers are described by the two's compliment system (-1 = 0xFFFF in a 16-bit value and 0xFFFFFFFF in a 32-bit value).

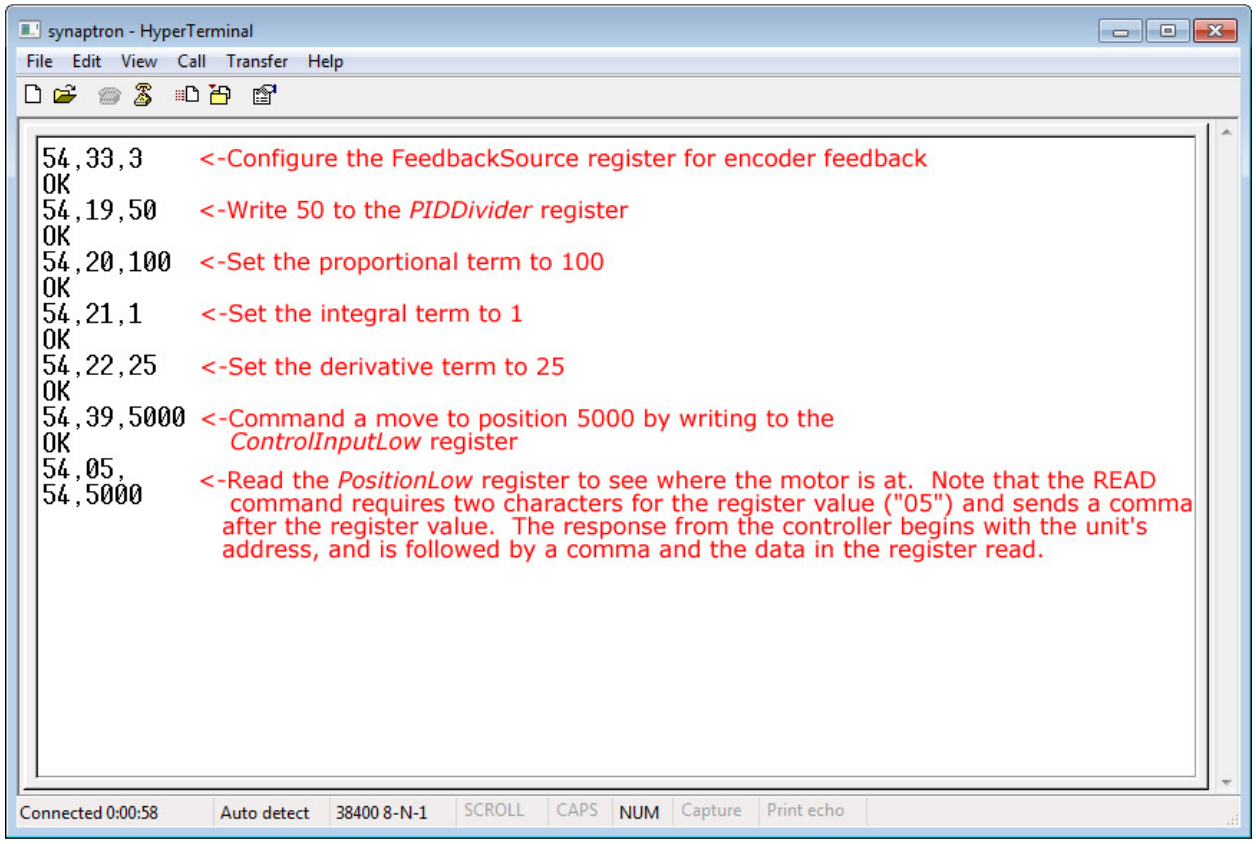
1.5. Communication LED: The red LED on the unit will blink when a command is received successfully.

1.6 Quick Start: Using the RS232 connection shown section 1.1 and a terminal program can get you started quickly. A USB-RS232 converter may be used to connect the module to a PC USB port. In this example Hyperterminal is used to communicate with the controller using the ASCII interface. Start Hyperterminal and set the COM port settings as shown below. The select file/properties/settings and click the ASCII Setup button. Check the checkboxes shown.



The following communication example assumes that a quadrature encoder and motor are correctly connected to the controller. It also assumes that the RS232 interface is correctly connected.

In this example we set the controller to use the quadrature encoder as its feedback source. The device defaults to using the *ControlInput* registers as the control source. After the PID settings are adjusted a move is implemented by writing to the *ControlInputLow* register. This is followed by a READ command that displays the contents in the *PositionLow* register indicating that the move was successful.



See section 3.0 for a detailed description of ASCII commands, and section 5.0 for details related to the various registers accessible in the controller.

2.0 Binary Commands: The communication protocol consists of two commands, they are WRITE and READ. These commands are used to modify or monitor the contents of the Synaptron's internal registers.

2.1 Binary WRITE Command: The WRITE command consists of an address word, register word, data packet, and a checksum byte. If sent correctly the Synaptron will respond with an acknowledge byte.

Address Word- The first byte is a dummy byte and should be 0. The lower byte is the unit's address. Addresses must be greater than 53 and less than 100. Address 54 is the default address, 99 is reserved for the broadcast commands. A broadcast command will be accepted by any unit, but not responded to.

Register Word- The first byte is a dummy byte and should be 0. The lower byte is the index value of the register you are writing data to. To write a 32-bit (4 byte) data packet the highest bit of the index value should be set.

Data Packet- Data packets are sent most-significant-byte first. If the most-significant-bit of the index byte is 0 then the data packet should be 16-bits sent as two separate bytes, MSB first. If the most-significant-bit of the index byte is 1 then the Synaptron expects a 32-bit value sent as 4 bytes.

