Inputs and Outputs Guide

Arduino PLC, ESP32 PLC Raspberry Pi PLC and Panel PC

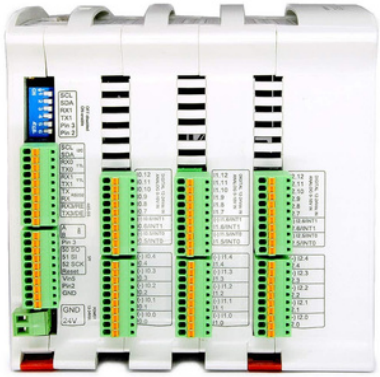
Industrial Shields®

Raspberry Pi

ARDUINO

Basics about digital inputs of an industrial PLC

Introduction



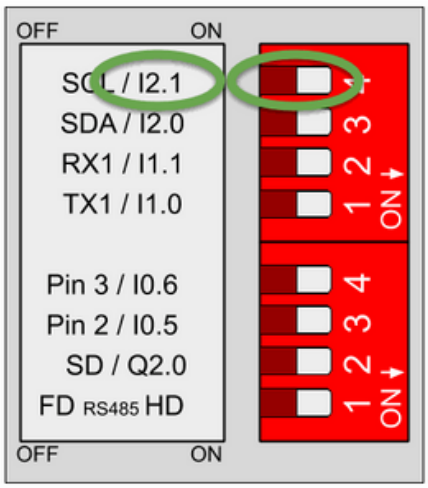
Thanks to this reading you will understand how to connect and configure the PLCs to be able to read the digital inputs correctly.

Configuring the switches

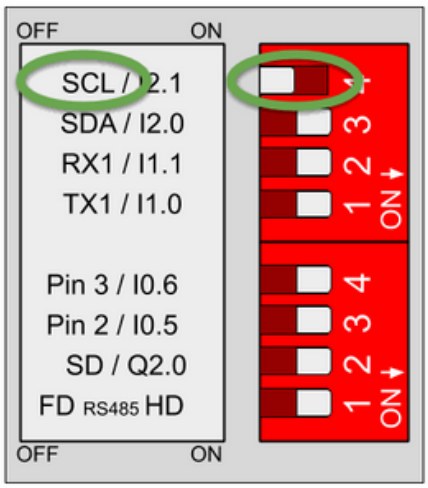
Almost all the digital inputs are always connected to the internal **Arduino**, but in a few cases, the user can choose a special peripheral configuration or a GPIO normal working. In these cases, the user can choose between two options through the switches.

Each switch can select only one configuration. For example, in this case, we are watching the **GPIOs** configuration of an M-Duino 57R+. If we put the switch to the right in the upper one, the input I2.1 will be activated and we will be able to work with this input as digital.

If the switch is in the left position, we will activate the **SCL** line which will be used for **I2C** communication. Keep in mind each switch has two different configurations: you must select the right or the left option.



I2.1 input enabled - SCL disabled



I2.1 input disabled - SCL enabled

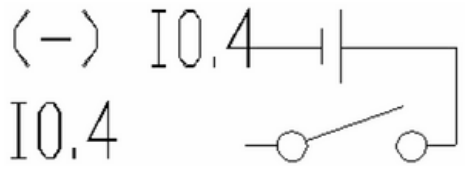
Basics about digital inputs of an industrial PLC

Input types

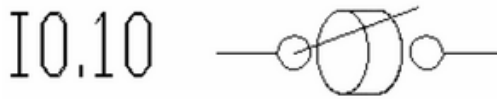
There are three different types of inputs in the **Industrial Shields** PLCs:

- 5V - 24V input
- 5V - 24V optoisolated input
- 5V input

Each one has a particular draw in the case of the **PLC**. Remember only the Pin 2 and Pin 3 are 5V compatibles:



5V - 24V optoisolated input



5V - 24V input

5V input

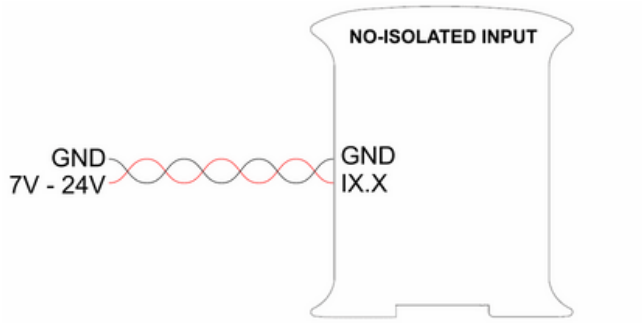
Hardware

Not all the inputs must be connected in the same way. While the non-isolated inputs must be referenced to the same ground as the **PLC**, the isolated inputs can be connected to the input grounds, allowing to isolate systems from the **PLC**.

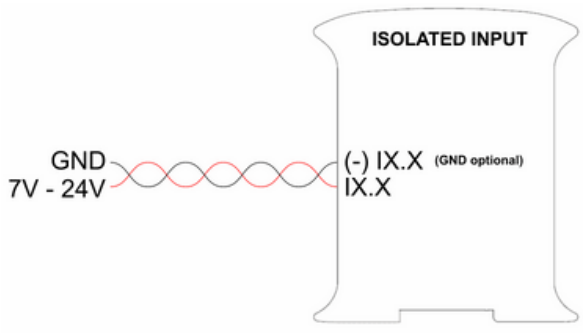
Anyway, the optoisolated input can be connected to the **PLC** ground as well.

The following images show how to connect the different inputs to the **PLC** for industrial automation:

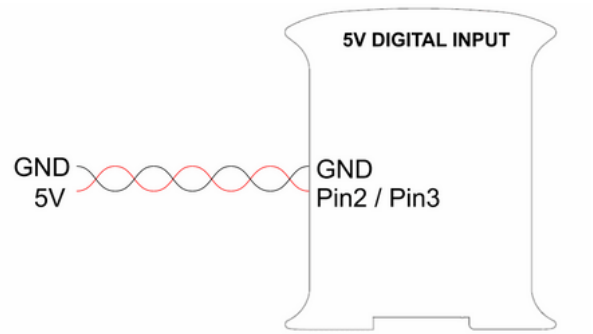
5V - 24V optoisolated input



5V - 24V input



5V input



Basics about digital inputs of an industrial PLC

Software

In order to program the digitals **GPIO**, we must keep in mind we can read the values with the following command:

```
digitalRead(GPIO);
```

This function returns "0" or "1" depending on the actual value of the input. **GPIO** is the name of the input. Imagine we want to know the state of the "I0.4" input, then, we must write this line:

```
digitalRead(I0_4);
```

The inputs "2" and "3" does not have a special name, and to read them we must write:

```
digitalRead(2);
digitalRead(3);
```

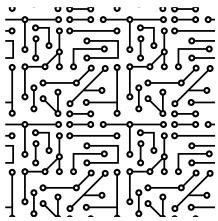
We must keep in mind we do not need to configure the digital inputs of the **PLC** as digital ones, except with the 5V compatible inputs. It means we must configure the inputs in the setup before read them: These statements must be defined within the Setup function:

```
pinMode(2,INPUT);
pinMode(3,INPUT);
```

ARDUINO

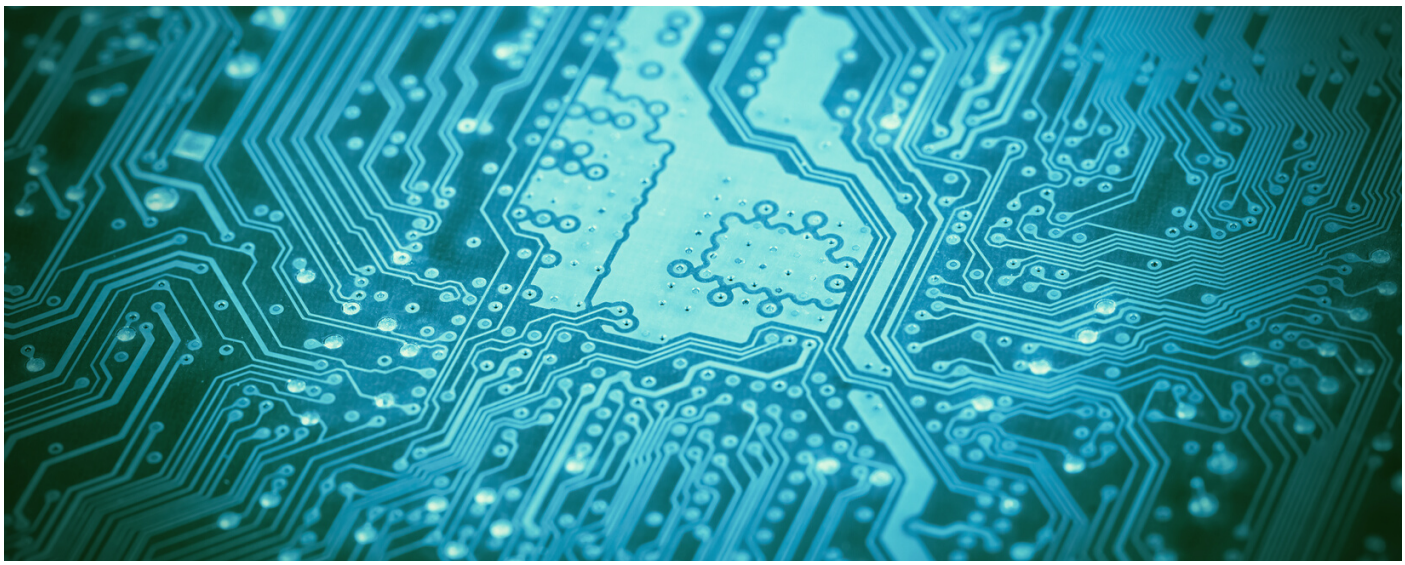
Basics about digital inputs of an industrial PLC

