Firstbot
Robot Controller

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.

Product Description:

The Firstbot robot controller is a versatile module for controlling robots and other electromechanical systems. The Firstbot includes Arduino-compatible hardware allowing you to program the system with the popular Arduino-compatible open-source software.

In addition to Arduino-compatible features the Firstbot has two onboard DC motor controllers, 4 RC servo input pins and 4 RC servo outputs. These functions are provided by a separate microcontroller whose firmware is also provided open-source for you to modify and improve.

- Open-source hardware and firmware
- Easy to use Arduino-compatible design
- 6-24VDC DC motor operation
- 4 RC servo signal inputs for connecting to RC receivers
- 4 RC servo signal outputs for driving servos
- 2 bi-direction DC motor drivers for controlling motors
- 5 analog inputs
- I2C and SPI ports
- 9 digital i/o
- Solder points for additional connections
- Footprints for programming headers

Arduino-Compatible:

This module contains 2 microcontrollers. The first is an Atmel ATmega328 that is pre-loaded with the open-source Arduino-compatible bootloader. The USB port may be used to program Arduino-compatible code into the controller. All i/o's of the ATmega328 are pulled out to solder points in a fashion similar to popular Atmel based products. Digital pins 2 and 3 are shared with the second microcontroller to create a serial interface to the motor/servo control functions.

Motor/Servo Control:

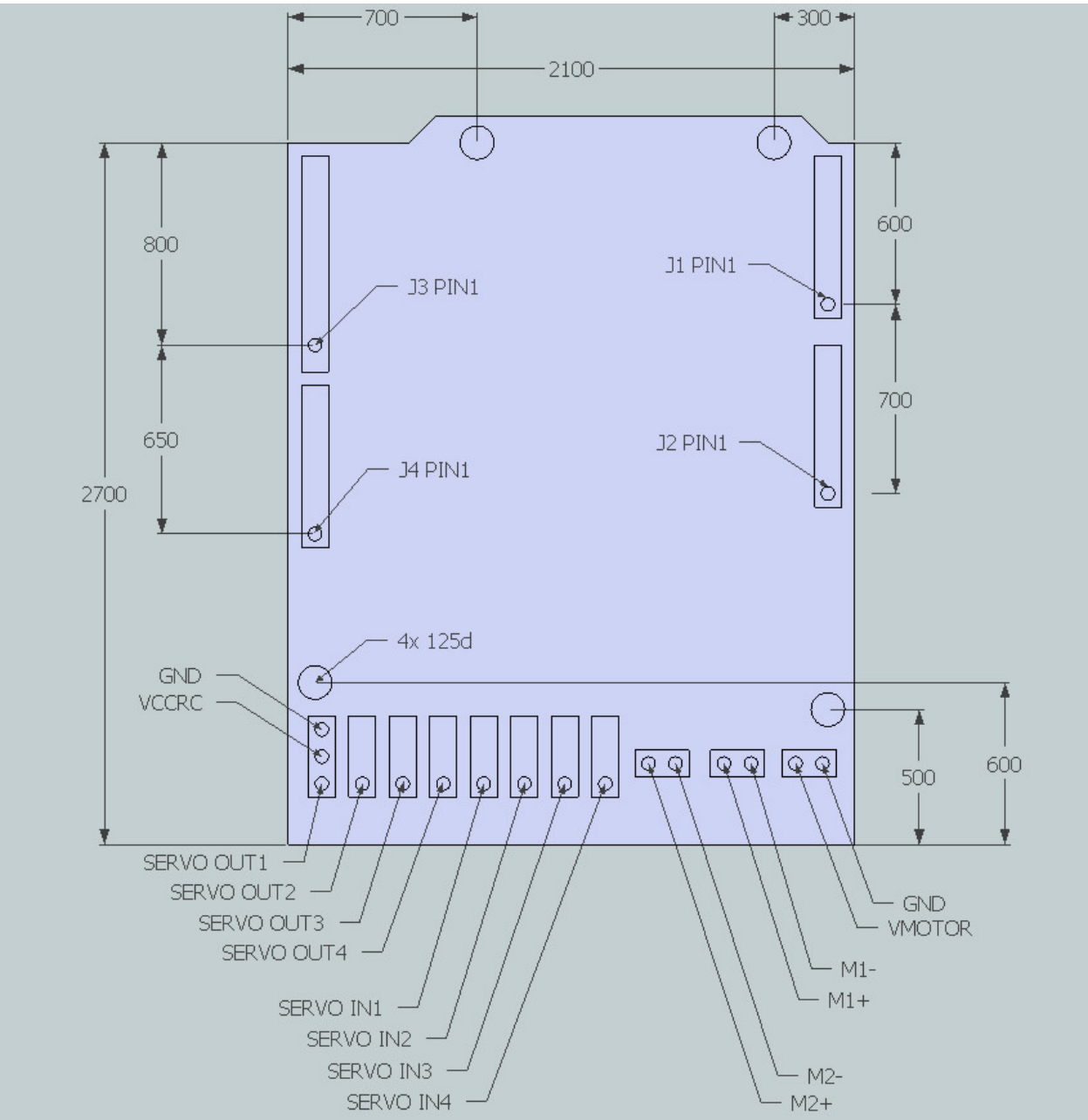
The second microcontroller is a Microchip PIC16F1829. This controller has custom firmware for reading and writing servo pulses and for interfacing to two Freescale MC33926 DC motor controllers. The firmware is open source and written using Microchip's free XC8 C compiler. The firmware implements a simple serial interface for controlling motors and servos (included at the end of this datasheet). The interface matches our BM011 module. Arduino-compatible application notes can be found at www.solutions-cubed.com

Custom Firmware:

Header footprints are located on the Firstbot that connect to programming pins on the ATmega328 and the PIC16F1829 (J7 and J11 respectively). Both can be programmed with custom firmware using development environments and programmers available from Atmel and Microchip.

