Welcome!

Thank you for purchasing our *AZ-Delivery IRF520 MOS Driver module*. On the following pages, you will be introduced to how to use and set-up this handy device.

Have fun!

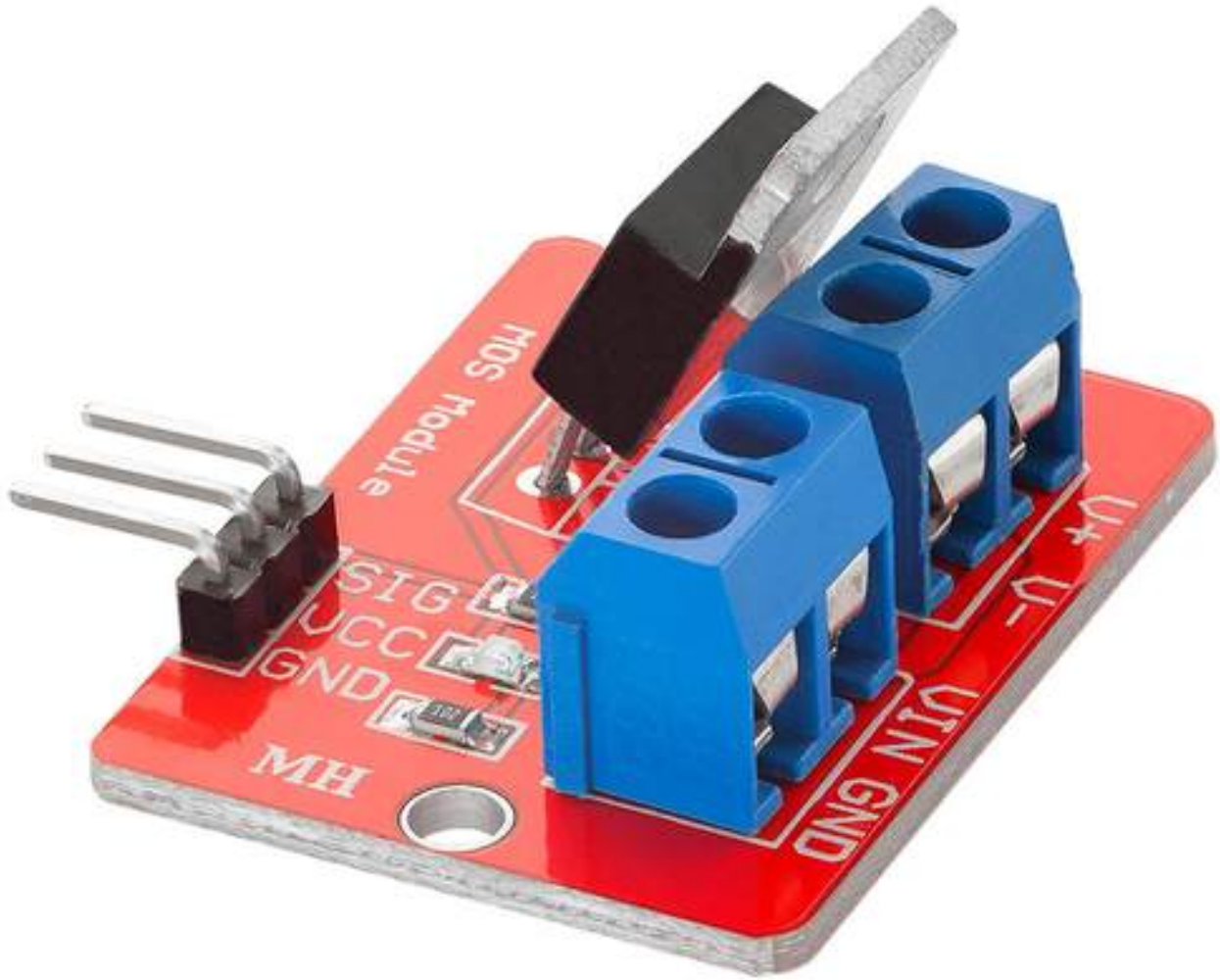


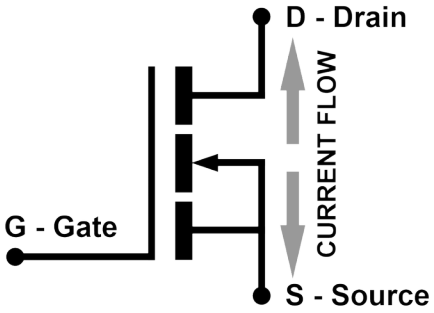
Table of Contents

Introduction

The IRF520 MOS Driver module is a breakout board for IRF520 MOSFET. It is designed to switch higher current and voltage DC loads and it is controlled with a single digital pin from the microcontroller.

The module is based on the IRF520 MOS-FET transistor. The MOSFET is a Metal-oxide Semiconductor Field Effect Transistor. It can be used for switching or amplifying the voltages in circuits.

Essentially the purpose of a MOSFET transistor is to control voltage and current flow between the source and the drain pins.



The MOSFET can be used in various applications such as high-current and high-speed power supplies, relay and solenoid drivers circuits, voltage regulators, step-up and step-down voltage converters, motor control, audio amplifiers, etc.