Software



Examples
You can see a read digital **GPIO** example in the following paragraph:

```
// Digital read example
// This example reads the I0_10, I0_2 and Pin 2 inputs, and shows via
serial if they are active

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);

    // Configure Pin 2 as a digital input
    pinMode(2, INPUT);
}

// Loop function
void loop()
{
    // Check Pin 2
    if (digitalRead(2))
        Serial.println("Pin 2 active");

    // Check I0_10
    if(digitalRead(I0_10))
        Serial.println("I0_10 active");

    // Check I0_2
    if(digitalRead(I0_2))
        Serial.println("I0_2 active");
```



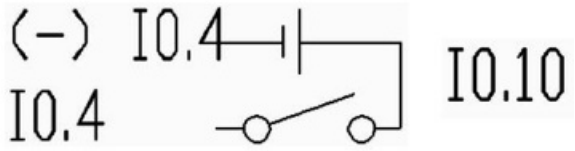
Digital inputs of a Raspberry PLC

Input types

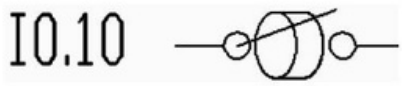
There are two different types of inputs in the **Raspberry Pi industrial PLC** devices:

- 5 Vdc - 24 Vdc input
- 5 Vdc - 24 Vdc optoisolated input

Each one has a particular draw in the case of the PLC:



5 - 24 Vdc Optoisolated Input

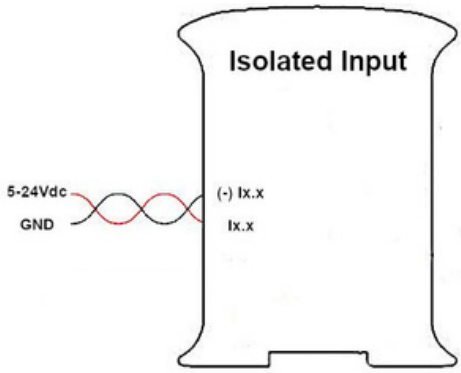


5 - 24 Vdc Input

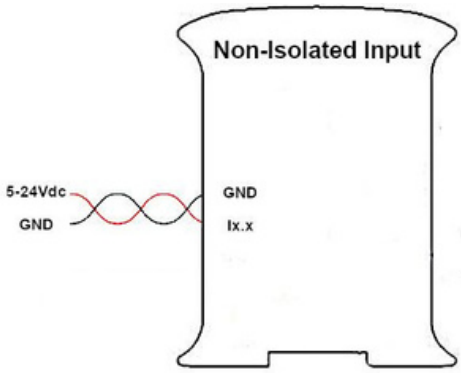
Hardware

Not all the inputs must be connected in the same way. While the non-isolated inputs must be referenced to the same ground as the **PLC**, the isolated inputs can be connected to the input grounds, allowing to isolate systems from the **PLC**. Anyway, the optoisolated input can be connected to the **PLC** ground as well.

The following images show how to connect the different inputs to the **PLC** for industrial automation:



5 - 24 Vdc Optoisolated Input



5 - 24 Vdc Input

Software

How to work with Bash Scripts

Raspberry Pi PLC has default bash scripts to work with the inputs. All the inputs and outputs scripts must be executed from the correct path.

It depends on the shield type of the I/O executed. In function of the shield of the I/O that you need to activate, you must execute the scripts from a specific path:

- Analog/Digital Shields

```
> cd /home/pi/test/analog
```

- Relay Shield

```
> cd /home/pi/test/relay
```

Digital inputs of a Raspberry PLC

The get-digital-input script will show the value of the selected input pin. It will only be provided the pin with which we are going to work. In order to call the function, we will do the following:

```
> ./get-digital-input <input>
```

Example for the I0.0 input returning a True value:

```
> ./get-digital-input I0.0
```

How to work with Python

The bash commands are the basis to work easily with the **Raspberry Pi PLC**. In order to work with python files, if you want to interact with the IOs of the PLC, you will have to call these scripts.

To edit the files you will be working with the Nano editor included by default and Python3.

```
nano digital_inputs.py
```

Python allows you to execute a shell command that is stored in a string using the subprocess library. In order to work with it, you will have to import it at the start of the file.

```
import subprocess
```

In this example, you will be reading the input given of the pin I0.0 **of the Raspberry Pi PLC**. In order to do it, you will implement a loop that will be constantly reading the input value. If it detects voltage, it will print a True value.

```
import subprocess
import time
print("Start")
while True:
          try:
                    x = subprocess.run(["./get-digital-
  input","I0.0"], stdout=subprocess.PIPE, text=True)
                    if '1' in x.stdout:
                              print(True)
                              time.sleep(1)
                    else:
                              print(False)
                              time.sleep(1)
          except KeyboardInterrupt:
                    print("\nExit")
                    break
```

In order to execute the Python program, you will call it as follows:

```
> python3 analog_outputs.py
```

To exit the program, just press ^C.

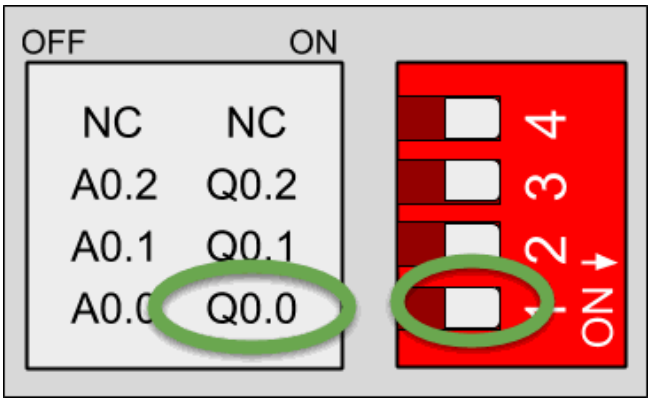
Basics about digital outputs of an industrial PLC

Reading this section, you will be able to understand how to connect and configure the digital outputs of your industrial Arduino PLC controller.

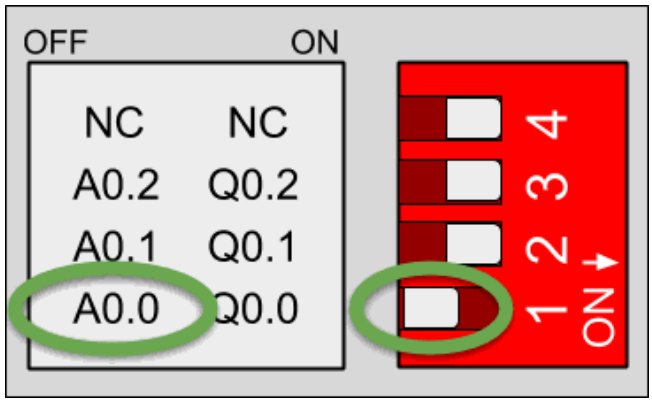
Configuring the switches

Most of the digital outputs are always connected to the internal **Arduino**, but in few cases, the user can choose between a special peripheral configuration or a **GPIO** by changing the position of the **Dip** Switches.

Each switch can select only one configuration. For example, in this case, we are watching the **GPIOs** configuration of an **M-Duino 21+**. If we put the switch to the right position (ON) in the lower one, the output **Q0.0** will be activated and we will be able to work this as digital.
If the switch is in the left position (OFF) we will activate the output as analog.
Keep in mind each switch has two different configurations: you must select the right (ON) or the left (OFF) option.



Q0.0 enabled - A0.0 disabled



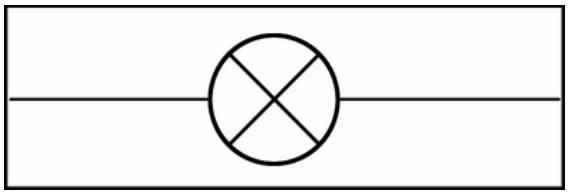
Q0.0 disabled - A0.0 enabled

Output types

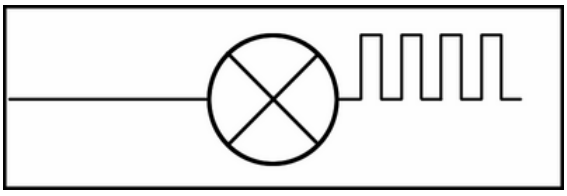
In all of the **Industrial Shields Arduino based PLCs**, digital outputs can work at:

- 5V -24V digital output

Digital outputs have a special draw in the case of the **PLC**. Keep in mind that the output that can handle PWM is the same as the other digital outputs



Digital output



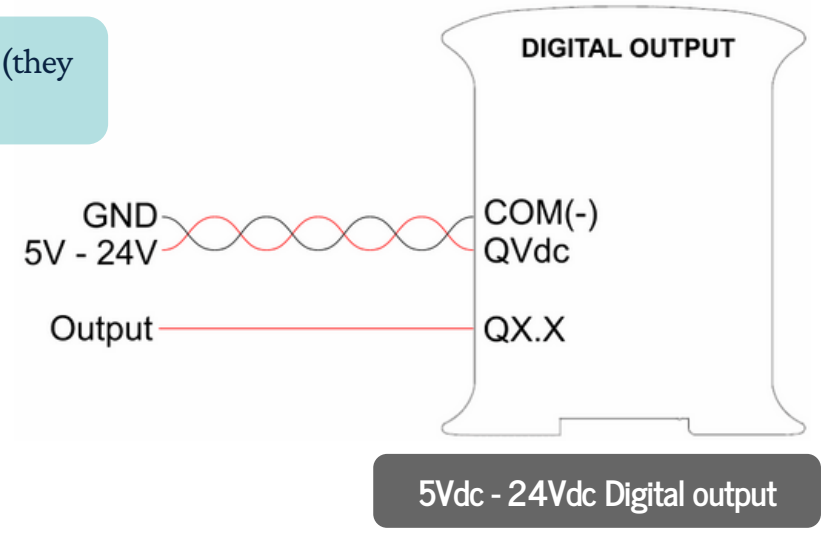
Digital output (PWM optional)

Basics about digital outputs of an industrial PLC

Hardware

All the digital outputs are optoisolated (they use the same **GNDs** as the **PLC**).