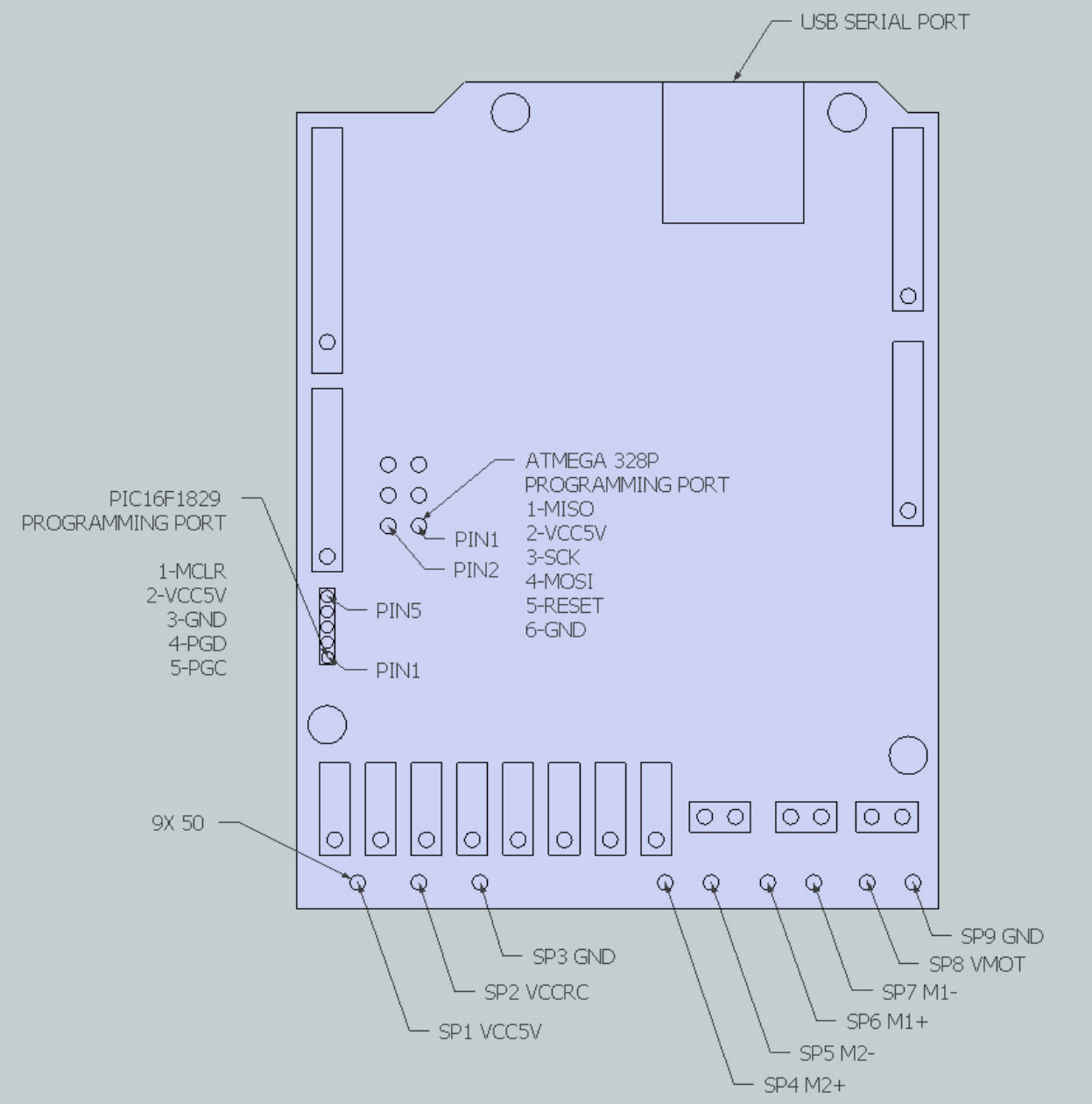
Dimensions:

All dimensions in mils; 1000mils = 1inch = 25.4mm.



Solder Points and Programming Ports:

All dimensions in mils; 1000mils = 1inch = 25.4mm.



Specifications:

Characteristic	Min	Typ	Max	Unit	Notes
VIN Operating voltage	6		24	V	Overall system and motor voltage supply
Onboard regulator 5VOUT	4.75	5	5.25	V	
Onboard regulator current source		100	250	mA	
Operating current		40		mA	No motors/servos attached
VDDRC	4	5	6		Voltage supply for servos
Operating temperature	0		+70	°C	

Motor and Servo Controller

Characteristic	Min	Typ	Max	Unit	Notes
VM Motor Voltage	6		24	V	Operating near the maximum voltage may require external protection from voltage spikes
Continuous Motor Current		1		A	At 90% duty cycle, 12V
Peak Motor Current			5	A	
Motor drive PWM frequency			7.8	KHz	
Servo pulse output/input resolution		10		uS	
Servo pulse output range	0		2550	uS	
Servo pulse output accuracy		+/-20		uS	can be fine-tuned via the Servo_Adjust register
Servo pulse input measurement range	450		2550	uS	register is loaded with a 0 to indicate the pulse width is outside of this range
Servo pulse input accuracyu		+/-20		uS	
Servo pulse output period	12	24	510	mS	this is user adjustable but defaults to 24mS

