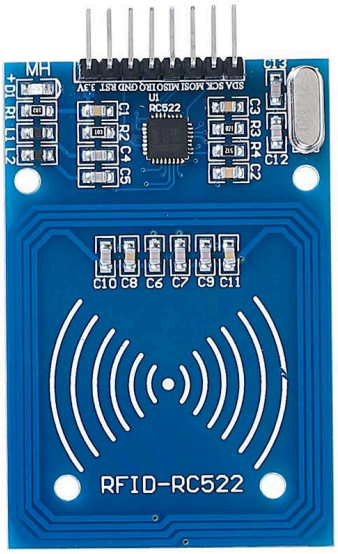
RFID-RC522 Quick Start Guide



THE EMBEDDERS

Steven Quinn
Don Abance
Sumrit Sanghera
Jun Hong

Last update: April 2024

This documents guides the users through:

1. Setting up SPI in C on the BeagleBone Green
2. Understanding the RC522 hardware
3. Translating that understanding into C code

Table of Contents

1. Setting up SPI

Background

As of April 2024, at this page there is a link to a BeagleBone Green (BBG) Serial Peripheral Interface (SPI) guide, a direct link to which can be found here (guide) and here (support files).

We recommend using the guide and support files as a starting point for using the RC522 RFID reader. The support files contain everything required to use SPI on the BBG to communicate with serial-based devices. There are also files which provide a foundation for using the SPI module to communicate with the RC522. This guide is meant as an expansion on that information. We want to provide users with a little more granularity in understanding how to work with the RFID module — since the SPI guide provides documentation about SPI, and supplies RFID-related code, but has no direct information about programming for the RC522.

Fully explaining how to read/write RFID tags would likely go over the 5 page limit for this guide. Instead, we will lay the groundwork for communication with the RC522, and give the user enough of a general intuitive understanding of its components to be able to take the next steps.

Device Trees 🌲

One of the first things to point out about the aforementioned SPI guide is that while they do show (pg.2) the various configurations for SPI on the BBG — SPI0, SPI1/CS0, SPI1/CS1 — there is no mention anywhere about device trees. This may be because of a difference in Linux versions. In our case, we discovered that a device tree had to be enabled to use SPI, and there was scarce mention of this online.

To check if SPI is enabled, try `ls /dev/` on your BBG. If the device tree is enabled, you should see the various SPI interfaces in the list: **spidev0.0**, **spidev1.0**, and **spidev1.1**, and possibly others. If there is no sign of these, you will have to enable the device tree; refer to the Zen Cape Audio Guide (current link) for steps. The device trees are named as such:

```
BB-SPIDEV0-00A0.dtbo
```

A list of them can be viewed with the following command:

```
ls /lib/firmware | grep -E "SPIDEV.*dtbo"
```

You will want to enable the one that reflects the relevant SPI interface/pinout that you have chosen (i.e. spidev0, or spidev1) based on the SPI guide's table. *Note: A reboot will be required after modifying your uEnv.txt file.*

Once SPI is enabled on your BeagleBone, use the SPI guide and its provided sample files as a basis to set up SPI communication in C. We will use this to communicate with the RFID reader.

2. Getting started with the RC522

We recommend consulting this page for a good explanation of understanding all of the pins on the RC522. Note that because we are using the BeagleBone's SPI interface, the differentiation of functions between pins will mostly be abstracted away from. Beyond that, the MFRC522 datasheet contains everything that one should know about the RFID module. However, it is quite dry and almost 100 pages, so we will do our best to distill the most important aspects of it.

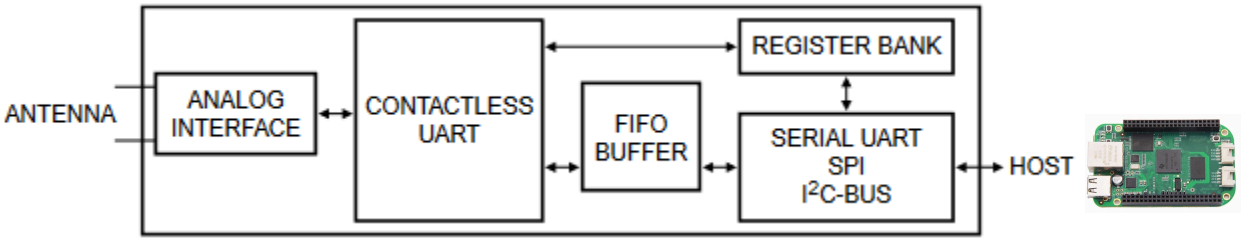
SPI Guide Provided Files

As mentioned previously, we will build off the SPI guide's included code. This includes the following information:

- Initializing SPI
- SPI data transfer
- Example code for reading/writing RC522 registers

One important thing to note is that the SPI guide's provided RC522 C code does not include initialization of the RFID module. This is required before one can begin to properly communicate with the RC522, which may be confusing for potential users. As such, we will use this guide to talk about the components one must understand in order to initialize the module.

