LESSON 11

RFID SYSTEMS

OBJECTIVE

In this lesson you will learn to work with radio frequency identification (RFID) systems including how to both read and write RFID tags.

MATERIALS

- Raspberry Pi connected to a monitor, keyboard, mouse, and internet access
- Assembled Circuit from Lesson B-10
- 1 x RFID Reader
- 1 x RFID Tag
- 1 x RFID Card

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- Administrative File Management (Lesson B-1)  
- Using Github to clone libraries (Lesson B-8)
Lesson

In this lesson, you will learn about RFID systems, their components, and some of their applications. You will connect an RFID reader that will write and read RFID cards, as well as create a program that will read an RFID card and perform an action based on the value that's read from the card.

RFID TECHNOLOGY

RFID stands for radio-frequency identification. This system was originally designed for powering a device using radio waves. The RFID receiver was powered by a transmitter sending radio waves at a very specific frequency. The receiver did not contain its own power source, instead it contained special circuitry to convert the incoming radio waves into power that could be used to power up additional circuits in the receiver.

Around 1970, new ideas and patents started to emerge from this existing RFID technology. Proposed use cases involved automated toll collection systems, banking, security, and medical applications, as well as many others.

RFID technology can now be seen everywhere. Automated toll collection is installed on many of highways. Most pets have RFID implants that allow for identification of the animal without any external markings like a collar or tags. Many businesses rely on RFID cards to manage employee access control, instead of handing out physical keys to the building.

HOW RFID WORKS

Modern RFID systems operate using readers and tags. The reader contains a radio transmitter and receiver, commonly referred to as a transceiver, as well as other circuitry to decode signals received from scanned tags. The reader is always broadcasting radio signals and waiting to hear back from a tag.

Tags, or cards, contain an antenna, circuitry to convert radio waves to DC power, a radio transceiver, as well as a tiny amount of storage that can be used to hold data specific to that card. When the card receives a radio signal in a specific frequency, the antenna will harvest energy from that signal, and power up the storage chip and the transceiver.
The card's transceiver will then send the contents of its tag back to the reader, which can happen over different distances based on the type of tag. Passive tags do not contain a power source and their distance is generally limited to anywhere between a few millimeters to a couple of inches, based on the design of the tag.

Active tags contain a battery that can be used to boost the tag's transmission distance. Some of these tags can be read from hundreds of meters away, but the tags are much larger than passive tags due to containing a battery, larger antenna, and more radio amplification circuitry.

Since passive tags don't contain a power source or much circuitry, they can be much smaller. Some versions are flat stickers that can be stuck to products in a store, and some are almost as small as a grain of rice and can be implanted under an animal's skin.

RFID can be used in many applications, but the underlying technology is the same. The reader sends out radio waves, and the tag responds with values stored within the tag.

**RFID FOR ACCESS CONTROL**

When used for building access control, a central computer or controller is used to monitor one or more card readers. This controller will also have the ability to unlock doors electronically. This means that the controller is the heart of the access control system.

The controller is where access control permissions are managed. It stores all tag or keycard information, along with what areas or doors that keycard is permitted to access.

The software inside the controller operates much like some of the if/else programs you have created throughout this course. When a keycard is scanned, it's unique value is used to check a list of permissions stored inside the controller. For example:

User A is issued card number 123. Card 123 is allowed to access door 1 and door 2.

User B is issued card number 456. Card 456 is only allowed to access door 2.

If card 123 is scanned at door 1 or 2 then access will be granted. If card 456 is scanned at door 2 then access will be granted, but when scanned at door 1, access will be denied. This is nice because access to areas can be granted or revoked just by changing the permissions in the main controller, instead of handing out individual metal keys, or changing locks.
SECURITY CONCERNS OF RFID

While more convenient, RFID does have some inherent security risks associated with the technology. With a metal key in your pocket there is little to no risk of anyone duplicating a key while it’s in your pocket. The same cannot be said for RFID cards.

Although encryption is used to secure data storage and communication on some higher end cards, most cards are not very secure. These less secure cards will respond to any reader that requests its data, given that reader is operating in the right frequency range. The tags in your kit operate in the 13.56MHz range, so they will start transmitting their stored data anytime they receive a 13.56MHz signal.

