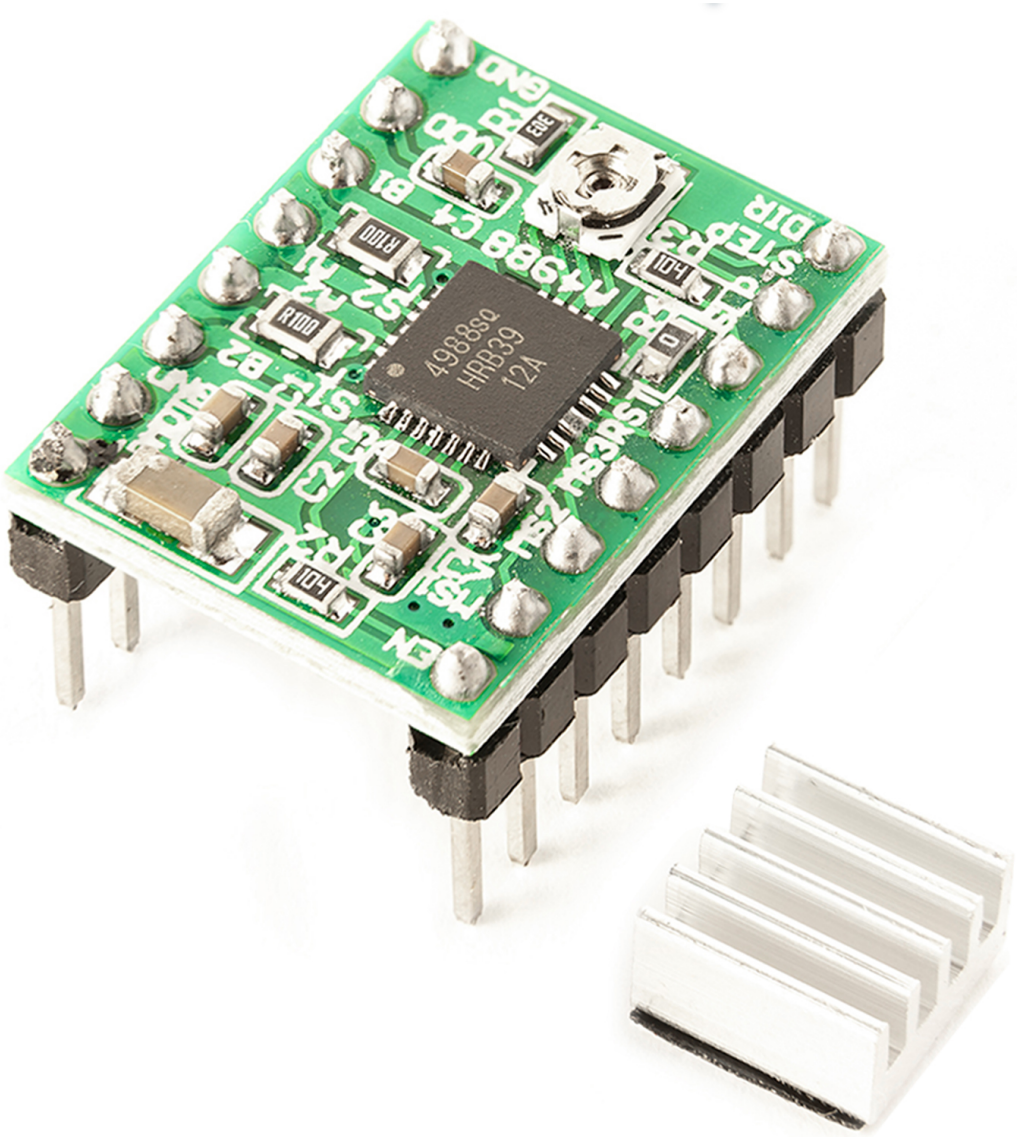
Welcome!

Thank you very much for purchasing our AZ-Delivery A4988 Stepper Motor Driver. On the following pages, we will introduce you to how to use and setup this handy device.

Have fun!



