PN532 NFC/RFID Guide

by Hayden Mai
Last Updated: April 10, 2025

Guide has been tested on

BeagleY-AI (Target):	**Debian 12.8**
PC OS (host):	**Debian 12.8**

This document guides the user through:

1. Wiring to the Zen Hat's PYMNL based on SPI/I2C protocol
2. Cross-compiling libnfc and configuration support
3. Read/write for a 13.56MHz MIFARE Classic 1k card/tag

Table of Contents

Formatting

1. Commands for the host Linux's console are shown as:
   `(host)$ echo "Hello PC world!"`
2. Commands for the target (BeagleY-AI) Linux's console are shown as:
   `(byai)$ echo "Hello embedded world!"`
3. Almost all commands are case sensitive.

1. RFID Intro

This guide covers the basics of the PN532 module using the libnfc library and the procedures to read and write data to a MIFARE Classic 1k tag. The PN532 reader module supports both Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI) communication protocols. However, this guide has only been tested for SPI communication on the BeagleY-AI platform.

Previous PN532 RFID guides focus on different programming languages such as C++ or Python, and utilize an alternative method to communicate with the module. In contrast, this guide aims to provide a practical extension to lecture materials while addressing potential knowledge gaps.

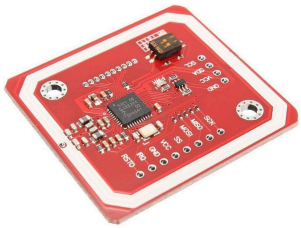

Figure 1. The PN532 module used in this guide

2. Wiring and Configuration

1. For SPI/I2C communication, the dual in-line package (DIP) switches on the PN532 modules need to be configured as follows:

Communication Protocol	SEL0	SEL1
I2C (Figure 2)	1	0
SPI (Figure 3)	0	1

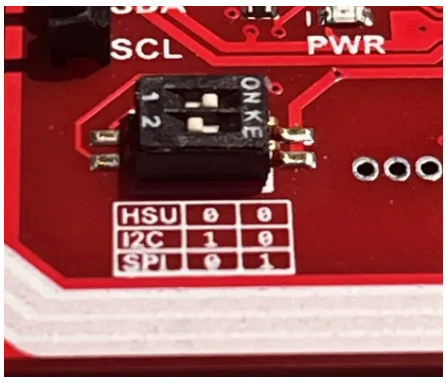

Figure 2. I2C Mode

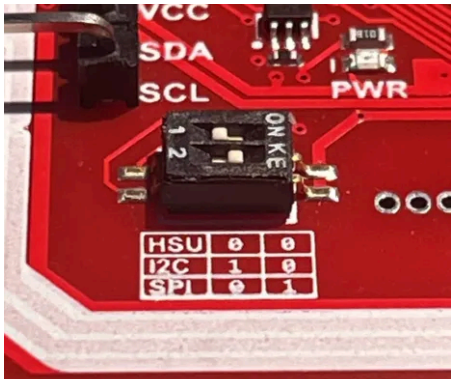

Figure 3. SPI Mode

2. The PN532 requires either 3.3V or 5V, as well as connecting the PN532 pins to the Zen Hat pins based on the communication mode:

SPI Pin Configuration

PN532 Pin	**Zen Hat Pin**
SCK	SPI0 SCK
MISO	SPI0 MISO
MOSI	SPI0 MOSI
SS	SPI0 CE1
VCC	3.3/5V
GND	GND

I2C Pin Configuration

PN532 Pin	**Zen Hat Pin**
SDA	I2C1 SDA
SCL	I2C1 SCL
VCC	3.3V/5V
GND	GND

3. Installing `libnfc`

To begin reading tags using the PN532, we will be using the libnfc library as the interface between Linux and the NFC module. Note: Only SPI was fully tested for this section.

1. Install libnfc on target:
   `(byai)$` **`sudo apt install libnfc-bin libnfc-examples`**

2. Configure libnfc to use SPI port:
   `(byai)$` **`nano /etc/nfc/libnfc.conf`**

```
…
# Set log level (default: error)
# Valid log levels are (in order of verbosity): 0 (none), 1
(error), 2 (info), 3 (debug)
# Note: if you compiled with --enable-debug option, the default
log level is "debug"
log_level = 1

# Manually set default device (no default)
# To set a default device, you must set both name and
connstring for your device
# Note: if autoscan is enabled, default device will be the
first device available in device list.
device.name = "PN532 via SPI" # Can be anything you'd like
device.connstring = "pn532_spi:/dev/spidev0.1:50000" # SPI
```

- If you are using I2C, replace the last line with:
  `device.connstring = "pn532_i2c:/dev/i2c-1" # I2C`

3. Check if libnfc is successfully configured:
```
(byai)$ nfc-list
nfc-list uses libnfc 1.8.0
NFC device: PN532 via SPI opened
```

4. To read a card, you can either have a tag already on the module and run the command in step 3, or you can poll for a tag instead. An example read of a MIFARE Classic 1k tag:
```
(byai)$ nfc-list
... (first 2 line omitted)
1 ISO14443A passive target(s) found:
ISO/IEC 14443A (106 kbps) target:
     ATQA (SENS_RES): 00  04
     UID (NFCID1): bb  69  f9  04
     SAK (SEL_RES): 08

(byai)$ nfc-poll
... (first 2 line omitted)
NFC device will poll during 36000 ms (20 pollings of 300 ms for 6
modulations)
ISO/IEC 14443A (106 kbps) target:
     ATQA (SENS_RES): 00  04
     UID (NFCID1): bb  69  f9  04
     SAK (SEL_RES): 08
Waiting for card removing...nfc_initiator_target_is_present:
Target Released
Done.
```

5. Cross compiling libnfc on host for C development:
```
(host)$ sudo apt install libnfc-dev:arm64
```
- Include `#include <nfc/nfc.h>` when coding.
- While compiling, add the linker option `-lnfc` flag.
- Refer to the included sample code for CMake compilation.