The following image shows how to connect a digital output to the **PLC**:



5Vdc - 24Vdc Digital output

Software

In order to program the digital outputs, we must keep in mind that we can write the values with the following command:

```
digitalWrite(GPIO,value);
```

This function writes a "HIGH" or "LOW" in the "GPIO" selected. Imagine we want to write a "HIGH" in the "Q0.6" output, then, we must write this line:

```
digitalWrite(Q0_6,HIGH);
```

We must know we do not need to configure the digital outputs as digital. Industrial Shields' libraries do all the work for us.
Example
You can see a digital **GPIO** written in the following paragraph:

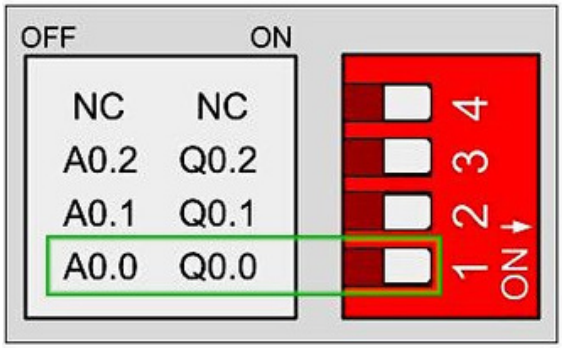
```
// Digital write example
// This example writes the Q0_0 and shows via serial the value

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);
}

// Loop function
void loop()
{
    Serial.println("1");
    digitalWrite(Q0_0, HIGH);
    Serial.println("0");
    digitalWrite(Q0_0, LOW);
}
```

Digital outputs of Raspberry PLC

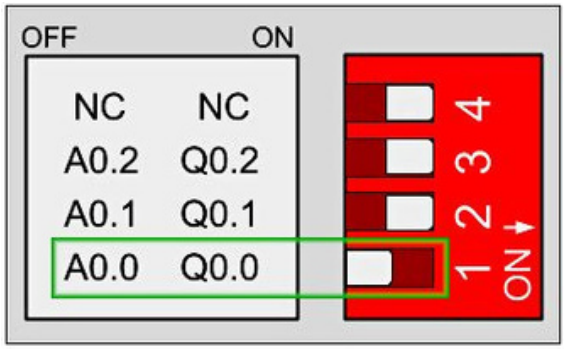
Configuring the switches

Most of the digital outputs are always connected to the internal **Raspberry Pi**, but in a few cases, the user can choose between a special peripheral configuration or a GPIO by changing the position of the Dip Switches.

Each switch can select only one configuration. For example, in this case you are watching the GPIOs configuration of an open source PLC Raspberry Pi 21+. If you put the switch to the right position (ON) in the lower one, the output Q0.0 will be activated and you will be able to work as digital. If the switch is in the left position (OFF), you will activate the output as analog. Keep in mind each switch has two different configurations: you must select the right (ON) or the left (OFF) option.



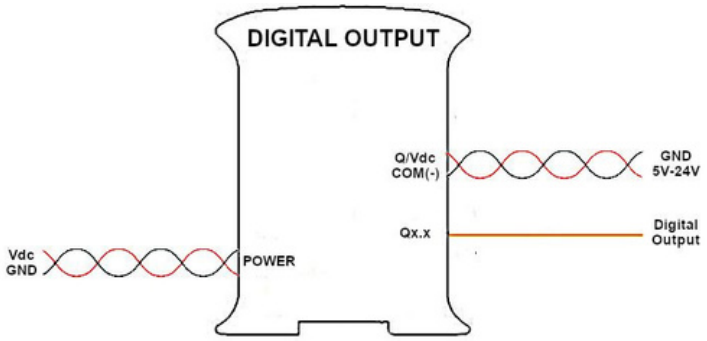
A0.0 Desactivado - Q0.0 Activado



A0.0 Activado - Q0.0 Desactivado

Hardware

Todas las salidas digitales están optoaisladas (utilizan las mismas GND que el PLC). La imagen siguiente muestra cómo conectar una salida digital a su PLC:



Salida de 5-24 Vdc

Digital outputs of Raspberry PLC

Software

How to work with Bash Scripts

Raspberry Pi PLC has default bash scripts for working with the inputs. All the inputs and outputs scripts must be executed from the correct path. It depends on the shield type of the I/O executed. In function of the shield of the I/O that you need to activate, you must execute the scripts from a specific path:

- Analog/Digital Shields

```
> cd /home/pi/test/analog
```

- Relay Shield

```
> cd /home/pi/test/relay
```

The set function will initialize the pin. You will provide the pin with which you are going to work and the value that will be set. For the digital option, a logical 1 will turn on the pin while a 0 will stop it.
By default, if not value option is provided, it will be initialized as a 1 for the Digital outputs. If any other options are chosen, an error code will warn you. In order to call the function, you will do the following:

```
> ./set-digital-output <output> <value>
```

There are some pins that both can work as digital or analog. In this case, if you have used these pins before in either digital or analogic and you want to switch their mode, you must call the set function, providing a stop to the value option; otherwise, there will be a system error. If a reboot is done, it is not necessary to do it.

The pins which can operate with both Analog/Digital configurations are:

Q0.5	Q1.5	Q2.5
Q0.6	Q1.6	Q2.6
Q0.7	Q1.7	Q2.7

Ejemplo:

```
> ./set-digital-output Q0.5 1
> ./set-digital-output Q0.5 0
> ./set-digital-output Q0.5 stop
> ./set-analog-output A0.5 50
```

Digital outputs of Raspberry PLC

How to work with Python

The bash commands are the base for easily working with the **Raspberry Pi based PLC**. In order to work with python files, if you want to interact with the IOs of the PLC, you will have to call these scripts.

To edit the files, you will be working with the Nano editor included by default and Python3.

```
nano digital_inputs.py
```

Python allows you to execute a shell command that is stored in a string using the os.system() function. In order to work with it, you will need to import its library at the beginning of the file. In addition, you will need to include the time library to summon a 2 seconds delay.

```
import os
import time
```

In this example program, you will be blinking some of the industrial **Raspberry Pi PLC** digital LEDs. In order to do it, you will implement a loop that will constantly open up and shut them off in an interval of 2 seconds.

```
import os
import time
os.system("echo Start")
while True:
          try:
                os.system("sudo ./set-digital-output Q0.0 1")
                os.system("sudo ./set-digital-output Q0.1 1")
                os.system("sudo ./set-digital-output Q0.2 1")
                os.system("sudo ./set-digital-output Q0.3 1")
                os.system("sudo ./set-digital-output Q0.4 1")
                os.system("sudo ./set-digital-output Q0.5 1")
                time.sleep(2)
                os.system("sudo ./set-digital-output Q0.0 0")
                os.system("sudo ./set-digital-output Q0.1 0")
                os.system("sudo ./set-digital-output Q0.2 0")
                os.system("sudo ./set-digital-output Q0.3 0")
                os.system("sudo ./set-digital-output Q0.4 0")
                os.system("sudo ./set-digital-output Q0.5 0")
                time.sleep(2)
          except KeyboardInterrupt:
                os.system("sudo ./set-digital-output Q0.0 0")
                os.system("sudo ./set-digital-output Q0.1 0")
                os.system("sudo ./set-digital-output Q0.2 0")
                os.system("sudo ./set-digital-output Q0.3 0")
                os.system("sudo ./set-digital-output Q0.4 0")
                os.system("sudo ./set-digital-output Q0.5 0")
                os.system("echo End")
                break
```

In order to execute the Python program, you will call it as the follow:

```
> python3 analog_outputs.py
```

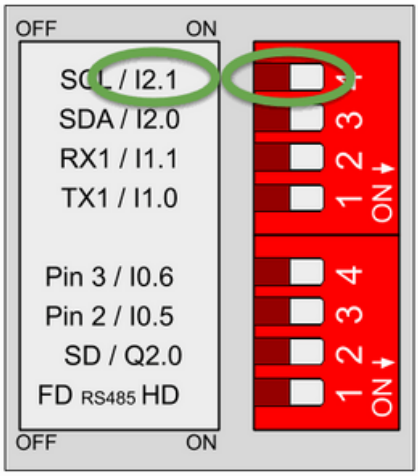
To exit the program, just press ^C.

