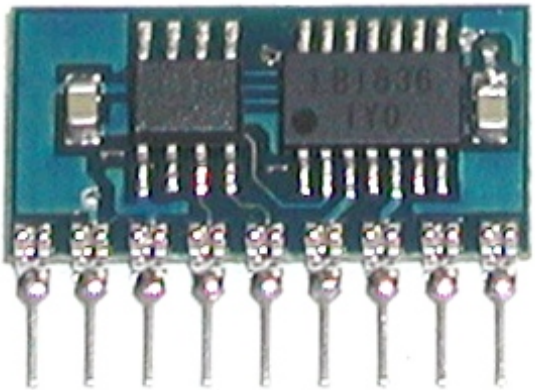
Micro Dual Serial Motor Controller

User's Guide



Contents:

http://www.pololu.com/

SMC02A

Important Safety Warning

The motor controller module is not intended for young children! Younger users should use this module only under adult supervision. **By using this product, you agree not to hold Pololu liable for any injury or damage related to the use or to the performance of this product. This product is not designed for, and should not be used in, applications where the malfunction of the product could cause injury or damage.**

Contacting Pololu

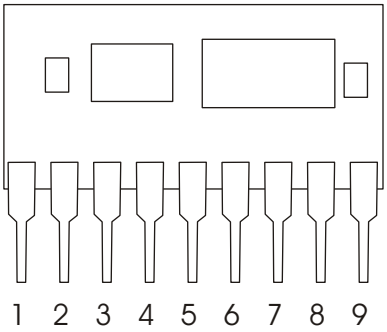
You can check the Pololu web site at `http://www.pololu.com/` for the latest information about the motor controller, including color pictures, application examples, and troubleshooting tips.

We would be delighted to hear from you about your project and about your experience with our motor controller. You can contact us through our online feedback form or by email at `support@pololu.com`. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

Module Pinout

The function of each of the nine module pins is listed in the table below. With the module components facing you and the pins facing down, the pins are numbered left to right.

PIN	FUNCTION
1	motor supply (1.8-9.0 V)
2	ground (0 V)
3	logic supply (3.0-5.5 V)
4	serial control input
5	reset
6	motor 1, positive output
7	motor 1, negative output
8	motor 0, negative output
9	motor 0, positive output



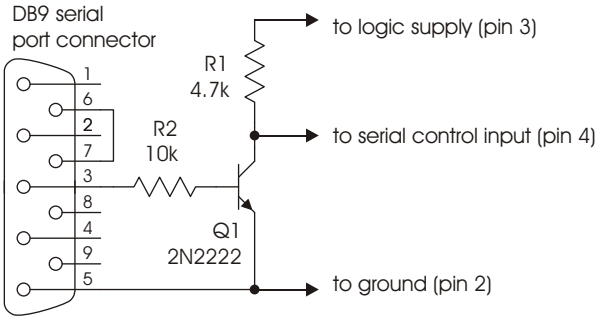
Connecting the Motor Controller

The motor controller module has pins with a standard 0.1" spacing. You can hook up your motor controller with a solderless breadboard, by soldering wires directly onto the pins, or even with your own custom printed circuit board. You can also use a socket to allow you to plug your motor controller in without directly soldering it.

Connecting Power (pins 1-3). Connect the ground pin (pin 2) to a ground terminal on your main controller unit, which might be a small microcontroller. Connect the positive terminal of your motor power source to pin 1. Connect your logic power supply to pin 3. The logic power supply is the voltage at which your main controller operates, such as 5 V. You can connect the same power source to both logic and motor supplies. For example, if you are using a microcontroller that can run at 4.5 V, you could run both your logic and motors off of 3 1.5-volt batteries. **Warning**: make sure the motor supply does not exceed 9 volts and the logic supply does not exceed 5.5 volts.

