

AZ-Delivery

Welcome!

Thank you for purchasing our *AZ-Delivery RFID RC522 Reader with RFID Chip and RFID Card*. On the following pages, we will introduce you to how to use and set-up this handy device.

Have fun!



Az-Delivery

RFID means radio-frequency identification. RFID uses electromagnetic fields to transfer data over short distances. RFID tags are used in many industries, for example, an RFID tag attached to an automobile during production can be used to track its progress through the assembly line, RFID-tagged pharmaceuticals can be tracked through warehouses, implanting RFID microchips in livestock and pets allows positive identification of animals, etc.

An RFID system is made of two parts: a tag and a reader. RFID tags are embedded with a transmitter and a receiver. The RFID component on the tags have two parts: a microchip that stores and processes information, and an antenna to receive and transmit a signal. The tag contains the specific serial number for one specific object and it does not need to be within direct line-of-sight of the reader to be tracked.

To read the information encoded on a tag, a radio transmitter-receiver called a reader emits a signal to the tag using an antenna. The tag responds with the information written in its memory bank. The reader will then transmit the read results to a microcontroller.

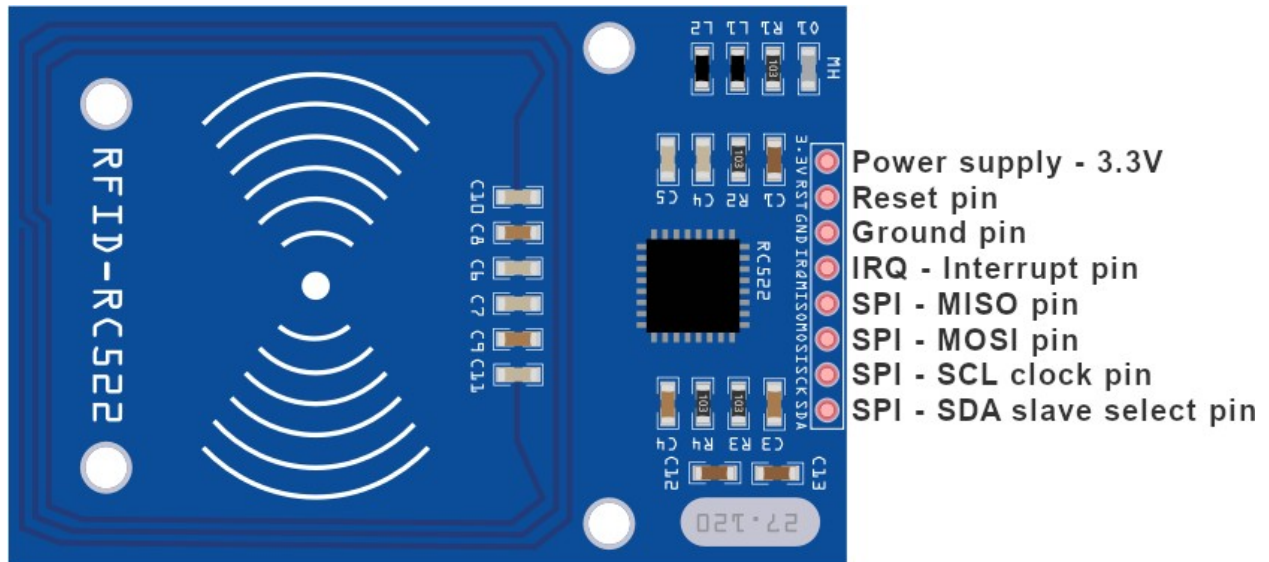


Specifications:

» Chip:	MFRC522
» Operating frequency:	13.56MHz
» Power supply voltage:	3.3V
» Current:	13 - 26mA
» Read Range:	Approx 30mm
» Communication:	SPI interface
» Max Data Transfer Rate:	10Mbit/s
» Dimensions:	40 x 60mm [1.6 x 2.4in]

The reader is a near-field device, not an actual radio. The way the reader works is not by sending and receiving radio signals, but by detecting small change in impedance of the tuned antenna circuit caused by the RFID tag modulating its tuned circuit. Move the RFID tag out from the near field of the antenna and there is no effect to measure. The reader is not really a receiver, but an amplified version of the fluctuations of the average output voltage in the tuned circuit.

The pinout of the reader



Power supply voltage is 3.3V! Do not connect 5V to this pin, or else you could destroy the device!

The reader uses SPI interface to communicate with a microcontroller.

Interrupt IRQ pin is on *HIGH* state all the time, and when the interrupt event happens, it changes its state to *LOW* for a brief period of time. Interrupt event happens when RFID tag is detected in the near-field of the reader.