Stepper motor, or step motor is a type of motor where shaft of the motor rotates in steps. The stepper motor is a DC motor without brushes, namely a brushless DC motor. Moving the shaft in steps is very useful because the shaft can be positioned ~~very~~ precisely without any feedback position measurement.

All electromotors consist of rotor and stator. In stepper motors, rotor is generally a permanent magnet, which is surrounded by the coils of the stator. To rotate a rotor, we need to turn on or off coils of the stator in a certain order. When a current flows through coils of the stator, electromagnetic field is produced, which turns rotor or “permanent magnet” in the direction of electromagnetic field.

By construction there are three different types of stepper motors:

» permanent magnet stepper motors,
» variable reluctance stepper motors and
» hybrid synchronous stepper motors.

(we will not cover stepper motor constructions in this eBook)

Driving modes of stepper motors

To drive the stepper motor, there are several driving modes, or excitation modes:

» **Wave drive mode**, in this mode we activate just one coil of the stator at a time, then the next one, and so on. In this mode, only one coil is activated, and to move the rotor to the next step, we turn on coils one by one consecutively. When we turn on the second coil, first coil is turned off, etc.

» **Full step drive mode** provides much higher torque output because we always have two active coils at a given time. This mode is better explained on the image on the next page.

» **Half step drive mode** is used for increasing the resolution of the stepper motor. This mode arose from the combination of the previous two modes. Here we activate one coil, and near the end of active state of that coil, we activate next coil. When the second coil is activated we turn off first coil, and so on. With this mode we get double the resolution with the same motor construction.

» **Microstepping mode** is the most common method of controlling stepper motors nowadays. In this mode we provide variable current to the coils in the form of a sine waves. This will provide smooth motion of the rotor, decrease the stress of the parts and increase the accuracy of the stepper motor.

Another way of increasing the resolution of the stepper motor is by increasing the number of poles of the rotor and the number of poles of the stator.

