Motor Shield

How to rev up

December 15, 2010 0:00

To do...

- Jan 2009 - New library with M3 and M4 speed swap fixed.Ugh! Also tested with up to IDE 13 so should be fewer problems
- Watch out! On some (much) older boards, Ground and VCC are swapped on the silkscreen next to the analog pins! The silkscreen should be "+5, GND, A0-5". Use a multimeter to check if you're confused

Powering your DC motors, voltage and current requirements

Motors need a lot of energy, especially cheap motors since they're less efficient. The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this shield are designed to run from **4.5V to 36V**. (However, in theory they should be OK down to about 2.5V?)

Current requirements: The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 600 mA per motor, with 1.2A peak current. Note that once you head towards 1A you'll probably want to put a heatsink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

On using the SN754410: Some people use the SN754410 motor driver chip because it is pin-compatible, has output diodes and can provide 1A per motor, 2A peak. After careful reading of the datasheet and discussion with TI tech support and power engineers it appears that **the output diodes were designed for ESD protection only** and that using them as kickback-protection is a hack and not guaranteed for performance. For that reason the kit does not come with the SN754410 and instead uses the L293D with integrated kickback-protection diodes. If you're willing to risk it, and need the extra currrent, feel free to buy SN754410's and replace the provided chips.

Need more power? Buy another set of L293D drivers and solder them right on top of the ones on the board (piggyback). Voila, double the current capability! You can solder 2 more chips on top before it probably isnt going to get you much benefit

You can't run motors off of a 9V battery so don't even waste your time/batteries! Use a big Lead Acid or NiMH battery pack. Its also very much suggested that you set up two power supplies (split supply) one for the Arduino and one for the motors. **99% of 'weird motor problems'** are due to noise on the power line from sharing power supplies and/or not having a powerful enough supply!

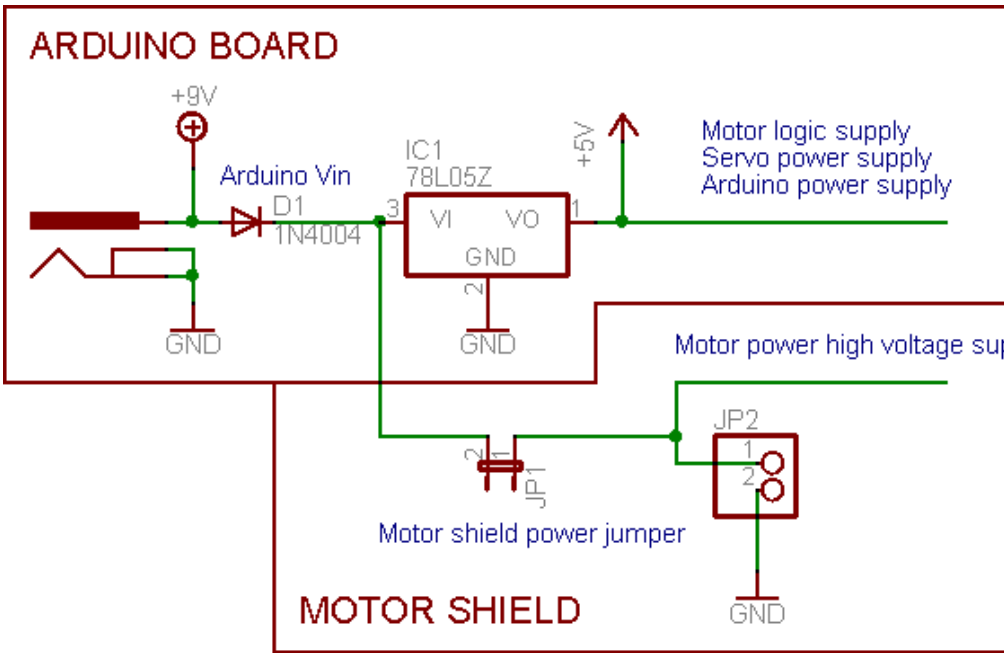
How to set up the Arduino + Shield for powering motors

Servos are powered off of the same regulated 5V that the Arduino uses. This is OK for the small hobby servos suggested. If you want something beefier, cut the trace going to + on the servo connectors and wire up your own 5-6V supply!

The DC motors are powered off of a 'high voltage supply' and NOT the regulated 5V. **Don't connect the motor power supply to the 5V line**. This is a very very very bad idea unless you are sure you know what you're doing!