Az-Delivery

How to set-up Arduino IDE

If you did not install Arduino IDE already, this is how to do it. Go to the link: <https://www.arduino.cc/en/Main/Software> and download installation file for your operating system platform.

Download the Arduino IDE



The screenshot shows the Arduino IDE download page. On the left, there is a teal circle with a white infinity symbol containing a minus and a plus sign. To its right, the text reads: **ARDUINO 1.8.9**, followed by a description of the IDE as open-source software written in Java, based on Processing, and compatible with any Arduino board. It refers to the 'Getting Started' page for installation instructions. On the right side, there is a teal sidebar with links for different operating systems: Windows (installer and ZIP file), Windows app (with a 'Get' button), Mac OS X (10.8 Mountain Lion or newer), and Linux (32 bits, 64 bits, ARM 32 bits, and ARM 64 bits). At the bottom of the sidebar are links for Release Notes, Source Code, and Checksums (sha512).

ARDUINO 1.8.9

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
[Get](#)

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

[Release Notes](#)
[Source Code](#)
[Checksums \(sha512\)](#)

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

Az-Delivery

For Linux, download file with extension *".tar.xz"*, which then you need to extract. When you extract it, go to the extracted directory, and open terminal in that directory. You need to run two *".sh"* scripts, first called *"arduino-linux-setup.sh"*, and second called *"install.sh"*.

To run first script in terminal, run the following command:

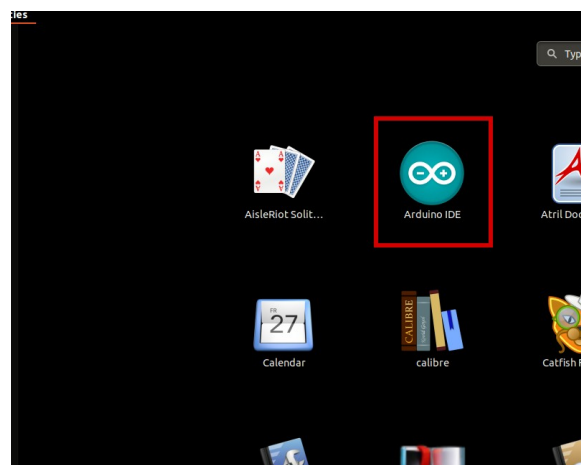
sh arduino-linux-setup.sh user_name

user_name - is the name of super user in the Linux operating system. After this, you will be prompted to provide password for the super user. Wait for a few minutes for script to complete everything.

After installation of the first script, run the second called *"install.sh"* script. In terminal, run the following command:

sh install.sh

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

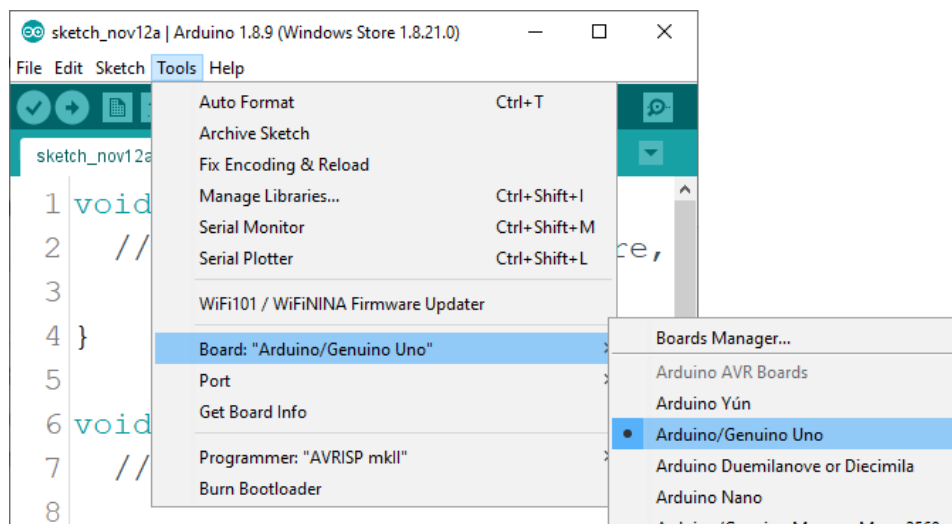


Az-Delivery

Next thing is to check if your PC can detect the Uno board. Open freshly installed *Arduino IDE*, and go to:

Tools > Board > {your board name here}

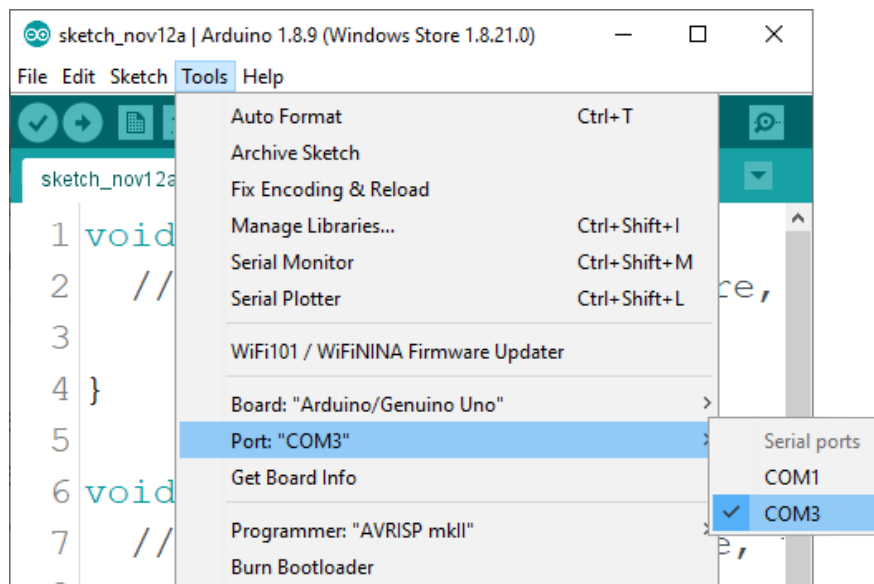
{your board name here} should be the *Arduino/Genuino Uno*, as you can see on the image below:



Az-Delivery

After this you need to select the port on which the Arduino board is connected. Go to: *Tools > Port > {port name goes here}*

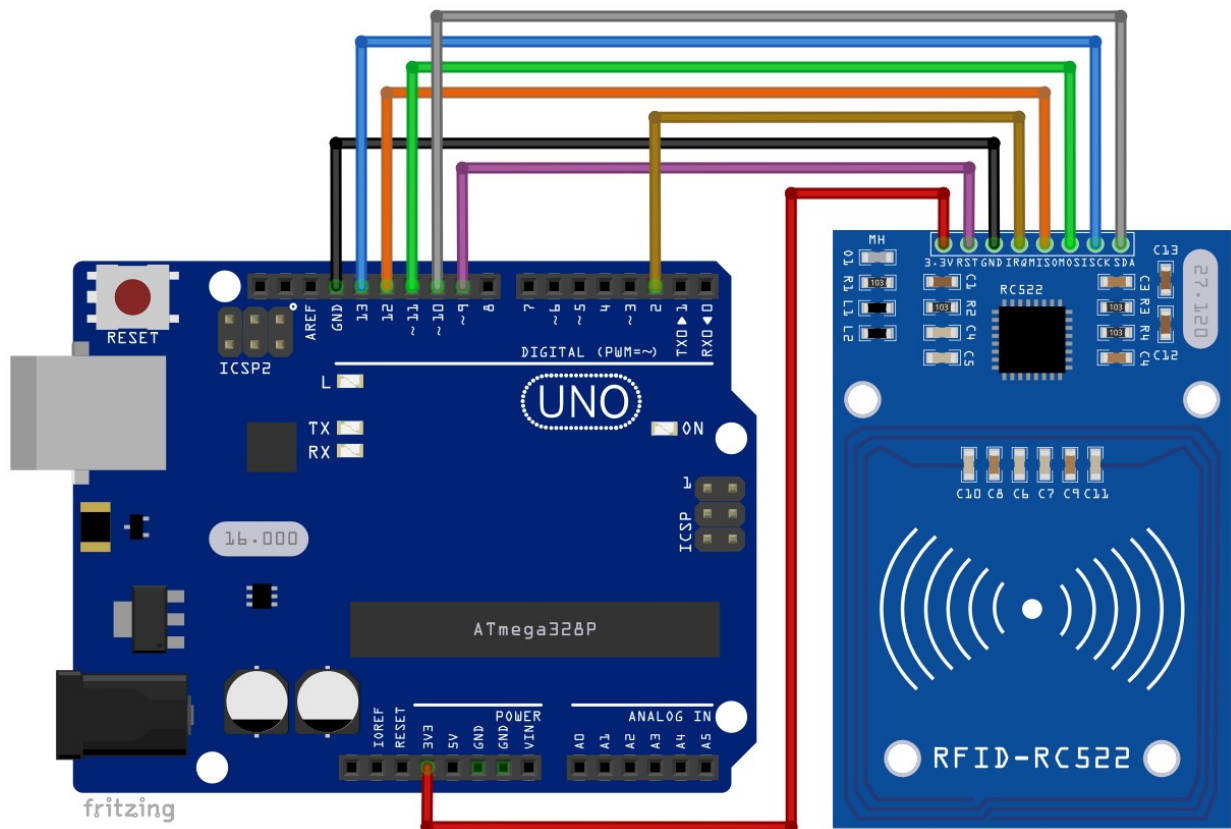
If you connected the Uno board on the usb port, there should be several port names. Because we are using *Arduino IDE* on *Windows*, port names are like on image below:



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

Connecting the reader with Uno

Connect the reader with the Uno as shown on the connection diagram below:



Module pin > Uno pin

3.3V > **3.3V !**

RST > D9

GND > GND

IRQ > D2

MISO > D12

MOSI > D11

SCK > D13

SDA (SS) > D10

Red wire

Purple wire

Black wire

Ochre wire

Orange wire

Green wire

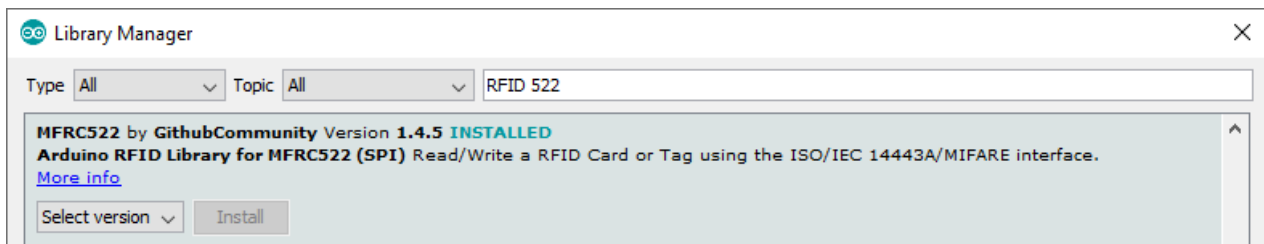
Blue wire

Gray wire

Az-Delivery

The library for Arduino IDE

To use the module with Uno it is recommended to download and install library for it. Open Arduino IDE and go to: *Tools > Manage Libraries*. New window will open, type “*RFID 522*” in the search box, and install library called “*MFRC522*” made by “*GithubCommunity*”, as shown on the image below:



With the library comes many example sketches. We will use one example sketch, called “*MinimalInterrupt*”. To open it, go to:

File > Examples > MFRC522 > MinimalInterrupt

Az-Delivery

Sketch example:

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN 9 // Configurable, see typical pin layout above
#define SS_PIN 10 // Configurable, see typical pin layout above
#define IRQ_PIN 2 // Configurable, depends on hardware
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance
MFRC522::MIFARE_Key key;
volatile bool bNewInt = false;
byte regVal = 0x7F;

void setup() {
  Serial.begin(9600); // Initialize serial communications with the PC
  SPI.begin(); // Init SPI bus
  mfrc522.PCD_Init(); // Init MFRC522 card

  // Read and printout the MFRC522 version (valid values 0x91 & 0x92)
  Serial.print(F("Ver: 0x"));
  byte readReg = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
  Serial.println(readReg, HEX);

  pinMode(IRQ_PIN, INPUT_PULLUP); // Setup the IRQ pin
  // Allow the irq to be propagated to the IRQ pin
  regVal = 0xA0; // Rx IRQ
  mfrc522.PCD_WriteRegister(mfrc522.ComIEnReg, regVal);

  bNewInt = false; // Interrupt flag
  // Activate the interrupt
  attachInterrupt(digitalPinToInterrupt(IRQ_PIN), readCard, FALLING);

  do { // Clear a spurious interrupt at start
    ;
  } while (!bNewInt);
  bNewInt = false;
  Serial.println(F("End setup"));
}
```

Az-Delivery

```
void loop() {
  if (bNewInt) { // New read interrupt
    Serial.print(F("Interrupt. "));
    mfrc522.PICC_ReadCardSerial(); // Read the tag data
    // Show some details of the PICC
    Serial.print(F("Card UID:"));
    dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
    Serial.println();
    clearInt(mfrc522);
    mfrc522.PICC_HaltA();
    bNewInt = false;
  }
  activateRec(mfrc522);
  delay(100);
}