The IRF520 MOS Driver module consists of IRF520 MOSFET, two resistors, LED and block screw terminal connectors.

Specifications

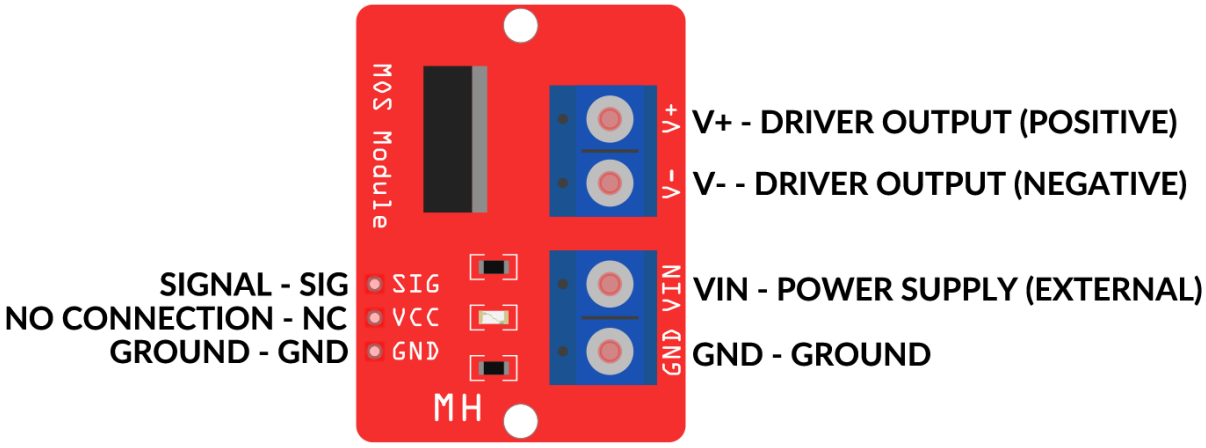
Output load voltage	0 - 24V DC
Output load current	less then 5A
Signal input voltage	3.3V - 5V
Interface	Digital
Mounting holes diameter	3mm
Dimesions	33x24x12mm (1.3x0.9x0.5in)

The module has on-board LED that is used as indicator when the signal is applied from the microcontroler to the module. The signals applied can be somewhat visually distinctive such as simple ON/OFF or slower PWM signal which looks like a fade-in/fade-out effect.

NOTE: Under the higher switching frequency and higher current load (over 1A) the MOSFET requires a heatsink. Under these conditions the MOSFET produces a lot of termal energy (heat) that needs to be channeled elsewhere. When termal, voltage and current capabilities are near the overload, the MOSFET can easily burn out, so it is preferable not to exceed the MOSFET specified limitations. For further information refer to the manufacturers datasheet.

The pinout

The IRF520 MOS Driver module has seven pins. The pinout is shown on the following image:



SIG - The signal pin is the analog input pin through which the module is controled

VCC - This pin actually is not connected anywhere and it can be designated as NC - No connection

GND - Ground pin, connected to the GND pin on the block screw terminal (right side of the image) internally.

VIN - External power supply pin for the device connected at the V+ and V- driver output pins.

V+ - This is the postive terminal for the device that is controled with the module.

V- - This is the negative terminal for the device that is controled with the module.

How to set-up Arduino IDE

If the Arduino IDE is not installed, follow the *link* and download the installation file for the operating system of choice. The Arduino IDE version used for this ebook is 1.8.12.

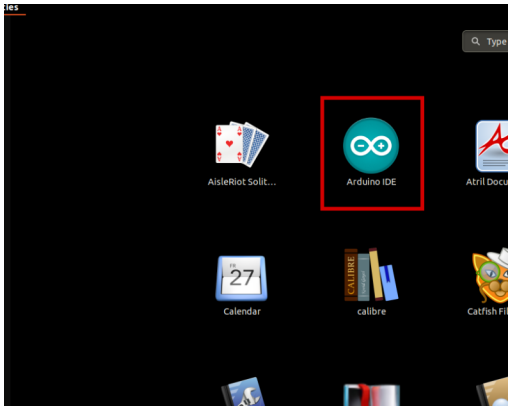
For *Windows* users, double click on the downloaded *.exe* file and follow the instructions in the installation window.

For *Linux* users, download a file with the extension *.tar.xz*, which has to be extracted. When it is extracted, go to the extracted directory and open the terminal in that directory. Two *.sh* scripts have to be executed, the first called *arduino-linux-setup.sh* and the second called *install.sh*.

To run the first script in the terminal, open the terminal in the extracted directory and run the following command:
sh arduino-linux-setup.sh user_name
user_name - is the name of a superuser in the Linux operating system. A password for the superuser has to be entered when the command is started. Wait for a few minutes for the script to complete everything.