Pin Functions and Notes

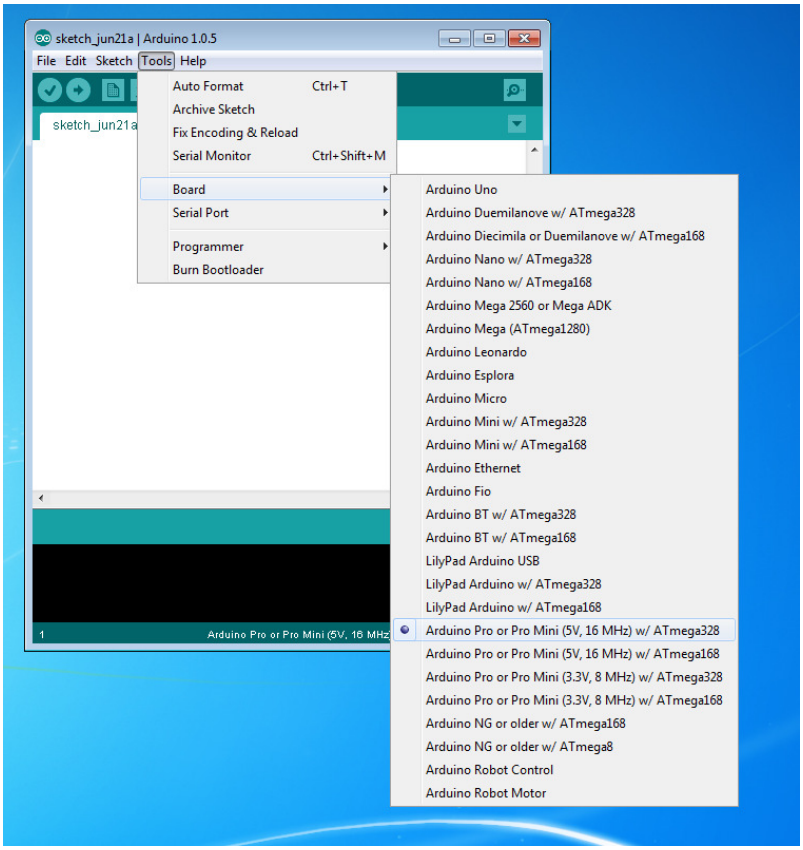
#	Name	Maximum Voltage	Notes
J1			**1x6, single row 0.1" spacing header**
1	A0	5V	10-bit analog input, or digital i/o
2	A1	5V	10-bit analog input, or digital i/o
3	A2	5V	10-bit analog input, or digital i/o
4	A3	5V	10-bit analog input, or digital i/o
5	A4-SDA	5V	10-bit analog input, digital i/o, or I2C data line
6	A5-SCL	5V	10-bit analog input, digital i/o, or I2C clock line
J2			**1x6, single row 0.1" spacing header**
1	RESET	5V	Pull low for reset, shared with USB serial UART circuitry
2	NC	N/A	
3	5V	5V	5V supply can source 100mA current
4	GND	0V	Ground return
5	GND	0V	Ground return
6	VIN	24V	Motor and system supply, connected to the VM pins
J3			**1x8, single row 0.1" spacing header**
1	7	5V	Digital i/o
2	6-PWM	5V	Digital i/o, PWM out
3	5-PWM	5V	Digital i/o, PWM out
4	4	5V	Digital i/o
5	3-PWM-DIN	5V	Digital i/o, PWM out, serial data input to PIC microcontroller
6	2-DOUT	5V	Digital i/o, serial data output to PIC microcontroller
7	1-TX	5V	Serial port transmit pin, shared with USB serial UART circuitry
8	0-RX	5V	Serial port receive pin, shared with USB serial UART circuitry
J4			**1x6, single row 0.1" spacing header**
1	13-LED-SCK	5V	Digital i/o, tied to LED through 1KΩ resistor, SPI clock
2	12-PWM-MISO	5V	Digital i/o, SPI master-out-slave-in
3	11-PWM-MOSI	5V	Digital i/o, PWM out, SPI master-in-slave-out
4	10-PWM-SS	5V	Digital i/o, PWM out, SPI slave select
5	9-PWM	5V	Digital i/o, PWM out
6	8	5V	Digital i/o
J5			**USB mini type B**
J7	**Atmega programming**		**2x3 dual row 0.1" spacing header, square pad indicated hole 1**
1	MISO	5V	Atmega programming interface master-in-slave-out
2	VCC5V	5V	5V supply
3	SCK	5V	Atmega clock
4	MOSI	5V	Atmega programming interface master-out-slave-in
5	RESET	5V	Atmega programming interface reset, shared with USB serial UART
6	GND	0V	Ground return
J8			**1x2 single row 0.1" spacing header, square pad indicated hole 1**
1	VM	24V	Motor and main supply voltage (6-18VDC), connected to VIN pin
2	GND	0V	Ground return
J9			**1x2 single row 0.1" spacing header, square pad indicated hole 1**
1	M1-	24V	Motor 1 negative input
2	M1+	24V	Motor 1 positive input
J10			**1x2 single row 0.1" spacing header, square pad indicated hole 1**
1	M2-	24V	Motor 2 negative input
2	M2+	24V	Motor 2 positive input

Pin Functions and Notes (continued)