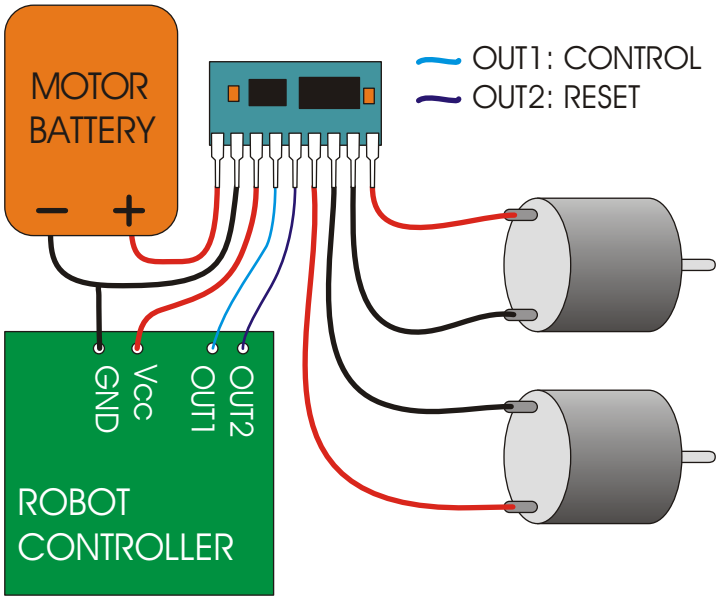
Reset Input (pin 5). The reset input is optional, but you may need to use it to ensure that spurious signals sent when your main controller turns on do not cause the motor controller to detect the baud rate incorrectly. Connect this pin to a digital output on your main controller. The line should normally be kept high (at the logic supply voltage), but bringing it low (to 0 V) for at least 2 microseconds resets the motor controller to its initial state (all motors off, waiting for its first serial command).

Serial Input (pin 4). Use a pin on your main controller that can be used as a logic-level, asynchronous serial output. Serial data can be sent down this line 8 bits at a time, with no parity bit, at any rate between 1200 and 19200 baud. *Once you choose a baud rate, you cannot change it until the motor controller is reset.* **Important note:** unlike RS-232 serial lines (the standard for serial ports used to connect devices to personal computers), this line uses logic voltages between 0 and the supply voltage. The higher voltages used on RS-232 lines will damage the motor controller. If you need to convert RS-232 levels to TTL levels, you will need to use a level converter such as the MAX220 (made by Maxim). You could also use the simple circuit shown below. **When building circuits that connect to a PC, be especially careful because you could potentially destroy the PC's serial port. Before attempting to connect your own electronics to a computer, make sure you know what you are doing!**



Connecting the Motors (pins 6-9). Connect one or two motors to the pins 6 through 9. You probably don't need to worry too much about the polarity, but the pins 6 and 9 go positive when the controller receives "forward" commands. If you find out that your motors turn in different directions than you expect, you can flip the wiring or just switch the forward and reverse commands on your robot controller program.

A typical setup is shown in the diagram below. The green box labeled "robot controller" represents a main control unit that includes a battery that is not shown. This robot controller could be a microcontroller or a device such as the BASIC Stamp from Parallax. Keep in mind that the wiring you use for the motor outputs and power connections should be capable of conducting several amps. We recommend using at least 26 gauge wire (remember, smaller numbers mean bigger wires!).

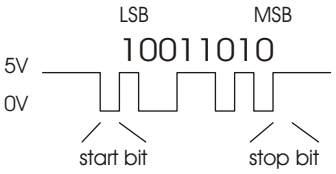


Using the Motor Controller

To set the speed and direction of a motor, send a four-byte command with the following structure to the motor controller's asynchronous serial input, pin 4.

start byte = 0x80	device type = 0x00	motor # and direction	motor speed

You must send the four-byte command eight bits at a time (with no parity bit) at a constant baud rate ranging from 1200 to 19200 baud. The serial bits must be *non-inverted*, meaning that a zero is sent as a low voltage, and a one is sent as a high voltage, as shown in the diagram to the right. (The PC-connection circuit on the previous page corrects the inverted signal coming out of PC serial ports.) *Commands sent to the serial input* ***must*** *conform to the above format (described in detail below) or else the motor controller and other devices connected to the serial line may behave unexpectedly.* This motor controller *interface protocol* is compatible with other Pololu serial devices such as our servo controller, so you can control multiple Pololu serial devices on a single line.