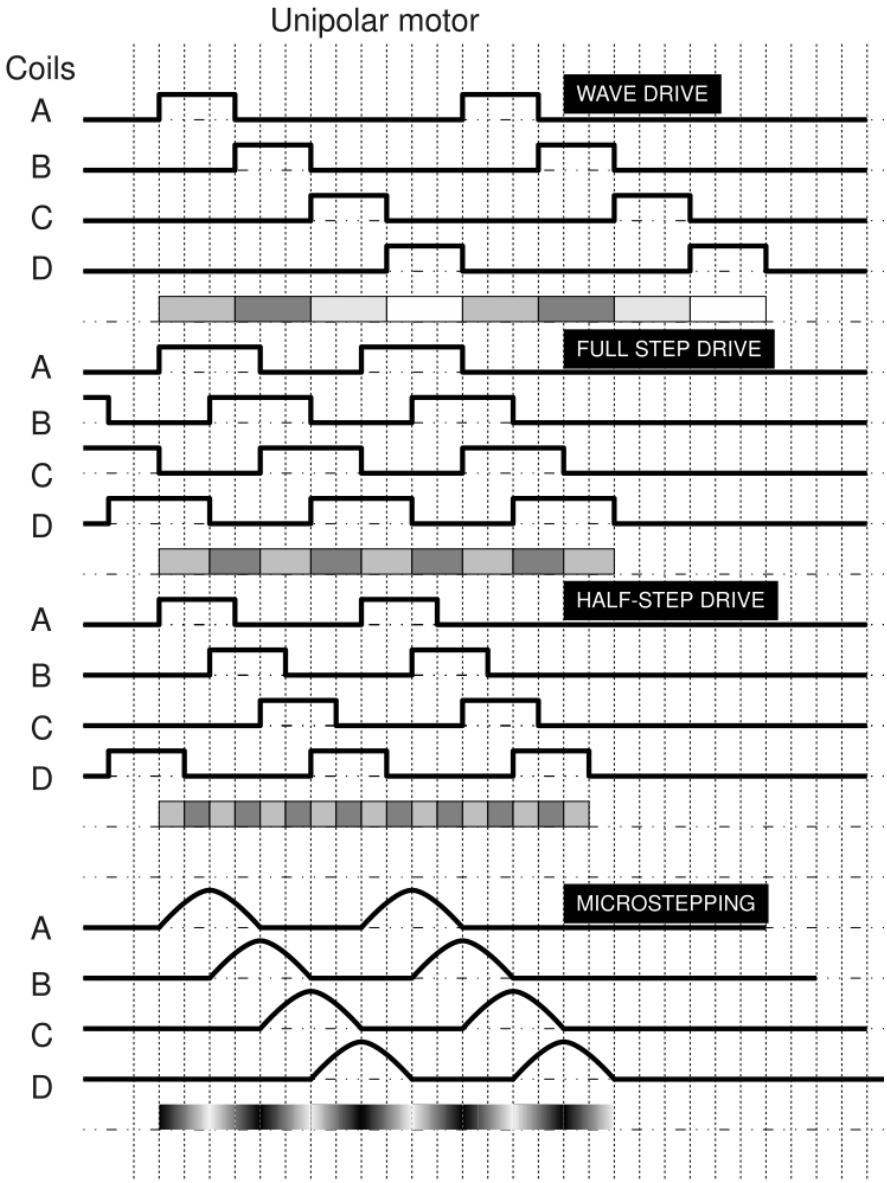


Image is taken from wikipedia article about stepper motors.

https://en.wikipedia.org/wiki/Stepper_motor

Stepper motor drivers

If you want to make some device where you will need to precisely control the rotation of the motor shaft, like in 3D printers or any other CNC machines, or robotic arms, etc., you will need a bunch of stepper motors and most importantly the stepper motor drivers.

To use microcontroller board to control a bunch of stepper motors is impractical and in most cases impossible. So we need a driver electronic circuit for every stepper motor.

In this eBook we will cover one of these devices, which is called "*A4988 stepper motor driver*". This device can control the rotational speed and spinning direction of motor shaft, and can drive stepper motor in several excitation modes. We can even use this device for microstepping excitation modes. With this stepper motor driver we can control only bipolar stepper motors.

The main chip on this device is an integrated circuit "*A4988*" manufactured by "*Allegro*". The driver has built-in translator for easy operation. This reduces the number of control pins to just two, one for controlling the steps and other for controlling rotational direction. The driver offers five different step resolutions, or excitation modes: full step, and four microstepping modes: half step, quarter step, eighth step, and sixteenth step.

Specifications

- » Min - max. logic voltage: 3V - 5.5V
- » Nominal current per phase: 1A
- » Maximal current per phase: 2A with passive cooling, alu heatsink
- » Min - max. motor output voltage: 8V - 35V

The A4988 actually requires two power supply connections. One for logic pins, and one for motor power supply:

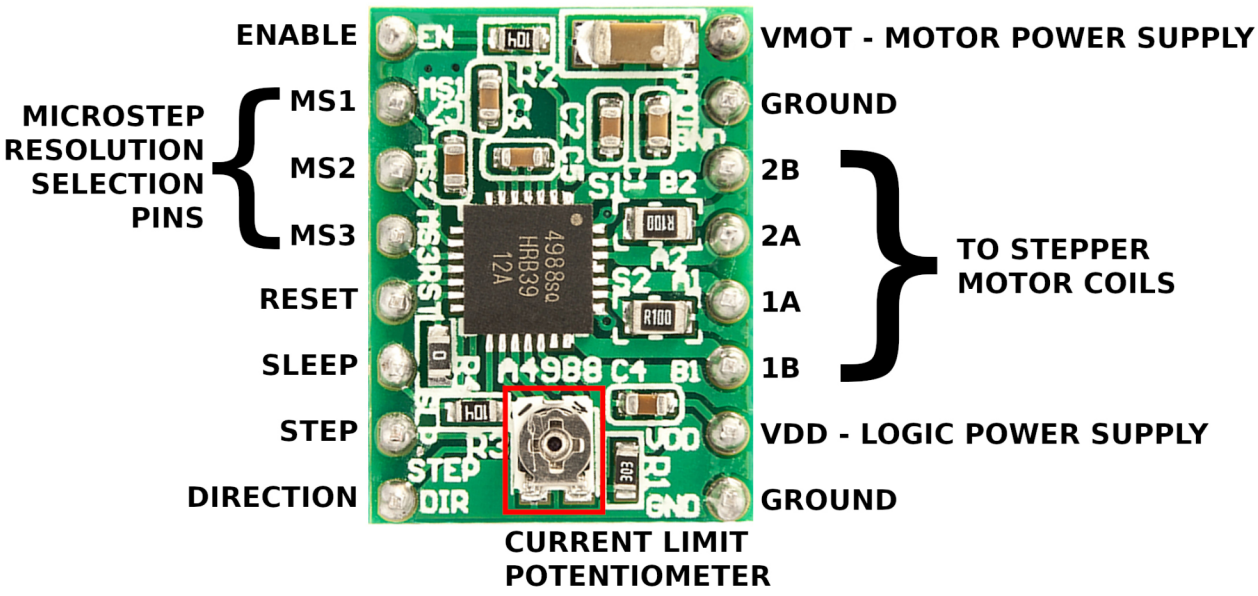
VDD and *GND* is used for driving the internal logic of the driver, (from 3V to 5.5V), which means that we can use either Atmega328P Board or Raspberry Pi as controling unit for this device.

VMOT and *GND* are used for powering the motor, from 8V to 35V. According to the datasheet, the motor supply requires an appropriate decoupling capacitor close to the board, capable of sustaining 4A current.

WARNING: This driver has low ESR ceramic capacitors on board, which makes it vulnerable to voltage spikes. In some cases, these spikes can exceed the 35V (maximum voltage rating of A4988), which can potentially permanently damage the board and/or the stepper motor.

One way to protect the driver from such spikes is to put a large 100µF electrolytic capacitor (or at least 47µF) across the power supply pins of the motor.

A4988 pinout



Microstep Selection Pins

The A4988 driver enables microstepping by allowing smaller step positions. This is achieved by energizing the coils with a variable current. For example, if you choose to drive a "*NEMA17*" stepper motor, it has a *1.8°* step angle or *200* steps per full revolution; in quarter step mode this motor will give *800* microsteps per full revolution.

The A4988 driver has three microstep resolution selection input pins:

» MS1

» MS2

» MS3

By setting appropriate logic levels to these pins we can set the drive mode of the motor to the one of these five modes:.

MS1	MS2	MS3	Microstep Resolution
LOW	LOW	LOW	Full step
HIGH	LOW	LOW	Half step
LOW	HIGH	LOW	Quarter step
HIGH	HIGH	LOW	Eighth step
HIGH	HIGH	HIGH	Sixteenth step

These three microstep selection pins are pulled *LOW* by internal pull-down resistors, so if we leave all of them disconnected, the motor will operate in full step mode.

Difference between microstepping and full step modes

Microstepping excitation modes are all modes where the shaft of the motor moves between the hardware steps. These modes position shaft of the motor in between the steps, creating more steps, and smooth motion of the shaft.

Half step excitation mode is a combination of full step and wave drive. This results in half of the basic step angle. This smaller step angle provides smoother operation due to the increased resolution of the angle. Half step produces about 15% less torque than the full step, however modified half stepping eliminates the torque decrease by increasing the current applied to the motor when a single coil is energized.

Microstepping can divide a motor's basic step ~~by~~ up to 256 times, making small steps smaller. A microstepping drive uses two variable current sine waves 90° apart, this is perfect for enabling smooth running of the motor. You will notice that the motor runs quietly and with no real detectable stepping action.

By controlling the direction and amplitude of the current flow in each coil of the stator, the resolution increases and the characteristics of the motor improve, giving less vibration and smoother operation. Because the sine waves work together there is a smooth transition from one coil to the other. When current increases in one coil, the current in the next coil decreases, which results in a smooth step progression and maintained torque output.