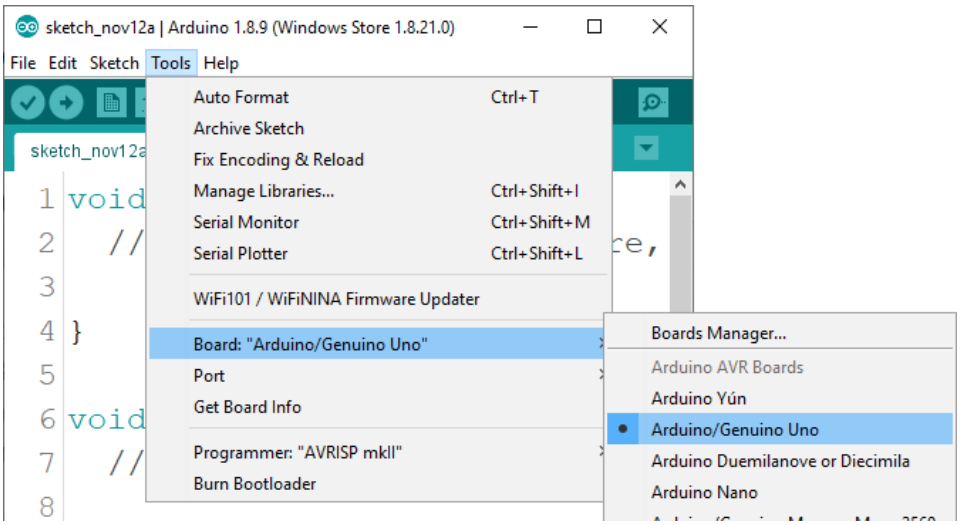
The second script, called *install.sh*, has to be used after the installation of the first script. Run the following command in the terminal (extracted directory): **sh install.sh**

After the installation of these scripts, go to the *All Apps*, where the *Arduino IDE* is installed.



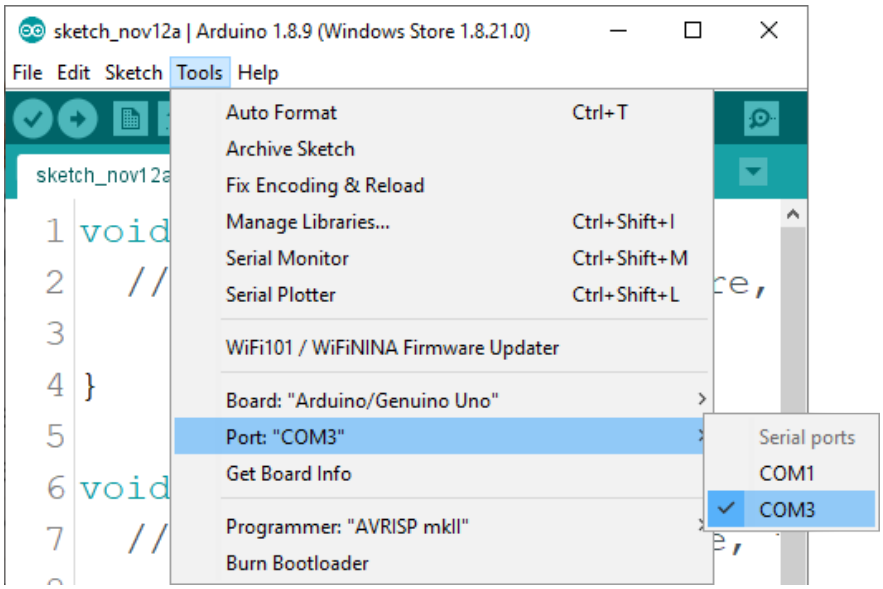
Almost all operating systems come with a text editor preinstalled (for example, *Windows* comes with *Notepad*, *Linux Ubuntu* comes with *Gedit*, *Linux Raspbian* comes with *Leafpad*, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect an Atmega328p board. Open freshly installed Arduino IDE, and go to:
Tools > Board > {your board name here}
{your board name here} should be the *Arduino/Genuino Uno*, as it can be seen on the following image:



The port to which the Atmega328p board is connected has to be selected. Go to: *Tools > Port > {port name goes here}*
and when the Atmega328p board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

If the Arduino IDE is used on Windows, port names are as follows:



For *Linux* users, for example port name is */dev/ttyUSBx*, where *x* represents integer number between *0* and *9*.