Checksum- The checksum byte is calculated by summing all preceding bytes and subtracting the result from 0. The lowest 8 bits are used as the checksum. Adding all bytes in the command together (including the checksum) should result in the lowest 8-bits equaling 0.

Response- If the command is accepted the unit will return a 0x06, or ACK as a response.

Example: This example shows is a WRITE command in 16-bit mode writing the 10,000 to register 5 (the *PositionLow* register). Note that the highest bit of the index byte is clear indicating that this is a 16-bit command.

Data Sent to Synaptron						
Address Word		**Register Word**		**Data Packet**		**Checksum**
Dummy Byte	Address Byte	Dummy Byte	Index Byte	Data MSB	Data LSB	Checksum
0	54	0	5	39	16	142
Data Received from Synaptron						
Response						
Ack Byte						
6						

Example: This example shows a WRITE command in 32-bit mode, writing 0x186A0 (100,000) to register 0x06 (the *PositionHigh* register). Byte values are shown in hexadecimal. **Note that the highest bit of the index register is set indicating to the Synaptron that this is a 32-bit WRITE.**

Data Sent to Synaptron								
Address Word		**Register Word**		**Data Packet**				**Checksum**
Dummy Byte	Address Byte	Dummy Byte	Index Byte	Data MSB	Data	Data	Data LSB	Checksum
0x00	0x36	0x00	0x**8**6	0x00	0x01	0x86	0xA0	0x1D
Data Received from Synaptron								
Response								
Ack Byte								
0x06								

2.2 Binary READ Command: The READ command consists of an address word, register word, and a checksum byte. If sent correctly the Synaptron will respond with the requested register data.

Address Word- The first byte is a dummy byte and should be 0. The lower byte is the unit's address. Addresses must be greater than 53 and less than 99. Address 99 is reserved for the broadcast commands.

Register Word- The first byte is a dummy byte and should be 0. The lower byte is the index value of the register you are reading data from. To read a 32-bit (4 byte) data packet the highest bit of the index value should be set.

Checksum- The checksum byte is calculated by summing all preceding bytes and subtracting the result from 0. The lowest 8 bits are used as the checksum.

READ Response:
Address Word- The first byte is a dummy byte and should be 0. The lower byte is the unit's address.

Data Packet – A 16-bit READ will return two bytes, MSB first. A 32-bit read will return 4 bytes MSB first.

Checksum – Summing all bytes of the READ response including the checksum should result in the lowest 8-bits being 0.

Example: This example shows a READ command reading register 5 (the *PositionLow* register), where 10,000 is the returned value. Byte values are shown in hexadecimal, and 16-bit mode is used.

Data Sent to Synaptron				
Address Word		**Register Word**		**Checksum**
Dummy Byte	Address Byte	Dummy Byte	Index Byte	Checksum Byte
0x00	0x36	0x00	0x05	0xC5
Data Received from Synaptron				
Address Word		**Data Packet**		**Checksum**
Dummy Byte	Address Byte	Data MSB	Data LSB	Checksum Byte
0x00	0x36	0x27	0x10	0x93

Example: This example shows a READ command reading register 6 (the *PositionHigh* register), where 100,000 is the returned value. 32-bit mode is used. Note that the highest bit of the index byte is set indicating to the Synaptron to return 32-bits of data. The data returned reside in registers 6 (highest 16 bits) and 5 (lowest 16 bits).

Data Sent to Synaptron						
Address Word		**Register Word**				**Checksum**
Dummy Byte	Address Byte	Dummy Byte		Index Byte		Checksum Byte
0	54	0		134		68
Data Received from Synaptron						
Address Word		**Data Packet**				**Checksum**
Dummy Byte	Address Byte	Data MSBa	Data	Data	Data LSB	Checksum Byte
0	54	0	1	134	160	163

3.0 ASCII Commands: The communication protocol consists of two commands, they are WRITE and READ. These commands are used to modify or monitor the contents of the Synaptron's internal registers.

3.1 ASCII WRITE Command: The WRITE command consists of address characters, a delimiter, register characters, a delimiter, the data characters, and the termination characters. If sent correctly the Synaptron will respond with an acknowledge string.

Address Characters and delimiter- Two characters representing an address between "54" and "99" should be sent. "54" is the default unit address and "99" represents a broadcast command. The address characters should be followed by a comma that acts as a delimiter.

Register Characters and delimiter- Two or three characters representing the register that the data is destined for. If writing a 32-bit value 128 should be added to the register value. For example, if writing a 32-bit value to registers 10 and 9 the register characters would be 10 + 128 = "138". The register characters should be followed by a comma that acts as a delimiter.

Data String- Unless operating in 32-bit mode values should less than 32767 but more than -32768. No thousands indicator should be used and negative numbers should be preceded by a "-".

Termination Characters- The ASCII command string is terminated by a carriage return (<cr> = 0x0d) followed by a line feed (<lf> = 0x0a).

Response- The Synaptron it will return a "0K"<cr><lf> as a response.

Example: This example shows is a WRITE command in 16-bit mode writing the 10,000 to register 5 (the *PositionLow* register).

Data Sent to Synaptron					
Address Characters + delimiter		**Register Characters + delimiter**		**Data Characters**	**Termination**
Address	delimiter	Register	delimiter	Data	Termination
"54"	","	"05"	","	"10000"	<cr><lf>
Data Received from Synaptron					
Response					
"OK"<cr><lf>					

3.2 ASCII Read Command: The READ command consists of address characters, a delimiter, register characters, a delimiter, and the termination characters. If sent correctly the Synaptron will respond with a response string.

