Contents

1 Serial communication with SPI

In this lab, we'll learn how to connect external parts to a single board computer using a digital communication bus, specifically: using an SPI communication bus. At the end of this lab, you should be able to:

- Connect a peripheral device or sensor to the SPI bus on the Raspberry Pi.
- Use a datasheet to identify the registers, and values to write to those registers, to configure and use a peripheral device or sensor.
- Inspect the digital waveform transmitted over the SPI bus, and identify important features (most and least significant bits, start condition, address, clock, etc.)

1.1 Notes

- In this lab, you will create some breadboard circuits with exposed pins and wires. Please be especially careful not to accidentally create connections that shouldn't be connected (e.g. short circuits). Also, check your work carefully before connecting any breadboard circuit to a board, to avoid damaging the board.
- Read each subsection of this lab manual in its entirety before you start following the instructions in it. Some instructions are modified by explanations that come afterwards.
- Although you may work with a partner, this collaboration is limited to discussion. Your partner is not allowed to construct or modify your circuit, log in to your Pi, or run commands or write code on your Pi. Similarly, you are not allowed to do these things for your partner. (You *are* encouraged to collaborate by screen-sharing or showing video of your circuit to debug and discuss problems together.)
- For your lab report, you must submit data, code, and screenshots from your own experiment. You are not allowed to use your lab partner's data, code, or screenshots.
- For any question in the lab report that is marked "Individual work", you should *not* collaborate with your lab partner or anyone else (even via discussion). You can use your notes, the lab manual, or the lecture slides and video to help you answer these questions.

1.2 Parts

In this lab, we'll use the following parts:

- Pi, SD card, and power supply. We will insert the SD card, connect the power supply, and log in to the Pi over SSH.
- Breadboard and jumper cables
- MFRC522 module and tags (one tag is a white card; the other is a blue plastic tag on a keychain.)

1.2.1 MFRC522 contactless communication module

The MFRC522 is a writer/reader for reading and writing to passive "tags" via contactless communication.

Figur 27 in the MFRC522 datasheet shows how the IC is typically used, in combination with an oscillator and antenna. The photograph of the MFRC522 module in your kit is annotated to show the location of these key components on the board:

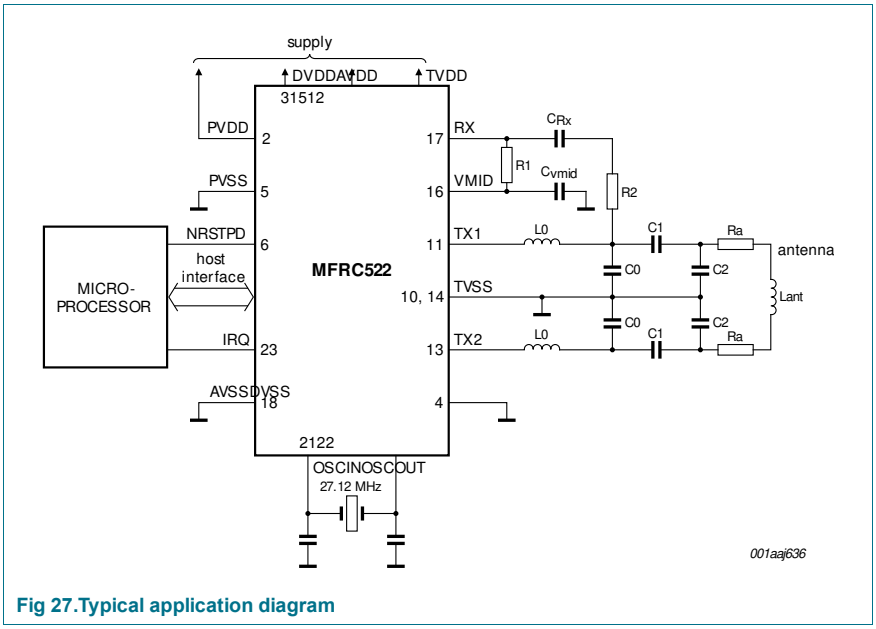


Fig 27.Typical application diagram

Figure 1: Application diagram from the MFRC522 datasheet (left), and its main components - the MFRC522 IC, the antenna, and the 27.12MHz crystal oscillator - on the board in your kit (right).