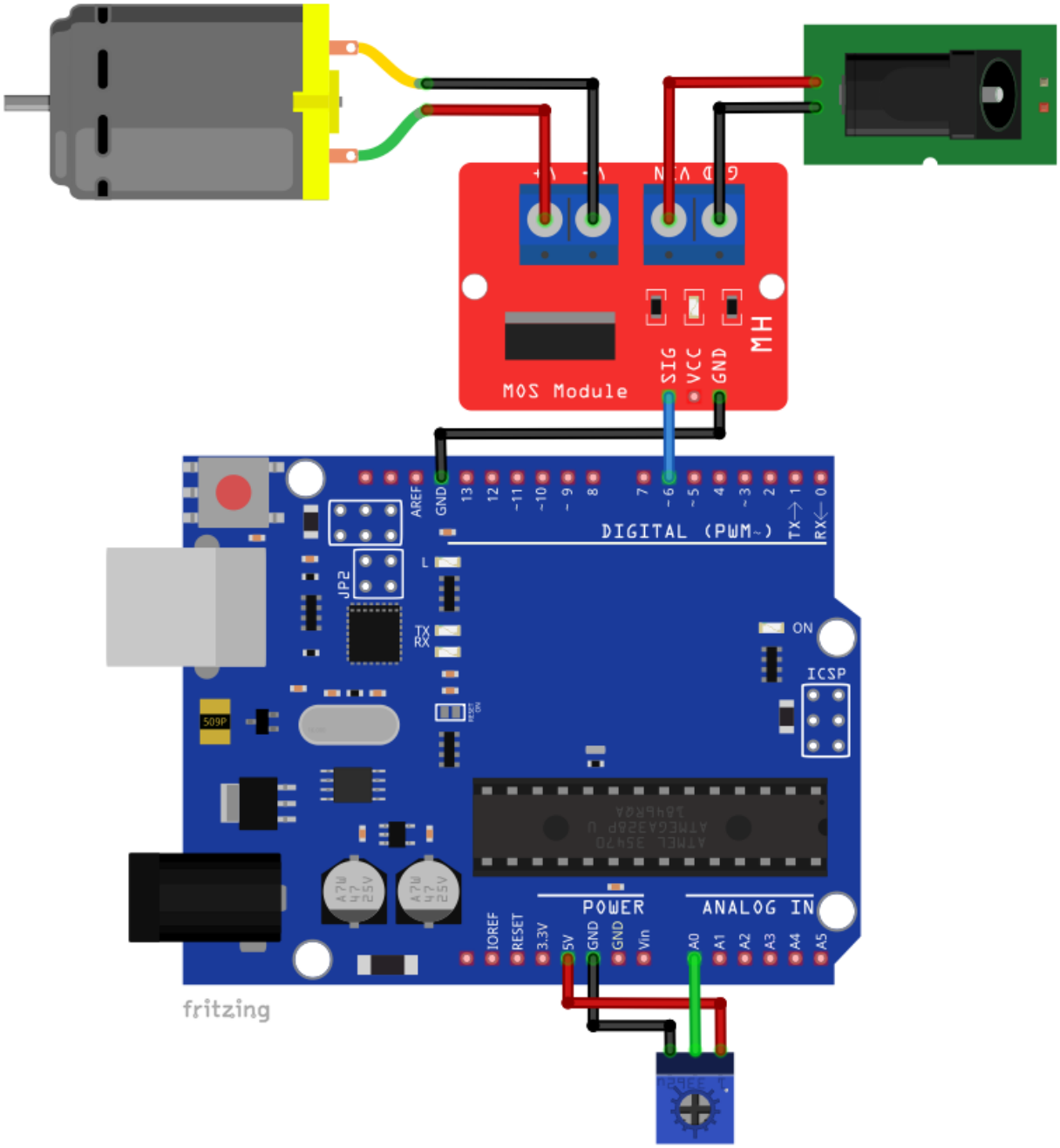
How to set-up the Raspberry Pi and Python

For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without the need for a *PC* screen Monitor, mouse or keyboard. The only things that are used in this mode are the Raspberry Pi itself, power supply and internet connection. All of this is explained minutely in the free eBook: *Raspberry Pi Quick Startup Guide*

The *Raspbian* operating system comes with *Python* preinstalled.

Connecting the module with Atmega328p

Connect the module with the Atmega328p as shown on the following connection diagram:



Module pin	Mc pin	Wire color
VCC	NC (no connection)	Red wire
GND	GND	Black wire
SIG	6	Blue wire
Module pin	**Battery (External power supply)**	**Wire color**
VIN	DC Connector (+)	Red wire
GND	DC Connector (-)	Black wire
Module pin	**Motor pin**	**Wire color**
V+	1	Red wire
V-	2	Black wire
Mc pin	**Potentiometer pin**	**Wire color**
5V	1	Red wire
A0	2 (center pin)	Green wire
GND	3	Black wire

NOTE: As an example here, the DC motor is controled by Atmega328p via MOS Driver. Depending on a motor type that is used, the external DC power supply has to match the voltage of the motor. The voltage from the DC power supply must not exceed the required motor voltage and the module output load voltage (24V), otherwise, the module or the motor can be damaged!

Library for Arduino IDE

To use the module with Atmega328p it is recommended to download an external library. The most simple library that is used in this eBook is called the `HCMotor`, version 0.1 which can be downloaded on the following *link*.