Address Characters and delimiter- Two characters representing an address between "54" and "99" should be sent, with "99" representing a broadcast command. The address characters should be followed by a comma that acts as a delimiter.

Register Characters and delimiter- Two or three characters representing the register that is to be read from. If reading a 32-bit value 128 should be added to the register value. For example, if reading a 32-bit value from registers 10 and 9 the register characters would be 10 + 128 = "138". The register characters should be followed by a comma that acts as a delimiter.

Termination Characters- The ASCII command string is terminated by a carriage return (<cr> = 0x0d) followed by a line feed (<lf> = 0x0a).

Response String- The response begins with the unit's address and a delimiting character. Unless operating in 32-bit mode values should less than 32767 but more than -32768. No thousands indicator should be used and negative numbers should be preceded by a "-". The response string will terminate with a <cr><lf>.

Example: This example shows a READ command reading register 5 (the *PositionLow* register), where 10,000 is the returned value. 16-bit mode is used.

Data Sent to Synaptron				
Address Characters + delimiter		**Register Characters + delimiter**		**Termination**
Address	delimiter	Register	delimiter	Termination
"54"	","	"05"	","	<cr><lf>
Data Received from Synaptron				
Response				
"54,10000"<cr><lf>				

Example: This example shows a READ command reading register 6 (the *PositionHigh* register), where 100,000 is the returned value. 32-bit mode is used. Note that 128 is added to the register value.

Data Sent to Synaptron				
Address Characters + delimiter		**Register Characters + delimiter**		**Termination**
Address	delimiter	Register	delimiter	Termination
"54"	","	"134"	","	<cr><lf>
Data Received from Synaptron				
Response				
"54,100000"<cr><lf>				

4.0 16-bit and 32-bit mode: Synaptron defaults to accepting 16-bit values (-32,768 to +32,767) for register writes, and returning 16-bit values for register reads. In some cases it may be desirable to write or read 32-bit values. For example, if 32-bit positions are used reading the *PositionLow* and *PositionHigh* registers separately could allow one register to change while the READ command for the other is being implemented. Reading these registers as a single 32-bit value prevents this. In order to use 32-bit values the *Function.F_32Pos* bit should be set via the serial interface. To indicate to the Synaptron that you are reading or writing 32-bit values the most significant bit of the index byte should be set. A simple way to accomplish this is to add 128 (hexadecimal 0x80) to the index value of the register you are writing to or reading from.

Binary Communication Examples:
Example: 32-bit vs. 16-bit WRITE. Writing 10,000 to *PositionHigh* (index 6) and *PositionLow* (index 5) registers.

16-bit example: Data sent - 0,54,0,5,39,16,142					
Data received – 6					
Data sent - 0,54,0,6,0,0,196					
Data received – 6					
32-bit example: Data sent - 0,54,0,134,0,0,39,16,13					
Data received – 6					
Register contents		**16-bit mode**		**32-bit mode**	
Register 6 *(PositionHigh)*	High byte	0	0	0	10,000
	Low byte	0		0	
Register 5 *(PositionLow)*	High byte	39	10,000	39	
	Low byte	16		16	

Example: 32-bit vs. 16-bit READ. Reading from the *PositionHigh* (index 6) and *PositionLow* (index 5) registers.

16-bit example: Data sent - 0,54,0,5,197					
Data received – 0,54,39,16,147					
Data sent - 0,54,0,6,196					
Data received – 0,54,0,0,202					
32-bit example: data sent 0,54,0,134,68					
Data received – 0,54,0,0,39,16,147					
Register contents		**16-bit mode**		**32-bit mode**	
Register 6 *(PositionHigh)*	High byte	0	0	0	10,000
	Low byte	0		0	
Register 5 *(PositionLow)*	High byte	39	10,000	39	
	Low byte	16		16	

5.0 Register Descriptions:

Index	Register Name	Default Value	Min. Value	Max Value
0	*FlashCycles*	9998	0	9998
1	*UnitAddress*	54	54	98
2	*Command*	0	65	71
3	*Function*	0	See register	See register
4	*Status*	0	See register	See register
5	*PositionLow*	0	-32768	32767
6	*PositionHigh*	0	-32768	32767
7	*Velocity*	0	-32768	32767
8	*Acceleration*	0	-32768	32767
9	*NegativeLimitLow*	0	-32768	32767
10	*NegativeLimitHigh*	0	-32768	32767
11	*PositiveLimitLow*	0	-32768	32767
12	*PositiveLimitHigh*	0	-32768	32767
13	*ControlLoopRate*	4	1	32767
14	*NegativePWMLimit*	-3685	-32768	0
15	*PositivePWMLimit*	3685	0	32767
16	*PWMFrequency*	20000	3000	20000
17	*PWMOutput*	0	-32768	32767
18	*MaxDutyCycle*	3685	See register	See register
19	*PIDDivider*	1	1	32767
20	*PTerm*	1	0	32767
21	*ITerm*	0	0	32767
22	*DTerm*	0	0	32767
23	*ErrorBand*	0	0	32767
24	*AnalogSampleCount*	16	1	64
25	*AnalogControl*	0	0	3300 (4096)
26	*AnalogFeedback*	0	0	3300 (4096)
27	*ControlSource*	0	0	2
28	*ControlMultiplier*	1	-32768	32767
29	*ConctrolDivider*	1	-32768	32767
30	*ControlOffset*	0	-32768	32767
31	*ControlResultLow*	0	-32768	32767
32	*ControlResultHigh*	0	-32768	32767
33	*FeedbackSource*	0	0	4
34	*FeedbackMultiplier*	1	-32768	32767
35	*FeedbackDivider*	1	-32768	32767
36	*FeedbackOffset*	0	-32768	32767
37	*FeedbackResultLow*	0	-32768	32767
38	*FeedbackResultHigh*	0	-32768	32767
39	*ControlInputLow*	0	-32768	32767
40	*ControlInputHigh*	0	-32768	32767
41	*BaudValue*	3	0	4
42	*Signal*	0	0	32767
43	*SignalTimeBase*	1	0	3
44	*Ticks*	0	0	32767
45	*Current*	0	0	32767
46	*CurrentMultiplier*	129	0	32767
47	*CurrentDivider*	100	0	32767
48	*NegativeCurrentLimit*	-1500	-32768	0
49	*PositiveCurrentLimit*	1500	0	32767
50	*IndexLow*	0	-32768	32767
51	*IndexHigh*	0	-32768	32767
52	*Function2*	0	-32768	32767
53	*VelocityLimit*	0	0	32767
54	*Reg54*	0	-32768	32767
55	*Reg55*	0	-32768	32767