#	Name	Maximum Voltage	Notes
J11	**PIC programming**		**1x5 single row 0.05" spacing header**
1	MCLR	5V	Master reset PIC programming interface
2	VCC5V	5V	On board 5V supply output
3	GND	5V	Ground return
4	PGD	5V	PIC programming data
5	PGC	5V	PIC programming clock
J12	**Servo In 1**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Accepts servo signal up to 2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J13	**Servo Out 1**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Outputs servo signal 500uS-2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J14	**Servo In 2**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Accepts servo signal up to 2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J15	**Servo Out 2**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Outputs servo signal 500uS-2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J16	**Servo In 3**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Accepts servo signal up to 2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J17	**Servo Out 3**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Outputs servo signal 500uS-2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J18	**Servo In 4**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Accepts servo signal up to 2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
J19	**Servo Out 4**		**1x3 single row 0.1" spacing header**
1	Signal	5V	Outputs servo signal 500uS-2500uS
2	VCCRC	6V	RC servo voltage supply from SP2 (solder point 2)
3	GND	0V	Ground return
	Solder Points		**0.05" solder points for wiring**
SP1	VCC5V	5V	5V supply can source 100mA
SP2	VCCRC	6V	Supply input for RC servo voltage
SP3	GND	0V	Ground return
SP4	M2+	24V	Motor 2 positive input
SP5	M2-	24V	Motor 2 negative input
SP6	M1+	24V	Motor 1 positive input
SP7	M1-	24V	Motor 1 negative input
SP8	VMOT	24V	Motor supply and main power supply
SP9	GND	0V	Ground return

User Notes/Tips

1. The Firstbot will upload programs similar to an Arduino Pro, 5V 16MHz, with an ATmega328. That board should be selected when uploading programs with the open source Arduino-compatible software.



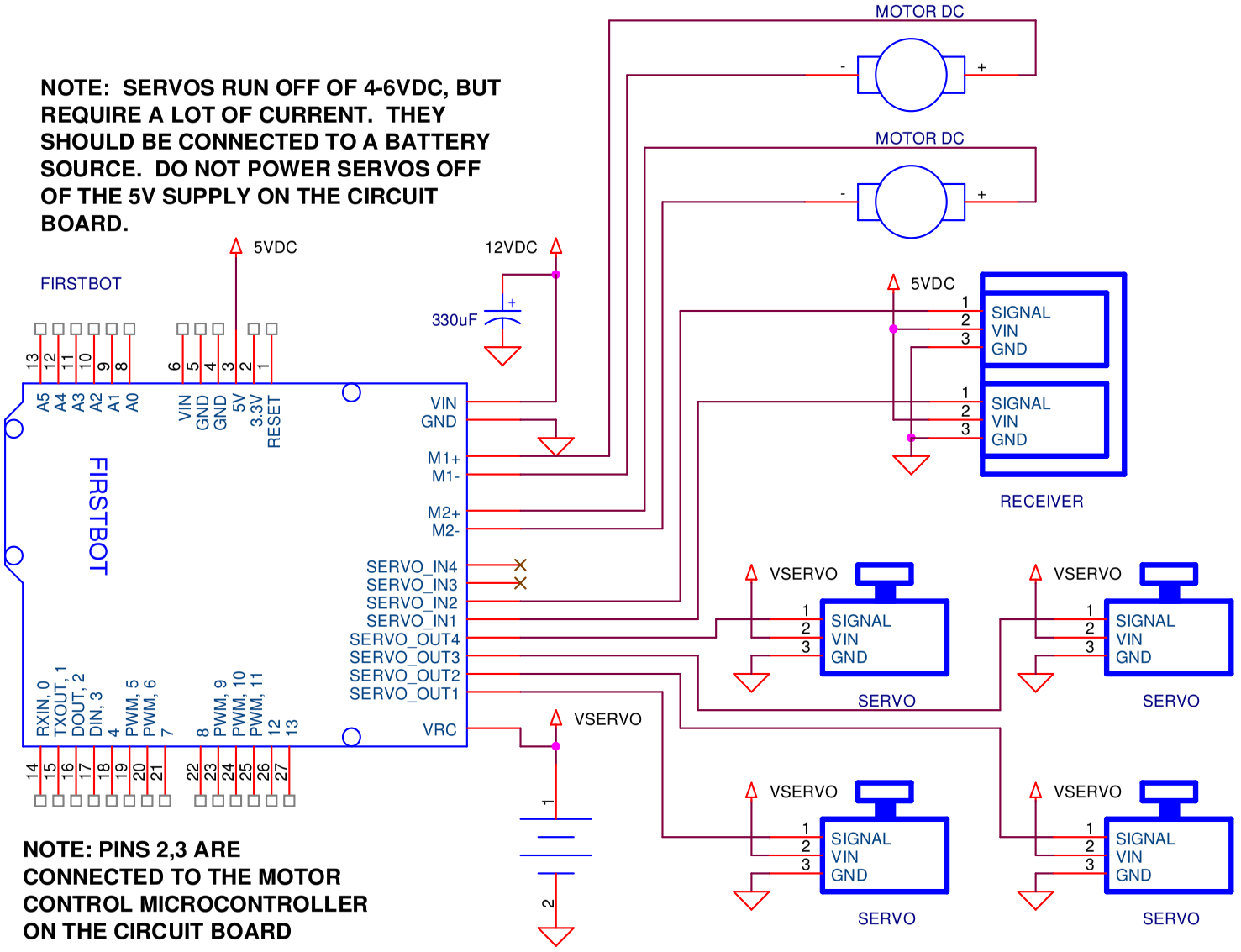
2. To upload new programs into the Atmega328 via the USB port power must be applied to the VM/VIN pins.