// MFRC522 interrupt serving routine
void readCard() {
  bNewInt = true;
}

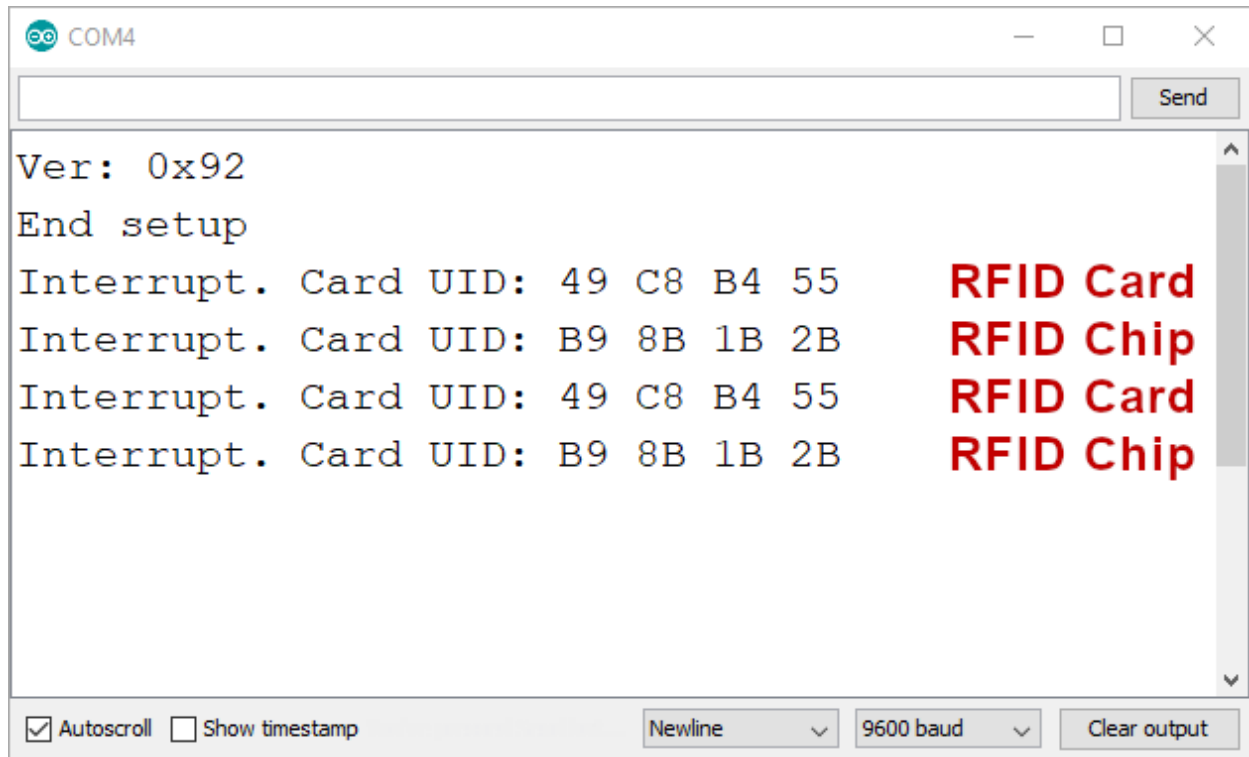
// Helper routine to dump a byte array as hex values to Serial
void dump_byte_array(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
}

// The function sending to the MFRC522 the needed commands to activate the reception
void activateRec(MFRC522 mfrc522) {
  mfrc522.PCD_WriteRegister(mfrc522.FIFODataReg, mfrc522.PICC_CMD_REQA);
  mfrc522.PCD_WriteRegister(mfrc522.CommandReg, mfrc522.PCD_Transceive);
  mfrc522.PCD_WriteRegister(mfrc522.BitFramingReg, 0x87);
}

// The function to clear the pending interrupt bits after ISR
void clearInt(MFRC522 mfrc522) {
  mfrc522.PCD_WriteRegister(mfrc522.ComIrqReg, 0x7F);
}
```

Az-Delivery

When you upload the sketch to the Uno, open Serial Monitor, (*Tools > Serial Monitor*). The output should look like the output on the image below:



When you bring RFID Card or Chip near the reader, an interrupt happens, and the UID number of the Card/Chip is sent to the Serial Monitor.

At the beginning of the sketch two libraries are imported, *SPI* for SPI interface and *MFRC522* for RFID reader. Then three macros are created, one for reset pin number, second for slave select pin number and last for interrupt pin number.

After that "*mfr522*" object, which represents the reader in the software and *key* variable are created. The *key* variable is used to store UID number of the specific card.

Az-Delivery

Then the *bNewInt* variable is created. When interrupt happens, the state of *bNewInt* variable changes to *true*, which indicates that interrupt happened. Then in the code we check for the state of *bNewInt* variable and print the corresponding output to the Serial Monitor (more about this later in the text).

Then we create *regVal* variable, which is used to set-up registers of the chip in the RFID reader.

In the *setup()* function, first we set-up Serial Interface with baud rate of *9600*. After, we set-up SPI interface and initialize the *mfr522* object.

Later we output the MFRC522 reader version on the Serial Monitor. Valid numbers are *0x91* and *0x92*.

Then we set-up pin mode of the interrupt pin to *INPUT* with internal *PULL UP* resistor. After this, we attach the interrupt routine with following line of the code:

```
attachInterrupt(digitalPinToInterrupt(IRQ_PIN), readCard, FALLING)
```

Where the first argument is a built-in function:

digitalPinToInterrupt()

which sets the *IRQ_PIN* (digital I/O pin 2), to listen for interrupt event. The second argument is the function that will be executed when an interrupt happens. And the third argument is the edge of digital signal, which represents the interrupt event itself.

Az-Delivery

For the purpose of this example we are using *FALLING* edge of the digital signal, that comes on the *IRQ_PIN*, as interrupt event. The value of this argument can also be *RISING* or *BOTH*, but we are not using that in our example.

At the end of *setup()* function we clear wrong interrupt readings with following lines of the code:

```
do { ; } while (!bNewInt);  
bNewInt = false;
```

In the *loop()* function there is one *if* statement, where we check the state of *bNewInt* variable. If *bNewInt* state is *true*, then interrupt happened, and we print output to the Serial Monitor. Output contains the message "*Interrupt. Card UID: **number***", where *number* is the UID number of the RFID card that caused interrupt. At the end of the *if* statement, we set the *bNewInt* variable to *false*.

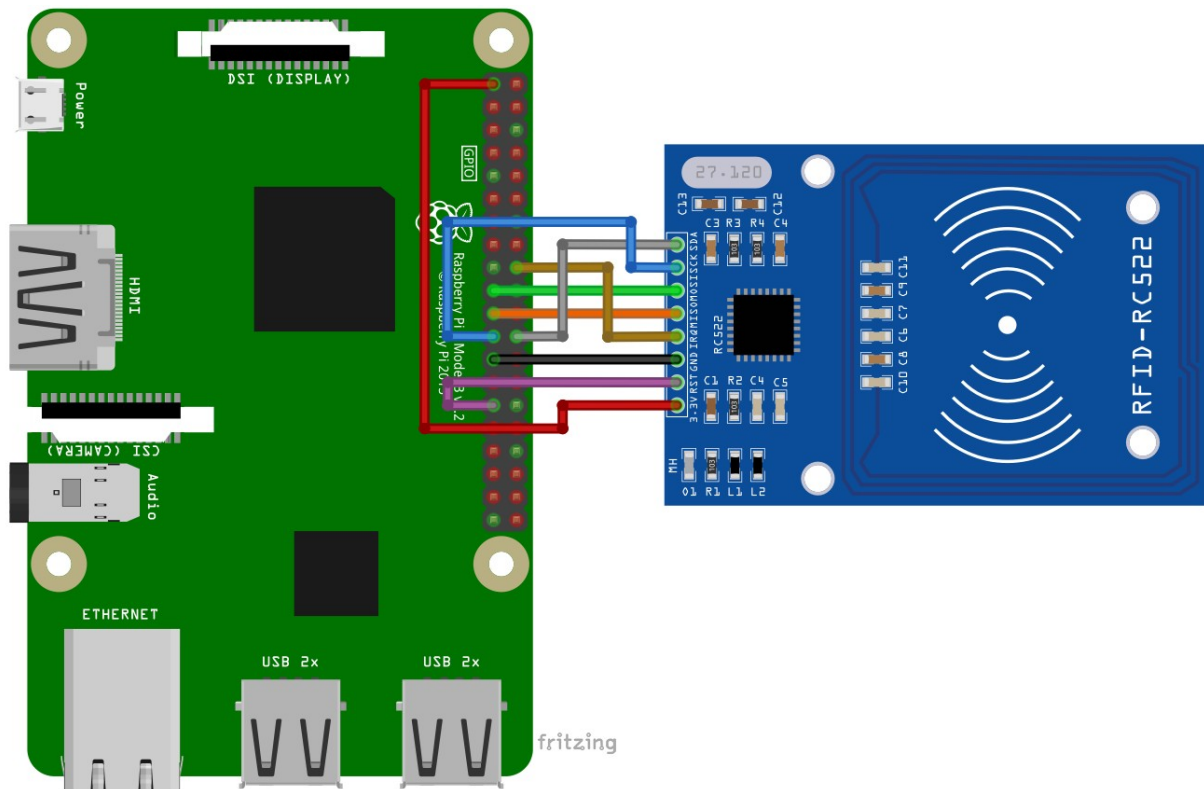
At the end of *loop()* function, we set-up registers of the reader chip again, and wait for 100 milliseconds.

After *loop()* function, we create *readCard()* function, which is executed when the interrupt happens. In the *readCard()* function we only set the state of *bNewInt* variable to *true*.

After this we create three functions which are used to set-up registers of the reader chip, and to communicate with the reader.

Connecting the reader with Raspberry Pi

Connect the reader with Raspberry Pi as shown on the connection diagram below:



Module pin	>	Raspberry Pi pin
------------	---	------------------

SDA (SS)	>	GPIO8 [pin 24]
----------	---	----------------

SCK	>	GPIO11 [pin 23]
-----	---	-----------------

MOSI	>	GPIO10 [pin 19]
------	---	-----------------

MISO	>	GPIO9 [pin 21]
------	---	----------------

IRQ	>	GPIO24 [pin 18]
-----	---	-----------------

GND	>	GND [pin 25]
-----	---	--------------

RST	>	GPIO25 [pin 22]
-----	---	-----------------

3.3V	>	3V3 [pin 1]
------	---	-------------

Gray wire

Blue wire

Green wire

Orange wire

Ochre wire

Black wire

Purple wire

Red wire



The library for Python

In order to use the reader with the Raspberry Pi it is recommended to download a library for Python. The library we are going to use is called “*pi-rc522*”. To install this library, open a terminal and run the following commands, one by one:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo pip3 install pi-rc522
```