0. FlashCycles – Register settings are stored in non-volatile flash memory. This memory has a limited number of erase-write cycles. Each time you implement a STORE command the number of cycles counts down. When the number reaches 0 the manufacturer erase-write cycle limitations has been reached (note: bootloading in a firmware update will reset this value to 9998).

1. UnitAddress – Serial communication is directed to specific address values. This allows multiple units to share a serial DIN line. This register determines which address the unit will accept commands from and respond to. A unit will also accept commands sent address 99, but will not issue any response. 99 is the broadcast address.

2. Command – Writing specific values to this register implements various commands. The *Command* register will always READ "0".

Command Table

Command	Value	Description
Read All Registers	65	This command will return all register values at one time. If binary communication is used the first byte will be a 0 and the second the unit's address. This will be followed by 2 bytes for each register starting with index 0. The last byte sent will be a checksum. None of the register values are separated by delimiters so an accurate count of received data is be necessary separate the data into internal registers. If the command is implemented using ASCII communication mode the returned string will be the address of the unit followed by a comma, and then each register's contents (starting with index 0) separated by commas. **Note: In ASCII mode this command may take several milliseconds to process. This can impact the control loop timing.**
Restore defaults	66	This command restores the default register settings and stores them in memory.
Store registers	67	This command stores register values in non-volatile memory. These values will become the new power-on settings. Note that the *FlashCycles* register is decremented each time this command is sent. **Care should be taken not to overuse the flash memory write cycles by reducing *FlashCycles* to 0.**
Reset H-bridge	68	If the motor control H-bridge enters a fault condition this command will reset the H-bridge. This is typically indicated when the *Status.S_BrakeFault* bit is set. This has no impact on conditions that cause the firmware controlled current limit functionality to be active.
Disable H-bridge	69	Turns off the motor control H-bridge.
Read firmware	70	Returns the firmware revision in the product. The format of the response matches a READ response whichever communication mode is used. For binary communication the data packet will be 16-bits long, MSB first.
Reset Microcontroller	71	Causes the microcontroller to enter an infinite loop forcing a watchdog timer reset condition. This may take up to 2 seconds, and the motor will be stopped during the reset period.
Clear Reset Flag	72	Clears the *Status.S_Reset* bit

3. Function – Setting or clearing individual bits in this register adjusts the device functionality. ***Function* Register Bits**

Bit	Name	Description
0	F_32Pos (32-bit mode)	0: Module maintains position as a 16-bit value from -32,768 to +32,767 and will rollover when those thresholds are crossed.
		1: Module maintains a 32-bit position.
1	F_SoftLimits (Software limits)	0: Software limits are disabled.
		1: Software limits are enabled. Software limits act as virtual limit switches Movement is restricted past the positive limit moving forward and the negative limit moving in reverse.
2	F_HardLimits (Hardware Limits)	0: Hardware limits are disabled.
		1: Switch closures (connections to ground) at pins 6 and 7 cause the motor to stop. When asserted, movement is restricted past the positive limit moving forward and the negative limit moving in reverse.
3	F_SwapAB (Swap Encoder Channel A and Channel B)	0: Quadrature encoder channel A leading channel B indicates forward movement.
		1: Quadrature encoder channel B leading channel A indicates forward movement.
4	F_StopMotor (Stop Motor)	0: No effect.
		1: Prevents the motor from moving.
5	F_EnableOnBrake (Re-enable On Fault Condition)	0: If the _BRAKE/_FAULT pin (pin 8) is 0V the controller will not attempt to reset the H-bridge.
		1: If the _BRAKE/_FAULT pin (pin 8) is not 0V the module will try to clear the fault condition by resetting the H-bridge.
6	F_IgnoreBrake (Ignore Brake Input)	0: When 0V is present on pin 8 the controller will attempt to stop the motor.
		1: The controller will ignore the brake pin when controlling the motor.
7	F_RawSignal (Use Raw Signal Value)	0: Positive pulse width at the SIGNAL pin (pin 24) is converted to 1uS per bit.
		1: Positive pulse width at the SIGNAL pin (pin 24) is left in its raw measurement state as determined by the *SignalTimeBase* register.
8	F_ConvertToFreq (Convert Signal to Frequency)	0: The positive pulse duration of the signal at pin 24 is reported in the *SIGNAL* register.
		1: Signal at pin 24 is converted to a frequency. SIGNAL at pin 24 must have a 50% duty cycle, and the *Function.F_RawSignal* bit must be clear, for this to be accurate.
9	F_RawVoltages (Use Raw Voltage)	0: Voltages at ANA CONTROL and ANA FEEDBACK (pins 19 and 20) are converted to 1mV per bit (0-3300).
		1: Voltages pins 19 and 20 are not converted increasing resolution to 0.81mV per bit (0-4095).
10	F_ClosedLoopEB (Closed Loop Error Band)	0: *ErrorBand* register setting has no impact on closed loop control.
		1: If the control and feedback settings are within +/- *ErrorBand* of each other the PID error is set to 0. This causes the proportional and derivative terms to quickly reach 0 and the integral term to follow shortly thereafter. This functionality may have adverse impact to velocity control systems.
11	F_OpenLoopEB (Open-Loop Error Band)	0: No effect.
		1: *PWMOutput* signals generated within +/- *ErrorBand* of 0 are set to 0. This creates a dead band around 0. The PWM control outside of the dead band area is not proportional. For example, with the *ControlSource* and *FeedbackSource* set to 0 and the *ErrorBand* is set to 100 a *ControlInputLow* of 100 would result in a 0 in the *PWMOutput* register, but a *ContolInputLow* of 101 would result in an output of 101.

