RM00221

NTAG 5 - Android Application development

Rev. 1.0 — 9 January 2020 Reference manual

531810 COMPANY PUBLIC

Document information

Information	Content
Keywords	NTAG 5, Android, ISO/IEC 15693, NFC, Pass-Through, PWM, GPIO, I2C Master
Abstract	This application note describes the usage of the Android Application developed to interact with the NTAG 5. There are four different use cases defined to explore some of the new features added in this product.

Revision history

Rev	Date	Description
v.1.0	20200109	First official released version

1 Introduction

1.1 Objective

This document aims to describe the main functionalities of the Android application that demonstrates several features of the NTAG 5 tag.

The Android Application is divided in four use cases. This document describes each use case in detail, presenting the most important parts of the code snippets.

1.2 Target audience

This document is intended for Android developers who are working with the NTAG 5. NXP has created a reference Android Application to demonstrate four different use cases of the new tag. Thus, an Android and Java background is needed in order to have a complete understanding of the document.

2 NTAG 5 overview

2.1 General description

NTAG 5 is ISO/IEC 15693 and NFC Forum Type 5 Tag compliant IC with an EEPROM, SRAM and I^2C host interface. This ensures information exchange with all NFC Forum Devices with a tap. With this ability, the tag offers a long-reading range and privacy due to close proximity with mobile devices.

2.2 Features and applications

NTAG 5 is equipped with a reliable and robust memory. It has a 2048-byte EEPROM for the user memory and a 256-byte SRAM for frequently changing data and Pass-Through mode.

Regarding security there are several ways to protect the IC:

- 32-bit password to access the memory from NFC and I^2C interfaces.
- Optional 64-bit password only to read/write the tag from NFC perspective.
- 128-bit AES mutual authentication from NFC perspective before doing memory operations.

The NTAG 5 has a configurable contact interface with an I^2C standard slave mode (with 100 kHz and fast 400 kHz operating frequencies) and a transparent I^2C master channel. Other interesting features in the contact interface are the configurable event detection pin, two GPIO channels, two PWM channels and energy harvesting for low-power budget applications.

Apart from these features, NTAG 5 has several innovative features such as Active Load Modulation, NFC and I^2C Disable, Privacy mode and DESTROY command.

Taking into account the IC features, it can be used in quite different applications: Lighting, Smart home, Consumer, Industrial, Gaming or Smart metering.

3 NTAG 5 – Use cases definition in Android App

3.1 Use cases definition

3.1.1 I^2C master channel use case

This use case aims to show the I^2C master channel feature of the NTAG 5. Using I^2C Master interface, I²C slave devices like sensors or memories can be connected to the NTAG 5 without an external microcontroller. Using energy harvesting capability, an I²C device can be powered by NTAG 5.

The response from the I^2C slave to the commands will be stored in the SRAM and can be read afterwards from NFC perspective.

In this use case, the I^2C Master channel will be used to retrieve different data from the combo sensor (FXOS8700CQ) included in the FRDM-KW41Z board. Since the MCU is not used to retrieve the data from the sensor, this use case will not have a KW41Z counterpart, only the Android application use case.

A sequence diagram of the use case can be seen in Figure 1.

It is important to remark that there are several I^2C commands pre-defined to communicate with the combo sensor (like sensor configuration, get accelerometer or get magnetometer data). But there will also be the possibility to write custom commands to retrieve other data from the sensor. The corresponding code snippet section will explain the way these commands are formed.