Now we are ready to start making Python script.

Az-Delivery

Python script:

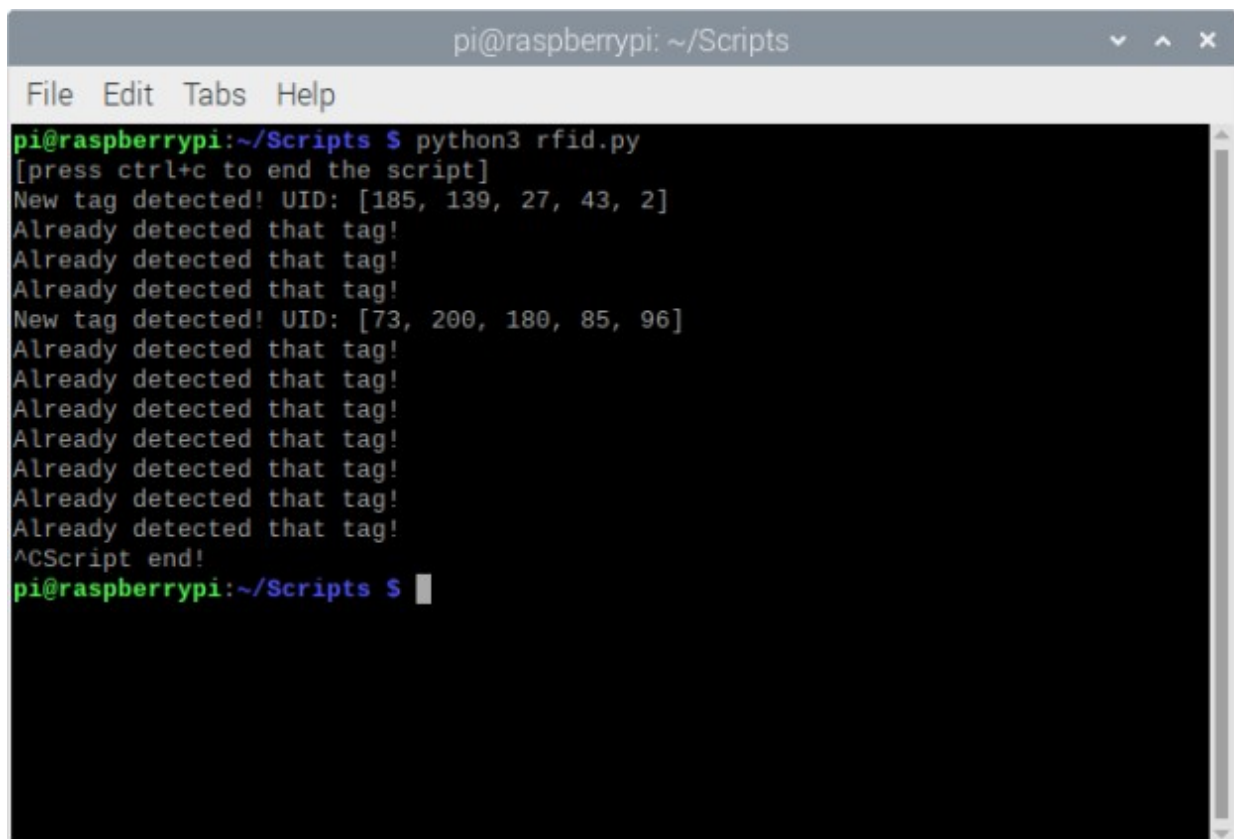
```
from pirc522 import RFID
from time import sleep
reader = RFID()
keys = list()
key_read = True
print('[press ctrl+c to end the script]')
try:
    while True:
        reader.wait_for_tag()
        error, tag_type = reader.request()
        if not error:
            error, uid = reader.anticoll()
            if not error:
                if len(keys) > 0:
                    for key in keys:
                        if key == uid:
                            key_read = False
                            break
                        else:
                            key_read = True
                else:
                    key_read = True
            if key_read:
                keys.append(uid)
                print('New tag detected! UID: {}'.format(uid))
                reader.stop_crypto() # always call this when done working
                key_read = False
            else:
                print('Already detected that tag!')
        sleep(0.1)
# Scavenging work after the end of the program
except KeyboardInterrupt:
    print('Script end!')
finally:
    reader.cleanup() # Calls GPIO cleanup
```