12	F_Encoder4X (encoder count 4X)	0: The quadrature encoder input is multiplied by 2. Example, a 500CPR encoder produces 1000 counts per revolution in the *Position* registers.
		1: The quadrature encoder input is multiplied by 4. Example, a 500CPR encoder produces 2000 counts per revolution in the *Position* registers.
13	F_PosPowerUp (load position on power up)	0: The *Position* registers are set to 0 on power up.
		1: The Position registers are loaded with the last value recorded in memory by a Store command (see *Command* register) when the module is powered up.
14	F_VelocityLimit	0: No effect
		1: Only when *ControlSource* = 0, the *VelocityLimit* register value is used to limit the speed of position changes
15	F_AlternatePID	0: No effect
		1: The output of the PID algorithm is added to the last PWM motor drive signal. Should only be used when *FeedbackSource* = 5

4. Status – The various bits of this register will be set or cleared based on the unit's current operating status.

***Status* Register Bits**

Bit	Name	Description
0	S_SoftNegLim (negative software limit)	0: Nothing.
		1: Motor moving in negative direction has reached the negative software limit position in the *NegativeLimit* registers. Motor is stopped. *Function.F_SoftLimits* must be set for this function to work.
1	S_SoftPosLim (positive software limit)	0: Nothing
		1: Motor moving in positive direction has reached the positive software limit position in the *PositiveLimit* registers. Motor is stopped. *Function.F_SoftLimits* must be set for this function to work.
2	S_HardNegLim (negative hardare limit)	0: Negative hardware limit input (pin 6) is not grounded.
		1: Negative hardware limit input (pin 6) has been connected to ground. Motor will be prevented from moving in negative direction if *Function.HardLimits* bit is set.
3	S_HardPosLim (positive hardware limit)	0: Positive hardware limit input (pin 7) is not grounded.
		1: Positive hardware limit input (pin 7) has been connected to ground. Motor will be prevented from moving in positive direction if *Function.HardLimits* bit is set.
4	S_BrakeFault (Brake or Fault condition)	0: Brake (pin 8) input or Fault (signal from H-bridge) is not grounded.
		1: Brake (pin 8) input or Fault (signal from H-bridge) has been connected to ground.
5	S_Boot (Bootloader pin)	0: Boot (pin 23) input is not grounded.
		1: Boot (pin 23) input has been connected to ground. If this pin is connected to ground at power up the module will enter bootloader mode and will not be able to communicate normally. If the Boot pin is grounded after power up the internal registers will be set to their default values. Registers can only be reset in this manner once each power cylce.
6	S_ErrorBand (Error band indicator)	0: System is not within the error band.
		1: In a closed loop system (*FeedbackSource* does not equal 0) this bit indicates that absolute value(*ControlResult – FeedbackResult)* <= *ErrorBand.* In an open loop system (*FeedbackSource* = 0) with *Function.OpenLoopEB* set this bit indicates that absolute value(*ControlResult)* <= *ErrorBand.*

7	S_UserCurrentLimit (User set current limit)	0: Nothing
		1: *Current* register is less than *NegativeCurrentLimit* register or greater than *PositiveCurrentLimit* register. Motor current limiting function is active and PWM signal is being limited.
8	S_IntCurrentLimit (Internal current limit)	0: Nothing
		1: Internal measurements indicate a current greater than approximately 5A. Motor is stopped while condition persists.
9	S_Index (Index pin input)	0: Encoder position register does not match last known index position.
		1: Encoder position is within +/-2 positions of the last known index position (+/-4 counts if *Function.F_Encoder4X* is set).
10	S_Reset (reset flag)	0: Flag has been cleared and a reset has not occurred.
		1: A reset has occurred. This flag is set anytime the operating system initializes. It can only be cleared by writing a 72 to the *Command* register.
11	Unused	
12	unused	
13	unused	
14	unused	
15	unused	

5. PositionLow – This holds the lower 16 bits of the position value from a quadrature encoder connected to pins 3,4, and 5 (if an index connection is used). This is a 2x count of the actual encoder. For example, a 500CPR encoder will create a position value of 1000 for a full revolution (500x2).

6. PositionHigh – In 32-bit mode this holds the upper 16-bits of the position value. In 16-bit mode this register will always be read as a 0.

7. Velocity – The change in the quadrature encoder position from one control loop to the next. The counts per second for a specific velocity can be calculated with this equation:

Counts Per Second = *Velocity*/(250uS * *ControlLoopRate*)

8. Acceleration – The change in velocity from one control loop period to the next.

9. NegativeLimitLow – The lower 16-bits of this software limit setting. If software limits are enabled through the *Function* register movement to values below this setting will be restricted.