Step and direction pins

STEP input pin controls the steps of the motor. Each *HIGH* pulse sent to this pin, steps the motor by number of microsteps set by microstep selection pins. The faster the pulses, the faster the motor will rotate.

DIR input pin controls the rotation direction of the motor shaft. Pulling it *HIGH* drives the shaft clockwise and pulling it *LOW* drives the shaft counterclockwise. If you ~~just~~ want the shaft to rotate in a single direction, you can connect the *DIR* pin directly to *VCC* or *GND* accordingly.

NOTE

The *STEP* and *DIR* pins are not pulled to any particular voltage internally, so you should not leave them floating in your application.

Power enabling pins

EN pin, when pulled *LOW* the driver is enabled. By default this pin is pulled *LOW* so the driver is always enabled, unless you explicitly pull it *HIGH* to disable the driver.

SLP pin, pulling this pin *LOW* puts the driver in sleep mode, minimizing the power consumption. You can use this when the motor is not in use, to conserve power.

RST pin, when pulled *LOW*, all *STEP* pins are ignored, until you pull it *HIGH*. It also resets the driver by setting the internal translator to a predefined "*home*" state. "*Home*" state is the initial position from where the motor starts and it differs depending on the microstep resolution.

NOTE

If you are not using the *RST* pin, you can connect it to the *SLEEP* pin to bring it *HIGH* and enable the driver.

Output pins

Output pins of the driver are 1A, 1B, 2A and 2B. Each output pin can deliver currents up to 2A. However, the amount of current supplied to the motor depends on the power supply, cooling system and current limiting settings.

If you do not know the pinout of your bipolar stepper motor, you could test it by using the multimeter. Turn the switch on you multimeter to continuity test (position labeled with the sound sign). First, test if multimeter is working by connecting both nodes of the multimeter together. You should hear a beep if it works. The bipolar stepper motor has two coils and four wires, two wires per coil (one for input and one for output of the coil). A coil is just a wrapped wire, so we check the continuity between two wires of the motor. Simply connect with one node of multimeter to any wire of the motor, and connect second node to any other wire of the motor. If you hear a beep, those two wires are part of one coil, and the other two wires are part of the second coil.

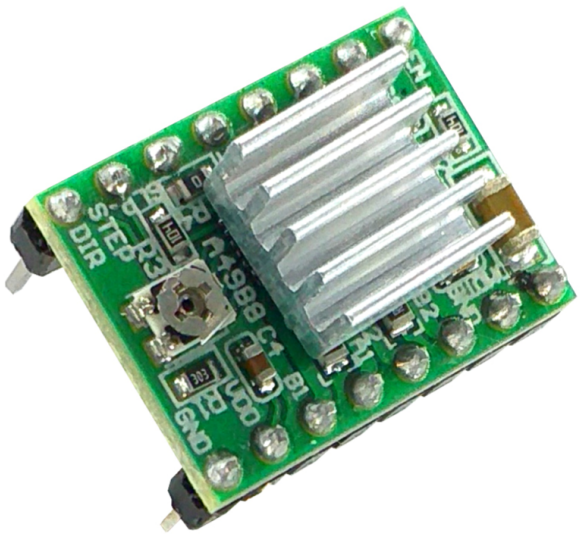
Stepper motors in general have similar names for its wires. But wire names are different for different manufacturers. Here are some examples of different wire names, and how those wires are connected with A4988 driver:

Stepper motor pins					**>**	**Driver pins**
A’	A	1	A1	A+	>	1A
A’’	C	3	A2	A-	>	1B
B'	B	2	B1	B+	>	2A
B’’	D	4	B2	B-	>	2B

(1A and 1B for the first coil, and 2A and 2B for the second coil)

Cooling System - Aluminum Heatsink

Excessive power dissipation of the driver chip results in a rise of temperature that can go beyond the capacity of chip, probably damaging it. Even if the driver IC has a maximum current rating of *2A* per coil, the chip can only supply currents around *1A* per coil without getting overheated. For achieving more than *1A* per coil, a heatsink or other cooling method is required. Our A4988 driver module comes with an aluminum heatsink. It is advisable to install it before using the module.