Az-Delivery

Save the script by the name *"rfid.py"* into default script directory. To run the script open terminal in the directory where you saved the script and run the following command:

python3 rfid.py

The output should look like the output on the image below:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 rfid.py
[press ctrl+c to end the script]
New tag detected! UID: [185, 139, 27, 43, 2]
Already detected that tag!
Already detected that tag!
Already detected that tag!
New tag detected! UID: [73, 200, 180, 85, 96]
Already detected that tag!
Already detected that tag!
Already detected that tag!
Already detected that tag!
Already detected that tag!
Already detected that tag!
Already detected that tag!
^CScript end!
pi@raspberrypi:~/Scripts $
```

To end the script press *"CTRL + C"*.

Az-Delivery

At the beginning of the script we import two libraries, one called *pir522* that we just installed and the other is for time.

Then we create reader object, using the following line of the code:

```
reader = RFID( )
```

You can connect the *SDA* pin of the reader to *CE1* (GPIO7 [pin 26]) instead of *CE0* (GPIO8 [pin 24]). When you do that you have to create reader object with the following line of the code:

```
reader = RFID(bus=0, device=1)
```

Also, you can connect *RST* pin to any other free GPIO pin. If you do this you have to create reader object with the following line of the code:

```
reader = RFID(pin_rst=number)
```

Where *number* is the Raspberry Pi board pin number (not GPIO number, but pin number).

Furthermore, you can connect the *IRQ* pin to any other free GPIO pin. If you do this you have to create reader object with the following line of the code:

```
reader = RFID(pin_irq=number)
```

Where *number* is the Raspberry Pi board pin number (not GPIO number, but pin number).

After this we create a list called *keys* where we store new UID keys. Also we create a boolean variable called *key_read* used to check if key is already in the *keys* list.

Az-Delivery

Then we create *try-except-finally* block. We do this in order to detect keyboard interrupt. Keyboard interrupt happens when we press *CTRL + C* on the keyboard. *finally* part of *try-except-finally* block is used to clean the GPIO pins of any used interfaces or defined pin modes after keyboard interrupt.

In the *try* part of *try-except-finally* block we create infinite loop (*while True:*). In this loop first we wait for tag to be detected. When tag is detected, an interrupt will happen on *IRQ_PIN*. When interrupt happens, we can read the tag data.

reader.request() function returns a tuple of two elements. The first element is a boolean value representing error (*True* there is error, *False* there is no error), and the second element represents the type of tag (which we do not use in the script).

Then we check if there is an error. If there are no errors we proceed to read UID of the tag.

reader.anticlaim() function returns a tuple of two elements. The first element is a boolean value representing error (*True* there is error, *False* there is no error), and the second element is the UID number of the tag.

Az-Delivery

Then, again we check if there is an error while reading UID. If there are no errors we proceed with checking if UID number is already in the *keys* list. If the UID number is not in the *keys* list, we add a new number, and print message “*New tag detected! UID: **number***”, where *number* is UID number of the tag. If UID number of the tag is in the *keys* list, we print message “*Already detected that tag!*”.

When we are done with reading data sent by reader, we have to run the following function:

```
reader.stop_crypto()
```

At the end of infinite loop block we call *sleep(0.1)*, or waiting time interval of *100* milliseconds, a waiting time interval between two loops of the infinite loop block.

You've done it!

Now you can use your module for various projects.



Now is the 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 products for Arduino and Raspberry Pi, 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>