10. NegativeLimitHigh – In 32-bit mode this holds the upper 16-bits of the negative limit value. In 16-bit mode this register will always be read as a 0.

11. PositiveLimitLow – The lower 16-bits of this software limit setting. If software limits are enabled through the *Function* register movement to values above this setting will be restricted.

12. PositiveLimitHigh – In 32-bit mode this holds the upper 16-bits of the positive limit value. In 16-bit mode this register will always be read as a 0.

13. ControlLoopRate – This register sets the period of time the control loop waits between motor speed changes. It is in increments of 250uS, and defaults to 4 (1mS). Adjusting this register up or down can impact the responsiveness of closed loop systems, and can also impact measurements of velocity and acceleration. Values below the default setting may negatively impact velocity and acceleration measurements.

14. NegativePWMLimit – Can be used to set the lower limit of the PWM output signal that drives the motor. For example, it could be set to 0 to prevent the motor from moving in reverse. It may also be used to limit the drive output to the motor. Setting it to 50% of the *-MaxDutyCycle* can allow you to limit a closed loop system's drive to 50% when moving in reverse. For full scale drive signals the value should be less than, or equal to, *–MaxDutyCycle.*

15. PositivePWMLimit – Can be used to set the upper limit of the PWM output signal that drives the motor. For example, it could be set to 0 to prevent the motor from moving forward. It may also be used to limit the drive output to the motor. Setting it to 50% of the *MaxDutyCycle* register can allow you to limit a closed loop system's drive to 50% when moving forward. For full scale drive signals the value should be greater than, or equal to, *MaxDutyCycle.*

16. PWMFrequency – Adjusts the frequency of the PWM signal (in Hertz). Adjusting this value will also modify the PWM signal resolution and will cause the *MaxDutyCycle* register value to change. It may be necessary to modify the *NegativePWMLimit* and *PositivePWMLimit* registers. Values below 10,000Hz can usually be heard by the human ear.

17. PWMOutput – Displays the value of the signal driving the motor. A negative value is reverse, and positive is forward. The duty cycle of the signal may be calculated with this equation:

$$\frac{PWMOutput}{MaxDutyCycle} * 100$$

18. MaxDutyCycle – Absolute value of the maximum allowed duty cycle value. This number is calculated from the *PWMFrequency* setting. The default value is 3685 which represents 3685 steps of control. A *PWMOutput* of 1842 would represent 50% duty cycle moving forward, while a *PWMOutput* of -1842 represents 50% duty cycle in reverse.

19. PIDDivider - Divides the output of the PID filter.

20. PTerm – The proportional gain setting of the PID filter. The error signal (the control source value minus the feedback source value) is multiplied by this term to provide a motor drive signal proportional to the error signal. This term usually has the most impact on motor drive.

21. ITerm - The integral gain setting of the PID filter. This term is multiplied by a continuing summation of error signals. It is used to allow small errors to be acted upon over time by the PID filter. Typically this will be a very small value in comparison to the *PTerm*. Large *ITerm* values will cause oscillation of the output signal.

22. DTerm – The derivative gain setting of the PID filter. This term is multiplied by the difference between a previous error signal and the current error signal. This creates 'drag' on abrupt changes in the motor drive signal.

23. ErrorBand – There are two uses of the *ErrorBand* register. The first occurs when the *Function.ClosedLoopEB* bit is set. When this bit is set if the absolute value of the *ControlResult-FeebackResult* is less than the *ErrorBand* register, then the PID error is set to 0. This reduces hunting by the motor controller when it is near the commanded value.

The second use occurs when the *Function.OpenLoopEB* bit is set. When this bit is set the *PWMOutput* is set to 0 when the absolute value of the *PWMOutput* register is less than or equal to the *ErrorBand* register.

24. AnalogSampleCount – The analog samples used to fill the *AnalogControl*, *AnalogFeedback*, and *Current* registers may be sampled up to 64 times before an average is taken. The sample rate is not based on the *ControlLoopRate*, but higher counts numbers would naturally increase the time required to update each measurement.

25. AnalogControl – This register holds the analog measurement of the voltage at the ANA CONTROL pin (pin 19). This voltage may be used as a control signal input. The input voltage must be between 0-3.3VDC. The register contents will default to 1 count per mV, or a value from 0-3300. Setting the *Function.RawVoltages* bit will fill the register with the raw 12-bit conversion (0-4095 or 0.81mV per bit).

26. AnalogFeedback – This register holds the analog measurement of the voltage at the ANA FEEDBACK pin (pin 20). This voltage may be used as a feedback signal input. The input voltage must be between 0-3.3VDC. The register contents will default to 1 count per mV, or a value from 0-3300. Setting the *Function.RawVoltages* bit will fill the register with the raw 12-bit conversion (0-4095 or 0.81mV per bit).

27. ControlSource – This register determines the source of the control signal.

Control Sources

Register Value	Source	Comments
0	*ControlInput* registers	The control signal is derived from contents written to the *ControlInputLow* and *ControlInputHigh* registers.
1	Analog (*AnalogControl* register)	The control signal is derived from the analog voltage at pin 19.
2	Signal (*Signal* register)	The control signal is derived from the signal or frequency at pin 24.

28-32. ControlMultiplier* through *ControlResult - These control registers are used to adjust the input values in the following manner.

$$\boldsymbol{ControlResult} = (\boldsymbol{ControlSource\ value} + \boldsymbol{ControlOffset}) * \frac{\boldsymbol{ControlMultiplier}}{\boldsymbol{ControlDivider}}$$

The results of the adjustment are stored in the *ControlResult* registers. The results are used as the control signal for open and closed loop control.