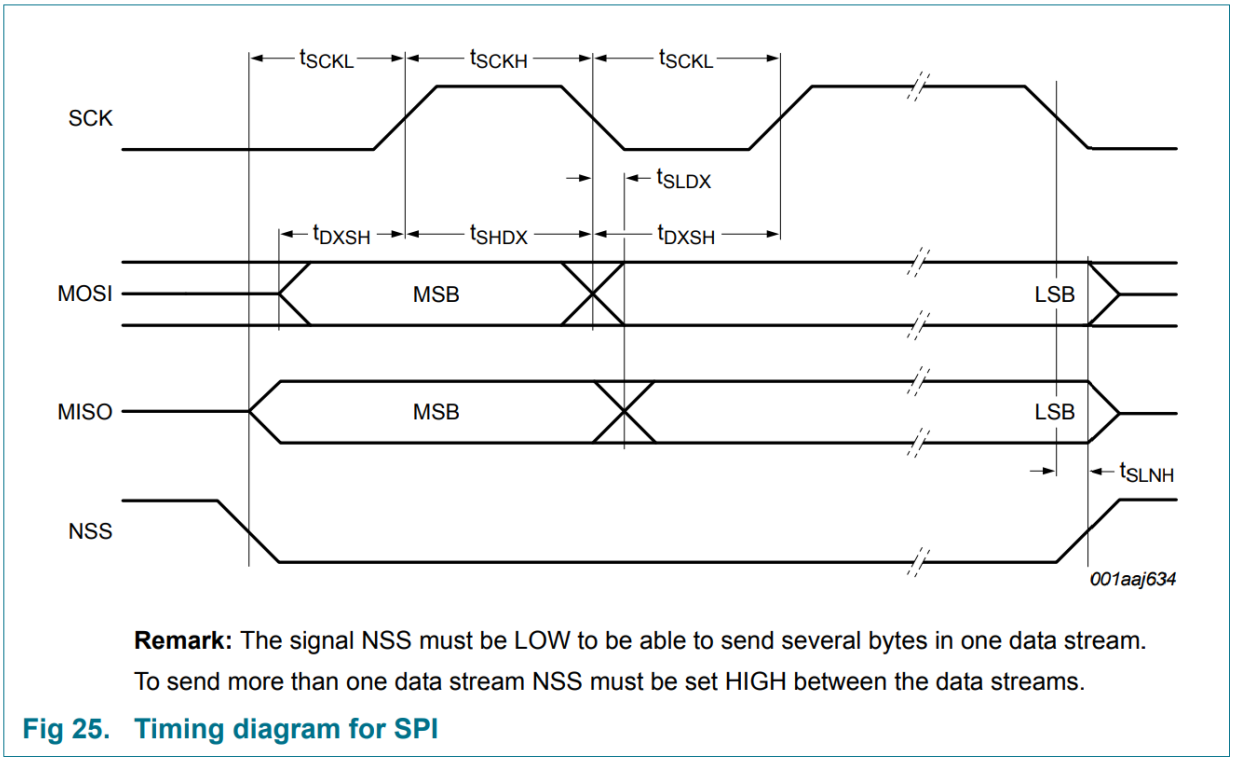
Contactless communication systems (such as MIFARE, RFID and NFC systems) use two devices, and transceiver (reader) and a transponder (tag). In our case, the tag is *passive* - it does not have any battery or other power source.

How does it work without its own power source? The reader generates a high frequency electromagnetic field. When the tag moves within close proximity of this field, electrons move through the tag's antenna, powering a small chip inside the tag which then modulates the electromagnetic field with data stored in the chip. The reader can detect the resulting change in the electromagnetic field, "reading" the data stored in the tag.

What about the tags themselves? The tags in our kits of have 1KB of memory, organized in 16 *sectors*. The sectors are numbered from 0 to 15. Each sector is divided into *blocks*, numbered from 0 to 3, which can store 16 bytes of data each, numbered from 0 to 15. However, the last block of each sector is reserved for access control, so only blocks 0, 1, and 2 of each sector are available for data. Furthermore, the first block of the first sector is used to store information about the IC manufacturer and a unique ID for the tag, and is also not available for data.

1.3 SPI interface with MFRC522

The MFRC522 module that we are using has an SPI interface (as well as some other serial communication busses, but on our breakout boards, the chip is hard-wired to use SPI mode). Figure 25 in the datasheet shows the SPI timing diagram:



Remark: The signal NSS must be LOW to be able to send several bytes in one data stream. To send more than one data stream NSS must be set HIGH between the data streams.

Fig 25. Timing diagram for SPI

Figure 2: SPI timing diagram for MFRC522

Lab report (individual work): When you connect the MFRC522 to the SPI interface on the Pi, which device is the "controller" and which is the "peripheral"?