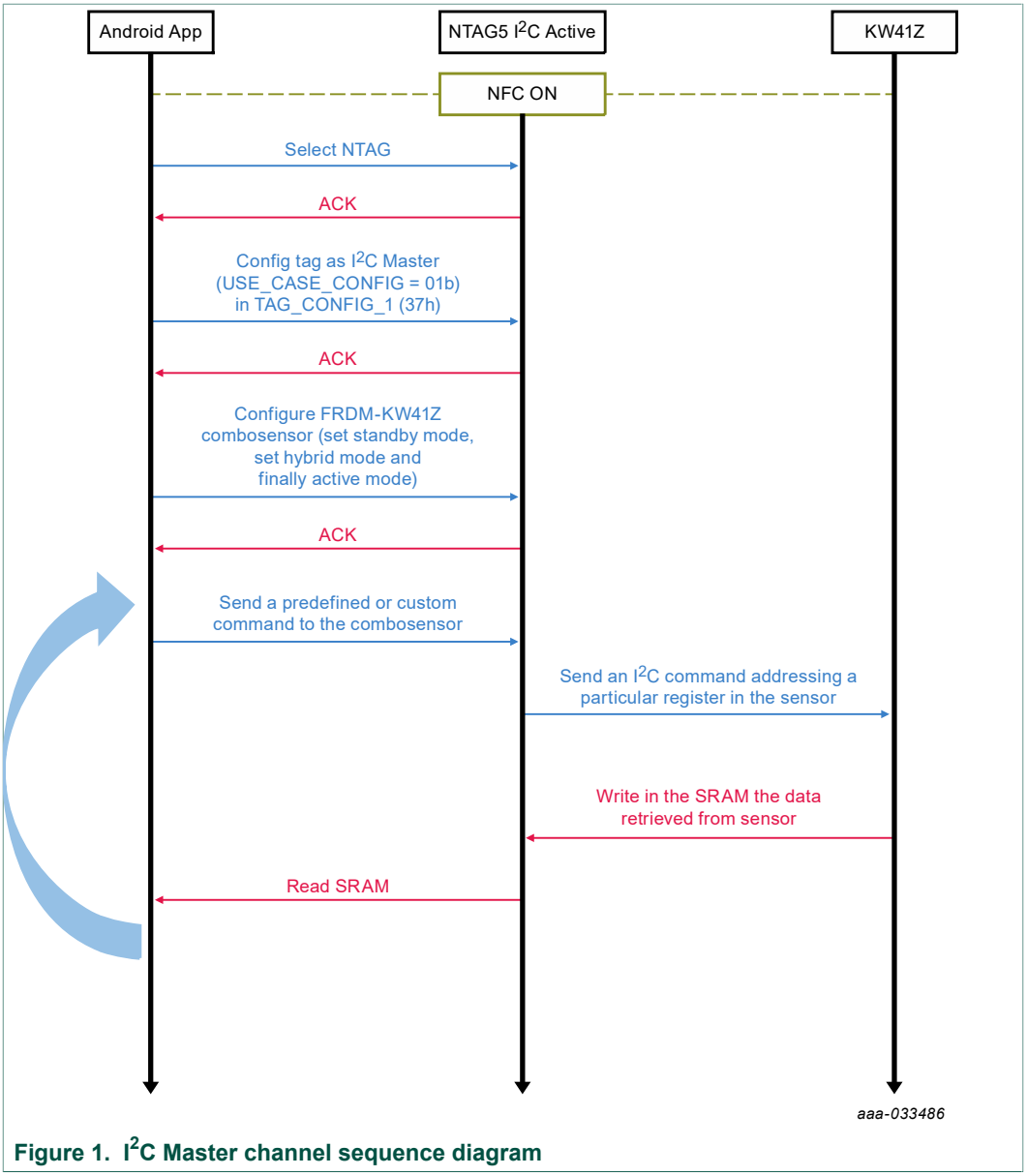

Figure 1. I²C Master channel sequence diagram

3.1.2 GPIO use case

This use case aims to demonstrate the GPIO feature of the NTAG 5. In this feature, it is possible to use the I²C lines as general-purpose input/output lines. The NTAG 5 has the I²C lines (SCL/SDA) multiplexed with two GPIO/PWM lines.

These GPIO lines are managed via configuration/session registers. In the input mode, the status of the pad will be available in the corresponding session bit. In the output mode, the status can be written in the corresponding session bit.

There are two different channels available to be configured as GPIO. In this use case, the channel 0 is configured as GPIO output to show the capabilities of the NTAG in this mode and channel 1 is configured as GPIO input.

For a better understanding of the use case, a sequence diagram is shown in Figure 2. First of all, the tag needs to be configured in GPIO/PWM mode. This is necessary

because the I^{2}C lines and the GPIO/PWM lines are multiplexed and the tag needs to select one of them. Once the IC is correctly configured, it is possible to read the configuration for the two GPIO channels.