3. Pins 2 and 3 of the ATmega328 are shared with PIC16F1829 as serial communication lines. There are 270 ohm resistors isolating the connections. In most cases these i/os should not be used for external connections.

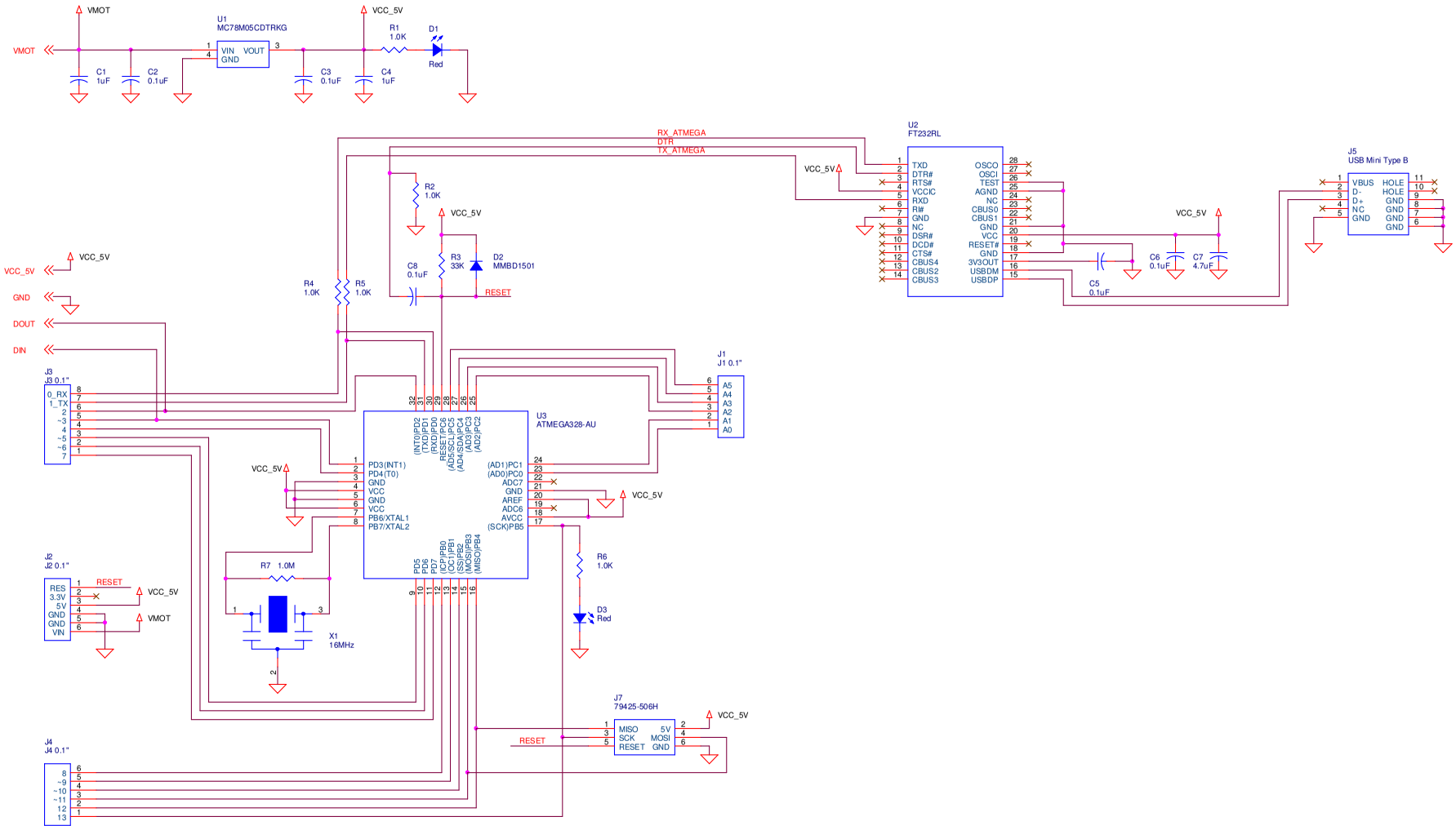
4. The serial communication protocol for the PIC16F1829 matches out BM011 motor/servo controller (as does the firmware).