Lab report (individual work): Does the MFRC522 expect the clock signal to be LOW or HIGH when idle? Which edge of the clock signal - rising or falling edge - is the "leading" vs "trailing" edge? Does the MFRC522 expect data to change on the leading edge and be stable on the trailing edge, or change on the trailing edge and be stable on the leading edge? What clock phase does the SPI interface on the MFRC522 expect? What clock polarity?

1.3.1 Enable SPI on Raspberry Pi OS

The SPI driver is disabled by default on Raspberry Pi OS. To enable it, run

```
sudo raspi-config
```

and choose Interfacing Options » SPI, select "Yes", then "OK", then "Finish".

Verify that it is enabled by running

```
lsmod | grep spi
```

This lists all currently loaded kernel modules (including drivers) and then filters the output to show only those with `spi` in the name. You should see an entry for `spidev` and `spi_bcm2835`.

1.3.2 Install the library

We'll use an open source library for the MFRC522 module.

First, make a new directory:

```
mkdir ~/lab-spi
```

Next, install a prerequisite Python SPI library. Run

```
cd ~/lab-spi
git clone https://github.com/lthiery/SPI-Py.git
cd SPI-Py
```

Now, install the SPI library with

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

Then, return to your home directory, and retrieve and install the MFRC522 library:

```
cd ~/lab-spi
git clone https://github.com/pimylifeup/MFRC522-python
cd MFRC522-python
sudo python3 setup.py install
```

1.3.3 Connect the MFRC522

Connect the MFRC522 module to the Raspberry Pi or Pi Zero W as shown in the diagram.

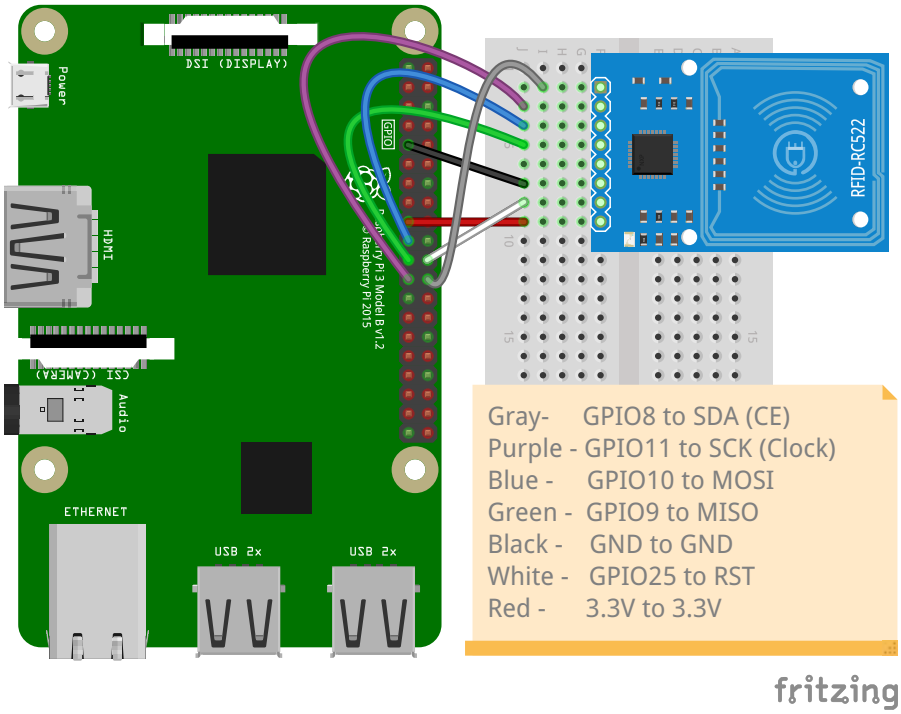


Figure 3: Connect the Pi to the MFRC522 module.

- Gray - GPIO8 (SPI CE0) to SDA on the MFRC522 module
- Purple - GPIO11 to SCK on the MFRC522 module (SCK a.k.a. clock line)
- Blue - GPIO10 (SPI SDA) to MOSI on the MFRC522 module
- Green - GPIO9 (SPI SDI) to MISO on the MFRC522 module
- Black - GND to GND on the MFRC522 module
- White - GPIO25 to RST on the MFRC522 module
- Red - 3.3V to 3.3V on the MFRC522 module

1.3.4 Get card ID

First, we'll use the module to get the card ID. We will use this as an opportunity to dig into the library and see how it works.