There are two places you can get your motor 'high voltage supply' from. One is the DC jack on the Arduino board and the other is the 2-terminal block on the shield that is labeled **EXT_PWR**. The DC Jack on the Arduino has a protection diode so you won't be able to mess things up too bad if you plug in the wrong kind of power. However the **EXT_PWR terminals on the shield do not have a protection diode** (for a fairly good reason). **Be utterly careful not to plug it in backwards** or you will destroy the motor shield and/or your Arduino!

Here's how it works:



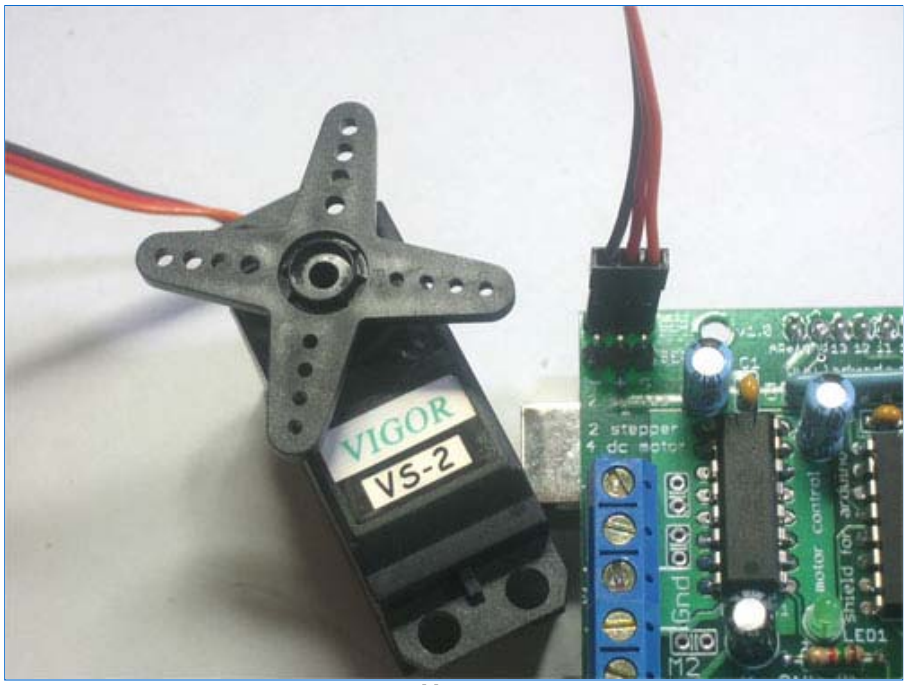
If you would like to have a **single DC power supply for the Arduino and motors**, simply plug it into the DC jack on the Arduino or the 2-pin PWR_EXT block on the shield. Place the power jumper on the motor shield.
If you have a Diecimila Arduino, set the Arduino power source jumper to EXT.
Note that you may have problems with Arduino resets if the battery supply is not able to provide constant power, and it is not a suggested way of powering your motor project

If you would like to have the **Arduino powered off of USB** and the **motors powered off of a DC power supply**, plug in the USB cable. Then connect the motor supply to the PWR_EXT block on the shield. Do not place the jumper on the shield. This is a suggested method of powering your motor project
(If you have a Diecimila Arduino, don't forget to set the Arduino power jumper to USB. If you have a Diecimila, you can alternately do the following: plug the DC power supply into the Arduino, and place the jumper on the motor shield.)

If you would like to have **2 seperate DC power supplies for the Arduino and motors**. Plug in the supply for the Arduino into the DC jack, and connect the motor supply to the PWR_EXT block. Make sure the jumper is removed from the motor shield.
If you have a Diecimila Arduino, set the Arduino jumper to EXT. This is a suggested method of powering your motor project

Either way, if you want to use the DC motor/Stepper system the motor shield LED should be lit indicating good motor power