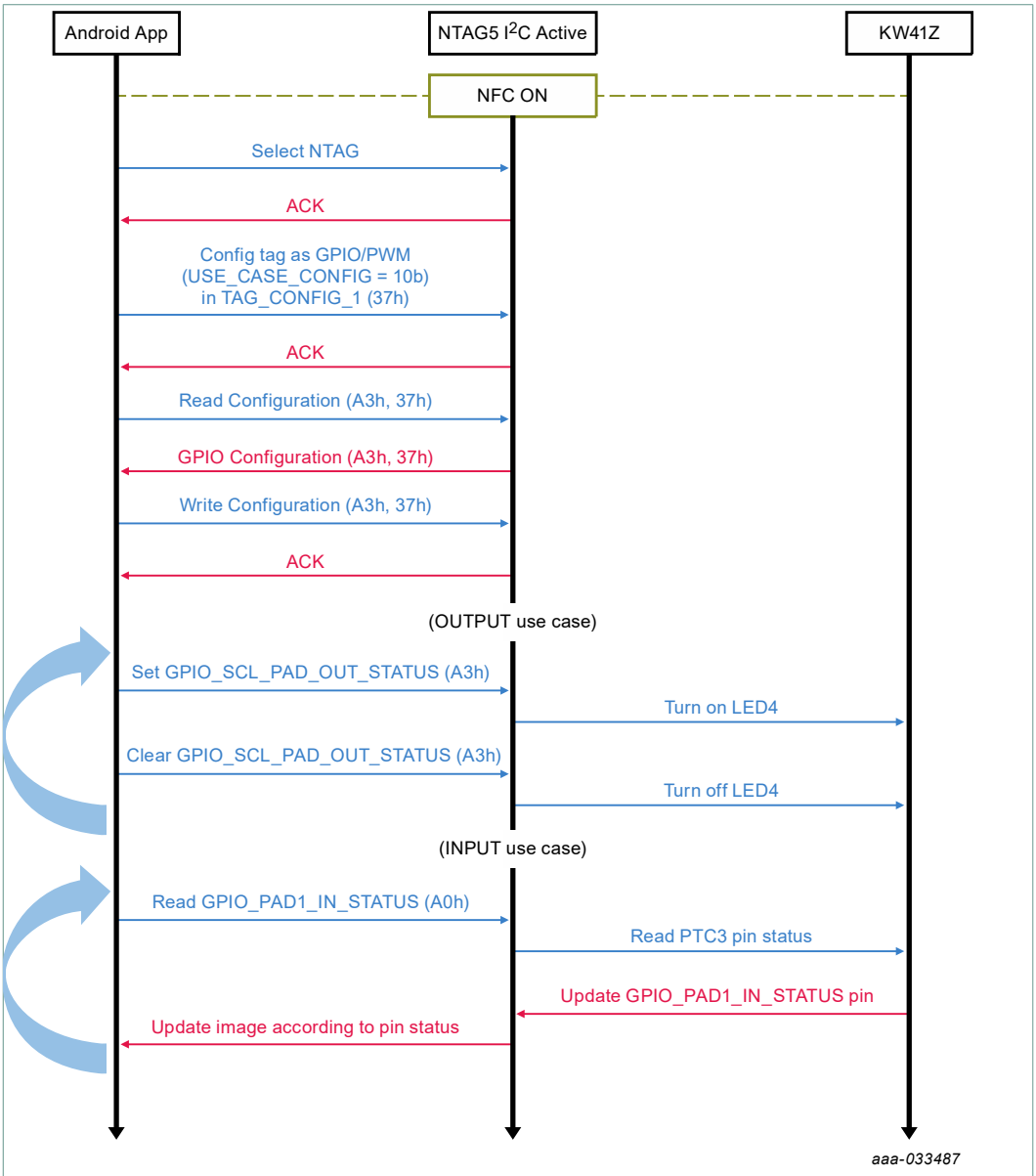

Figure 2. GPIO use case sequence diagram

3.1.2.1 GPIO output

In the GPIO output, the idea is to use the KW41Z board with the Android application. First, it is necessary to flash the FRDM-KW41Z board with the corresponding binary file. Then select the GPIO use case in the Android app menu. As previously stated, the GPIO output is configured in channel 0 (this line is multiplexed with the SCL line of the I^{2}C interface).