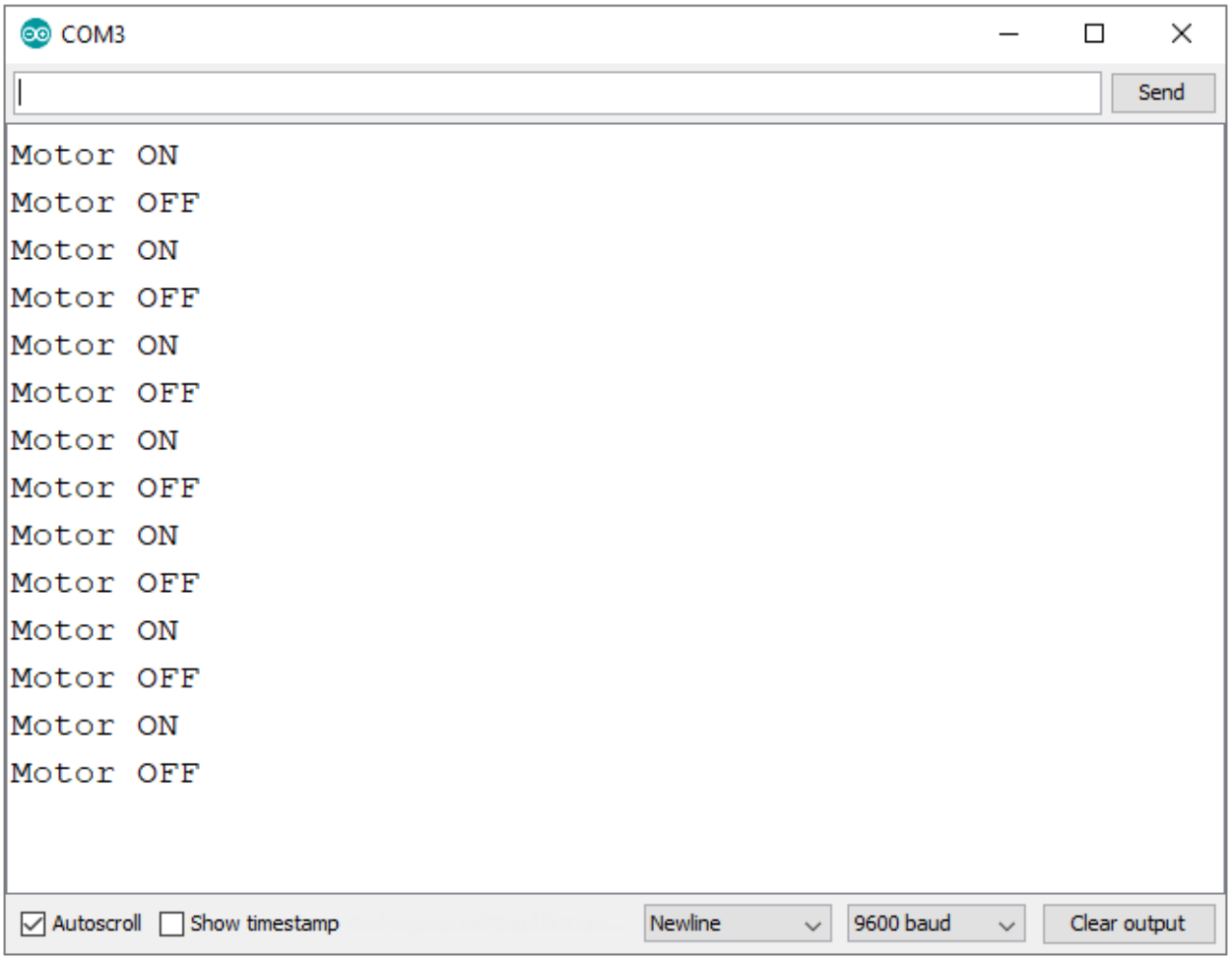
When the *.zip* file is downloaded, open Arduino IDE and go to:
Sketch > Include Library > Add .ZIP Library
and add the downloaded zip file.

Sketch examples

Simple ON/OFF example

```
void setup() {
  // Set digital pin 6 to OUTPUT
  pinMode(6, OUTPUT);
}

void loop() {
  Serial.begin(9600);
  // Turn the Motor ON (The logic level is HIGH)
  digitalWrite(6, HIGH);
  Serial.print("Motor ON");
  // One second pause
  delay(1000);
  // Turn the Motor OFF (The logic level is LOW)
  digitalWrite(6, LOW);
  Serial.print("Motor OFF");
  // One second pause
  delay(1000);
}
```

Upload the sketch to the Atmega328p and run the Serial Monitor (*Tools* > *Serial Monitor*). The result should look like as on the following image:



The sketch starts with the *setup()* function where the mode of digital pin 6 on the Atmega328p is initialized and set to OUTPUT.