Basics about analog inputs of an industrial PLC

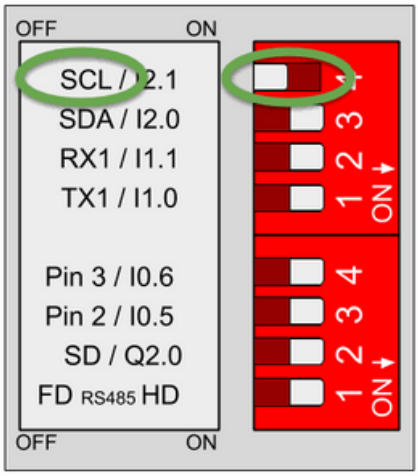
Configuring the switches

Most of the analog inputs are always connected to the internal **Arduino**, but in a few cases, the user can choose between a special peripheral configuration or a **GPIO** by changing the position of the **Dip Switches**.

Each switch can select only one configuration. For example, in this case, we are watching the **GPIOs** configuration of an **M-Duino 57R+**. If we put the switch to the right position (ON) in the upper one, the input I2.1 will be activated and we will be able to work with this as input.
If the switch is in the left position (OFF) we will activate the SCL line which will be used for I2C communication. Keep in mind that each switch has two different configurations: you must select the right (ON) or the left (OFF) option.



I2.1 input enabled - SCL disabled



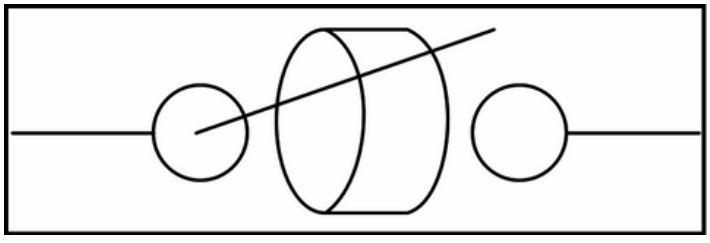
I2.1 input disabled- SCL enabled

Input types

In all of the **Industrial Shields Arduino based PLCs,** analog inputs can work at:

- 0V - 10V analog input

Analog inputs have a special draw in the case of the **PLC**:

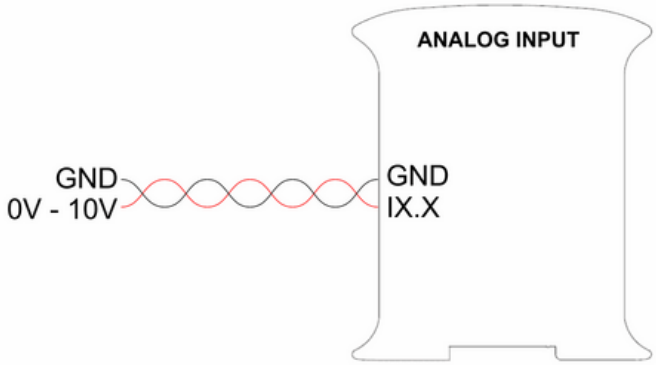


0V - 10Vdc Analog input

Basics about analog inputs of an industrial PLC

Hardware

All the analog inputs are not opto-isolated (they use the same **GNDs** as the **PLC**).



The following image shows how to connect an analog input to the **PLC**:

0V - 10Vdc Analog input

Software

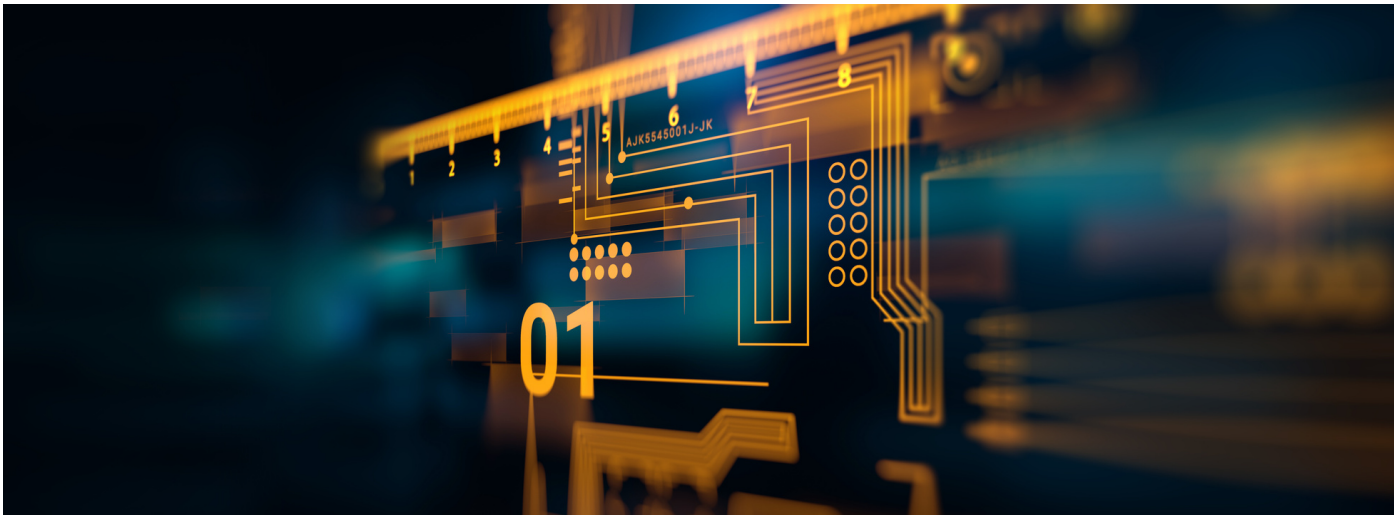
In order to program the analog **GPIOs**, we must keep in mind that we can read the values with the following command:

```
analogRead(GPIO);
```

This function returns a value between 0 and 1023 depending on the applied voltage level to the input (0V it is equal to 0, and 10V is equal to 1023).
GPIO is the name of the input. Imagine we want to know the state of the "I0.12" input, then, we must write this line:

```
analogRead(I0_12);
```

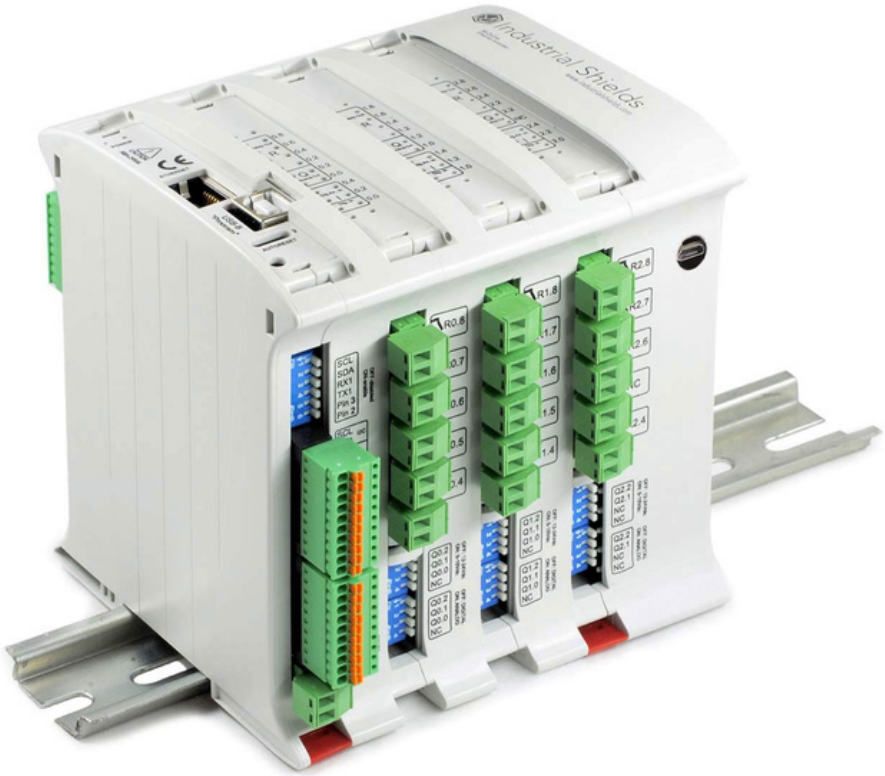
We must know that we do not need to configure the analog inputs as analog. **Industrial Shields'** libraries do all the work for us.



Basics about analog inputs of an industrial PLC

Example
You can see a read analog GPIO example in the following paragraph:

```
// Analog read example
// This example reads the I0_12 and shows via serial the value

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);
}

// Loop function
void loop()
{
    int value = analogRead(I0_12);
    Serial.println(value);
}
```



ARDUINO

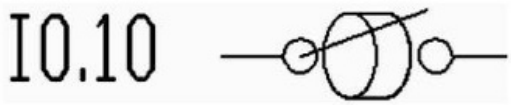
Analog inputs of Raspberry PLC

Input types

On all **Industrial Shields PLCs**, analog inputs can work at:

- 0 Vdc - 10 Vdc input

Each of them has a particular drawing in the case of the PLC:



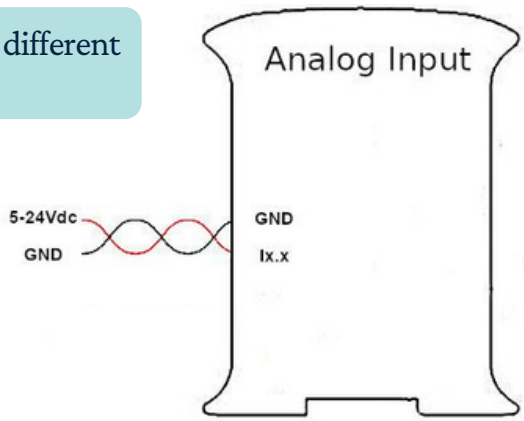
0 - 10 Vdc Analog Input

Hardware

Not all the inputs must be connected in the same way. While non-isolated inputs must be referenced to the same ground as the PLC, isolated inputs can be connected to input grounds, allowing the PLC systems to be isolated. Anyway, the optoisolated input can be connected to the PLC ground as well.