5. Application notes are available for download at www.solutions-cubed.com.

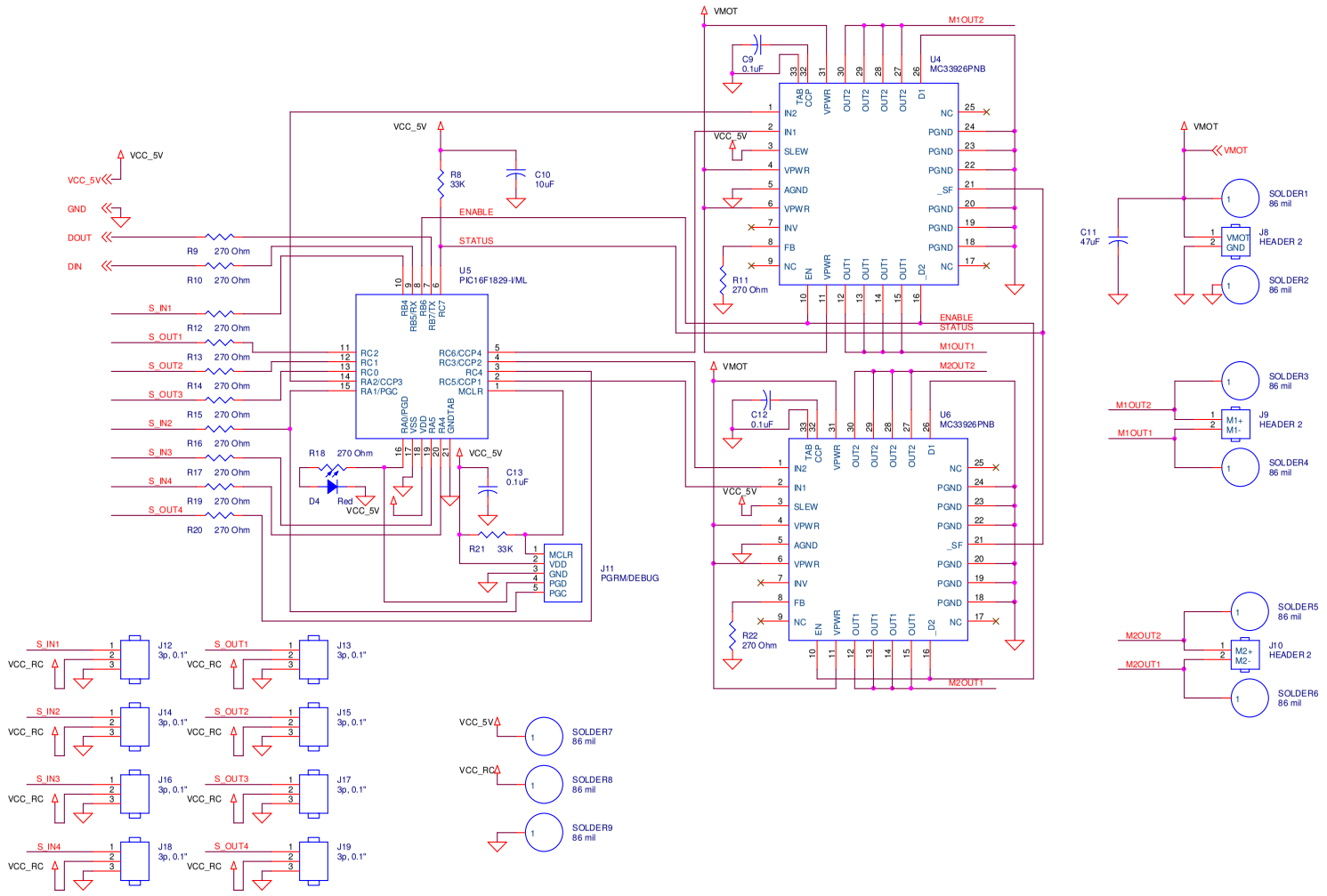
Application Schematics



Schematic (Atmega328 Section):



Schematic (PIC16F1829 Section):



Example Code for Serial Interface to PIC16F1829

```
/*
 *Firstbot - BM011 Dual Motor Quad Servo Controller Communication example

 *The circuit:
 * RX is digital pin 2 (connect to TX of other device)
 * TX is digital pin 3 (connect to RX of other device)

 */
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3); // Receive data on 2, send data on 3
byte SerialTXBuffer[10];
byte SerialRXBuffer[10];

void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(2400);
  mySerial.begin(9600);
}

/* WriteRegister:
Writes a single byte, "Value",  to the register pointed at by "Index".
Returns the response
*/
byte WriteRegister(byte Index, byte Value)
{
byte i = 0;
byte checksum = 0;
byte ack = 0;

SerialTXBuffer[0] = 210;
SerialTXBuffer[1] = 1;
SerialTXBuffer[2] = 3;
SerialTXBuffer[3] = Index;
SerialTXBuffer[4] = Value;

for (i=0;i<6;i++)
  {
  if (i!=5)
    {
    mySerial.write(SerialTXBuffer[i]);
    checksum += SerialTXBuffer[i];
    }
  else
    mySerial.write(checksum);
  }
  delay(5);

            if (mySerial.available())
    ack = mySerial.read();
   return ack;
}
```

```
/* ReadRegister:
Reads a single byte from the register pointed at by "Index".
Returns the response
*/
byte ReadRegister(byte Index)
{
byte i = 0;
byte checksum = 0;
byte response;

SerialTXBuffer[0] = 209;
SerialTXBuffer[1] = 1;
SerialTXBuffer[2] = 2;
SerialTXBuffer[3] = Index;
SerialTXBuffer[4] = 1;

for (i=0;i<6;i++)
  {
  if (i!=5)
    {
    mySerial.write(SerialTXBuffer[i]);
    checksum += SerialTXBuffer[i];
    }
  else
    mySerial.write(checksum);
  }
  for (i=0;i<4;i++)
    {
    do{
    }while(!mySerial.available());
    SerialRXBuffer[i] = mySerial.read();
    }

  response = SerialRXBuffer[2];
  return response;
}

void loop() // run over and over
{
  // Set motor1 to +50% and motor2 to -50% duty cycle
  WriteRegister(9,190);
  delay(10);
  WriteRegister(10,67);
  delay(10);

  // Set each servo out register to 1.25mS pulse width
  WriteRegister(11,125);
  delay(10);
  WriteRegister(12,125);
  delay(10);
  WriteRegister(13,125);
  delay(10);
  WriteRegister(14,125);
  delay(10);

  // Read all servo pulse input registers and display them  using the serial monitor
    Serial.print("Servo In1=");
    Serial.print(ReadRegister(5));
    delay(100);

    Serial.print(" Servo In2=");
    Serial.print(ReadRegister(6));
    delay(100);

    Serial.print(" Servo In3=");
    Serial.print(ReadRegister(7));
    delay(100);

    Serial.print(" Servo In4=");
    Serial.println(ReadRegister(8));
    delay(100);
}
```

Communication Protocol for PIC16F1829

Hardware Usage:

The asynchronous serial communication takes place at 9600BPS using eight data bits, no parity, and one stop bit (9600, n, 8, 1). Data is sent LSB first. There is no hardware flow control. Serial data is received at the DIN pin. Data is sent out on the DOUT pin.