This is where the problem occurs. Your card doesn't know the difference between the RFID reader in your kit and any other 13.56MHz reader it sees. Imagine that you wired your front door to be conveniently controlled by an RFID lock that only opened for a tag that sent out the string Open Sesame. Everything is working great and you no longer have to carry keys, using only the tag to unlock your front door.

Since your tag will respond to any 13.56MHz reader, it’s possible that your tag might get scanned by someone hoping to duplicate your tag in order to access your house. Someone close enough to your tag in line at a coffee shop could send a 13.56MHz signal your way, and your tag would respond to them with Open Sesame. They can now duplicate or clone your card and unlock your front door with their copy of your card.

Your tag won't know that it was scanned and the reader on your front door won't know the difference, since it's only looking for the string Open Sesame to unlock the door. This attack is fairly uncommon, but it is made possible by the convenience of RFID. There are RFID shielding devices available that make this type of attack impossible, but this security does come at the cost of convenience.

These RFID shielding devices completely enclose your card in a material that radio waves cannot penetrate. This shielding can be built directly into wallets, purses, or smaller pouches and can hold a single card or tag. While inside this shielding, your card cannot be read by an attacker, but it also cannot be read by your front door reader either. You would need to remove your card from this shielded pouch in order for the radio waves emitted by the reader to interact with the card, and for the card to send its data back to the reader.

As with everything, there are trade-offs between security, convenience, and cost. The benefits of deploying an RFID system would need to be weighed against the possible cloning and misuse of a card. More secure RFID systems can lessen the risk, if not eliminate, the possibility of these attacks, but this will require a higher-level system with encryption that will come at a higher financial cost.
MFRC522 TAG READER

The RFID reader included in your kit runs on the MFRC522 chipset which can read and write 13.56MHz tags. The Raspberry Pi requires installation of a special library to communicate with the MFRC522, but once installed, communication with the device is very simple over the SPI bus.

These libraries do not support software SPI, so hardware SPI will be used to communicate with the reader. Using hardware SPI means that specific GPIO pins will be used for this communication, and that a setting in the Raspberry Pi will need to be changed in order to enable hardware SPI. In the activities for this lesson, you will install this library and change the SPI setting once the reader is wired up.

Here is a pinout of the MFRC522 board, as well as signal descriptions:

CE0 – Chip Enable
SCLK – Serial Clock
MOSI – Master Out, Slave In
MISO – Master In, Slave Out
IRQ – Interrupt, not used in Raspberry Pi with hardware SPI
Ground – Connect to Ground
Reset – Will trigger a reset of the MFRC522
3.3V – Connect to 3.3V supply
There are many RFID cards on the market, but not all are compatible with this reader. If you plan to purchase additional cards for other projects, ensure they belong to one of the families listed below:

- MIFARE1 S50
- MIFARE1 S70
- MIFARE Ultralight
- MIFARE Pro
- MIFARE DESFire

Cards may also list compatibility with ISO14443A, which means they will also work with this reader. If you plan to use a card not listed above, please do your research to ensure it will be compatible with the MFRC522 reader before purchasing.

**RFID TAGS**

Tags can come in many shapes and sizes and can store different amounts of information. The tags in your kit will hold 1KB (one kilobyte) of information. This is not a ton of room, but it is enough to store enough information to identify the card with a reader. Your program can then respond however you would like to the presence of the card.

The tags and library you will be using in this lesson support the reading of two values:

The card's UID value, or Unique ID, is a unique 12 to 13-digit value that is assigned at the time the card is manufactured. This value can be read but cannot be modified.

The card's text value is a 48-character field that can store any data you like. This could be anything from a single letter, number, or character, up to a complex string of all these combined. This value can be read and written by the MFRC522 reader.
READING TEXT FROM THE TAG

One thing to note about the text value is that it is 48 characters long, and shorter strings written to the card will be formatted to use all 48 of these characters. If you intend to write the value `card` to your tag, only four of the character slots will be used, leaving 44 unused.

