EEE174 –CPE185 INTRODUCTION TO MICROPROCESSORS

LAB 10 – RFID

Lab Session: Wednesday 6:30PM - 9:10PM

Section 32385

Lab Instructor: Sean Kennedy

Student Name: Andrew Robertson

TABLE OF CONTENTS

PART 1

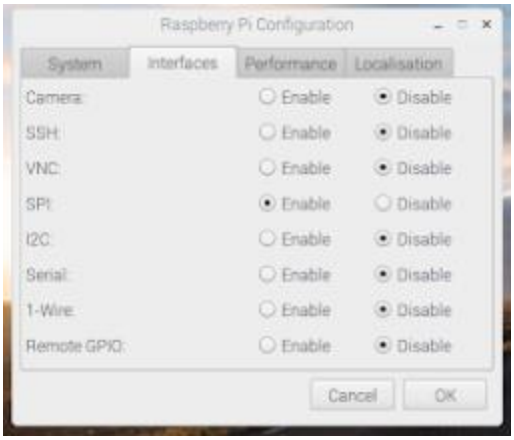
OVERVIEW

This lab is part 5 of 5 for the series of do it yourself labs geared towards helping with the final project. This piece is unrelated to our final project - I am exploring RFID technology to see how it works. This lab will be exploring the use of a Mifare S50 high frequency passive RFID card and Mifare MFRC522 IC.

LAB DISCUSSION

WORK PERFORMED / SOLUTION:

Looking at the datasheet and requirements for this module provided by FreeNove, this device will be using SPI (Serial Peripheral Interface) for its method of communication. Luckily they also mentioned this is disabled by default on Raspbian and should be enabled before continuing otherwise I would have had no idea what was wrong. The instructions provided are out of date and show screenshots and file paths for an older version of Raspbian so I would like share updated instructions. To enable the SPI interface, navigate to the Raspbian logo, then preferences, followed by Raspberry Pi configuration, and last replicate this page:



Once this is completed, we can move forward to testing the module and the SPI interface by using the code that has been supplied by the packager. When navigating to the Packagers Git repository, I am also instructed to install a certain Library for Python functionality with SPI and this code. I used a feature new to me to do so by using the Git clone command to download the files followed by installation of the library.

```
pi@raspberrypi:~ $ git clone https://github.com/lthiery/SPI-Py.git
Cloning into 'SPI-Py'...
remote: Counting objects: 85, done.
remote: Total 85 (delta 0), reused 0 (delta 0), pack-reused 85
Unpacking objects: 100% (85/85), done.
pi@raspberrypi:~ $ cd SPI-Py
pi@raspberrypi:~/SPI-Py $ sudo python setup.py install
```

Now that I've seen how awesome the Git clone command is in person I proceeded to download their entire repository for my parts kit that had a PDF reference for all parts and provided test code. With the pinout reference in front of me I then proceeded to get the circuit ready to test, once done I then ran the test code and tried reading and writing values to the RFID card. Before this lab I had no clue how RFID cards works but its basically a wireless RAM device. It's a very near field wireless communication but contactless just the same. The image is a bit large to fit here but is Figure 2 under listing files, showing a series of commands to scan (detect and link) a card, read from certain sectors, write to those sectors, clear sectors, and halt (disconnect) from the card.

Figure 1 under listing files shows how these sectors are organized. From reading the PDF provided I've learned that the vendor code cannot be altered and contains the devices serial number. There are also two passwords per sector under a control block, and different passwords grant different levels of access. With Password B one can modify data blocks but with Password A, data and control blocks can be modified. Using password A is much like being a super user in Linux, there is heightened access but with this comes greater danger.

LISTING FILES(S):