In the Android app: there is a bulb image at the left side of the screen. This image is clickable. Pushing the bulb image sets the output session register of the NTAG and then

LED4 in the FRDM-KW41Z board should be turned on. When clicking on 'Clear GPIO output' button, the value in the session register will be cleared and both LED4 in the board and bulb image in the apps will turn off.

Configuration: Channel 0 has the possibility to configure the slew rate of the signal. The slew rate is the speed a signal goes from high to low (and the opposite way). At a higher slew rate, it is possible to have some noise or distortion in the signal. At the frequency the NTAG is operating, there is not a large impact regarding noise or distortion in the signal. So it is recommended to configure the tag with High speed in the slew rate.

In the FRDM-KW41Z: the PTC2 pin is configured as an input pin and will trigger an interrupt at rising and falling edges of the pin. When there is a rising edge (i.e. the output session register in the NTAG is set to 1), the LED 3 will be illuminating in the KW41Z board. At a falling edge, the register will be set to 0 and the LED 3 will be turned off.

3.1.2.2 GPIO input

As previously stated for the GPIO output use case, it is necessary to flash the FRDM-KW41Z board with the GPIO binary file. Also select the GPIO use case in the Android app menu.

The GPIO input use case has the opposite behavior of the GPIO output. In this case, the Channel 1 is configured as input (SDA line of the I^2C interface). In the Android app, it will be necessary to configure the PAD and slew rate; the possible configurations will be explained later.

In the Android app: There is a button image at the left side of the screen. This button is **not** clickable. This button represents the status of the SW3 button on the FRDM-KW41Z board. Each time the SW3 button on the FRDM-KW41Z board is pushed, the session register of the NTAG is set to 1 and then the button of the Android app will also be displayed as pushed. On the other hand, when releasing the SW3 button in the FRDM-KW41Z board, the session register is cleared and the Android app will release the button of the image.

Configuration: Channel 1 has the possibility to configure the slew rate of the signal (the same recommendation as in Section 4.2.1 applies here) and the PAD configuration. Regarding the PAD configuration, there are four options. The first option '*Receiver disabled'* will disable the GPIO pin in the tag and the GPIO communication with the FRDM-KW41Z board will not be possible so this configuration should only be used in case the user wants to disable GPIO input communication. The configurations '*Plain input with weak pull-up*' and '*Plain input with weak pull-down*' are used when no external resistor is connected to the board and to avoid a floating pin. In case of having an external resistor connected to the board, it is recommended to use the 'Plain input' configuration and avoid weak pull-up/pull-down.

In the FRDM-KW41Z: the PTC3 pin is configured as an output pin. Each time the SW3 button is pushed, the PTC3 pin will set the corresponding session register of the NTAG. When SW3 button is released, the value of session register is cleared.

3.1.3 PWM use case

This use case aims to demonstrate the PWM generation capabilities of the NTAG 5. PWM signals can be used to control electrical devices. In this case, the PWM generated by the NTAG 5 will be used to control the brightness of two LEDs in the FRDM-KW41Z board.

There are two different PWM lines available in the NTAG 5, as previously mentioned. These PWM lines are multiplexed with the I^2C and GPIO lines.

Before running this use case, it is necessary to flash the FRDM-KW41Z board with the corresponding PWM binary file. In the Android app menu, the PWM use case option has to be selected.

In the Android app: in this use case, the screen displays both PWM channels with the same configuration options. As is shown in the sequence diagram in Figure 5, at first it is necessary to configure the IC with GPIO/PWM use case. Once the tag is correctly configured, it is possible to read the PWM configuration registers to check the existing data. It is also possible to write these registers again to set new parameters to the PWM channels.

The explanation of how the NTAG 5 configures a PWM signal can be found in section 8.3.4 in the product data sheet [Datasheet] and [Application Note].