These unused values cannot be left empty, so spaces will be used to fill up the rest of the characters available. This means that when trying to write a short value like `'card'`, the actual value written to the card will be:

`'card                                            '`

This won't normally be an issue unless you want your program to act on this value when read back from the card.

This wouldn't normally be a problem unless you want to make an action happen in your program based on this text value:

`'card' does not equal 'card                                            '`

If `'card                                            '` is read from the text field of the tag, and your program is looking for the string `'card'`, then the strings will not match, and the program will not operate as expected.

Leading spaces refer to spaces that occur before the information in your string:

`'        card'`

Trailing spaces refer to spaces that occur after the information in your string:

`'card                                            '`

Python has a built-in function to get rid of these extra spaces, and you will learn more about it in the next section.
REMOMING TRAILING SPACES IN PYTHON

As you found in the last section, sometimes it is necessary to remove leading and trailing spaces, also known as whitespace, from a string. Fortunately, Python makes this easy with the `strip()` command:

```
short_var = long_var.strip()
```

In this example, the string `long_var` will be stripped of all leading and trailing spaces, and saved as `short_var`.

```
long_var = '      42 Electronics      '
short_var = long_var.strip()
print(short_var)
```

The `strip()` command will only strip leading and trailing spaces from the string, and not the one between the words. After stripping and being saved as `short_var`, the print command will print `42 Electronics` to the console, instead of the longer version with extra spaces.

There are also some other variations of the `strip()` command:

- `lstrip()` Removes only leading whitespace from a string
- `rstrip()` Removes only trailing whitespace from a string

You may have noticed that the `rstrip()` command could also be used to eliminate trailing whitespace from the text value read from a card. This is true, but `strip()` will ensure that any extra spaces, whether before or after the value in the string, are removed.
DETERMINING PROGRAM TYPE: SHEBANG OR #!

You haven’t seen them in programs yet, but a shebang line can be used at the beginning of a program to help some systems identify what type of program is below. The line will be the very first line of a program, and it will start with the characters #! which are referred to as a shebang. The line of code might look something like this:

```bash
#!/usr/bin/env python
```

or

```bash
#!/usr/bin/env python3
```

In Unix-like operating systems, this line can be used to determine the path to locate the program that should be used to execute the program, and which program should be used to run the program. Both examples above specify the path for finding Python as /usr/bin/env, but they specify different versions of Python. The first example specifies Python, but not which version. Without modifications to your system, Raspbian will use Python 2 for a file containing this type of shebang. The second example specifies that the program must run in Python 3.

These lines of code are only used when the Python version is not specified when running the program on the command-line. Starting the program using `sudo python program.py` or `sudo python3 program.py` will override this line, and it will only be treated as any other comment in the program. This is the same behavior we see in Thonny. The environment setting in Thonny will determine which version of Python is used to run a file, not the shebang line.

This won’t normally be an issue for you since programs so far have been run in either Thonny using Python 3, or in the command-line using Python 2. This information can be helpful if you’re trying to build a project you found online, and odd errors seem to be popping up.

A shebang including of something like `#!/usr/bin/env python3` or `#!/usr/bin/env python2.6` means that program was intended to specifically run in that version of Python. Attempting to run a program intended for Python 2.6 using Thonny, where the default is Python 3, will likely result in program errors.
READING AND WRITING TAGS

In order to read and write tags, you will install a couple of libraries in Activity #1 that will enable your Pi to communicate with the MFRC522. One of the libraries is SPI-Py which is takes care of the communication framework needed for the SPI bus. The second library is MFRC522-python which is used to greatly simplify sending data to and from the MFRC522,

Once these libraries are installed, and the SimpleMFRC522 module is imported into your program, working with the reader becomes very easy:

```python
reader = SimpleMFRC522.SimpleMFRC522()
```

This line of code will allow you to refer to the reader as `reader` in your program, instead of the much longer name above.

```python
id, text = reader.read()
```

This command will read the id number and text value stored on the tag and set them equal to `id` and `text`. These variables can then be printed or used in other ways throughout your program.

```python
reader.write('card')
```

The command above can be used to modify the text value stored on the tag. In this example, `card` will be written to the tags text value. The id number field is fixed so there is no command available for changing that value.