Servos



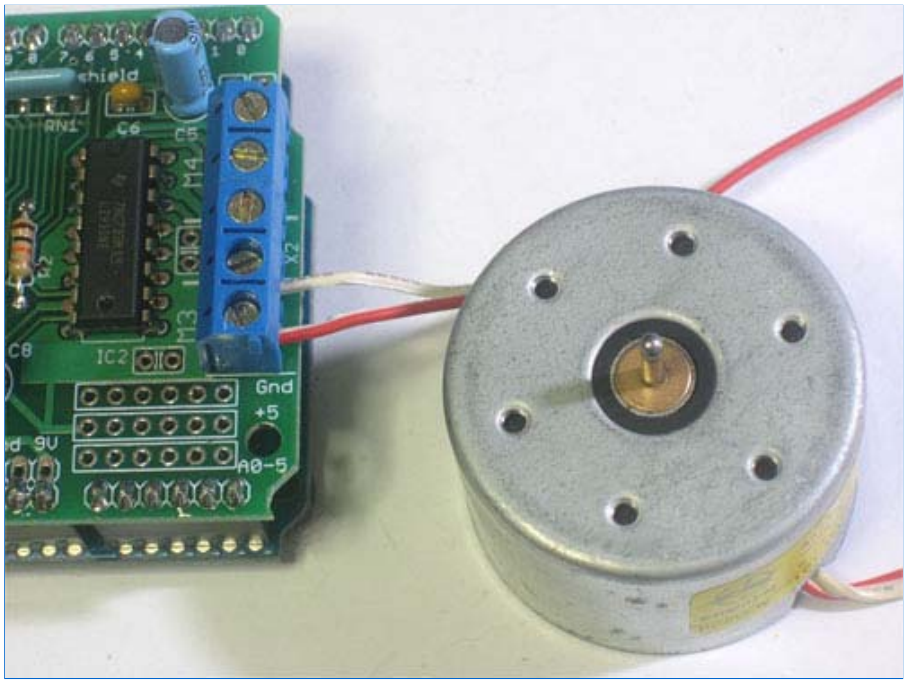
Hobby servo

Hobby servos are the easiest way to get going with motor control. They have a 3-pin 0.1" female header connection with +5V, ground and signal inputs. The motor shield simply brings out the 16bit PWM output lines to 2 3-pin headers so that its easy to plug in and go. They can take a lot of power so a 9V battery wont last more than a few minutes!

The nice thing about using the onboard PWM is that its very precise and goes about its business in the background. You can use the built in Servo library

Using the servos is easy, please read the official Arduino documentation for how to use them and see the example Servo sketches in the IDE

DC Motors



DC motor

DC motors are used for all sort of robotic projects. The motor shield can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed

can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

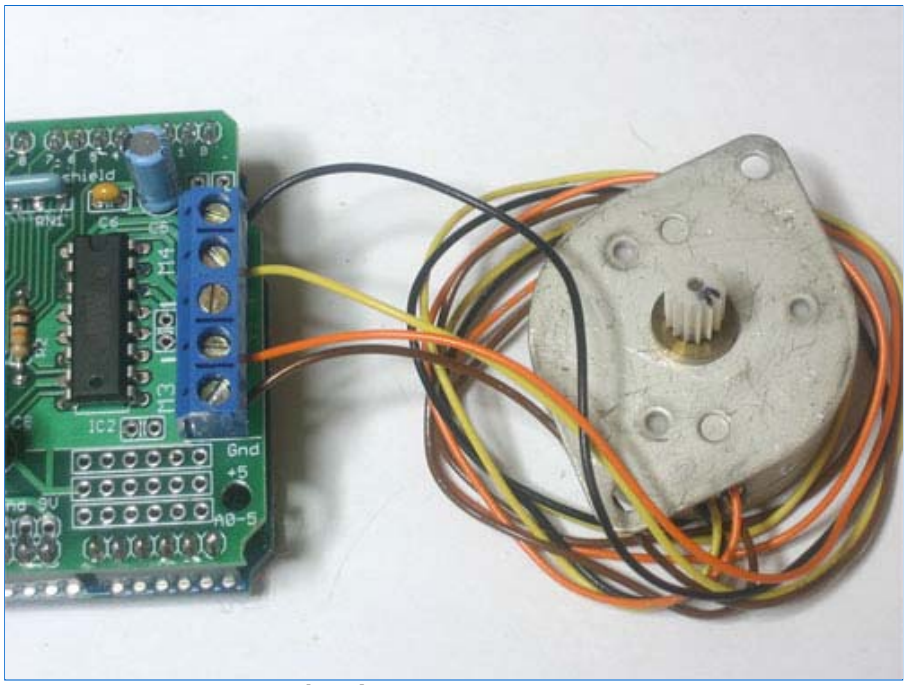
Note that the H-bridge chip is not really meant for driving loads over 0.6A or that peak over 1.2A so this is for *small* motors. Check the datasheet for information about the motor to verify its OK.