Example: Assume that you have an analog signal of 1-2VDC, and you want it to control a motor in full forward and full reverse with no feedback.

Register	Value	Comments
ControlSource	1	Selects *AnalogControl* as the feedback source
FeedbackSource	0	Selects no feedback source, or open loop control
PWMFrequency	20000	Determines PWM resolution with a maximum of +/-3685
Function.RawVoltages	1	Set this bit to increase the resolution of the *AnalogControl* signal to a full 12-bits or 0.81mV per bit.
ControlOffset	-1850	We want 1.5V to equal a duty cycle of 0% (motor off). 1.5/0.81mV is roughly 1850.
ControlMultiplier	590	We want 2V to equal full scale forward, or ~3685. (2V-1.5V)/0.81mV = 617. ControlMultiplier/ControlDivider = 3685/617 = 5.97. Since all the math is done using integers we set the multiplier to 590 and the divider to 100. This will give us an approximation of full scale control and fine adjustments may be mad later
ControlDivider	100	See above.

If the settings are correct then using an input of 1V should result in full speed reverse.

$$\boldsymbol{ControlResult} = \left(\frac{\boldsymbol{1V}}{\boldsymbol{0.81mV}}\right) - \boldsymbol{1850}) * \frac{\boldsymbol{590}}{\boldsymbol{100}}$$

$\boldsymbol{ControlResult} = \boldsymbol{-3631}$**, or about 99% duty cycle in reverse.**

33. FeedbackSource – This register determines the source of the feedback signal.

Feedback Sources

Register Value	Source	Comments
0	No feedback	Selected to create an open-loop controller.
1	Analog (*AnalogFeedback* register)	The feedback signal is derived from the analog voltage at pin 20.
2	Signal (*Signal* register)	The feedback signal is derived from the signal or frequency at pin 24.
3	Quadrature Encoder (*Position* registers)	The feedback signal is derived from an encoder attached to pins 3 (CHA) and 4 (CHB).
4	Current (*Current* register)	The feedback signal is derived from the internal current measurement circuit.
5	Velocity (*Velocity* register)	The feedback signal is derived from the velocity as calculated by the change in encoder position from one control loop period to the next.

34-38. FeedbackMultiplier* through *FeedbackResult - These feedback registers are used to adjust the input values in the following manner.

$$\boldsymbol{FeedbackResult} = (\boldsymbol{FeedbackSource\ value} + \boldsymbol{FeedbackOffset}) * \frac{\boldsymbol{FeedbackMultiplier}}{\boldsymbol{FeedbackDivider}}$$

The results of the adjustment are stored in the *FeedbackResult* registers. The results are used as the feedback signal used for closed loop control.

Example: Assume that you have an analog feedback signal of 1-2VDC, and you want to match it to a control source voltage of 0-3.3V.

Register	Value	Comments
ControlSource	1	Selects *AnalogControl* as the control source
FeedbackSource	1	Selects *AnalogFeedback* as the feedback source
PWMFrequency	20000	Determines PWM resolution with a maximum of +/-3685
Function.RawVoltages	0	Clear this bit to have a resolution of 1mV per bit. The analog measurement will now range from 0mV to 3300mV
FeedbackOffset	-1000	We want 1.0V to equal an output of 0. So we subtract 1000mV from the feedback source.
FeedbackMultiplier	33	We want 2V to equal a full scale analog measurement, or 3300mV. Since all the math is done using integers we set the multiplier to 33 and the divider to 10, giving us a scaling of 3.3.
FeedbackDivider	10	See above.

If the settings are correct then using an input of 1V should result in a feedback signal of 0, and 2V will result in 3300.

$$\boldsymbol{FeedbackResult} = ((\boldsymbol{1000mV}) - \boldsymbol{1000mV}) * \frac{\boldsymbol{33}}{\boldsymbol{10}} = \boldsymbol{0}$$

$$\boldsymbol{FeedbackResult} = ((\boldsymbol{2000mV}) - \boldsymbol{1000mV}) * \frac{\boldsymbol{33}}{\boldsymbol{10}} = \boldsymbol{3300}$$

39. ControlInputLow - This register contains the lower 16-bits of the serial control register. If the *ControlSource* is 0 (serial control) then the contents of this register are used to set motor speed or desired control values.

40. ControlInputHigh – This register contains the upper 16-bits of the serial control register. If the *Function.F_32Pos* bit is clear this register will be 0.

41. BaudValue – You may modify the serial communication baud rate by changing the value in this register. The register defaults to 3 (9.6Kbps). When modifying the value the WRITE command will be acknowledged at the original baud rate, but all subsequent commands should occur using the new baud rate.

Register Value	Baud Rate	Time for Read All command (65) to be executed in binary communication mode. Includes time to send command.
0	115.2Kbps	~10ms
1	57.6Kbps	~20ms
2	38.4Kbps	~30ms
3	9.6Kbps	~115ms
4	1.2Kbps	~925ms

42. Signal - The positive pulse width of the signal at pin 24 will be stored in this register. The register defaults to 1uS per bit, so a 1mS signal would be read as 1000. Setting the *Function.F_RawSignal* bit can allow greater resolution and accuracy of signal measurements. Adjusting the *SignalTimeBase* register value can also adjust resolution and the length of signals that can be measured. If the *Function.F_ConvertToFreq* bit is set, and the *Function.F_RawSignal* is clear the signal measurement will be converted to a frequency. For the frequency conversion to be accurate the signal must have a 50% duty cycle.