That is the extent of the commands that we need to interact with the reader. Using these two commands you can change the text value of a tag, read the id and text value of a tag, and then have your program take any actions you would like, based on the tag data that is presented.
ACTIVITIES

In the following activities you will connect the RFID reader to your Raspberry Pi, install two libraries that will be used to communicate with the reader, and write a couple of programs that will interact with the reader.

ACTIVITY #1 – ADDING THE RFID READER AND SOFTWARE

In this activity you will connect the RFID reader and install software that will allow you to read and write tags. The circuit from Activity #10, Lesson #2 will be used as the starting point for this lesson.

STEP #1

*Shut down the Pi and disconnect power before proceeding.*

To make room for the new components, the phototransistor, potentiometer, and any associated components will need to be removed from the breadboard.

Using the circuit from Lesson #10, Activity #2 as a starting point, remove the phototransistor, potentiometer, resistor, and any associated jumper wires. The circuit should now look like this:
STEP #2

The MCP3008 will also need to be removed from the breadboard. Exercise a lot of caution when removing the IC from the breadboard. If the IC comes out of the board unevenly it can cause the pins to bend, and they may break when straightened.

Gently inserting a small, flat screwdriver under alternating sides of the IC is the safest way to remove the IC.

Remove the IC and associated jumper wires from the breadboard. If you're at all unsure on the best way to do this, watch the short video on the Level B Resource Page before attempting to remove the IC. The breadboard should now look like this:
STEP #3

Now that you have room on the breadboard, it's time to install and connect the reader.

Install the 8-pin connector of the reader into H49 through H56. The reader should be oriented such that the main body of the reader is above covering columns A through H.
STEP #4

The reader is now ready to be connected to the wedge. Make the following connections between the reader and the wedge:

3.3V – J49 to P1-41
Reset – J50 to J11
Ground – J51 to N2-51
No connection – J52

MISO – J53 to C11
MOSI – J54 to C10
SCLK – J55 to C12
SDA – J56 to J12

Double-check all connections with this photo before proceeding to the next step.
STEP #5
Now that the reader is connected it's time to power on the Pi and enable hardware SPI.

Power on your Raspberry Pi.

Once it's up and running click on the raspberry in the top-left corner, select Preferences, and then select Raspberry Pi configuration from the bottom of the list.
Once inside the configuration utility, select the Interfaces tab, and select the Enabled radio button next to SPI. This will turn on hardware SPI in the Raspberry Pi.

Reboot your Raspberry Pi to allow the SPI setting change to take effect.
STEP #6

Before installing the required libraries, make sure your Pi is fully updated so the libraries will have access to the latest versions of the Raspberry pi software packages.

Open a Terminal window by clicking the terminal button in the top menu bar. Once open, use the command `sudo apt-get update` to ensure your Pi knows the latest version numbers of all packages. Once that completes, run `sudo apt-get upgrade` to upgrade any required packages to the latest version. Answer y to any questions about free disk space that will be consumed by the upgrades.
Now that your Pi is fully updated, it's time to clone the required libraries from GitHub.

In your existing Terminal window, type the following commands, pressing enter after each command:

First, ensure you are still located in the /home/pi directory:

```
cd ~
```

Next, clone SPI-Py from the 42 Electronics GitHub repository:

```
git clone https://github.com/42electronics/SPI-Py.git
```

Now change directories into SPI-Py:

```
cd SPI-Py
```

The last step is to execute the setup.py install script for python3:

```
sudo python3 setup.py install
```

SPI-Py will now be installed for use in the Python 3 environment, which you can run from within Thonny. If you encounter any errors during this process, start over from the beginning of this step.
STEP #8

The last library to clone will be MFRC522-python. This library does not require the install step, just cloning from GitHub.

In your existing Terminal window, type the following commands, pressing enter after each command:

First, move yourself back to the /home/pi directory:

```
cd ~
```

Next, clone MFRC522-python from the 42 Electronics GitHub repository:

```
git clone https://github.com/42electronics/MFRC522-python.git
```

If you encounter any errors during this process, start over from the beginning of this step.
ACTIVITY #2 – READING AND WRITING TAGS

In this activity you will connect the read information from and write information to the tags included in your kit.