Communication Sequence Overview:

The serial protocol is a master/slave interface. This module is the slave device and unless otherwise noted does not initiate communication.

There are four types of communication packets. They are Read Packets, Write Packet, ACK Packets, and Reply Packets. The master always initiates a communication sequence with a command and a slave always terminates the communications sequence with a reply or ACK.

Packet Byte	Representation
Operation	Read = 209, Write = 210
Address	default is 1
Length	3 to 30
Packet Data	Message specific
Checksum	sum of all characters in packet proceeding the checksum

Figure 5.2: Packet information

Operation: There are two operations: read or write. A read operation allows the master to specify which registers to read from the module. A write operation allows the master to write to specific registers.

Address: The default device address is 1. This value may be changed using a write operation.

Length: This byte tells the receiver how many bytes are *following* the length byte. This includes the checksum.

Packet Data: The specific data that is necessary for a complete message.

Checksum: A simple sum of *all* of the bytes in the packet before the checksum. For example, if the packet data is 209, 1, 3, 0, 1, the checksum would be 214.

Since the checksum is a single byte value only the remainder is sent for values greater than 255. If the sum of a packet's bytes is 257, the checksum would be 1 (257/256 = 1 with a remainder of 1). If the sum is 512 the checksum is 0 (512/256 = 2 with a remainder of 0).

In most cases when you are summing values in a byte sized variable any overflow is lost, and the remainder is all that is left. This is the checksum.

Read Operation
A read operation allows the master to read specific registers from the module.

Operation Byte	Device Address	Length	Start Index	Number of Registers	Checksum

Figure: Form of a read command

The "Start Index" is the register index value that the read should begin from. For example, if you want to start your read at register 4, then the Start Index is 4. The "Number of Registers" is number of registers to read, including the register identified by the Start Index. The example below shows a read command of register 4. A reply packet will be returned when the read operation is accepted.

Master Sends: 209, 1, 3, 4, 1, 218
Slave returns: reply packet

Write Operation
A write operation allows the master to write specific registers to the module.

Operation Byte	Device Address	Length	Start Index	Data1	Data N	Checksum

Figure: Form of a write command

The "Start Index" identifies the first register that will be written to. The "Data1" register is the data to write to the Start Index register. With the Write command it is possible to write to more than one register at a time. The data will be written to sequential register starting at the Start Index. In the figure above the "Data N" byte will be written to the "Start Index" +1 register. The example below shows a write of 150 to register 11 and 151 to register 12.

Master Sends: 210, 1, 4, 11, 150, 151, 15
Slave returns: ack

You'll notice that the sum of the bytes in the packet is 527. Since the checksum is a byte sized value only the overflow is sent (527 = hexadecimal 0x20F, therefore the lower byte is sent, 0x0F, or decimal 15.

Reply Packet
The module sends Reply Packets to the Master in response to a Read Operation. Each Reply Packet will begin with the slave address (default to "1"). Next is a length byte, followed by the message data. The last byte in the packet is the checksum.

Device Address	Length	Data1	DataN	Checksum

Figure: Reply packet representation

The "Data" in the reply packet is the register data that the master requested using a read command.

ACK Packet
The module always sends an ACK packet to the Master in response to a Write Operation. Each ACK Packet is a single byte.

ACK
6

Figure: ACK packet representation

The module sends the ACK packet *after* the requested write is completed. If the master does not receive an ACK after an appropriate period of time, it may assume that the write operation did not work.

Error Detection:
Error detection is accomplished by inspection of the received data and making sure that the data was received in a timely and appropriate fashion. Inspection of the data packets will be performed by…

- verifying that all elements of the packet are present
- making sure that the message is the correct length
- verifying the checksum
- verifying that the message is supported by the Slave
- testing all values with limited range
- inter-character time

The inter-character time is the time between successive characters (bytes) in the same packet. The maximum time allowed is 2ms.

Serial Communication Examples:
Example 1 - Setting Motor1 (register 9) to full speed reverse (1) and Motor2 (register 10) to full speed forward (254).

Master Sends: 210, 1, 4, 9, 1, 254, 223
Slave returns: 6

Example 2 - Setting Servo_Out1, Servo_Out2 (registers 11, 12) to 1mS (100), and Servo_Out3, Servo_Out4 (registers 13, 14) to 1.5mS (150).

Master Sends: 210, 1, 6, 11, 100, 100, 150, 150, 216
Slave returns: 6

Example 3 – Reading Servo_In1 (register 5) through Servo_In4 whose contents are 1mS, 2mS, 1.25mS and 1.75mS respectively.

Master Sends: 209, 1, 3, 5, 4, 222
Slave returns: 1, 5, 100, 200, 125, 175, 94

Register Definitions:

This module is controlled by reading from and writing to internal registers via the previously described serial interface. The registers are defined below.

