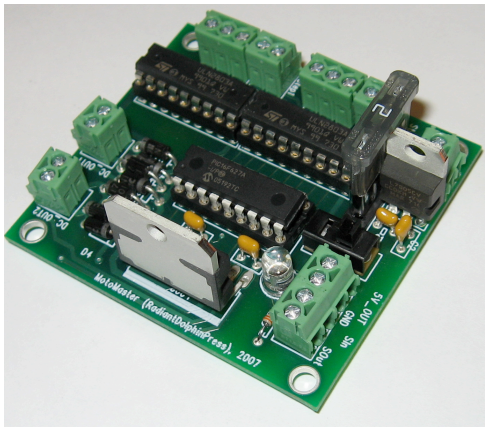
MotoMaster

Multi-motor serial driver
Developed at the University of California, San Diego
By Alex Simpkins

Radiant Dolphin Press Robotics
www.robotics.radiantdolphinpress.com

Document Rev 1.1, Feb. 14, 2008

Description:



The MotoMaster is a PWM-based serial motor driver capable of driving up to 10 motors at independent speeds. Two of the outputs can be bidirectional. In addition, MotoMaster can drive two unipolar stepper motors, each drawing up to 1A per coil, and one bipolar stepper motor drawing up to 2A per coil. It is controlled via a simple asynchronous serial interface at 50kbps (8-bit, non-inverted, no parity), which requires only a single digital IO pin from a microprocessor, or rs232 using a simple external circuit. The MotoMaster can also provide a time-base output (via serial out) for systems which do not have timer interrupts such as the Basic Stamp Microprocessor from Parallax.

These motor drivers are simple to use. Interfacing one with a basic stamp involves merely connecting two wires (ground and 1 I/O pin). The stamp controls this driver with simple asynchronous serial communication. One line of code!

Features:

- Minimally two wire serial communication – 50kbps, 8 bits, no parity, noninverted, one wire for serial input, one wire for ground (time-base output requires three wires – one for input, one for output, one for ground)
- All inputs and outputs have screw terminal blocks for easy connections
- All outputs are protected by a standard easy to replace fast-blow auto fuse
- Bipolar dual h-bridge for driving 2 DC motors at varying speeds and bipolar directions
- Dual darlington transistor arrays drive up to 8 DC motors at 1A current draw each, or outputs can be paralleled for higher current driving capability
- Darlington outputs can also serve to drive and control 2 unipolar stepper motors at 1A each, or 1 2A stepper motor

- Steppers can be automatically controlled for speed, direction, number of steps, continuous rotation, and individual coils for teaching stepper technology in instructional settings, or for the purpose of additional logic outputs
- An onboard 5V regulator can drive external circuitry up to 1A loads
- Convenient on-off switch disconnects logic power to the microprocessor and related logic circuitry, bringing the motors to a stop
- Possible motor configurations (less motors can also be driven – ie one DC or one stepper), note current ratings are 'up to' that maximum, more combinations are possible
    - 8 output lines for controlling external logic circuitry (Voltage needs to be buffered appropriately, 2 bidirectional DC 2A ea.
    - 8 unidirectional DC 1A ea, 2 bidirectional DC 2A ea.,
    - 4 unidirectional DC 2A ea, 2 bidirectional 2A ea.,
    - 1 unipolar 4-phase stepper 1A per coil, 4 unidirectional DC 1A ea., 2 bidirectional DC 2A ea,
    - 2 unipolar 4-phase stepper 1A per coil, 2 bidirectional DC 2A ea.
    - 2 unipolar 4-phase steppers 1A per coil, 1 bipolar 2-phase stepper 2A per coil
    - 1 unipolar 4-phase stepper 2A per coil, 2 bidirectional DC 2A ea.
    - 1 unipolar 4-phase stepper 2A per coil, 1 bipolar 2-phase stepper 2A per coil

Layout:

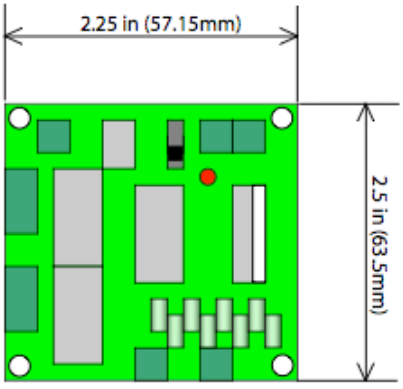


Figure 1: Basic dimensions for MotoMaster Rev.1.0

Typical ratings:

Rating	*Symbol*	*Min*	*Typ.*	*Max.*	*Units*	*Notes*

Output (continuous) from h-bridge outputs *(each output)*	V_{mh}	7	7.2	35	Volts (V)	
	i_{hb}	0A	--	+-2A	Amps (A)	
Input *(power source)*	V_{in}	7	10	35	Volts (V)	
	i_{in}	0	~2	12	Amps (A)	
Outputs (continuous) from Darlington *(see section on Darlington array)*	V_{da}	0	7	35	Volts (V)	
	i_{da}	0	--	1	Amps (A)	[1]
	i_{daTOT}	0	--	8	Amps (A)	[2]
Onboard voltage regulator	V_{reg}	4.95	5.0	5.05	Volts (V)	
	i_{reg}	0	--	1	Amps (A)	

[1] Per Pin
[2] All pins in parallel

Pinout:

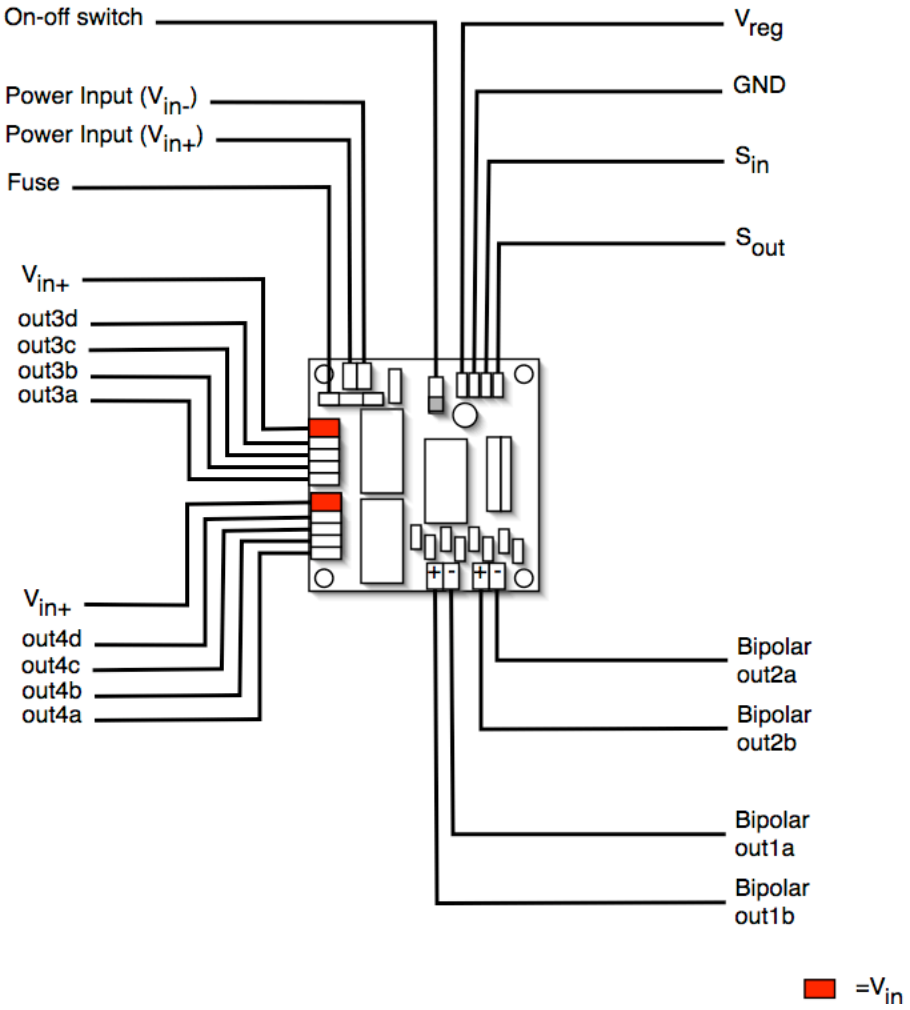


Figure 2: Pinout for MotoMaster Rev.1.0

Connection Diagrams:

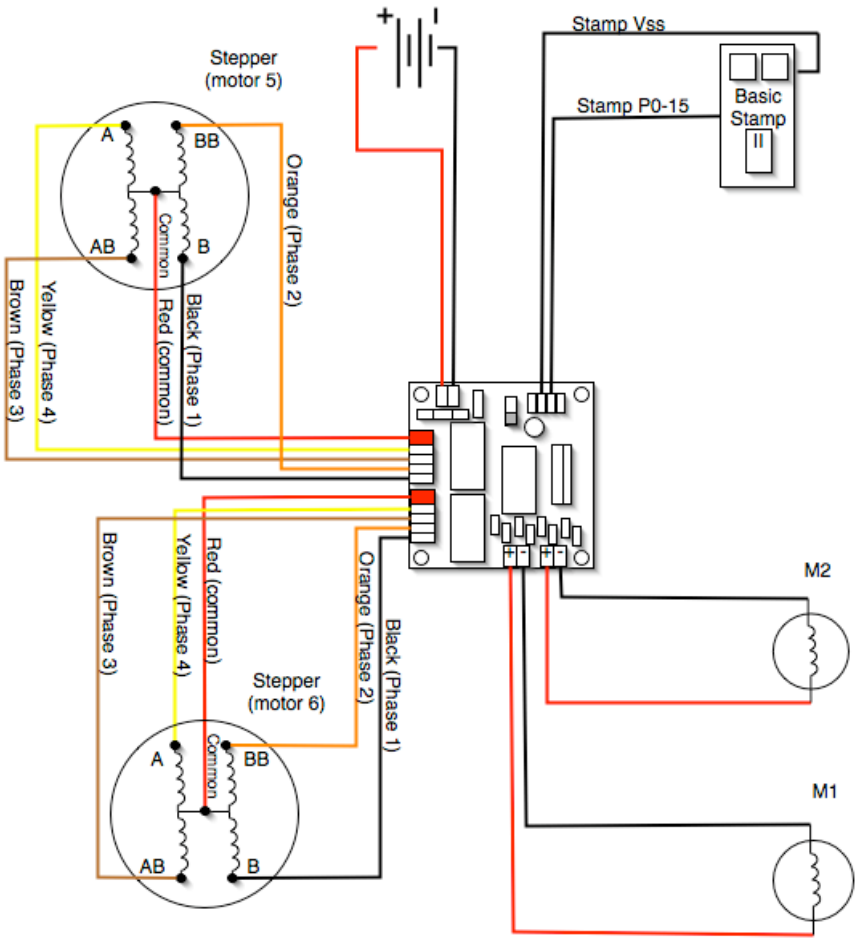


*Figure 3: Connection diagram for 2 dc motors and 2 unipolar stepper motors (color code for stepper shown is for the Mitsumi from parallax, see **Table 1** for conversion to the Howard industries stepper)*

