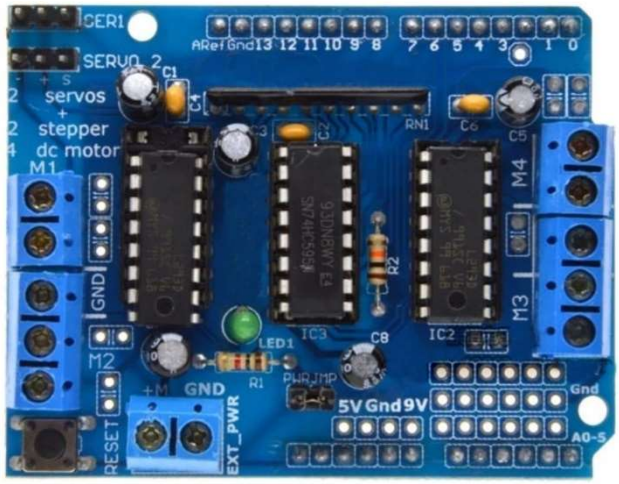
Motor shield L293D v1

Content

Info: [Adafruit motor shield](#) (library: Adafruit motor)



DC-motors

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 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, orMOTOR12_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.

The AF_DCMotor class provides speed and direction control for up to four DC motors when used with the Adafruit Motor Shield. To use this in a sketch you must first add the following line at the beginning of your sketch:

```
#include <AFMotor.h>
```

AF_DCMotor motorname (portnum, freq)

This is the constructor for a DC motor. Call this constructor once for each motor in your sketch. Each motor instance must have a different name as in the example below.

Parameters:

- port num - selects which channel (1-4) of the motor controller the motor will be connected to
- freq - selects the PWM frequency. If no frequency is specified, 1KHz is used by default.

  Frequencies for channel 1 & 2 are:

  - MOTOR12_64KHZ
  - MOTOR12_8KHZ
  - MOTOR12_2KHZ
  - MOTOR12_1KHZ

  Frequencies for channel 3 & 4 are:

  - MOTOR34_64KHZ
  - MOTOR34_8KHZ
  - MOTOR34_1KHZ

Example:

```
AF_DCMotor motor4(4);  // define motor on channel 4 with 1KHz default PWM
AF_DCMotor left_motor(1, MOTOR12_64KHZ);  // define motor on channel 1 with 64KHz PWM
```

Note: Higher frequencies will produce less audible hum in operation, but may result in lower torque with some motors.

setSpeed(speed)

Sets the speed of the motor.

Parameters:

- speed - Valid values for 'speed' are between 0 and 255 with 0 being off and 255 as full throttle.

Note: DC Motor response is not typically linear, and so the actual RPM will not necessarily be proportional to the programmed speed.

run(cmd)

Sets the run-mode of the motor.

Parameters:

cmd - the desired run mode for the motor. Valid values for cmd are:

- FORWARD - run forward (actual direction of rotation will depend on motor wiring)
- BACKWARD - run backwards (rotation will be in the opposite direction from FORWARD)
- RELEASE - Stop the motor. This removes power from the motor and is equivalent to setSpeed(0). The motor shield does not implement dynamic breaking, so the motor may take some time to spin down

Stepper motors

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: it's just like unipolar motors except there is 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 it's 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.

   Steps 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 it's 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.

Because the stepping commands 'block' - you have to instruct the Stepper motors each time you want them to move. If you want to have more of a 'background task' stepper control, check out AccelStepper library (install similarly to how you did with AFMotor) which has some examples for controlling two steppers simultaneously with varying accelleration

```
#include <AFMotor.h>

AF_Stepper motor(48, 2);

void setup() {
Serial.begin(9600);
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);
}
```

AF_Stepper steppername(steps, portnumber)

The AF_Stepper constructor defines a stepper motor. Call this once for each stepper motor in your sketch. Each stepper motor instance must have a unique name as in the example below.

Parameters:

- steps - declare the number of steps per revolution for your motor.
- num - declare how the motor will be wired to the shield.

Valid values for 'num' are 1 (channels 1 & 2) and 2 (channels 3 & 4).

Example:

```
AF_Stepper Stepper1(48, 1);  // A 48-step-per-revolution motor on channels 1 & 2
AF_Stepper Stepper2(200, 2);  // A 200-step-per-revolution motor on channels 3 & 4
```

step(steps, direction, style)

Step the motor.

Parameters:

- steps - the number of steps to turn
- direction - the direction of rotation (FORWARD or BACKWARD)
- style - the style of stepping:

Valid values for 'style' are:

- SINGLE - One coil is energized at a time.
- DOUBLE - Two coils are energized at a time for more torque.
- INTERLEAVE - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
- MICROSTEP - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

Note: Step is a synchronous command and will not return until all steps have completed. For concurrent motion of two motors, you must handle the step timing for both motors and use the "onestep()" function below.

```
Stepper1.step(100, FORWARD, DOUBLE);  // 100 steps forward using double coil stepping
Stepper2.step(100, BACKWARD, MICROSTEP);  // 100 steps backward using double
microstepping
```

setSpeed(RPMspeed)

set the speed of the motor

Parameters:

- Speed - the speed in RPM

Note: The resulting step speed is based on the 'steps' parameter in the constructor. If this does not match the number of steps for your motor, you actual speed will be off as well.

```
Stepper1.setSpeed(10);  // Set motor 1 speed to 10 rpm
Stepper2.setSpeed(30);  // Set motor 2 speed to 30 rpm
```

onestep(direction, stepstyle)

Single step the motor.

Parameters:

- direction - the direction of rotation (FORWARD or BACKWARD)
- stepstyle - the style of stepping:

Valid values for 'style' are:

- SINGLE - One coil is energized at a time.
- DOUBLE - Two coils are energized at a time for more torque.
- INTERLEAVE - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
- MICROSTEP - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

```
Stepper1.onestep(FORWARD, DOUBLE);  // take one step forward using double coil stepping
```

release()

Release the holding torque on the motor. This reduces heating and current demand, but the motor will not actively resist rotation.

```
Stepper1.release(); // stop rotation and turn off holding torque.
```