The following pictures show how to connect the different inputs to the PLC for industrial automation:

5 - 24 Vdc Input



Software

How to work with Bash Scripts

Raspberry Pi industrial PLC has default bash scripts for working with the inputs. All the inputs and outputs scripts must be executed from the correct path. It depends on the shield type of the I/O executed.

Depending on the shield of the I/O that you need to activate, you must execute the scripts from a specific path:

- Analog/Digital Shields

```
> cd /home/pi/test/analog
```

- Relay Shield

```
> cd /home/pi/test/relay
```

Analog inputs of Raspberry PLC

The get-digital-input script will show the value of the selected input pin. Only the pin we are going to work with will be provided. The return value will be in the range of 0 to 4096 (10 Vdc).
In order to call the function, we will do the following:

```
> ./get-analog-input <input>
```

Example for the I0.0 input returning a True value

```
> ./get-analog-input I0.12
4096
```

How to work with Python

The bash commands are the base for working easily with the industrial Raspberry PLC. In order to work with python files, if you want to interact with the IOs of the PLC, you will have to call these scripts.

To edit the files you will work with the Nano editor included by default and Python3.

```
nano digital_inputs.py
```

Python allows you to execute a shell command that is stored in a string using the subprocess library. In order to work with it, you will have to import it at the start of the file.

```
import subprocess
import time

def str2dec(string):
   return (string[0:-1])

def adc(value):
   return (10*int(str2dec(value)))/4096

if __name__ == "__main__":
   print("Start")
   while True:
      try:
         x = subprocess.run(["./get-analog-input","I0.12"],
         stdout=subprocess.PIPE, text=True):
         print(adc(x.stdout))
         time.sleep(1)
      except KeyboardInterrupt:
         print("\nExit")
         break
```

In order to execute the Python program, you will call it as the follow:

```
 > python3 analog_outputs.py
```

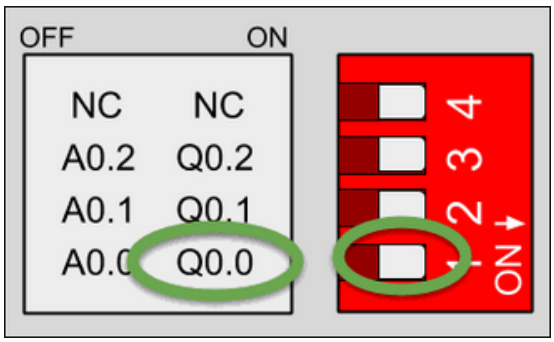
To exit the program, just press ^C.

Basics about analog outputs of an industrial PLC

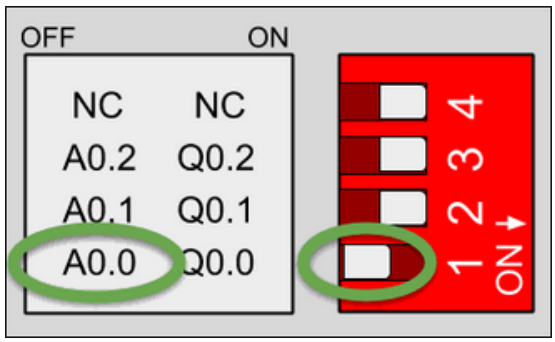
Configuring the switches

Many of the analog outputs are always connected to the internal Arduino, but in some cases, the user can choose between a special peripheral configuration or a GPIO by changing the position of the Dip switches.

Each switch can select only one configuration. For example, in this case you can see the configuration of a **GPIO** on an **M-Duino 21+**. If you put the switch in the bottom right corner (ON), the output Q0.0 will be activated and you will be able to work this digitally. If you switch to the left (OFF) position, you will activate the output as analogue. Note that each switch has two settings: you must select either the right (ON) or the left (OFF) option.



Q0.0 enabled - A0.0 disabled



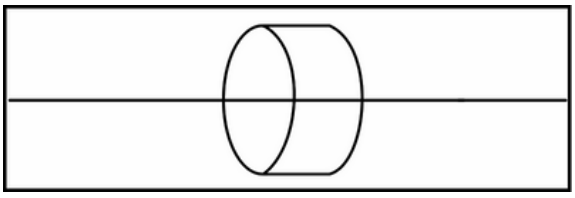
Q0.0 disabled - A0.0 enabled

Types of outputs

On all **Industrial Shields Arduino based PLCs**, analog outputs can operate on:

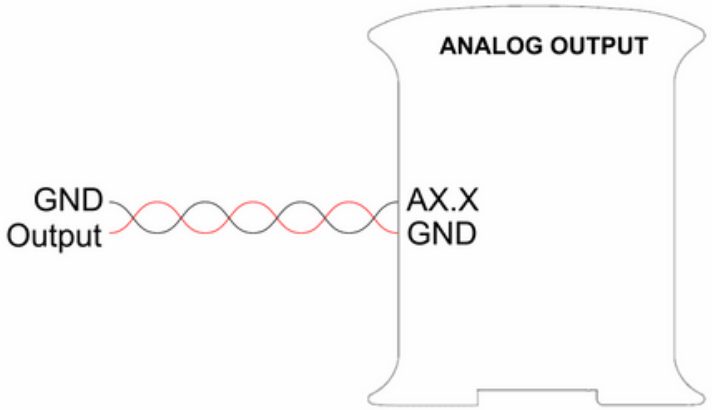
- Analog output 0V - 10V

Analog outputs have a special pattern on this type of PLC:



Analog output print

Analog output 0Vdc -10Vdc



Hardware

The following picture shows how to connect the analog output to the **PLC**:

Basic functions of the analog outputs of an industrial PLC

Software

To program the analog outputs, you must take into account that you can write the values with the following command:

```
analogWrite(GPIO,value);
```

This function sets the value of the analog output "A0.0" to 255 (i.e. 10V):

```
analogWrite(A0_0,255);
```

Example
You can see an analog **GPIO** written in the next paragraph:

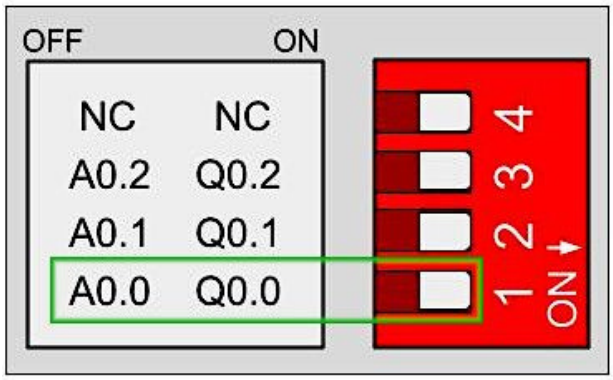
```
// Analog write example
// This example writes the A0_0 and shows via serial the value

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);
}

// Loop function
void loop()
{
    Serial.println("Value: 0");
    analogWrite(A0_0, 0);
    delay(1000);
    Serial.println("Value: 100");
    analogWrite(A0_0,100);
    delay(1000);
    Serial.println("Value: 255");
    analogWrite(A0_0,255);
    delay(1000);
}
```

Raspberry Industrial PLC analog outputs

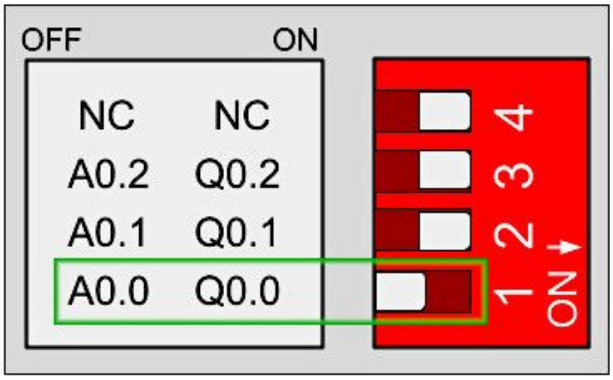
Switch configuration

Most of the digital outputs are always connected to the internal **Raspberry Pi**, but in some cases, users can choose between a special peripheral configuration or a GPIO by changing the position of the Dip Switches.

Each switch can select only one configuration. For example, in this case you can see the GPIO configuration of a **Raspberry Pi based PLC 21+**. If you set the switch to the right (ON) position at the bottom, it will activate the Q0.0 output and you will be able to work as digital. If the switch is in the left (OFF) position, it will activate the output as analog.
Note that each switch has two different settings: you must select either the right (ON) or the left (OFF) option.