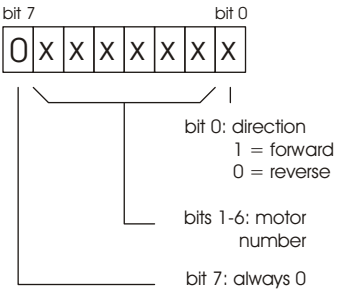


The Four-Byte Motor Controller Command

Byte 1: Start Byte. This byte should *always* be 0x80 (128 in decimal) to signify the beginning of a command. The start byte is the only byte with the highest bit (bit 7) set, and it alerts all devices on the serial line that a new command is being issued. All succeeding bytes sent down the serial line must have their highest bit cleared to zero.

Byte 2: Device Type. This byte identifies the device type for which the command is intended, and it should be 0x00 for commands sent to motor controllers. All devices that are not dual motor controllers ignore all subsequent bytes until another start byte is sent.

Byte 3: Motor Number and Direction. This byte has three parts, as shown in the diagram to the right:



Bit 0 specifies the direction of the motor. Set this bit to 1 to make the motor go forward; clear the bit to make it go backward.

Bits 1-6 specify the motor number. If you are using only two motors per serial line, you can use the default values of 0 for motor M1 and 1 for motor M2. If you want to control more than two motors, use numbers in the range of 2 to 63, as described in the section, "Controlling Multiple Motor Controllers with One Serial Line".

Byte 3: Motor Number and Direction (continued).

Bit 7 must be cleared since this is not a start byte.

To obtain the complete byte 3 value from a motor number and a direction, multiply the motor number by 2 and add 1 if the direction is forward. For example, to make motor 5 go forward, byte three should be 5 x 2 + 1 = 11. To make motor 1 go backward, byte 3 should be 1 x 2 = 2. (Two efficient ways to multiply by 2 in a microcontroller program are shifting left by one digit or adding the motor number to itself.)

Byte 4: Motor Speed. The most significant bit must be zero since this is not a start byte. The remaining seven bits specify the motor speed. The possible range of values for byte 4 is thus 0x00 to 0x7F (0 to 127 decimal). 0x00 turns the motor off, and 0x7F turns the motor fully on; intermediate values correspond to intermediate speeds.

Resetting the Motor Controller

The motor controller's optional reset line should normally be kept high at the logic supply voltage. Pull the reset line low to 0 V for at least 2 microseconds to reset the motor controller to its initial state (all motors off, waiting for the first serial command). You do not need to reset the motor controller to use it successfully. However, you may need to reset the motor controller to ensure that spurious signals sent when your robot controller turns on do not cause the motor controller to detect the baud rate incorrectly.

Controlling Multiple Motor Controllers with One Serial Line

To control a particular motor, you must specify its motor number in command byte 3. For all motor controller boards, motor 0 (pins 8 and 9) responds to commands for motor number 0, and motor 1 (pins 6 and 7) responds to commands for motor number 1. To control more than two motors with a single serial line, you need to use motor numbers 2 through 63. The back of your motor controller module has a label that ends in '-x', where x specifies the motor numbers to which your module will respond. When the label ends in '-1', the motor controller responds to numbers 2 and 3, when the label ends in '-2', the motor controller responds to numbers 4 and 5, and so on. Motor controllers that are not specially ordered typically respond to numbers 2 and 3; you need to order specially programmed motor controllers to use motor numbers 4 through 63.