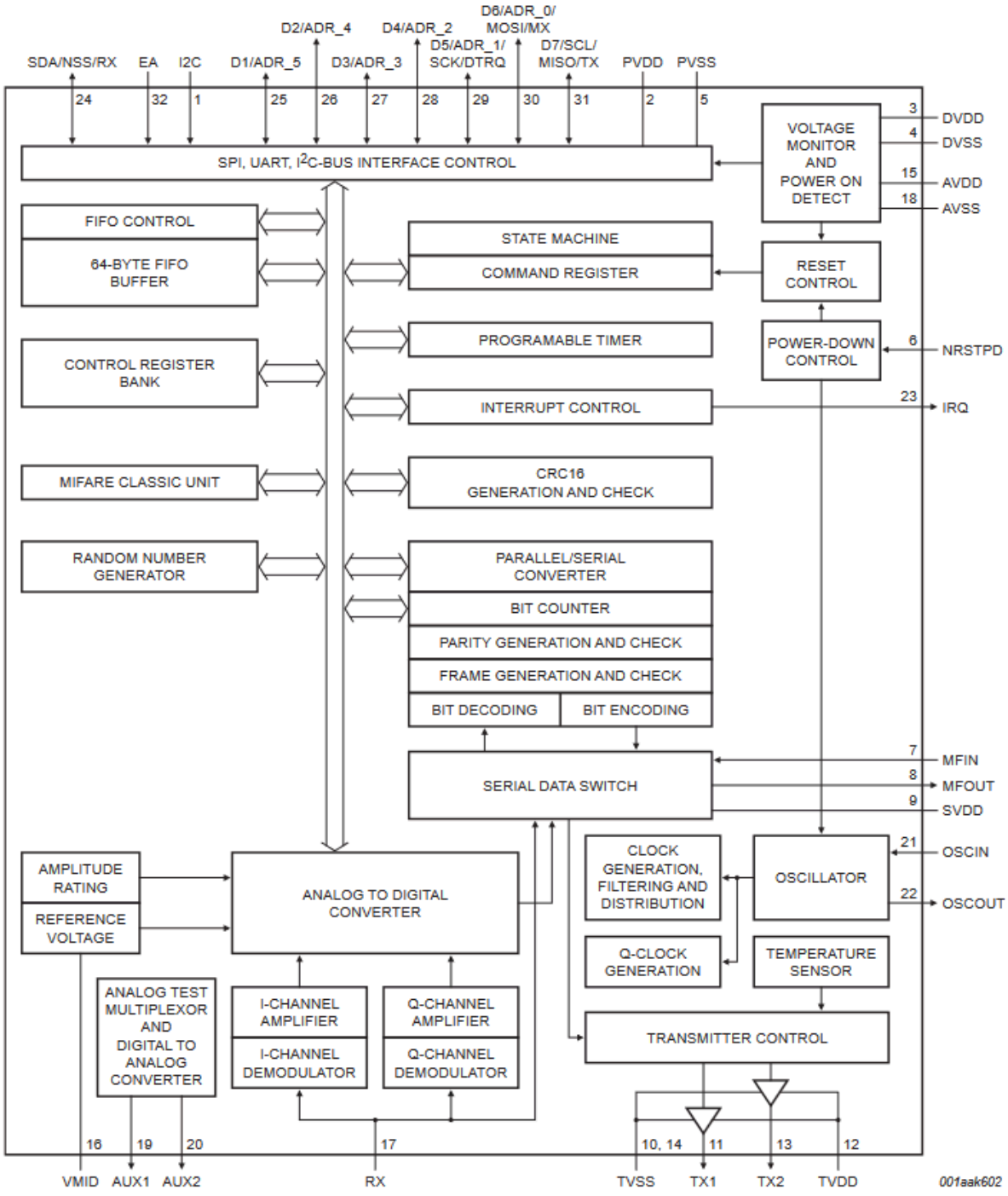
Understanding the RC522 Hardware Layout



Above is the "simplified block diagram" of the MFRC522 (see datasheet §6). As you can see, there are two primary ways that you, as the host, can interact with the module. One is the **Register Bank**, and the other is a **FIFO Buffer**.

The Register Bank is essentially a group of hardware registers that store bits of information (typically 1 byte total), where each bit determines a certain aspect of functionality for the module. When the host modifies bits in these registers, the controller (**Contactless UART** in the image) changes its behaviour according to what is depicted in the datasheet. Pages 35-70 (of 95) of the datasheet are entirely dedicated to laying out the functionality of each register's bits. In order to get your RC522 up and running, you have to modify the values of certain registers; more on that later.

The FIFO Buffer is used primarily as a means of bulk data transfer between host and controller. This is more efficient than using registers to get large amounts of information from the controller, because the alternative would be reading register information one byte at a time (one SPI request per byte). Therefore, when one is attempting to receive information from an RFID tag, the information is retrieved through the FIFO Buffer instead.