Current limit potentiometer

Before using the motor, there is a small adjustment that we need to make. We need to limit the maximum amount of current flowing through the stepper coils and prevent it from exceeding rated current of the motor. There is a small trimmer potentiometer on the A4988 driver that can be used to set the current limit. In order to set the current limit, you need to follow the next steps:

» take a look at the datasheet for your stepper motor. Note down rated current of the motor. In our case we are using "*NEMA17*" *200 steps/rev, 12V, 350mA*

» put the driver into full step mode by leaving the three microstep selection pins disconnected

» hold the motor at a fixed position by NOT clocking the *STEP* input. Do not leave the *STEP* input floating, connect it to logic power supply (*5V* for Atmega328P or *3.3V* for Raspberry Pi)

» place the ammeter in series with one of the coils on your stepper motor and measure the actual current

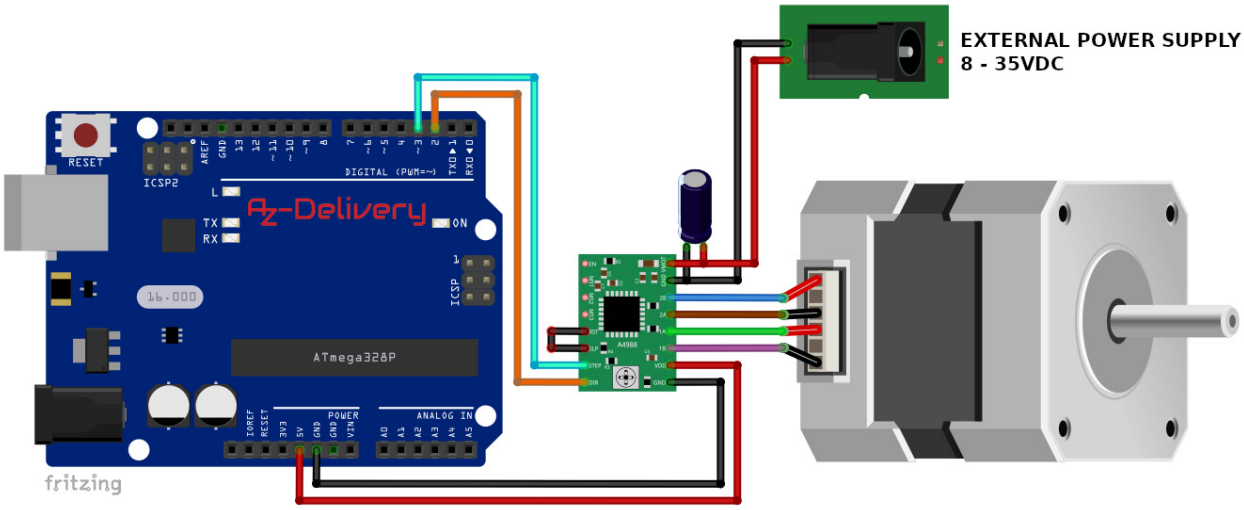
» take a small screwdriver and adjust ~~the~~ potentiometer until you reach rated current value from datasheet of the motor

NOTE: You will need to perform this adjustment again if you ever change the logic voltage (*VDD*).

WARNING: Connecting or disconnecting a stepper motor while the driver is powered can destroy the driver!!!

Connecting the driver with Atmega328P Board

Connect the driver with your microcontroller board like on connection diagram below:



Driver pin	**>**	**Board pin**		
VDD	>	5V		**Red wire**
GND	>	GND		**Black wire**
STEP	>	D3		**Cyan wire**
DIR	>	D2		**Orange wire**
VMOT	>	+	of external power supply	**Red wire**
GND	>	GND	of external power supply	**Black wire**

Connect *RST* pin to the *SLEEP* pin to keep the driver enabled (**Black wire**).

REMEMBER to put a large *100µF* decoupling electrolytic capacitor across motor power supply pins, as close to the board as possible, like on connection diagram above.