For example, to control six motors independently, you need three motor controller boards, each with different motor numbers. All three motor controllers respond to commands for motor numbers 0 and 1. For controlling the six motors independently, use motor numbers 2, 3, 4, 5, 6, and 7. (The exact numbers depend on which motor numbers you request when you specially order additional motor controllers.)

You can individually control up to 62 motors at a time with a single serial line using 31 motor controllers: one with the default program and 30 that are specially programmed.

Example BASIC Stamp II Program

This program, which can run on a BASIC Stamp II controller, makes motor 1 gradually speed up, then slow down, then speed up in the other direction, and then slow down again. For the code to work, pin 15 must be connected to the reset input (pin 5), and pin 14 must be connected to the serial input (pin 2). The interface code should look similar in other programming languages; the description below should help you in understanding the code and, if necessary, in translating it to other languages.

On line 1, the 8-bit variable `speed` is declared for later use. The motor controller is then reset by a low-going pulse on pin 15 (lines 2 and 3).

The first *for loop* on lines 4-7 causes motor 1 to gradually speed up. The serial output is created by the `serout` statement on line 5. The first parameter, 14, specifies the pin number through which to send the serial signal. The next parameter, `32`, sets up the serial characteristics to be 8 bits with no parity, non-inverted, at a baud rate of 19200. The four numbers in square brackets are the data to be sent, and they correspond to the four control bytes for the motor controller. The first two bytes should always be `$80` and `0`. The second `0` makes motor 1 go backward. The speed variable, which increases every time through the loop, is the only part of the command that changes, and that is what makes the motor gradually speed up. The `pause` statement on line 6 causes the program to wait for 20 ms (0.02 seconds) before sending the next command.

When the first loop ends, the motor is set to its full speed of 127. The second loop on lines 8-11 slows the motor back down by sending speeds from 127 down to 0. The next two loops on lines12-19 then repeat the process, except for the parameter value of `1` in byte three, which causes motor 1 to spin forward.

```
1        speed   var   byte
2        low 15         'reset motor controller
3        high 15
4        for speed = 0 to 127
5           serout 14,32,[$80, 0, 0,speed]
6           pause 20
7        next
8        for speed = 127 to 0
9           serout 14,32,[$80, 0, 0,speed]
10          pause 20
11       next
12       for speed = 0 to 127
13          serout 14,32,[$80, 0, 1,speed]
14          pause 20
15       next
16       for speed = 127 to 0
17          serout 14,32,[$80, 0, 1,speed]
18          pause 20
19       next
```

The Pololu Micro Dual Serial Motor Controller

For a robot to interact with its environment, it must be able to convert electrical signals into motion. However, the power requirements of *actuators*, electrical devices capable of producing motion, are typically so high that normal digital circuitry cannot drive them. In addition, precise motion control requires constantly changing the signals sent to the actuators, leaving the control circuitry with little time to attend to other tasks.

The Pololu motor controller bridges the gap between robot controllers and power-hungry actuators. Using one serial output from your robot controller, you can independently set each of two small DC motors (the kind typically found in remote-control cars and motorized toys) to go forward or backward at any of 127 different speeds. To control additional motors, you can connect multiple motor controllers to the same serial line. The motor controller is compatible with the Pololu Servo Controller, so you can control an almost unlimited number of motors and servos with one serial line.

Because of its small size, the motor controller is especially suited for small robots and mechanisms that use small motors that run at low voltages. With this motor controller, you can make a robot that runs off of just two 1.5 V batteries or three 1.2 V rechargeable cells!

Specifications

PCB size	0.90" x 0.45"
Motor ports	2
Speeds	127 forward, 127 backward, and off
Maximum current	1 A per motor (continuous)
Motor supply voltage	1.8-9.0 V
Logic supply voltage	3.0-5.5 V
PWM frequency	600 Hz
Serial baud rate	1200-19200 (automatically detected)