To connect a motor, simply solder two wires to the terminals and then connect them to either the **M1, M2, M3,** or **M4**. Then follow these steps in your sketch

1. Make sure you include **<AFMotor.h>**
2. Create the AF_DCMotor object with **AF_DCMotor(*motor#, frequency*)**, to setup the motor H-bridge and latches. The constructor takes two arguments.
   The first is which port the motor is connected to, **1, 2, 3** or **4**.
   frequency is how fast the speed controlling signal is.
   For motors 1 and 2 you can choose **MOTOR12_64KHZ**, **MOTOR12_8KHZ**, **MOTOR12_2KHZ**, or **MOTOR12_1KHZ**. A high speed like 64KHz wont be audible but a low speed like 1KHz will use less power. Motors 3 & 4 are only possible to run at 1KHz and will ignore any setting given
3. Then you can set the speed of the motor using **setSpeed(*speed*)** where the ***speed*** ranges from 0 (stopped) to 255 (full speed). You can set the speed whenever you want.
4. To run the motor, call **run(*direction*)** where ***direction*** is **FORWARD**, **BACKWARD** or **RELEASE**. Of course, the Arduino doesn't actually know if the motor is 'forward' or 'backward', so if you want to change which way it thinks is forward, simply swap the two wires from the motor to the shield.

```
#include <AFMotor.h>

AF_DCMotor motor(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Motor test!");

  motor.setSpeed(200);     // set the speed to 200/255
}

void loop() {
  Serial.print("tick");

  motor.run(FORWARD);      // turn it on going forward
  delay(1000);

  Serial.print("tock");
  motor.run(BACKWARD);     // the other way
  delay(1000);

  Serial.print("tack");
  motor.run(RELEASE);      // stopped
  delay(1000);
}
```

Steppers



A bi-polar stepper motor - 4 wires

Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This motor shield supports up to 2 stepper motors. The library works identically for bi-polar and uni-polar motors

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. Theres plenty of tutorials online on how to reverse engineer the coils pinout. The center taps should both be connected together to the GND terminal on the motor shield output block. then coil 1 should connect to one motor port (say M1 or M3) and coil 2 should connect to the other motor port (M2 or M4).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

Running a stepper is a little more intricate than running a DC motor but its still very easy

1. Make sure you include **<AFMotor.h>**
2. Create the stepper motor object with **AF_Stepper(*steps*, *stepper#*)** to setup the motor H-bridge and latches. ***Step****s* indicates how many steps per revolution the motor has. a 7.5degree/step motor has 360/7.5 = 48 steps. ***Stepper#*** is which port it is connected to. If you're using M1 and M2, its port 1. If you're using M3 and M4 its port 2
3. Set the speed of the motor using **setSpeed(*rpm*)** where ***rpm*** is how many revolutions per minute you want the stepper to turn.
4. Then every time you want the motor to move, call the **step(*#steps, direction, steptype*)** procedure. ***#steps*** is how many steps you'd like it to take**. *direction*** is either **FORWARD** or **BACKWARD** and the step type is **SINGLE, DOUBLE. INTERLEAVE** or **MICROSTEP**.
   "Single" means single-coil activation, "double" means 2 coils are activated at once (for higher torque) and "interleave" means that it alternates between single and double to get twice the resolution (but of course its half the speed). "Microstepping" is a method where the coils are PWM'd to create smooth motion between steps. Theres tons of information about the pros and cons of these different stepping methods in the resources page.
   You can use whichever stepping method you want, changing it "on the fly" to as you may want minimum power, more torque, or more precision.
5. By default, the motor will 'hold' the position after its done stepping. If you want to release all the coils, so that it can spin freely, call **release()**
6. The stepping commands are 'blocking' and will return once the steps have finished. If someone wants to be awesome and write a version of the library that does background stepping that would be cool! :)

```
#include <AFMotor.h>


AF_Stepper motor(48, 2);


void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Stepper test!");

  motor.setSpeed(10);  // 10 rpm

  motor.step(100, FORWARD, SINGLE);
  motor.release();
  delay(1000);
}

void loop() {
  motor.step(100, FORWARD, SINGLE);
  motor.step(100, BACKWARD, SINGLE);

  motor.step(100, FORWARD, DOUBLE);
  motor.step(100, BACKWARD, DOUBLE);

  motor.step(100, FORWARD, INTERLEAVE);
  motor.step(100, BACKWARD, INTERLEAVE);

  motor.step(100, FORWARD, MICROSTEP);
  motor.step(100, BACKWARD, MICROSTEP);
}
```

If you want two stepper motors to step at once you'll need to write something like this:

```
void doublestep (int steps, int direction, int style) {
  while (steps--) {
    motor1.step(1, direction, style);
    motor2.step(1, direction, style);
  }
}
```