Keep the microstep selection pins disconnected to operate the motor in full step mode, or connect appropriate *MS* pin to the *VDD* voltage, to operate in different excitation mode, like we discussed. But if you do this, then the motor shaft for the same *STEP* clock would work slower if you are using some excitation mode other than full step mode. That is because in other excitation modes we are using more steps, more steps = more time to go through them. So in order to use a motor with the same rotational speed, you have to increase clock on the *STEP* pin. With "*NEMA17"* stepper motor, we used *950us* for on state, and *950us* off state for the clock signal on the *STEP* pin. That is the minimum in the full step mode (all *MS* pins are disconnected). But with sixteenth step mode (all three *MS* pins connected to the *VDD*), these values go as low as *35us*. You will see this in the code.

We do not require any libraries for this driver to work. We will create our own sketch.

Code:

```
uint8_t stepPin = 2;
uint8_t dirPin = 3;
int steps = 1000;    // you should increase this if you are using
                     // some of microstepping modes
int usDelay = 950;   // minimal is 950 for full step mode and NEMA15 motor
                     // minimal is 35 for sixteenth step mode

void setup() {
  Serial.begin(9600);
  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
}

void loop() {
  digitalWrite(dirPin, HIGH); // motor direction cw
  for(int x = 0; x < steps; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(usDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(usDelay);
  }
  delay(1000);
  digitalWrite(dirPin, LOW); // motor direction ccw
  for(int x = 0; x < steps; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(usDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(usDelay);
  }
  delay(1000);
}
```

We start the sketch with defining the *STEP* and *DIR* pins that are connected to our Atmega328P Board. We define a variable called *steps* which we use for the number of steps for the motor shaft. In the setup function, we declare *STEP* and *DIR* pins as digital outputs. In the loop section we spin the motor clockwise and then spin it counterclockwise at an interval of two seconds.

To control the spinning direction of a motor we set the *DIR* pin either *HIGH* or *LOW*. *HIGH* input spins the motor clockwise and *LOW* spins it counterclockwise.

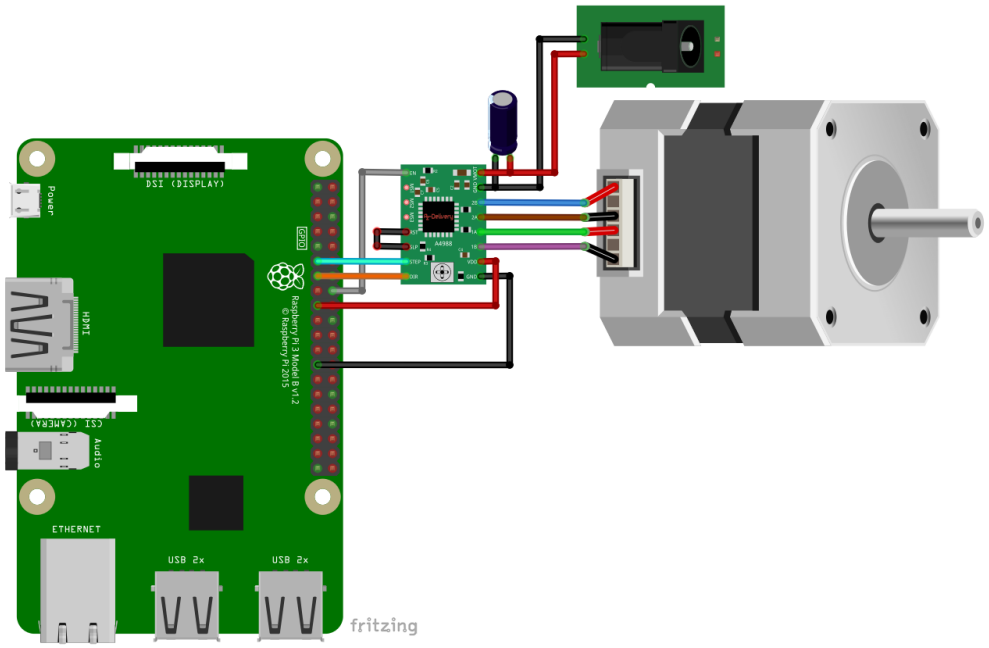
Rotation speed of a motor shaft is determined by the frequency of the pulses (clock signal) we send to the *STEP* pin. The higher the pulses, the faster the motor runs. The pulses are nothing but pulling the output *HIGH*, waiting for a bit then pulling it *LOW* and waiting again. This waiting period is defined by the variable *usDelay*. By changing the delay between the two pulses, you change the frequency of those pulses and hence the rotational speed of the motor shaft.