In the FRDM-KW41Z: PTC2 and PTC3 pins are configured as PWM channel 0 and channel 1 respectively. Both pins also have an interrupt to turn on/off LED3 and LED4 according to the input value of the corresponding pin.

A sequence diagram of the use case can be found in Figure 3.

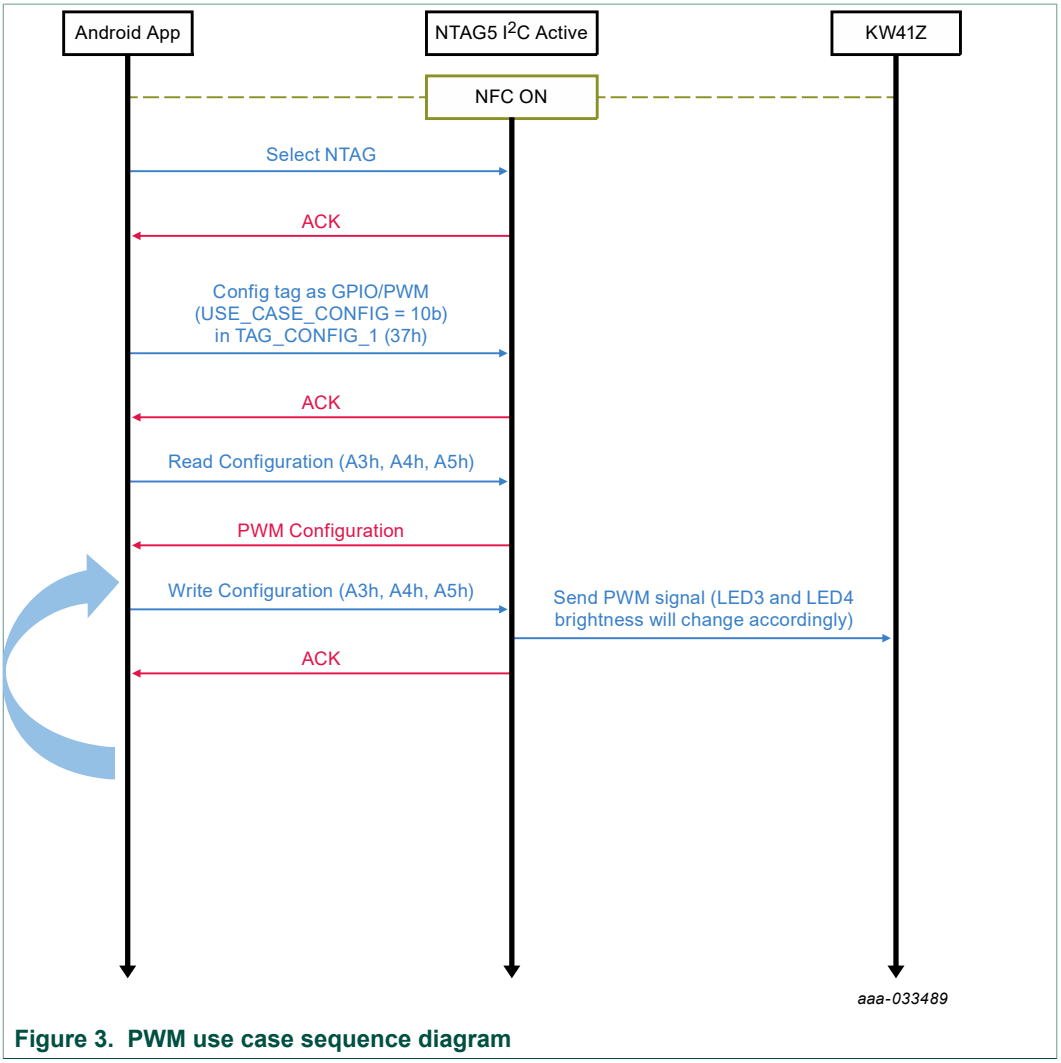

Figure 3. PWM use case sequence diagram

3.1.4 Pass-Through mode use case

This use case aims to show the Pass-Through mode functionality of the NTAG 5. The Pass-Through mode allows a faster data transfer between RF and I^2C interfaces using the 256-byte SRAM.