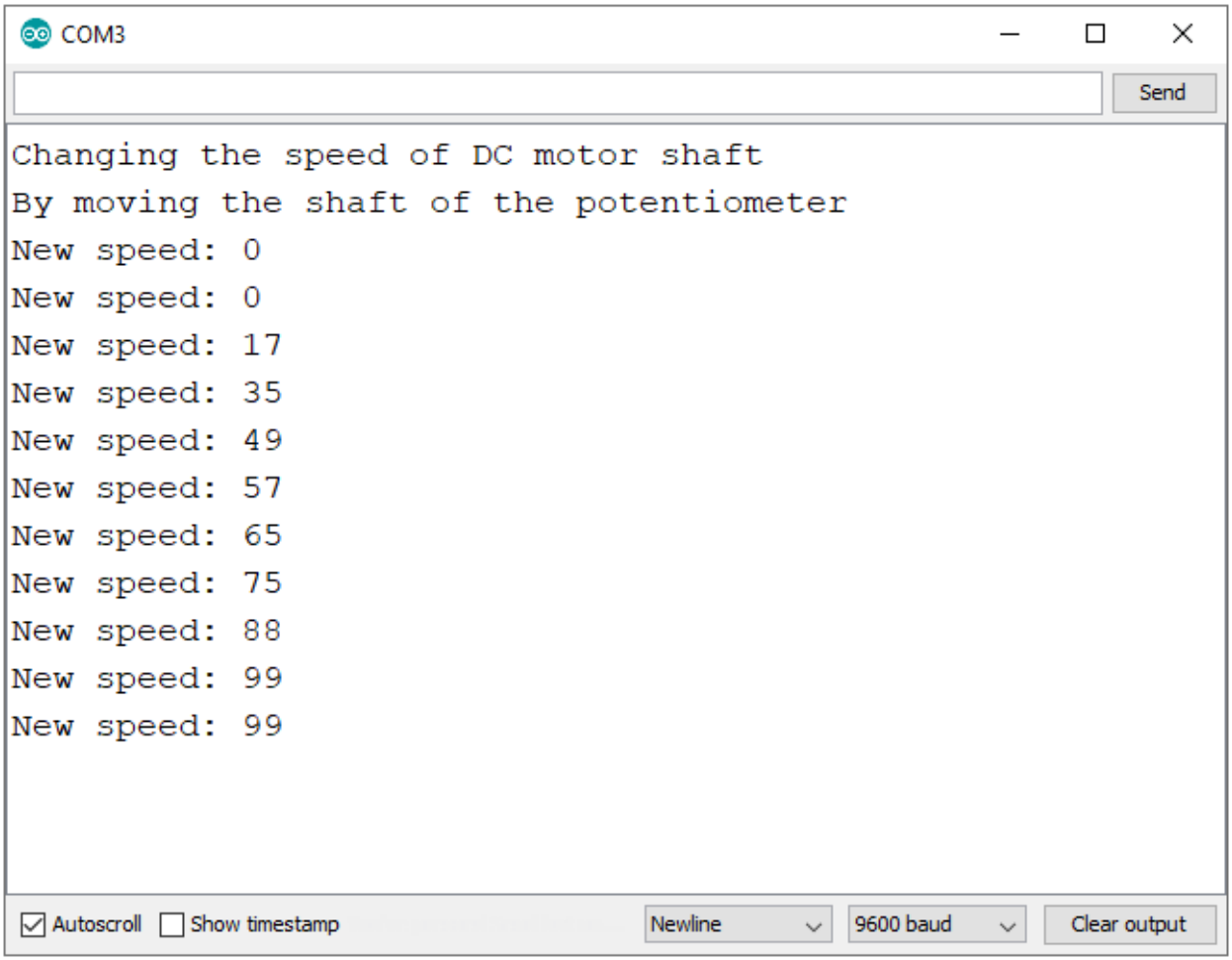
Next, in the *loop()* function the *digitalWrite()* function is used to turn the motor ON or OFF by setting the digital pin 6 voltage level to HIGH or LOW.

The motor is turned ON for one second and then turned OFF for one second. This is repeated until Atmega328p gets turned OFF.

Adjustable PWM example

```
#include "HCMotor.h"
#define MOTOR_PIN 6
#define POT_PIN A0

HCMotor HCMotor;

void setup() {
  Serial.begin(9600);
  HCMotor.Init();
  HCMotor.attach(0, DCMOTOR, MOTOR_PIN);
  HCMotor.DutyCycle(0, 100);
  Serial.println("Changing the speed of DC motor shaft");
  Serial.println("By moving the shaft of the potentiometer");
}
void loop() {
  uint16_t Speed = map(analogRead(POT_PIN), 0, 1024, 0, 100);
  HCMotor.OnTime(0, Speed);
  Serial.print("New speed: ");
  Serial.println(Speed);
  delay(1000);
}
```

Upload the sketch to the Atmega328p and run the Serial Monitor (*Tools* > *Serial Monitor*). The result should look like as on the following image:



At the beginning of the second sketch, the library called *HCMotor* is included.

Then, the macros called *MOTOR_PIN* and *POT_PIN* are created.

The macro called *MOTOR_PIN* represents the digital pin that is used to control the module. With this macro digital pin 6 is assigned for use.

The macro called *POT_PIN* represents the analog input of the Atmega328p to which the potentiometer is connected.

Next, the object called *HCMotor* is created.

Then, in the *setup()* function the *HCMotor* object is initialized with the following line of code:

```
HCMotor.Init();
```

After that, the library function called *attach()* is used and it has three arguments and returns no values. It is used to attach the PWM output (Digital pin) for a standard DC motor to a module *signal* (SIG) pin. The first argument represents the number of the motor, connected to specified pin (third argument). Values for this argument can be between `0` to `MAXMOTORS`. Second argument represents the type of the motor that is used. These values can be `DCMOTOR` or `STEPPER`. The third value represents pin of motor that is in use.

This module supports only DC loads, for example DC motors, thus the used value DCMOTOR is used for the second argument.

Then, the function called *DutyCycle()* is used to control the speed of the motor. It has two arguments and returns no values.