Above: A more detailed block diagram about each aspect of the device.

3. RC522 Communication

Initialization

Let's walk through the required steps for initializing the module.

Resets

The first thing to note when beginning work with the RC522 is that upon starting the device, the persistent storage on the registers will contain whatever they contained the last time it was used. Therefore, it is recommended to perform a reset on the device to make sure everything is in an expected state at the beginning of each program run.

There are two kinds of resets one can perform on the module. The first is a hard reset, used by driving current through the RST pin on the device. The second is by using the **SoftReset** (see §10.3) command in the **CommandReg**. For simplicity, we opted for the hard reset. To accomplish this on the BeagleBone, all that is required is to write a 0 then 1 to the RST GPIO pin. Make sure the GPIO pin is configured for "out" before doing so. *Note: This will also clear all data contained in the FIFO buffer.*
Here are an example set of commands for using P9.23 as RST via commandline:

```
config-pin P9_23 gpio
config-pin P9_23 out
echo 0 > /sys/class/gpio/gpio49/value
echo 1 > /sys/class/gpio/gpio49/value
```

The same commands can be run from a C program.

Timer

The RC522 has a timer unit, which is used for timing of certain communication events. This timer unit must be set up properly to enable data transmission with the device. This requires writing several values to **TModeReg**, **TPrescalerReg**, and both **TReload** registers. These values are calculated using several different formulas that are explained in §8.5 of the datasheet. This is one area where we will omit detail, as it would take a while to explain all the calculation steps. Here is what the C code looks like:

```
RFIDReader_writeReg(TModeReg,
(1 << TMODEREG_TAUTO_BIT) | (0x04 << TMODEREG_TPRESCALER_HI_BIT));
RFIDReader_writeReg(TPrescalerReg, 0x00);
RFIDReader_writeReg(TReloadRegH, 0x01);
RFIDReader_writeReg(TReloadRegL, 0x49);
```

Mode, ASK, TxControl

A few things are required for the ModeReg:

- ModeReg's **TxWaitRF** bit will make it such that the transmitter waits for the module to enable its. RF field before beginning data transmission.
- ModeReg's **PolMFin** bit will set the polarity of the MFIN pin, which is used to indicate that a signal is received.
- ModeReg's **CRCPReset** bit defines a preset value for the CRC coprocessor, which is used for CRC calculations utilized during data transmission. See datasheet §8.2.5 for more information on the CRC coprocessor.
- Using the predefined bit mask values makes this both easy to implement and understand for future programmers.

```
RFIDReader_writeReg(ModeReg,
(1 << MODEREG_TXWAITRF_BIT)
| (1 << MODEREG_POLMFIN_BIT)
 | (1 << MODEREG_CRCPRESET_BIT));
```

The **TxASKReg** is used to force 100% amplitude-shift keying (ASK) modulation, which basically makes it easier for the reader to differentiate between 1s and 0s in read signals (see Wikipedia for details).

```
RFIDReader_writeReg(TxASKReg, 1 << TXASKREG_FORCE100ASK_BIT);
```

One must also turn on the **TxControlReg's Tx2CW** bit to enable the RFID antenna.

```
uint8_t TxControl = RFIDReader_readReg(TxControlReg);
RFIDReader_readReg(TxControlReg, (TxControl | TXCONTROLREG_ANTENNA_ON_MASK));
```

FIFO Buffer

Once the aforementioned initialization steps have been taken, the next step is learning how to work with the FIFO Buffer. Because of the similarity with the provided readReg/writeReg functions, we will omit the steps (note that code for this is provided in the sample files). The general process is -- like reading/writing registers -- we pass in our buffers and desired length to send/receive, and using the **FIFODataReg** we signify whether we want to read or write. §8.3 of the datasheet explains in detail how the FIFO buffer is manipulated to read/write information. As was previously discussed, the FIFO buffer is necessary for receiving bulk data from nearby RFID tags.

There are also various status-related register bits that can be used to retrieve information about the FIFO buffer; including how many stored bytes, remaining capacity, etc.

4. Provided Sample Code

We have included some sample code which builds off the SPI guide's sample code, including RC522 initialization, and functions for working with the FIFO buffer. If the reader would like to work with the RC522, we recommend they understand the given code before running it. The associated sections of the datasheet are included as comments for clarity.

All that would be required to extend implementation for reading RFID tag information (i.e. UIDs) are the following steps:

1. Implementing the RC522's Transceive ability (see datasheet for details)
2. Using transceive with the correct register changes to get the appropriate information

5. Troubleshooting

- Should you encounter an issue with configuring your RST pin, make sure there are no other enabled device trees that are using that pin.
- If you encounter issues with the provided code, consult the references listed in the provided files.
- The RC522 should display a red LED when powered ON. Keep this in mind when making sure the device is operational.