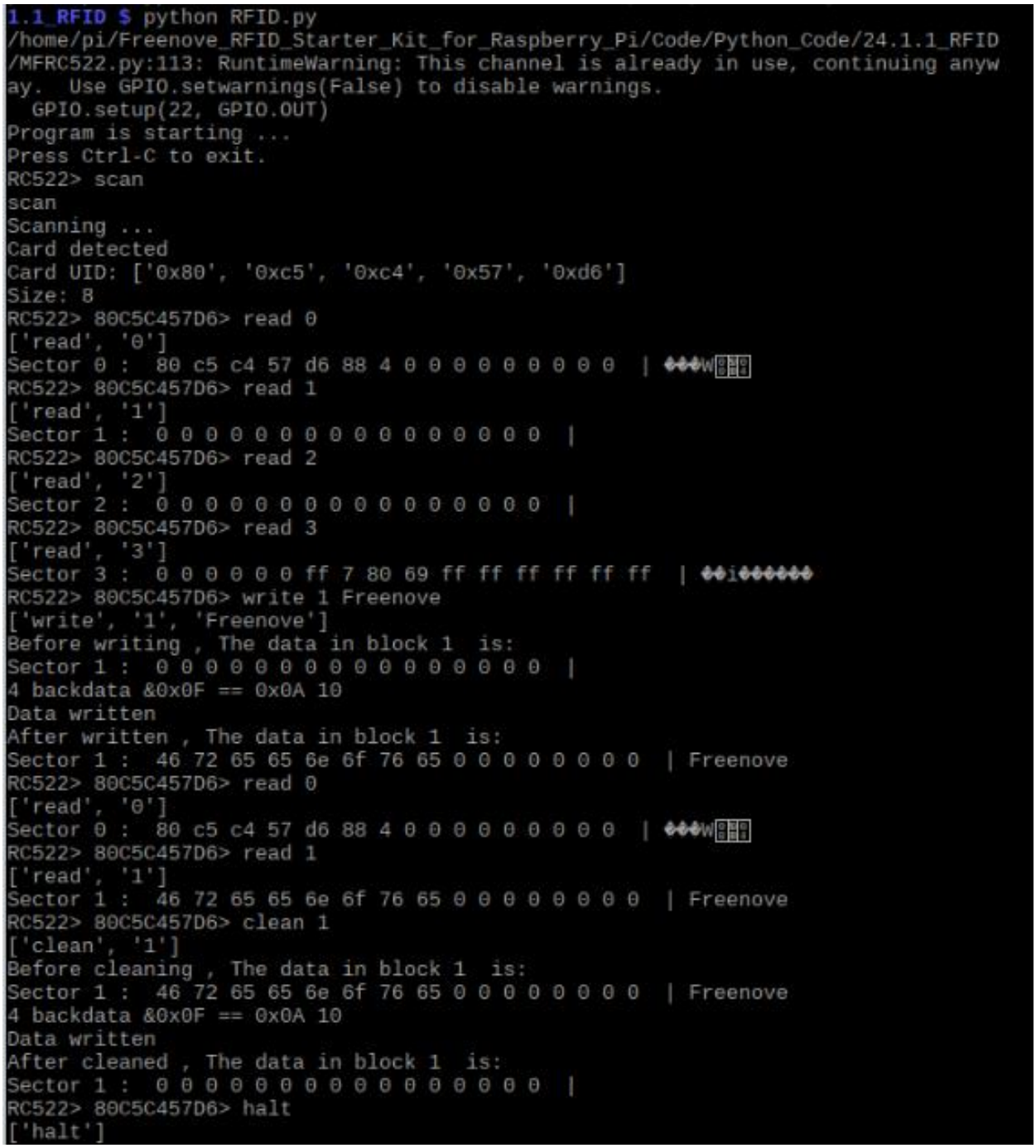
RFID code provided via:
https://github.com/Freenove/Freenove_RFID_Starter_Kit_for_Raspberry_Pi/tree/master/Code/Python_Code/24.1.1_RFID

RFID module Pinout provided via:
https://github.com/Freenove/Freenove_RFID_Starter_Kit_for_Raspberry_Pi/blob/master/Tutorial.pdf

Figure 1

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
.....	.....	.....	.....	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Figure 2


1.1_RFID $ python RFID.py
/home/pi/Freenove_RFID_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/24.1.1_RFID
/MFRC522.py:113: RuntimeWarning: This channel is already in use, continuing anyw
ay. Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(22, GPIO.OUT)
Program is starting ...
Press Ctrl-C to exit.
RC522> scan
scan
Scanning ...
Card detected
Card UID: ['0x80', '0xc5', '0xc4', '0x57', '0xd6']
Size: 8
RC522> 80C5C457D6> read 0
['read', '0']
Sector 0 : 80 c5 c4 57 d6 88 4 0 0 0 0 0 0 0 0 0 |
RC522> 80C5C457D6> read 1
['read', '1']
Sector 1 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
RC522> 80C5C457D6> read 2
['read', '2']
Sector 2 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
RC522> 80C5C457D6> read 3
['read', '3']
Sector 3 : 0 0 0 0 0 0 ff 7 80 69 ff ff ff ff ff ff |
RC522> 80C5C457D6> write 1 Freenove
['write', '1', 'Freenove']
Before writing , The data in block 1 is:
Sector 1 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
4 backdata &0x0F == 0x0A 10
Data written
After written , The data in block 1 is:
Sector 1 : 46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0 | Freenove
RC522> 80C5C457D6> read 0
['read', '0']
Sector 0 : 80 c5 c4 57 d6 88 4 0 0 0 0 0 0 0 0 0 |
RC522> 80C5C457D6> read 1
['read', '1']
Sector 1 : 46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0 | Freenove
RC522> 80C5C457D6> clean 1
['clean', '1']
Before cleaning , The data in block 1 is:
Sector 1 : 46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0 | Freenove
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1 is:
Sector 1 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
RC522> 80C5C457D6> halt
['halt']

CONCLUSION

Security checkpoints, hotels, and work environments are some examples of establishments that use RFID technology. After doing this lab I believe I can figure out how one of these devices works. Using the commands tested above it seems a finite state machine would allow for easy verification and access methods. State 0 would be to scan, and it would remain in that state until a card is scanned. State 1 would be to dump that card's data, check it against some sort of allowed listing and set the appropriate flag(s). Last, check flags, respond accordingly such as allowing entry or giving denial indication, then disconnect from card and return to state 0.