We tested this code with the *NEMA17* stepper motor. In the full step mode minimum value of *usDelay* variable is *950*. If you go lower than this, the shaft of the motor will not move. This means that the motor can not handle faster clock rates. If you set the sixteenth excitation mode, you get sixteen times more steps than in the full step mode. So in order to move shaft of the motor to the same position like in full step mode, we need to increase

variable *steps* sixteen times. And to get the same rotation speed as in full step mode (*usDelay* = *950*) we need to change the value of *usDelay* to *35* (which is minimum for this mode). If you set this variable lower than *35*, the shaft of the motor will not move.

Connecting the driver with Raspberry Pi

Connect the driver with your Raspberry Pi like on connection diagram below:



Driver pin	**>**	**Raspberry Pi pin**		
VDD	>	3.3V	[pin 17]	**Red wire**
GND	>	GND	[pin 30]	**Black wire**
STEP	>	GPIO17	[pin 11]	**Cyan wire**
DIR	>	GPIO27	[pin 15]	**Orange wire**
EN	>	GPIO23	[pin 16]	**Gray wire**
VMOT	>	+	of external power supply	**Red wire**

GND > GND of external power supply **Black wire**

Connect *RST* pin to the *SLEEP* pin to keep the driver enabled (**Black wire**). **REMEMBER** to put a large *100µF* decoupling electrolytic capacitor across as close to the board as possible, like on connection diagram above.

Like for microcontroller board, we do not need to install any libraries for this driver module. We will only create our Python script. Here is the code:

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
STEP = 17 # step pin
DIR = 27 # direction pin
EN = 23 # enable pin
GPIO.setup(STEP, GPIO.OUT)
GPIO.setup(DIR, GPIO.OUT)
GPIO.setup(EN, GPIO.OUT)
steps = 5000 # number of steps
usDelay = 950 # number of microseconds
uS = 0.000001 # one microsecond
GPIO.output(EN, GPIO.LOW)
print("[press ctrl+c to end the script]")
try: # Main program loop
    while True:
        GPIO.output(DIR, GPIO.HIGH) # cw direction
        for i in range(steps):
            GPIO.output(STEP, GPIO.HIGH)
            sleep(uS * usDelay)
            GPIO.output(STEP, GPIO.LOW)
            sleep(uS * usDelay)
        sleep(2)
        GPIO.output(DIR, GPIO.LOW) # ccw direction
        for i in range(steps):
            GPIO.output(STEP, GPIO.HIGH)
            sleep(uS * usDelay)
```

```
            GPIO.output(STEP, GPIO.LOW)
            sleep(uS * usDelay)
        sleep(2)
# Scavenging work after the end of the program
except KeyboardInterrupt:
    GPIO.output(EN, GPIO.HIGH)
```

We just transformed sketch into Python code. The only difference is the code for *EN* pin. This is the pin for enabling the driver. If it is pulled *LOW*, the driver is enabled, and if it is pulled *HIGH*, the driver is disabled. We need this because, *DIR* and *STEP* pins are floating, and if we end the script without connecting these pins to *GND* or *VDD*, driver would go crazy, and the shaft of the motor starts moving weird. So we have to disable the driver when the script ends. We do this in the *except* block of the code.

The rest of the code is the same as in the sketch, so we do not need to explain it.

You've done it!

You can now use your module for your projects.

Now it is time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find on the internet.

If you are looking for the high quality microelectronics and accessories, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us