Open a new file called `spi-id.py` inside the `lab-spi` directory, and put the following inside:

```
import RPi.GPIO as GPIO
import sys
import time
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

print("Hold a tag near the reader")
print("Reading tag in 1 second...")
time.sleep(1)
id = reader.read_id_no_block()
print(id)

GPIO.cleanup()
```

Save your script and exit the text editor. Place the tag against the reader. Then, run

```
python3 spi-id.py
```

and note the ID that is returned.

By default, the SPI interface operates too fast for `piscope` to be able to reliably show us what's going on on the SPI lines. We will tell the SPI interface to run slower so that we can watch it. Open the library file `MFRC522.py`:

```
nano ~/lab-spi/MFRC522-python/mfrc522/MFRC522.py
```

and find the line

```
    def __init__(self, bus=0, device=0, spd=1000000, pin_mode=10, pin_rst=-1,
        debugLevel='WARNING'):
```

Change it to

```
    def __init__(self, bus=0, device=0, spd=10000, pin_mode=10, pin_rst=-1,
        debugLevel='WARNING'):
```

(two fewer zeros!) Then save and close the file. Reinstall the library with

```
cd ~/lab-spi/MFRC522-python/
sudo python3 setup.py install
cd ~/lab-spi/
```

Run

```
sudo pigpiod
```

if it is not already running, and open PiScope:

```
piscope
```

Leave PiScope open to monitor the SPI lines. Then run your script again:

```
python3 spi-id.py
```

The SPI transactions in Piscope will appear in two groups - first, the read and write commands used to initialize the device, and then, after 1 second of idle time, the commands used to read the ID.

Section 8.1.2.2 and 8.1.2.3 of the datasheet describes how to write data to command registers on the module. Each transaction is two bytes: first, the controller sends the address of the register to write to, then it sends the command to write.

Look at the SPI transactions used to initialize the module. See if you can identify each step of the initialization process.

- Send 0x0F to register 0x01 (CommandReg)
- Send 0x8D to register 0x2A (TModeReg)
- Send 0x3E to register 0x2B (TPrescalerReg)
- Send 0x1E to register 0x2D (TReloadReg low bits)
- Send 0x00 to register 0x2C (TReloadReg high bits)
- Send 0x40 to register 0x15 (TxASKReg)
- Send 0x3D to register 0x11 (ModeReg)

Next, look at the second group of SPI transactions. Here, the Pi is reading the UID. Section 8.1.2.1 and 8.1.2.3 describes how to read data from the module.

In the Pi's SDI line (where data from the MFRC522 module is sent to the Pi), can you find the bytes of the UID? (Convert the UID returned by the script to hex, then look for each pair of hex digits as a single byte.)

Note that for each "read", the Pi first sends the address of the FIFODataReg (0x09), then reads a byte from the peripheral.

Lab report: Take a screenshot of the piscope display showing the 7 initialization commands, as described above. Annotate the screenshot - for each initialization command, label the register byte and command byte.

Lab report: Take a screenshot of the piscope display, zoomed in to the last byte the Pi reads from the module. On the screenshot, indicate each time when data is sampled from the SPI line. Label the address byte, and note which bit is the R/W bit, which are the address bits, and which is the unused bit. Then, label the data bit that is transferred from the module.

Lab report: Note the status of the CE0 line on the Pi during the transactions above. What is the purpose of the CE line?

1.3.5 Write to and read from a tag

Sometimes, we may want to read and write more than just the tag ID - we may want to store data on the tag.

Edit the MFRC522 library again to restore the original SPI bus speed, and install the library again.

Open a new file called `spi-mfrc522.py` inside the `lab-spi` directory, and put the following inside:

```
import RPi.GPIO as GPIO
import sys
import time
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

print("Hold a tag near the reader")

try:
    print("First, writing to the tag...")
    text = input('Enter new data to write to tag, then hit Enter: ')
    print("Place your tag next to the antenna to write")
    reader.write(text)
    print("Done writing! Remove your tag.")

    print("Wait 5 seconds...")
    time.sleep(5)

    print("Now, reading tag - place it next to the antenna...")
    id, text = reader.read()
    print(id)
    print(text)

finally:
    GPIO.cleanup()
```

Save your script and exit the text editor. Then, run

```
python3 spi-mfrc522.py
```

When prompted, enter some text to write to the tag. Then, hold the tag near the antenna. Keep it there until you have received confirmation that the new text was written.

After 5 seconds, the script will switch to "read" mode. Hold the tag near the reader, and wait until the tag ID and text is printed to the screen.

Use the zoom and arrow buttons in `piscope` to examine the SPI communication.