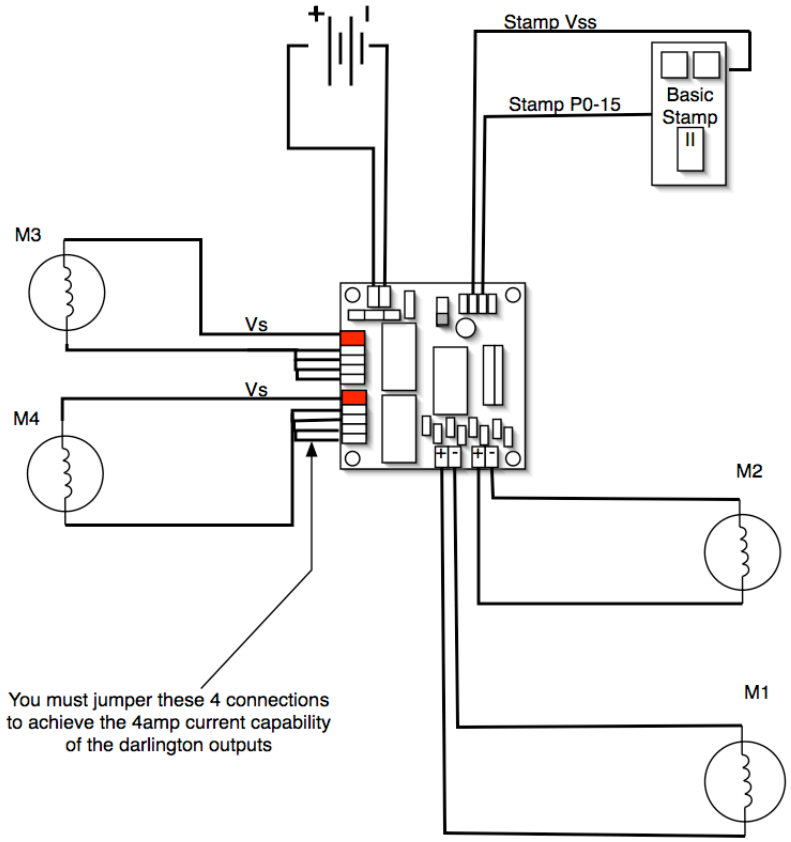


Figure 4: Connection diagram for 2 bidirectional dc motors (motor 1 and 2) and 2 unidirectional DC motor connections (motor 3 and 4)

Table 1: Connections for stepper motors

Manufacturer	**Mitsumi**	**Howard industries**
Common	Red	black
Outd	Yellow	White
Outc	Brown	Red
Outb	Orange	Green
Outa	Black	brown

Programming:

On a Basic Stamp II microcontroller:

2 DC Motors, bidirectional control (outs 1 and 2)

This is the easy part. The syntax for programming the motor driver is the following:

- **speed** is a number from 0 to 64, 64 being on full
- **motor #** is 1 or 2
- **direction** is "f" or "r" for forward or reverse, respectively
-

serout PIN#, 0, [MOTOR#, DIRECTION, SPEED]

Examples:

```
'for example, the following will turn on motor 1 full speed forward
direction:

serout 1, 0, [1,"f", 64]



'for example, the following will turn on both motors full speed forward
direction:

serout 1, 0, [1, "f", 64, 2, "f", 64]



'for example, the following will turn motor 1 reverse half speed:

serout 1, 0, [1, "r", 32]



'for example, the following will turn motor 1 off and motor 2 on almost
full speed forward

serout 1, 0, [1, "f", 0, 2, "f", 50]
```

Unipolar Stepper control examples and description, motor 5 and 6 (step 1 and 2):

Stepper motors are motor 5 and motor 6

you can control speed, direction, and number of steps up to 255 steps (each step is approximately 3.7 degrees). This frees your stamp to perform other activities during stepper motor control. It also serves to simplify your code.

COMMAND FORMAT:

The following are the command formats:

```
serout PIN#, 0, [5, SPEED AND DIRECTION, # OF STEPS]
```

with a forward command, the bit patterns output at the darlington are:

0011
0110
1100
1001

with a reverse command, the bit patterns are in opposite order:

1001
1100
0110
0011

'example:
'moves the stepper clockwise 10 steps at a moderately slow pace

```
serout 0, 0, [5, 50, 10]
```

SPEED AND DIRECTION:

speed and direction are set with one number. 0-128 is clockwise, 129-255 is counterclockwise.

Thus if you want to move the stepper counterclockwise very fast, you would use a number close to 255 (you may have to find the max speed that the stepper can rotate without missing steps, as the code is targeted at giving the best speed range possible, from slow to as fast as the motor is likely to be able to move without any load. If you

want
to rotate the motor counterclockwise, as slow as possible, you would use the number 129.

Here is a table of speeds and directions (you can use any number between, this is just a general guide to give you numerical intuition):

1 - clockwise, very slow
50 - clockwise, moderately slow
120 - clockwise, very fast

129 - counterclockwise, very slow
179 - counterclockwise, moderately slow
250 - counterclockwise, very fast

NUMBER OF STEPS:

The resolution of the current stepper motors used is 3.7 degrees per step. Thus if you wish the stepper to move 37 degrees you command 10 steps:

```
'serout 0, 0, [5, 50, 10]
```

NOTE:

The stepper's last pair of coils activated during the movement will remain energized until you deactivate the coils with the following command:

```
serout 0, 0, [5, 0, 0]
```

The coils remaining active is useful because the holding torque of the motor (the ability to resist torques) is a few orders of magnitude higher when energized than when not (confirm this by attempting to rotate the motor's shaft with your hand when the power is off and when the coils are energized, such as right after a rotation).