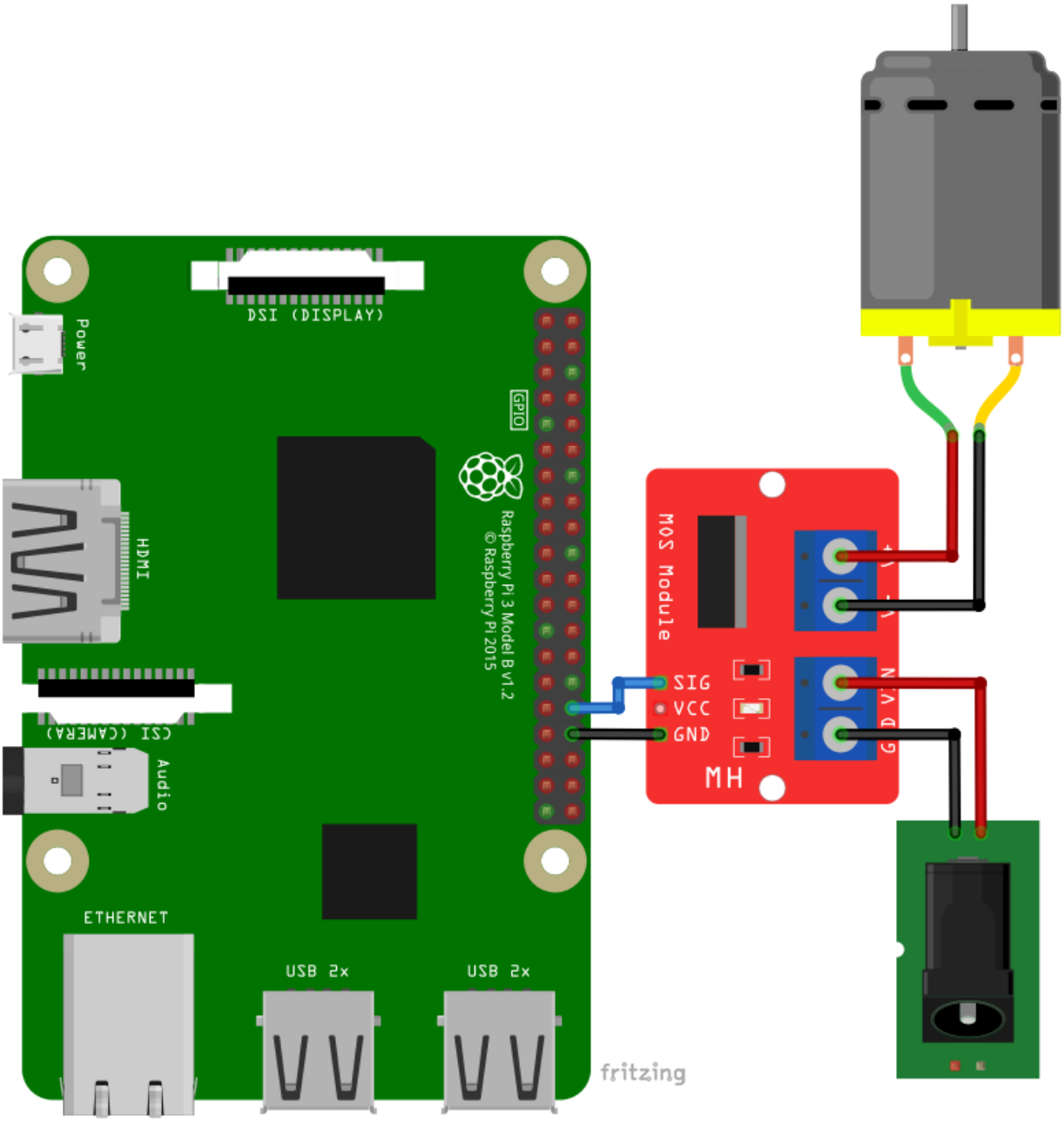
Next, in the *loop()* function, the variable *Speed* is created which is an integer value, and it is used to read the values from the analog pin (A0). The analog values range is from 0 to 1024. The values are then mapped to a range from 0 to 100 to reduce down the range. This is done with the following line of code:

```
Speed = map(analogRead(POT_PIN), 0, 1024, 0, 100);
```

Then, the function called *OnTime()* is used to set the duty cycle for DC motor, where: *MotorNum* is a value used to reference which motor is to be altered (if more then one are used).

Connecting the module with Raspberry Pi

Connect the module with the Raspberry Pi as shown on the following connection diagram:



Module pin	Raspberry Pi pin	Physical pin	Wire color
SIG	GPIO12	32	**Blue wire**
GND	GND	34	**Black wire**
Module pin	**Motor pin**		**Wire color**
V+	1		**Red wire**
V-	2		**Black wire**
Module pin	**DC Connector (External power supply)**		**Black wire**
VIN	DC Connector (+)		**Red wire**
GND	DC Connector (-)		**Black wire**

Libraries and tools for Python

To use the module with the Raspberry Pi, the library *RPi.GPIO* has to be installed. To install the library, open the terminal and run the following commands, one by one:

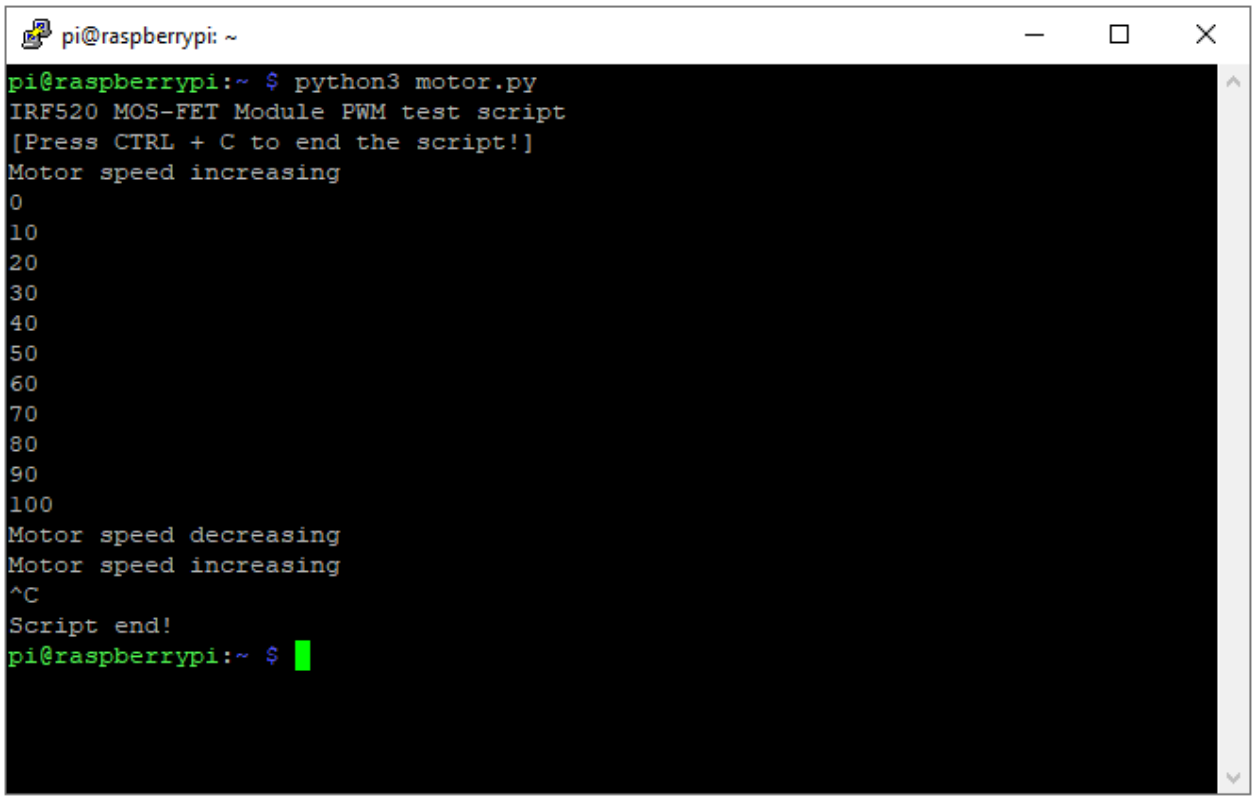
```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install python3-rpi.gpio
```

Python script

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(12, GPIO.OUT)
pwm = GPIO.PWM(12, 100)

print('IRF520 MOS-FET Module PWM test script')
print('[Press CTRL + C to end the script!]')

dc=0
pwm.start(dc)
try:
  while True:
    print('Motor speed increasing')
    for dc in range(0, 101, 10):
      pwm.ChangeDutyCycle(dc)
      time.sleep(1)
      print(dc) # printing raising PWM values
    print('Motor speed decreasing')
    for dc in range(95, 0, -10):
      pwm.ChangeDutyCycle(dc)
      time.sleep(0.05)

except KeyboardInterrupt:
  print('\nScript end!')

finally:
    pwm.stop()
    GPIO.cleanup()
```

Save the script by the name *motor.py*. To run the script open the terminal in the directory where the script is saved and run the following command: **python3 motor.py**

The result should look like as on the following image:



To stop the script press ‘CTRL + C’ on the keyboard.

The script starts with importing two libraries: *time* and *RPi.GPIO*.

Then, the GPIO pin naming is set to BCM, and the mode for GPIO pin on which *SIG* pin of MOS Driver is connected is set to OUTPUT.

Then, the object called *pwm* is created and it is initialized with the following line of code:

```
pwm = GPIO.PWM(12, 100)
```

where the GPIO pin 12 is used and the PWM with duty cycle is set up on this pin, with the frequency of 100Hz.

Next the variable called *dc* is created which represents the PWM duty cycle percentage. The starting value for the variable is 0%.

After this, the *try-except-finally* block of code is created. In the *try* block of code, the indefinite loop block (*while True:*) is created. Inside this block of code value of *dc* variable gets incremented by 10 every second. When this value reaches the value 100 it gets decremented by 10 to the value of zero, where the process is repeated.

The function called *ChangeDutyCycle()* changes the PWM duty cycle value by changing of *dc* variable with the following line of code: `pwm.ChangeDutyCycle(dc)`.

To stop the script press ‘CTRL + C’ on the keyboard. This is called keyboard interrupt. When keyboard interrupt happens, the `except` block of code is executed, displaying message `Script end!` in the terminal.

The `finally` block of code is executed at the script end. When the `finally` block of code is executed, the functions called `stop()` and `cleanup()` are executed. The `pwm.stop()` stops the PWM signal on the specified GPIO pin and `cleanup()` function disables all used GPIO interfaces and GPIO pin modes.

Now it is the time to learn and make your own projects. You can do that with the help of many example scripts and other tutorials, which can be found 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