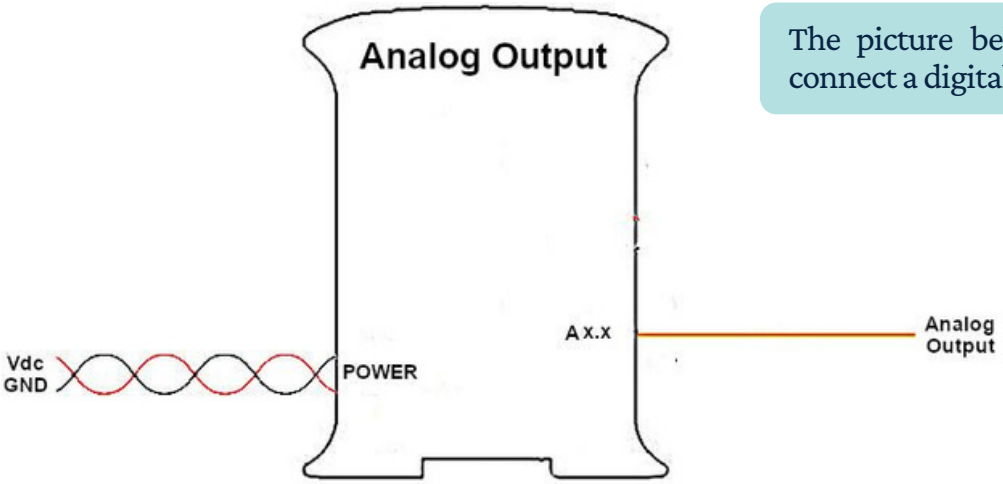
A0.0 Disabled - Q0.0 Enabled



A0.0 Enabled - Q0.0 Disabled

Hardware

The picture below shows how to connect a digital output to the PLC:



Raspberry Industrial PLC analog outputs

Software

Raspberry Pi

How to work with Bash Scripts

Raspberry Pi PLC has default bash scripts to work with the inputs. All input and output scripts must be run from the correct path. It depends on the type of I/O shield executed. Depending on the area of the I/O you need to activate, you must run the scripts from a specific path:

- Analog/Digital Zone

```
> cd /home/pi/test/analog
```

- Relay Zone

```
> cd /home/pi/test/relay
```

The set function will initialise the pin. You will provide the pin you are going to work with and the value to be set. For the analog option, the value will work in a range from 0 to 4095, this being the maximum possible value (10 Vdc).
By default, if no value option is provided, it will be initialised as 50% for the analogue outputs. If any other option is chosen, an error code will warn you. To call the function, you must do the following:

```
> ./set-digital-output <output> <value>
```

By default, if no value option is provided, it will be initialised as 50% for analog outputs. If you choose any other option, an error code will warn you. To call the function, you must do the following:
The pins that can work with both analogue/digital configurations are:

Q0.5	Q1.5	Q2.5
Q0.6	Q1.6	Q2.6
Q0.7	Q1.7	Q2.7

Example:

```
> ./set-analog-output A0.5 2048
> ./set-analog-output A0.5 4096
> ./set-analog-output A0.5 0
> ./set-analog-output A0.5 stop
> ./set-digital-output Q0.5 1
```

How to work with Python

The bash commands are the basis for working easily with the **Raspberry Pi industrial PLC.**
To work with python files, if you want to interact with the PLC IOs, you will have to call these scripts.

Raspberry Industrial PLC analog outputs

To edit the files, we work with the default Nano editor and Python3.

```
> nano analog_outputs.py
```

Python allows you to execute a shell command that is stored in a string using the **os.system()** function. In order to work with it, you will have to import its library at the beginning of the file. In addition, you will include the timing library to invoke a 2 second delay.

```
import os
import time
```

In this example program, you will change the values of the A0.5 output of the **Raspberry Pi industrial PLC**. To do this, you will implement a loop that will increment the output value by 25% every 2 seconds and reset it after reaching 100%.

```
import os
import time
os.system("echo Start")
while True:
    try:
        os.system("sudo ./set-analog-output A0.5 0")
                time.sleep(2)
        os.system("sudo ./set-analog-output A0.5 1024")
                time.sleep(2)
        os.system("sudo ./set-analog-output A0.5 2048")
                time.sleep(2)
        os.system("sudo ./set-analog-output A0.5 3072")
                time.sleep(2)
        os.system("sudo ./set-analog-output A0.5 4095")
                time.sleep(2)

    except KeyboardInterrupt:
        os.system("sudo ./set-analog-output A0.5 0")
        os.system("echo End")
        break
```

To run the Python program, you will call it as follows:

```
> python3 analog_outputs.py
```

To exit the programme, simply press ^C.

Raspberry Pi

Use of interrupt inputs on industrial Arduino boards

Introduction

Interrupts, as their name implies, are a method of stopping the process being executed by the processor in order to execute a smaller subroutine. This method has a lot of real-world application and is an important part of automation.

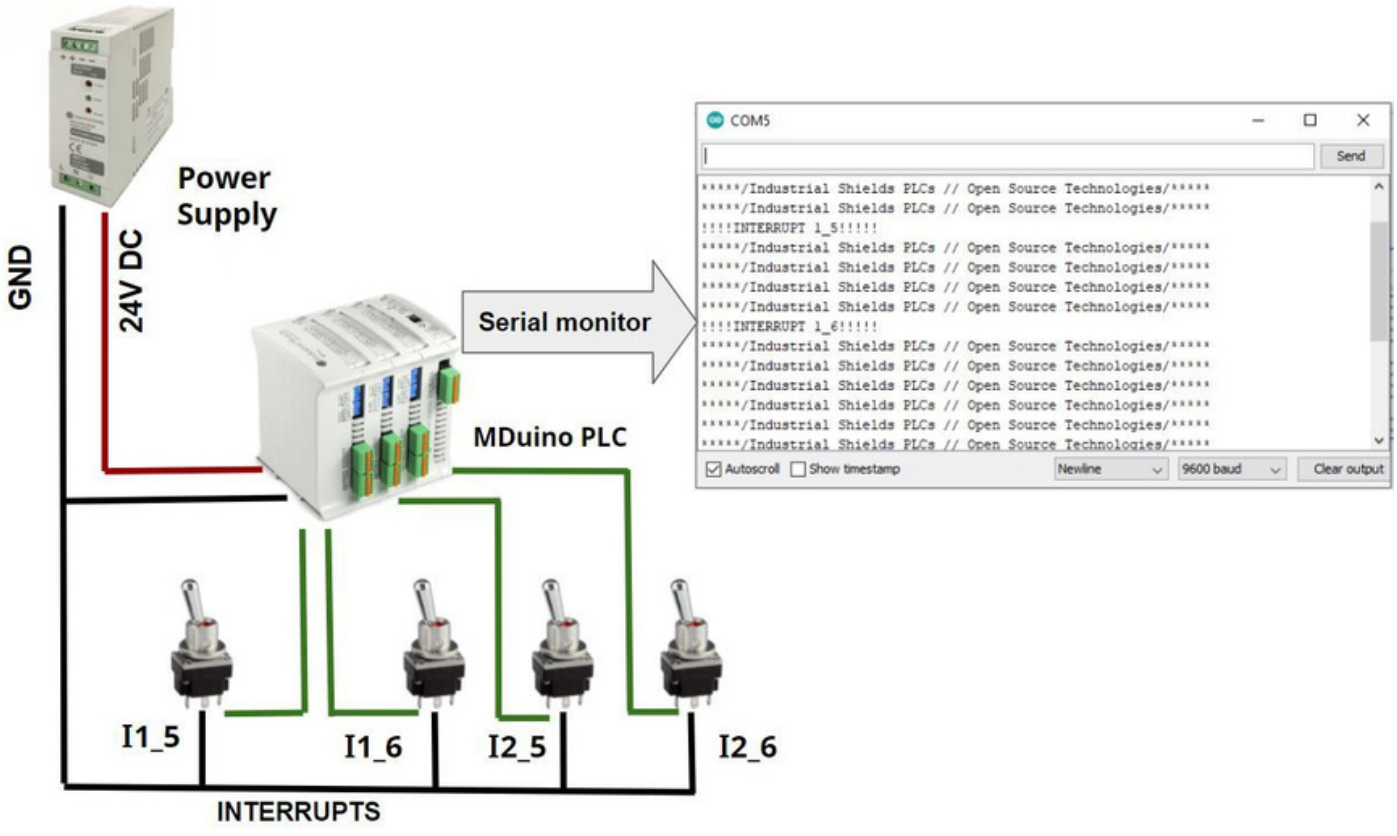
> These interrupts can be generated externally with the help of hardware such as a switch or a sensor, or they can be generated by software when a particular condition is met or a set of instructions has been executed.

In this reading, you will learn how to use **hardware interrupts on an Arduino PLC**. This is an overview with example code to demonstrate the capabilities with respect to interrupt handling and execution of the boards that support this feature.

For ease of understanding and demonstration, we will loop a text string on the serial monitor and interrupt it with some hardware buttons.

Hardware

In this guide the industrial controller to be used is an **M-DUINO PLC Arduino Ethernet 58 IOs Analog/Digital PLUS**. If you are using a different board, make sure the interrupt inputs are enabled and check the DIP switch status.