Individual control over each darlington output, grouped by 4 pins at a time:

```
'0 determines the serial settings, "i" signifies to the driver individual control, then
```

the next byte tells which of the motor outputs to control (3 or 4), while the final byte is the actual command. 0 is off, 1 is on for a particular pin.

```
serout pin#,0, ["i",3, %11000000]


'The byte order corresponds to the following:

'%out3a, out3b, out3c, out3d, don't care,don't care,don't care,don't care

'or the same for replacing 3 with 4...
```

Using darlington outputs to control a motor (motor 3 and motor 4):

```
'0 determines the serial settings, 3 or 4 is the motor number, 3 corresponds to the step 2 label, 4 corresponds to the step 1 label, 0-64 is the speed, and 0 is a dummy byte

serout pin#,0, [3,64, 0] 'turns on motor 3 full speed
```

Time-base output

There are two commands relevant for MotoMaster time-base. The first resets the current time measure, the second tells the MotoMaster to output the current time value at the Sout port. The resolution is 10msec, +-8e-6sec. The longest time you can measure without resetting the timer is 10.9 minutes approximately, calculated as follows (please note : the variable storing time is a 16 bit word):

$$T_{\max} = \left(2^{16}{}_{hundredths\_of\_a\_sec}\right)\left(\frac{1\text{sec}}{10_{hundredths\_of\_a\_sec}}\right)\left(\frac{1\text{min}}{60\,\text{sec}}\right) = 10.9\,\text{min}$$

IMPORTANT NOTES:

(1) One important point – YOU MUST IMMEDIATELY perform the serin command after sending the "t" byte command (serout pin, 0, ["t", 0, 0]). If you don't, the stamp won't read the time correctly or at all.

(2) If you are trying to move the stepper (motor 6) at the same time as reading the time, you may miss steps. A future version of the MotoMaster software will address this issue. You probably will not see this issue unless you are attempting to move the stepper (motor 6) at a high speed while reading time. When you are not trying to read time simultaneously with moving motor 6, there is no problem with missing steps

Getting the current time once:

```
'this code will get the time once
Time var word

'reset the MotoMaster timer:
serout pin#, 0, ["r", 0, 0]

'use the MotoMaster to output current relative time:
time var word ' the time value is stored in a 16-bit word
serout pin#, 0, ["t", 0, 0]
serin pin#, 0, [Time.highbyte, Time.lowbyte]

'if you want to display the variable from a basic stamp debug screen, use the
following line:
debug dec Time, cr
```

How to continuously update the time value in a loop:

```
Time var word

'reset the MotoMaster timer:
serout pin#, 0, ["r", 0, 0]

'use the MotoMaster to output current relative time:
time var word ' the time value is stored in a 16-bit word

timeloop:
   serout pin#, 0, ["t", 0, 0]
   serin pin#, 0, [Time.highbyte, Time.lowbyte]

  'if you want to display the variable from a basic stamp debug screen, use the
following line:
   debug dec Time, cr
goto timeloop
```

Application notes:

(1) Linearizing nonlinear effects of friction and other DC motor nonlinearities

Due to the physical characteristics of DC motors (friction and other nonlinearities) and diodes, even fast recovery diodes, PWM speed control of motors produces a slightly nonlinear response. This is an important thing to be aware of for proper design. One solution is to measure and feed back the speed of the motor using a differentiated position signal or a velocity sensor such as hall-effect-based sensor or tachometer. This is a model-reference

control design, and if you have the sensor feedback available, it will greatly linearize the response, attenuating effects of friction and other nonlinearities in the system.

(2) PWM, an introduction

What is PWM? PWM refers to pulse with modulation. It is the process by which an analog wave form can be created using a digital pulse train.

Control over speed is achieved by this method of PWM. This method is based on the fact that the motor, due to its inertia and LR (Inductance and resistance) characteristics, acts as a low pass filter. Thus, a repeating square wave input current of some frequency will be filtered to a DC current between the maximum voltage and zero volts. The proportion of ”on” time versus ”off” time of the PWM wave determines the average motor current. The period of one of the PWM wave cycles determines the smoothness of the DC average current, as a function also of motor characteristics (inductance and resistance).