The Pass-Through mode allows data transfer in two directions: from RF to I^2C interface and from I^2C to RF interface. The NTAG 5 IC has an arbitration mechanism; this arbiter locks the interface that has to communicate with the tag according to the Pass-Through direction and SRAM_DATA_READY register. This mode uses the Event Detection pin of the NTAG 5 to detect SRAM events.

Since the SRAM is activated in this use case, it is mandatory to connect the board to the Vcc power supply.

More information about Pass-Through mode can be found in sections 8.1.4.1, 8.1.5 and 8.4.5 of [link Datasheet, boost Datasheet].

The use case is started in the Android app by clicking the 'Start Demo' button in the Pass-Through view.

The main functionality of this use case is to write the SRAM five times in a Pass-Through direction, then configure it again to write five times in the opposite Pass-Through direction and repeat this process until the 'Stop Demo' button is pushed in the Android application.

The Pass-Through direction is *automatically changed* in the FRDM-KW41Z board each time the Pass-Through mode has done five operations in the corresponding direction.

The default Pass-Through direction is RF->I^2C. This direction is indicated with a blue color in the LED4 in the FRDM-KW41Z board. When the tag is configured with I^2C->RF direction the LED4 will illuminate in green.

In the Android app: the usage of the Android app is straightforward in this use case. To start using the demo, it is necessary to click the 'Start Demo' button in the Android view. This will start writing in the RF->I^2C direction; after a loop of five operations, the FRDM-KW41Z board will toggle the direction to I^2C->RF and the Android app will automatically switch to this mode as well. The application will be looping between both Pass-Through directions until the 'Stop Demo' button is clicked in the application.

Important: To run this use case correctly, it is necessary to connect the Event Detection wire to the PTC17 pin in FRDM-KW41Z board.

3.1.4.1 RF→ I^2C direction

This is the default direction of the Pass-Through mode for the FRDM-KW41Z board. When the board is flashed with the corresponding binary file, a blue LED is turned on; this blue LED indicates the Pass-Through direction.

At startup, the FRDM-KW41Z is waiting for a valid NFC connection. When the phone taps the antenna, an event is triggered in the FRDM-KW41Z and the Pass-Through configuration is written in the IC. In this direction, the SRAM is written via RF and read via I^2C.

When the tag is configured, the arbiter locks the RF interface. When the RF interface is locked the SRAM can only be written via RF. When the last byte of the SRAM has been written, the arbiter releases the RF interface and locks the I^2C interface, then the data can be read via I^2C.

This use case is illustrated in Figure 4.

In the FRDM-KW41Z: The Pass-Through configuration sets the following data in the TAG_CONFIG_1_REG: ARBITER_MODE = 10b, SRAM_MAPPING = 1b and PT_TRANSFER_DIR = 1b. More information about the Pass-Through configuration can be found in the product [link Datasheet, boost Datasheet].

In this Pass-Through direction, the FRDM-KW41Z board configures the 'LAST_BYTE_WRITTEN_BY_RF' event; this event is triggered when the last byte of the SRAM has been written by RF. When this happens, the I^2C interface is locked and the SRAM_DATA_READY bit is set. This means that the SRAM is ready to be read via I^2C.