Index	Register	Range	Default Value	Description
0	Firmware	1-255	NA	Firmware revision loaded into the module.
1	Address	0-255	1	The serial address this unit will respond to.
2	Command	0-255	0	Commands specific to the module may be implemented by writing to this register. See the detailed register definition.
3	Mode	0-255	0	Operating modes specific to the module may be implemented by writing to this register. See the detailed register definition.
4	Indicator	0-255	0	Indicators specific to this module may be read from this register. See the detailed register definition.
5	Servo_In1	45-255*	NA	The length of the pulse read at the associated pin in 10uS increments. The module will read pulses from 450-255uS. * Pulses outside the defined range result in an error code of 0 being loaded into this register.
6	Servo_In2	45-255*	NA	see above
7	Servo_In3	45-255*	NA	see above
8	Servo_In4	45-255*	NA	see above
9	Motor1	1-254	127	Writing to this register controls motor speed and direction. 127 is stopped or 0% duty cycle. 1 is full speed reverse or -99%, full forward is 254 or 99%. Speed ramping can be achieved by incrementally increasing or decreasing the value you are sending.
10	Motor2	1-254	127	see above
11	Servo_Out1	50-250	150	Controls the output pulse width in 10uS intervals. Most servos will accept values from 0.5-2.5mS. The pulse outputs are generated in sequence, Servo_Out1 then Servo_Out2, etc. They are generated every 2mS* Servo_Period.
12	Servo_Out2	50-250	150	see above
13	Servo_Out3	50-250	150	see above
14	Servo_Out4	50-250	150	see above

Register Definitions (continued):

Index	Register	Range	Default Value	Description
15	Servo_ Pace1	1-255	10	You can slow down the rate of change of the associated Servo_Out1 signal by modifying this value. This value sets the allowed change in output per Servo_Period. The default of value 10 allows you to change output pulse width 100uS every servo period. So changing the Servo_Out1 register from 100 to 200 would allow the pulse width to change from 1mS to 2mS in 240mS (Servo_Out1 change / Servo_Pace1 x Servo_Period). Setting the Servo_Pace1 value to 1 would allow you to change the output pulse from 1mS to 2mS in 2.4S ((2mS-1mS)/0.01mS x 24mS).
16	Servo_ Pace2	1-255	10	see above
17	Servo_ Pace3	1-255	10	see above
18	Servo_ Pace4	1-255	10	see above
19	Servo_ Period	6-255	12	This register establishes the period of time between the start of each output pulse. It is in 2mS intervals. Setting Servo_Period to 10 has each servo output pulse occur every 20mS. Some servos motors behave poorly if this value is shorted or extended significantly. Most servos expect a signal between 20-25mS. However, you can extend the period somewhat if you want to slow down the movement of you servos beyond what the Servo_Pacex registers allow.
20	Servo_ Adjust	0-255	254	Used to fine tune servo output pulse width. 254 subtracts 20uS from the pulse width commanded which typically accounts for interrupt and other programmatic delays.
21	Unused1	0-255	0	
22	Unused2	0-255	0	
23	Unused3	0-255	0	
24	Programmed	0-255	0	This register is used to determine if EEPROM is blank (first time power up)

Detailed Register Descriptions:

Command Register (index = 2): Writing values to the Command register implements specific commands. Once these commands are executed the register value is returned to 0.

Command	Value	Description
Restore	1	Restores internal registers to their default settings and stores those values in EEPROM so they become the power on default values.
Store	2	Stores the current register values in EEPROM so they become the power on default values.
Reset Controller	3	Forces the module's microcontroller into a watchdog timer reset.
Reset H-Bridge	4	Briefly disables and re-enables the motor drive circuitry to clear any fault conditions. This command also clears the H-bride fault status flag (bit 0 of the Indicator register).

Mode Register (index = 3): Setting specific bits in the Mode register causes the module to implement special operating modes. A bit is set by writing a "1" to it. Multiple special modes can be implemented at the same time, although not all modes work together.

Mode	Bit	Description
Disable Auto Reset	0	Setting this bit prevents the controller from automatically re-enabling the H-bridges if they shut down due to a fault condition. With this bit set the H-bridges will have to be reset using the Reset H-Bridge command (see Command register).
Servo_In3 Controls Motor 1	1	Setting this bit causes the controller to use the servo input signal at Servo_In3 as the drive signal for Motor1. Servo signals from 1-2mS will be used with 1mS equal to full speed reverse, 1.5mS stopped, and 2mS = full speed forward.
Servo_In4 Controls Motor 2	2	as above but Servo_In4 controls Motor2
Mix Servos for Motor Control	3	Servo signal inputs 3 and 4 are mixed to create drive signals for both motors. This allows you to create a skid steering robot with Servo_In3 controlling speed (forward and reverse) and Servo_In4 controlling direction (left and right).
Disable Servo Inputs	4	Setting this disables the code that reads pulses at the Servo_Inx pins. This allows you control the contents of the Servo_Inx registers through the serial interface and may be useful in testing special operating modes (above).

Indicator Register (index = 4): The Status register contains bit values that indicate important conditions in the module.

Indicator	Bit	Description
H-Bridge Fault	0	Set when one or both H-bridges have experienced a fault condition. This bit can only be cleared by resetting the H-bridges via the Command register (see Reset H-bridge command). This bit being set may not indicate a fault condition is present. Under normal operation the module will attempt to reset the H-bridges if a fault occurs.
	1	unused
	2	unused
	3	unused
	4	unused
	5	unused
	6	unused
	7	unused