Use of interrupt inputs on industrial Arduino boards

Syntax

To initialise the interrupt input on the board, you must use the attachInterrupt() function with the following parameters:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

- **digitalPinToInterrupt(pin) -->** Used to initialise the given pin and assign it as the interrupt.
- **pin -->** In this case, you will not use the Arduino pin number, but the ones written on their respective boards. For example, "I1_5" for the board you are using.
- **ISR --> ISR -->** This stands for Interrupt Service Routine, it is a function that is called when the interrupt is triggered. This must not take any parameters and returns nothing, however, it can pass global variables.
- **mode -->** Specifies when an interrupt is to be triggered.
  - **LOW** to activate the interrupt whenever the pin is low.
  - **CHANGE** to trigger the interrupt when the pin changes value.
  - **RISING** to trigger when the pin goes from low to high.
  - **FALLING** for when the pin goes from high to low.

Due, Zero and MKR1000 plates also allow:

  - **HIGH** to trigger the interrupt whenever the pin is high.

Use of interrupt inputs on industrial Arduino boards

Code

This code shows how to operate interrupts **I1_5, I1_6, I2_5 and I2_6 by applying different functions in each case.**

```
// Interrupt Example. Industrial Shields PLCs.
// Board used M-DUINO PLC Arduino Ethernet 58 IOs Analog/Digital PLUS
int val1,val2,val3,val4 = 0;
///////Setting up the board and the pins
void setup() {
  Serial.begin(9600L);

  //Initalizing interrupt I1_5
  attachInterrupt(digitalPinToInterrupt(I1_5), isrI1_5, LOW);

  //Initalizing interrupt I1_6
  attachInterrupt(digitalPinToInterrupt(I1_6), isrI1_6, CHANGE);

  //Initalizing interrupt I2_6
    attachInterrupt(digitalPinToInterrupt(I2_5), isrI2_5, RISING);

  //Initalizing interrupt I2_6
  attachInterrupt(digitalPinToInterrupt(I2_6), isrI2_6, FALLING);
}
/////// Printing a String every seconds in a loop continuously to interrupt
void loop() {
   Serial.println("*****/Industrial   Shields   PLCs   //   Open   Source
Technologies/*****");
 delay(1000);
}
///////Interrupt Service Routines
void isrI1_5(){
  Serial.println("!!!!INTERRUPT 1_5!!!!!"); //ISR for I1_5
  }
void isrI1_6 (){
  Serial.println("!!!!INTERRUPT 1_6!!!!!"); //ISR for I1_6
  }
void isrI2_5(){
  Serial.println("!!!!INTERRUPT 2_5!!!!!"); //ISR for I2_5
  }
void isrI2_6 (){
  Serial.println("!!!!INTERRUPT 2_6!!!!!"); //ISR for I2_6
  }
//////////////End//////////////
```

How to program Raspberry PLC interrupt inputs in Python

How to work with Python

Code Example

For this example, you need to import the libraries that you can see at the beginning of the code, taking into account that "signal", "sys" and "RPi.gpio" are essential to work with interrupt inputs in Python with a **Raspberry Pi PLC**. The INT_GPIO must be the GPIO of the **Raspberry Pi** that you are going to configure as an interrupt input, in this case 13, which is the INT.

If you do not know the mapping between the Raspberry Pi GPIOs and the I/O of your PLC, you can take a look at these tables, also included in the Datasheet and User Guide.

Equivalence table

Pinout

Raspberry Pinout	PLC Pinout
NC	-
GPIO2	SDA
GPIO3	SCL
GPIO4	INT21
GND	-
GPIO17	INT30
GPIO27	INT20
GPIO22	IRQ SPI 485
NC	-
GPIO10	MOSI 0
GPIO9	MISO 0
GPIO11	SCLK 0
GND	-
GPIO 0	-
GPIO5	IRQ SPI CAN
GPIO6	IRQ SPI ETH
GPIO13	INT10
GPIO19	MISO 1
GPIO26	FAN CONTROL
GND	-

Digital I/Os

Digital Inputs			
PLC Pinout	Chip ADDR	Chip INDEX	GPIO
Zone A			
I0.0	ADDR = 0x21	5	-
I0.1	ADDR = 0x21	3	-
I0.2	ADDR = 0x21	2	-
I0.3	ADDR = 0x21	1	-
I0.4	ADDR = 0x21	0	-
I0.5	-	-	GPIO = 13
I0.6	-	-	GPIO = 12
Zone B			
I1.0	ADDR = 0x20	2	-
I1.1	ADDR = 0x20	1	-
I1.2	ADDR = 0x20	0	-
I1.3	ADDR = 0x21	7	-
I1.4	ADDR = 0x21	6	-
I1.5	-	-	GPIO = 27
I1.6	-	-	GPIO = 4
Zone C			
I2.0	ADDR = 0x20	6	-
I2.1	ADDR = 0x20	5	-
I2.2	ADDR = 0x20	7	-
I2.3	ADDR = 0x20	4	-
I2.4	ADDR = 0x20	3	-
I2.5	-	-	GPIO = 17
I2.6	-	-	GPIO = 16

You should also know that a signal is a software interrupt delivered to a process. The operating system uses signals to report exceptional situations to a running program.

The first function is **signal_handler**, a function that has to be called if an event that triggers a signal is anticipated, and the operating system can be told to execute it when that particular type of signal arrives.
In this case, this handler does a **GPIO.cleanup()** and a **sys.exit(0)** when it detects a CTRL+C (command that sends a SIGINT).

The second function is called **int_activated_callback** and, inside it, you can put the code you want to be executed when the interrupt is activated.

Finally, there is the **GPIO.setmode**, onfiguring it following the layout of the BCM GPIO; el GPIO.setup, configuring it with the number of the GPIO interrupt inputs, whether the trigger edge is going to be **FALLING** o el **RISING**; the trigger callback and the bouncing time (which is the period that no interrupt is going to be triggered to avoid signal bouncing). The signal.signal is the function to trigger the signal_handler, explained above.

The last infinite loop is simply to test that you can be running other processes in your code while the interrupt is ready to be triggered.

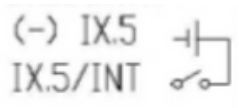
So, if you run this code, it will perform indefinite "Work" prints until the interrupt is triggered.

When a falling edge is detected on the signal, the previous prints will stop, the interrupt will trigger and you will see the "INT triggered" print once, then the "Job" prints will continue until another interrupt triggers (always respecting the 1000ms bounce time).

```
import signal
import sys
import time

import RPi.gpio as gpio
import BUTTON.GPIO as gpio

INT_GPIO = 13

def signal_handler(sif, frame):
    GPIO.cleanup()
    sys.exit(0)

def int_activated_callback(channel):
    print("INT activated")

if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
            GPIO.setup(INT_GPIO,   GPIO.FALLING,   callback=int_activated_callback,
bouncetime=1000)
    signal.signal(signal.SIGINT, signal_handler)
    while 1:
        print ("Work")
        time.sleep(0.1)
```

The interrupt has to be triggered by an activation of the input signal, either by a rising edge or a falling edge. To test this, you can connect the GND of the sensor to the optoisolated GND of the input you are going to use ((-)IX.5) and the output of the sensor to the interrupt input signal (IX.5/INT).

When the digital sensor is activated, you will see the activation of the interrupt as well. Here is an example of one of the PLC interrupt inputs, with the GND pin and the SIGNAL pin:

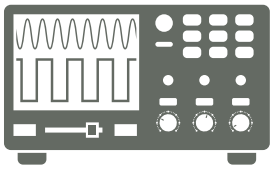


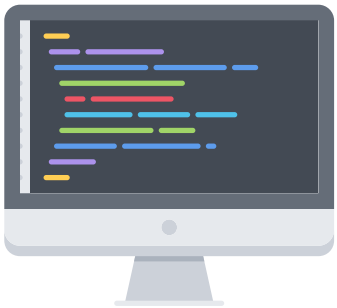
How to work with PWM outputs on the Raspberry industrial PLC

Introduction

Raspberry Pi based PLC family devices have a defined number of digital outputs. All of them can be programmed as **PWM** outputs, if necessary.

As we know, **PWM (Pulse Width Modulation)** is a type of voltage signal that is used to send information or to modify the amount of power sent by each load.





Explanation and use of the Bash Script

First of all, the Bash script you have to run to manage the PWM outputs is called "**set-analog-output**" located in the path "**/home/pi/test/analog/**". You must make sure that the output you want to configure as PWM is not configured as analogue or digital, so, to make sure, you can run the "stop" function to disable the corresponding input as digital or analogue (if it has been previously modified). To work as PWM, the DIP switch of the output in question must be in the "ON" position.

```
./set-analog-output A0.5 stop
```

Or:

```
./set-digital-output Q0.5 stop
```

To execute the script, the "**set-analog-output**" script must be called, but with a digital output and the pulse width as parameters. The pulse width is the high time period of the duty cycle and ranges from 0 to 4095 (12 bits).

For example, if you want a high time period of 25%, set 1024 and, if you want a high time period of 100%, set 4095.

```
./set-analog-output Q0.5 4095
```

Note: Refer to the User's Guide for PWM-compatible outputs.