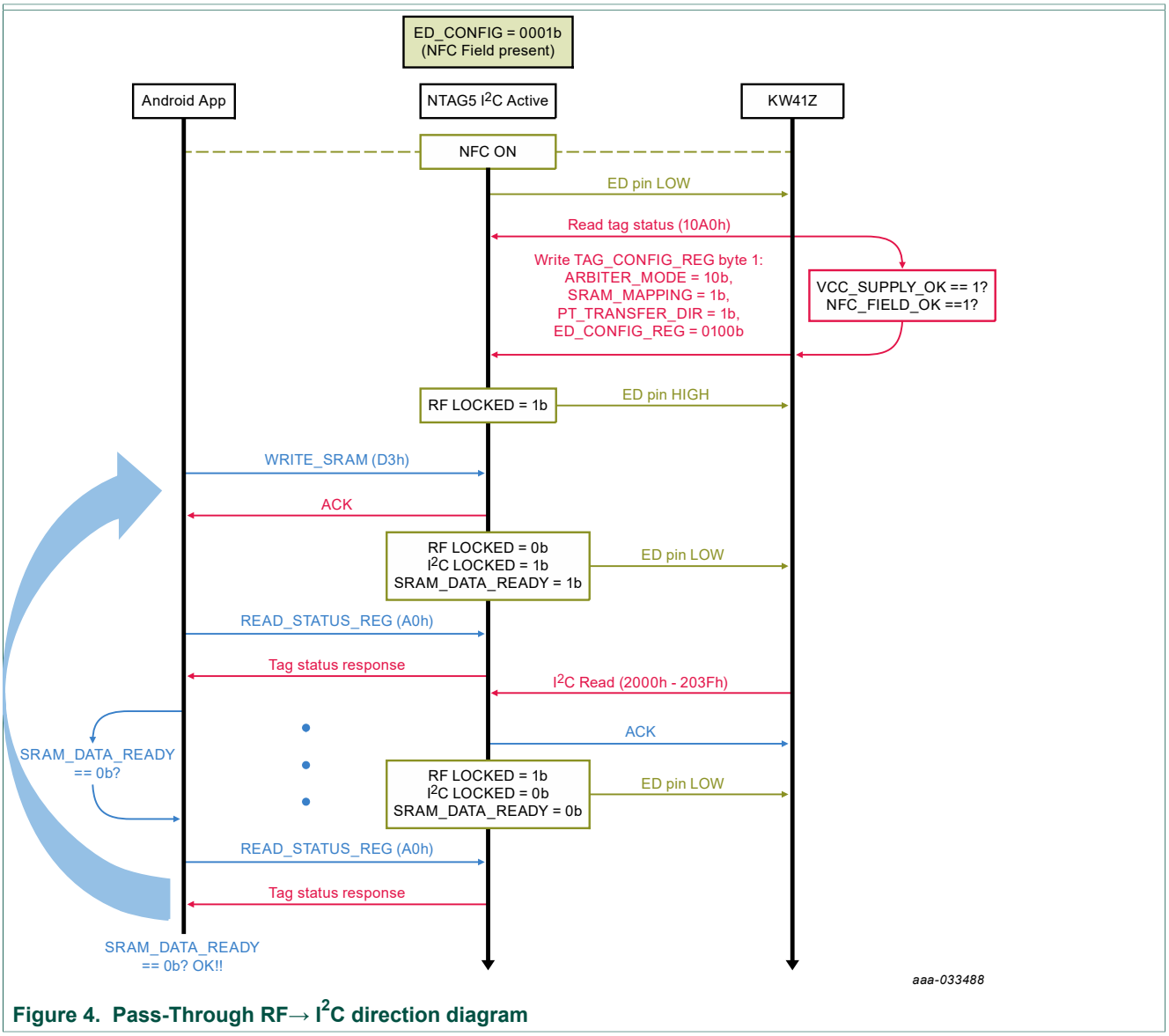

Figure 4. Pass-Through RF→ I^2C direction diagram

3.1.4.2 I^2C → RF direction

When this direction is set in the FRDM-KW41Z board, a green light is turned on in LED4.

This use case is illustrated in Figure 5.

In the FRDM-KW41Z: The Pass-Through configuration sets in the TAG_CONFIG_1_REG the following data: ARBITER_MODE = 10b, SRAM_MAPPING

= 1b and PT_TRANSFER_DIR = 0b. More information about the Pass-Through configuration can be found in the product [Data sheet] and [Application note].

In this case, the firmware configures the 'LAST_BYTE_READ_BY_RF' event; this event is triggered when the last byte of the SRAM has been read by RF. When this happens, the I^2C interface is locked and the SRAM_DATA_READY bit is cleared. This means that the SRAM is ready to be written via I^2C.