6. Troubleshooting:
   - If neither `nfc-list` nor `nfc-poll` works, try running the commands with **`sudo`**.
   - Check if the SPI overlay is loaded on the BeagleY-AI:
     - You should have two files: **`/dev/spidev0.0`** and **`/dev/spidev0.1`**.
     - If the files are not present, check `extlinux.conf`:
       `(byai)$ sudo nano /boot/firmware/extlinux/extlinux.conf`
       - The `fdtoverlays` line in the last section should contain:
         `/overlays/k3-am67a-beagley-ai-spidev0.dtbo`
   - Check if your jumper wires are correctly connected (See *2. Wiring and Configuration*).

4. Read/Write to a Tag

This portion of the guide will provide a quick rundown of read and write for a MIFARE Classic 1k tag using the PN532 module. For more information, refer to the provided C sample code and both the PN532 (section 7.3.8, pg. 130) and MIFARE Classic (section 8.6 - 9.1) datasheets.

Included with each tag is 1KB (1024 bytes) of EEPROM, it is organized as:

- 16 sectors, numbered from 0 to 15.
- Each sector contains 4 data blocks, numbered from 0 to 3.
  - The first data block (block 0) of the first sector (sector 0) is the **manufacturer block**. It contains the UID of the tag and manufacturer data.
  - The last data block (block 3) of each sector is a **sector trailer**, containing keys A & B and access bits (access conditions).
- Each block can store up to 16 bytes of data.

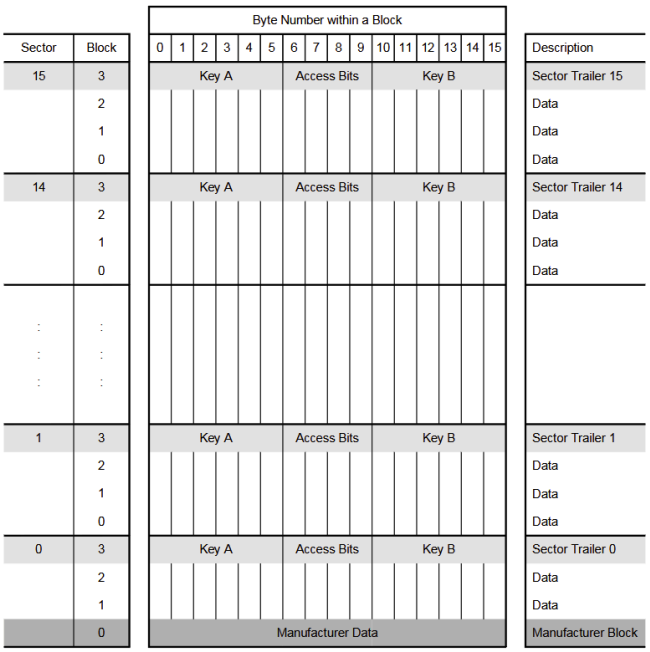

Figure 4. Memory layout of the MIFARE Classic 1k

4.1. Reading a Tag

In order to read data, the PN532 module must make a request to read a block. This can be done using the following data structure:

CMD	ADDR	DATA[16]

- **CMD** is the command byte, in this case is read (0x30).
- **ADDR** is the block number to access (e.g. block 1)
- **DATA** stores the 16 bytes of data read from the tag.

Note: In order to access a block, the RFID reader/writer must first send an authentication command (0x60/0x61) to any block in the same sector for read/write access. This only needs to be done once if your next access is within the same sector. (e.g. authenticate once to read blocks 4, 5, 6 in one go)

- To authenticate, the same data structure is used, however **DATA** contains authentication information:
  - Bytes **0 to 5** contain the 6 byte key (A or B).
  - Bytes **6 to 9** contain the 4 byte UID of the targeted card.
- By default, key A is `{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}`. Key B is not set.
- Key and access bits may be changed, refer to section 8.6.3 of the MIFARE Classic datasheet for more information about the sector trailer.

4.2. Writing to a Tag

Similarly, writing to a tag uses the same data structure, now with the write (0xA0) command as **CMD**, and the data to be written to the tag is stored **DATA**. Do note that changing access bits may change the write permission of the sector for a certain key. By default, authentication using key A will enable read/write access. Visit section 8.7.1 of the MIFARE Classic datasheet for further information.

4.3. List of Commands

Here are a few relevant commands you may use/encounter:

Command (CMD)	**Command code (hexadecimal)**	**Notes**
Authentication with Key A	0x60	
Authentication with Key B	0x61	Is not used by default
Read	0x30	
Write	0xA0	
Decrement	0xC0	Value block only
Increment	0xC1	Value block only
Restore	0xC2	Value blocks only
Transfer	0xB0	

More commands can be found in section 9.1 of the MIFARE Classic datasheet.

5. Next Steps

You've reached the end of this PN532 guide! Be sure to check out the included C sample code (using `libnfc-dev`) for read and write implementations that build on the concepts in this guide. While this guide covers the core concepts, it doesn't address every detail of PN532. For a more comprehensive understanding, referring to the official datasheets is highly recommended.

6. References

- PN532 Datasheet: https://www.nxp.com/docs/en/user-guide/141520.pdf
- MIFARE Classic Datasheet: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf
- libnfc library: https://github.com/nfc-tools/libnfc
- MIFARE Classic 1K Access Bits Calculator: http://calc.gmss.ru/Mifare1k/