How to work with PWM outputs on the Raspberry industrial PLC

The PWM parameters of the script do not have to be modified to ensure correct behaviour. Here you can see the script:


```
#!/bin/bash

# PWM period in nanoseconds
PERIOD="2000000"

case ${1} in
    A0.5) ADDR=40; INDEX=10 ;;
    A0.6) ADDR=40; INDEX=1 ;;
    A0.7) ADDR=40; INDEX=0 ;;
    A1.5) ADDR=40; INDEX=3 ;;
    A1.6) ADDR=40; INDEX=5 ;;
    A1.7) ADDR=40; INDEX=8 ;;
    A2.5) ADDR=41; INDEX=2 ;;
    A2.6) ADDR=41; INDEX=1 ;;
    A2.7) ADDR=41; INDEX=0 ;;
    Q0.0) ADDR=40; INDEX=15 ;;
    Q0.1) ADDR=40; INDEX=14 ;;
    Q0.2) ADDR=40; INDEX=13 ;;
    Q0.3) ADDR=40; INDEX=12 ;;
    Q0.4) ADDR=40; INDEX=11 ;;
    Q0.5) ADDR=40; INDEX=10 ;;
    Q0.6) ADDR=40; INDEX=1 ;;
    Q0.7) ADDR=40; INDEX=0 ;;
    Q1.0) ADDR=40; INDEX=2 ;;
    Q1.1) ADDR=40; INDEX=9 ;;
    Q1.2) ADDR=40; INDEX=6 ;;
    Q1.3) ADDR=40; INDEX=4 ;;
    Q1.4) ADDR=40; INDEX=7 ;;
    Q1.5) ADDR=40; INDEX=3 ;;
    Q1.6) ADDR=40; INDEX=5 ;;
    Q1.7) ADDR=40; INDEX=8 ;;
    Q2.0) ADDR=41; INDEX=6 ;;
    Q2.1) ADDR=41; INDEX=7 ;;
    Q2.2) ADDR=41; INDEX=5 ;;
    Q2.3) ADDR=41; INDEX=4 ;;
    Q2.4) ADDR=41; INDEX=3 ;;
    Q2.5) ADDR=41; INDEX=2 ;;
    Q2.6) ADDR=41; INDEX=1 ;;
    Q2.7) ADDR=41; INDEX=0 ;;
    *)
        echo "Output not defined" >&2
        exit 1
        ;;
esac

VALUE="${2:-50}"

if [ -z "${PWM}" ]; then
    CHIP_BASE_DIR="/sys/bus/i2c/devices/1-00${ADDR}/pwm"
    CHIP_NAME="$(ls ${CHIP_BASE_DIR})"
    CHIP_DIR="${CHIP_BASE_DIR}/${CHIP_NAME}"
    CHIP="${CHIP_NAME#pwmchip}"

    PWM="${INDEX}"
fi

if [ "${VALUE}" = "stop" ]; then
    echo "${PWM}" > ${CHIP_DIR}/unexport
    exit 0
fi

if [ ! -d ${CHIP_DIR}/pwm${PWM} ]; then
    echo "${PWM}" > ${CHIP_DIR}/export
fi

echo "${PERIOD}" > ${CHIP_DIR}/pwm${PWM}/period

DUTY_CYCLE="$((${2} * ${PERIOD} / 4095))"
echo "${DUTY_CYCLE}" > ${CHIP_DIR}/pwm${PWM}/duty_cycle
```

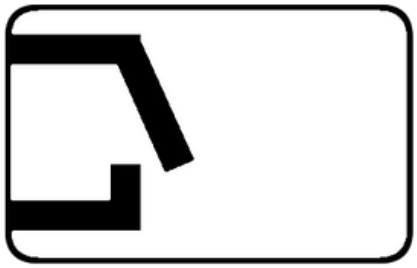
Basics of the internal relays of an industrial PLC

Introduction

Relay characteristics
There is only one type of relay in our PLCs.
These relays have the following characteristics:

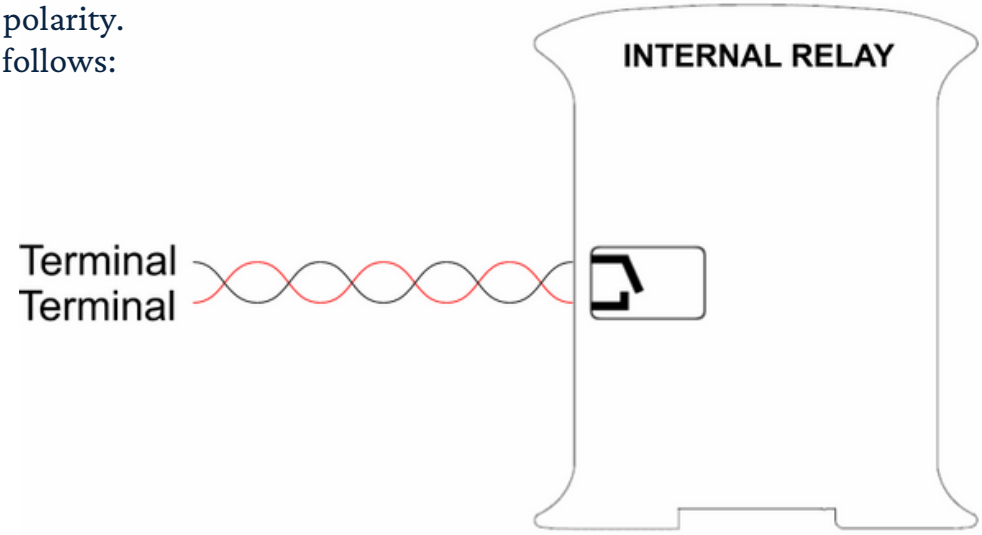
- Up to 5A working up to 250Vac
- Up to 3A working up to 30Vdc

The following illustration shows how to identify GPIO relays:



Hardware

The internal relays have no polarity.
They must be connected as follows:



Software

To program the internal relays, you should note that you can write the values using the following command:

```
digitalWrite(relay,value);
```

"Relay" has to be the reference of the target relay. The Ardbox family has the references as "R1", and for example, the M-Duino family has the reference as "R0.1" with the relay. You must write "HIGH" or "LOW" in the "value" parameter. **"HIGH" is equivalent to relay closed and "LOW" is equivalent to relay open.**

```
digitalWrite(R1,HIGH);    // Ardbox family
digitalWrite(R0_3,LOW);   // M-Duino family
```

Examples

You can see how to handle an internal relay in the Ardbox family in the example below:

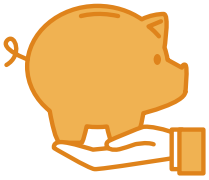
```
// Internal relay example in Ardbox family
// This example writes the R1 and shows via serial the state

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);
}

// Loop function
void loop()
{
    Serial.println("Opening");
    digitalWrite(R1, HIGH);
    delay(1000);
    Serial.println("Closing");
    digitalWrite(R1, LOW);
    delay(1000);
}
```

The following example shows how to operate an internal relay with the M-Duino family:

```
// Internal relay example in M-Duino family
// This example writes the R0_1 and shows via serial the state

// Setup function
void setup()
{
    // Set the speed of the serial port
    Serial.begin(9600UL);
}

// Loop function
void loop()
{
    Serial.println("Opening");
    digitalWrite(R_01, HIGH);
    delay(1000);
    Serial.println("Closing");
    digitalWrite(R_01, LOW);
    delay(1000);
}
```

Benefits of using Arduino, Raspberry Pi or ESP32 controllers

Direct impact on costs



Different platforms can be used to program Arduino-based equipment, most of which are free of charge.

No licence fees!

ARDUINO

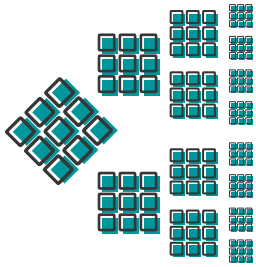
Arduino IDE, the original Arduino and the main software on the market for programming Arduino boards, and therefore Industrial Shields PLCs, is free to download.

https://www.arduino.cc/en/main/software

Quantity and quality of inputs and outputs

The range of industrial PLCs based on Arduino, Raspberry Pi or ESP32, completes a multi-featured range in terms of types and quantities of inputs and outputs. There are countless applications in which these controllers can be used, whether for monitoring, control or automation solutions.

In addition, there is the possibility of installation in master-slave mode, which greatly increases the number of available inputs and outputs.



Standard industrial communications, and more

In industrial environments, standard communications are required to facilitate connection between all types of solutions, hardware or software, in the fastest, cheapest, safest and most reliable way. Industrial Shields PLCs have these requirements, although there may be manufacturers or sectors with specific solutions.

I2C	Serial TTL (UART)	Wi-Fi & BLE	RS485 Half / Full Duplex	LoRa
SPI	Ethernet	GPRS / GSM	RS232	CANBus

... and more

Thanks to the flexibility of Industrial Shields, we have added to our product range specific solutions that our customers have demanded, such as:

LoRa® **Long Range (LoRa)**, a technology ideal for long-distance connections and for IoT networks where sensors that do not have mains power are required.

DALI Digital Addressable Lighting Interface **DALI**, a protocol created to control lighting systems (Digital Addressable Lighting Interface).

Conclusion

The advantages of the different PLC ranges, with the particularities of each CPU, the number of inputs and outputs, or specific accessories such as GPRS, WiFi, LoRa or DALI, ensure a wide range of possibilities. With rare exceptions where the specifications of the solution will be very unique, Industrial Shields PLCs are a great solution for industrial applications in all sectors, whether for automation, monitoring or control.

Need more information?

Contact us and **let's talk**

Our Support Team will help you by phone, email or via the ticket system.

Camí del Grau, 25
Sant Fruitós de Bages 08272 (Barcelona)
Spain

Tel: (+34) 938 760 191

industrialshields@industrialshields.com

https://www.industrialshields.com