STEP #1

You now have a local copy of the MFRC522-python repository.

Open File Manager by clicking on the folder icon in the top menu bar. File manager will open in the /home/pi directory. Double-click on the MFRC522-python directory to view its contents.

Some example programs called read.py and write.py have been included to allow you to quickly begin reading and writing tags.

Double click the file named read.py and it will open in Thonny:

```
#!/usr/bin/python3

import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()

try:
    id, text = reader.read()
    print(id)
    print(text)
finally:
    GPIO.cleanup()
```

Run read.py by clicking the run button in Thonny. Place the card near the reader and its id number and text value will be displayed in the console output.

The text field will be represented by 48 squares because this field is completely empty from the factory.

Scan the blue tag to ensure it scans properly, and that it displays a different id value than the card.
STEP #2

The cards can be read but you need to fix the empty text value fields by writing new data to the tags.

Navigate back to the File Manager and double-click on write.py. Write.py will open as a new tab in Thonny.

Make sure there are no tags near the reader when you run write.py. The program will write any tag within range and you want to make sure the tags get programmed correctly, so you can write a program in Activity #3 that will recognize these values.

You will now write the text value of card to the white card.

With no tags near the reader, run write.py in Thonny. In the Shell window of Thonny, enter card as the value to be written to the tag, and press enter. Place the white card near the reader when prompted, the tag will be written, and Tag written will be printed to the Shell for confirmation.

If any errors occur, or the program does not run as expected, stop write.py in Thonny and run it again. Do not proceed to the next step until the white card has successfully been written with the text value card.

STEP #3

Now that the card has been written you can work on the blue tag. Make sure to hold the metal keyrings on the tag to keep them from coming into contact with any metal connections on the circuit board of the reader.

Using the same process in the last step, write a text value of tag to the blue tag.
STEP #4

Confirm the tags were properly written by reading the values back.

Switch tabs in Thonny back to read.py and run that program. Scan the white card to ensure it reports its ID number and the text string **card** when scanned.

Run the program again and scan the blue tag to ensure it reports its ID number and the text string **tag** when scanned.

You now know that the text values loaded correctly and now you can create a program that can make decisions based on those values.

**NOTE:** The RFID reader’s proper operation relies on good connections to power, ground, and data lines on the Raspberry Pi. Poor or intermittent connections can cause the reader not to operate properly. If your reader stops reading or writing for no reason, carefully remove and reinstall each of the jumper wire connections in J49 through J56. If the problem with your reader is due to a bad jumper wire connection, this should fix the problem.

The reader will read a tag very quickly, but if you try to swipe a tag by the reader extremely fast it is possible to pull the tag away from the reader before it’s finished reading. This will result in a card read error that might look something like this:

```
>>> %Run read.py
AUTH_ERROR!!
AUTH_ERROR(status2reg & 0x88) != 0
1005969354382
```

If this happens, just scan the tag again, more slowly, and everything should work as expected.
ACTIVITY #3 – CREATING AN ACCESS CONTROL PROGRAM

In this activity you will create a program that reads the text values from scanned tags, and prints a message letting the user know if access is granted, or not.

STEP #1

Use the read.py program as a starting point, as it already contains everything needed to read tags.

With read.py open in Thonny, select File, and then select Save as from the dropdown menu. Enter a file name of access_control and click the Save button:

NOTE: Programs that need to read RFID tags must be located inside the MFRC-Python directory, as they will need direct access to the SimpleMFRC522.py and MFRC522.py files used to communicate with the reader. Attempting to run programs that require access to these communication files from anywhere else, will result in Python errors.
STEP #2

You now have a copy of read.py saved as access_control.py that can be modified without affecting the original file. The program is currently running a try: loop until it sees a tag, prints the id and text values, and runs a GPIO.cleanup() before exiting. Only one tag can be read before the program automatically exits. Let's add a while True: loop inside the try: loop to keep the program reading tags until you exit the program.

Add a while True: loop just below the try: loop. The addition is highlighted below:

```python
reader = SimpleMFRC522.SimpleMFRC522()
try:
    while True:
        id, text = reader.read()
        print(id)
        print(text)
        time.sleep(.3)
finally:
    GPIO.cleanup()
```

Run the program and read the card and tag.