Attempting to measure signals longer than the maximum allowed by the *SignalTimeBase* setting can cause this register to rollover giving inaccurate values.

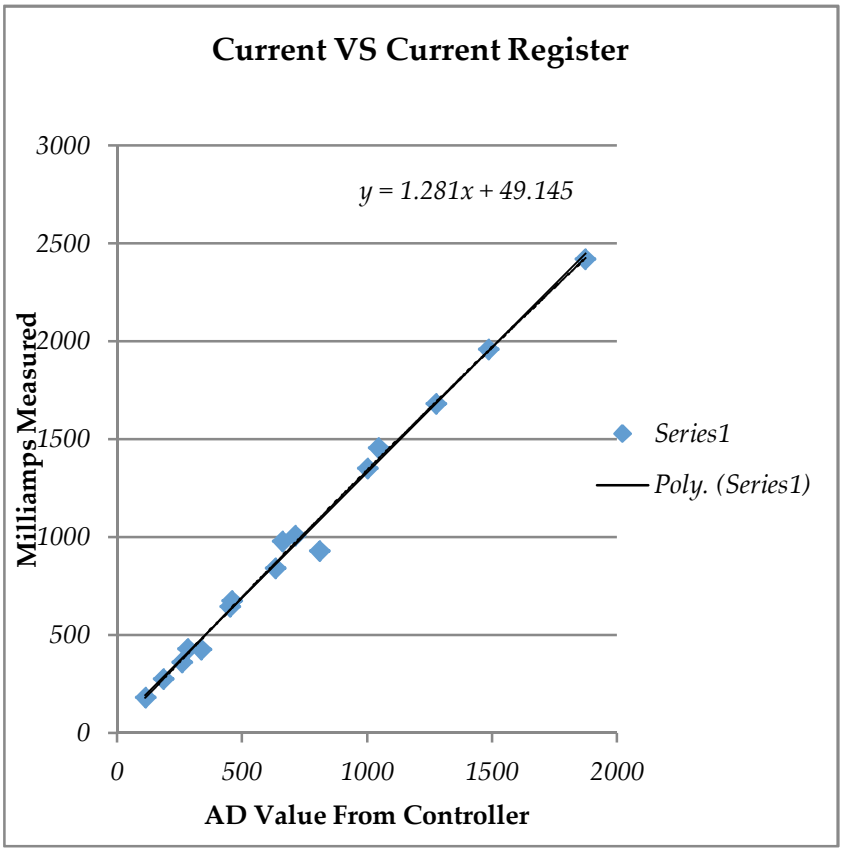
NOTE: Using the SIGNAL input as a control or feedback source will limit the update rate. The pulse measurement at the SIGNAL input is measured every 3 pulses. So a 5ms pulse would effectively create an update rate of 15ms regardless of the *ControlLoopRate* setting.

43. SignalTimeBase – The *SignalTimeBase* register allows the user to select different time bases for pulse-width or frequency measurement. You should select the setting that provides the greatest resolution, accuracy, and range of measurements that meets your needs. The *Function.F_RawSignal* bit has an impact on how the pulse count is made.

Register Value	**Maximum Pulse Width**		**Resolution**	
	F_RawSignal=0	*F_RawSignal=1*	*F_RawSignal=0*	*F_RawSignal=1*
0	1759uS	885uS	1uS	0.027uS per count
1	14075uS	7110uS	1uS	0.217uS per count
2	32767uS	57014uS	2uS	1.74uS per count
3	32767uS	227522uS	7us	6.95uS per count

44. Ticks – This register increments every time the control loop is executed (the default setting is 1ms). It will roll over at 32,767.

45. Current – The controller measures motor current and stores the result in this register. The measurement is most accurate between 500mA and 3A of load current, and is estimated to be +/-20% of the actual value in that range. Different loads and drive signals can have a significant impact on the current measured, so this value should be considered a first order approximation of load current. The value loaded into this register is modified by the *CurrentMultiplier* and *CurrentDivider* registers.



46-47. CurrentMultiplier and CurrentDivider – The current measurement made by the controller is multiplied by the *CurrentMultiplier* and then divided by the *CurrentDivider* prior to being stored in the *Current* register. The user may modify these register to create more accurate current measurements based on the conditions of their system. The system default is 129 for the multiplier and 100 for the divider. That results in a 1.29 multiplier of the analog current measurement, which has been shown to approximate milliamps in the range of 500mA to 3A.

$$\boldsymbol{Current} = \frac{\boldsymbol{CurrentMultiplier}}{\boldsymbol{CurrentDivider}} * \boldsymbol{Analog\ Current\ Measurement}$$

48-49. NegativeCurrentLimit and PositiveCurrentLimit– The user may set a maximum motor current while running in either direction. Each bit equals approximately 1mA of motor current, so a negative limit of -2000 is equal to -2000mA, or -2A. When this limit is reached the controller will restrict increases in the PWM drive signal to keep the motor current below the limit. See registers 45-47 about the accuracy of these measurements.

50. LastIndexLow – Holds the lower 16-bit position value associated with the last time the Index input (pin 5) was at a logic low. This is a 2x count of the actual encoder. For example, a 500CPR encoder with an index pin will create a "last index" position every 1000 counts. (500x2).

51. LastIndexHigh – Holds the upper 16-bit position value associated with the last time the Index input (pin 5) was at a logic low. In 16-bit mode of operation this register is always 0.

52. Function2- Unused.

53. VelocityLimit – When the *Function.F_VelocityLimit* bit is set and the *ControlSource* register = 0 the contents of the *VelocityLimit* register are used to limit motor movement per PID update.

54. Reg54 – Unused.

55. Reg55 – Unused.