You will notice the program is now reading without exiting after each tag, but it's reading one tag multiple times because this program can loop very quickly, and the tag might be near the reader during more than one loop.

Another problem with the program is that there is no longer away to exit the program gracefully. Press the stop button in Thonny or CTRL-C to stop the program. This will result in errors because the program was busy communicating with the reader when the program was ended. You will fix the looping and exit problems in the next step:
STEP #3

It's time to fix the duplicate read and exit issues that we created in the last step. You will add a `sleep` command to slow down the loop and modify the `finally:` command to catch keyboard exceptions.

Import the `time` module at the beginning of the program and add a `time.sleep(.3)` to the end of the loop. Also, change `finally:` to `except KeyboardInterrupt:` which will allow manually ending the program to trigger the `GPIO.cleanup()`.

Additions and changes are highlighted below:

```
#!/usr/bin/python3

import RPi.GPIO as GPIO
import SimpleMFRC522
import time

reader = SimpleMFRC522.SimpleMFRC522()

try:
    while True:
        id, text = reader.read()
        print(id)
        print(text)
        time.sleep(.3)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Run the program.

It can now scan tags reliably without duplicating reads and pressing stop in Thonny or CTRL-C will not generate errors, as `GPIO.cleanup()` is being triggered before the program exits.
STEP #4

Now that the program is reading cards without exiting, it needs to strip the trailing whitespace from the text value that's being read from the card. To do this you will run the `strip()` command on the value of text that's read from the card.

Replace the two existing print statements with a `strip()` command that will strip trailing whitespace from `text` and save it as the new value of `text`. Changes highlighted below:

```python
try:
    while True:
        id, text = reader.read()
        text = text.strip()
        time.sleep(.3)
```

The id number and text value will be read from the card and the text value will be stripped down to either 'card' or 'tag' based on the tag that was scanned.
STEP #5

You can now add some if statements to check which tag is being read and print whether the tag is granted or denied access. Go ahead and grant access to the card, but deny access to the tag, by adding two if statements.

Add two if statements directly under the strip() command that will check the value of text and print **ACCESS GRANTED** or **ACCESS DENIED** based on which tag is read:

```python
try:
    while True:
        id, text = reader.read()
        text = text.strip()
        if text == 'card':
            print('ACCESS GRANTED')
        if text == 'tag':
            print('ACCESS DENIED')
        time.sleep(.3)
```

Run the program and read both tags. The access messages will be printed each time a card is scanned to indicate **ACCESS GRANTED** or **ACCESS DENIED**.

This code is being kept very simple to illustrate the concept, but the print sections of the if statements in this program could be replaced with anything you like. By including GPIO pin setups at the beginning of your program you could light an LED when access is granted, play a noise through the piezo speaker when access is denied, or any combination of print statements and GPIO events that you might want.

In Lesson B-12, you will add more advanced program functionality, and circuitry that will allow for additional notification options.

- Leave this circuit built as it will be used as a starting point in Lesson B-12.
- Save the program to use as a starting point in Lesson B-12.
QUESTIONS FOR UNDERSTANDING

1. Is hardware SPI turned on by default on the Raspberry Pi, or does it require a menu change and reboot to become enabled?

2. What is the Python command that removes leading and trailing whitespace from a string?

3. Do all RFID tags work with all RFID readers?

Answers can be found on the next page.
ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

1. *Is hardware SPI turned on by default on the Raspberry Pi, or does it require a menu change and reboot to become enabled?*

   **ANSWER:** Hardware SPI requires a menu change and a reboot to be enabled.

2. *What is the Python command that removes leading and trailing whitespace from a string?*

   **ANSWER:** To remove both leading and trailing spaces, use the `strip()` command.

3. *Do all RFID tags work with all RFID readers?*

   **ANSWER:** No, not all readers and tags are compatible. It is important to check for compatibility between RFID readers and RFID tags.
CONCLUSION

In this lesson you learned to work with RFID readers, including how to read and write tags, and remove trailing spaces.

In the next lesson you will continue to work with RFID readers but add more advanced Python programming techniques to create more complicated program functionality.