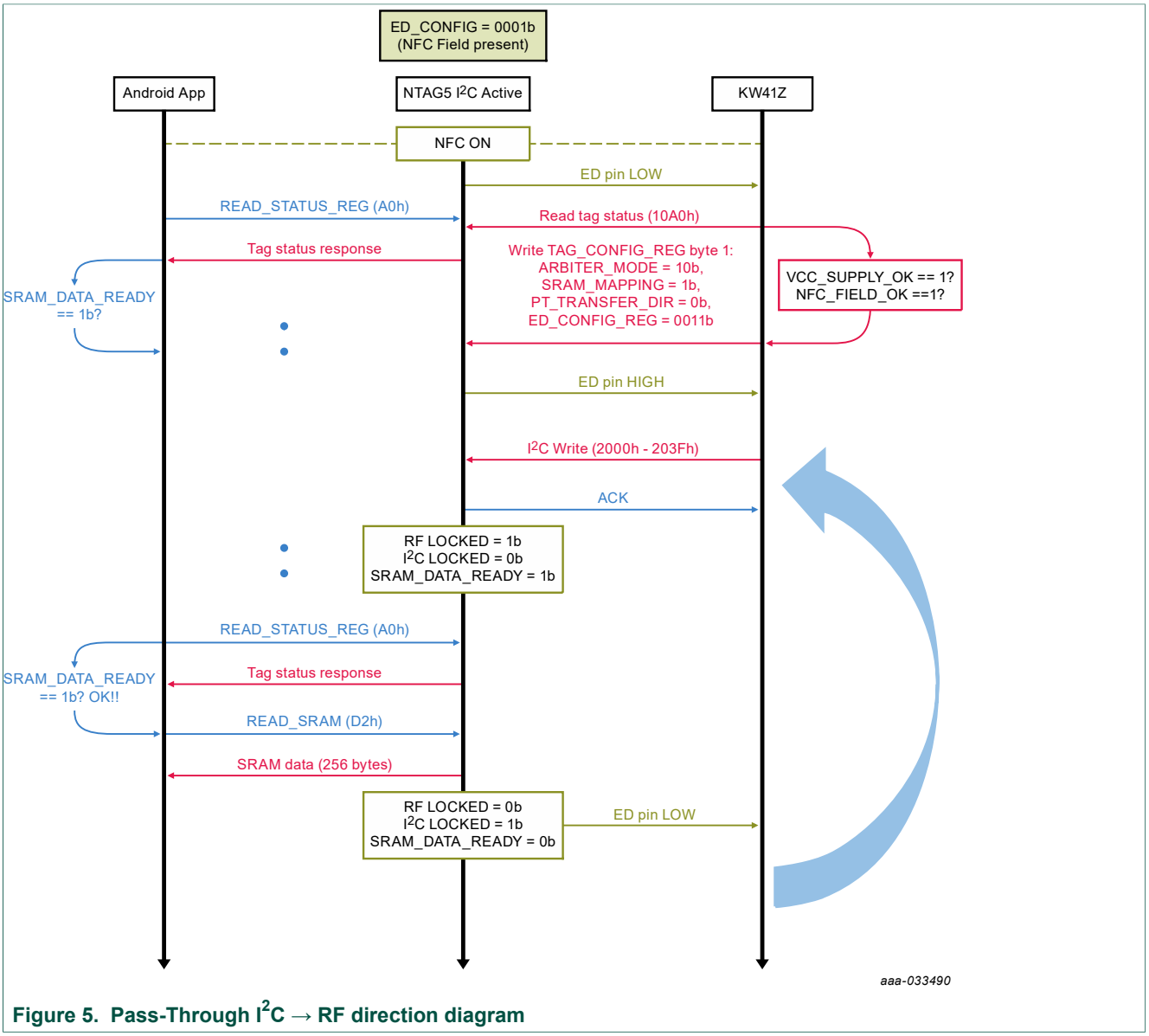

Figure 5. Pass-Through I^2C → RF direction diagram

3.1.5 ALM use case

The Active load Modulation use case is intended to show the capabilities of the NTAG 5 to change the ALM parameters through the FRDM-KW41Z and read the updated configuration using the Android App. The parameters that can be configured are listed in [Data sheet]. Figure 8 shows the diagram of this use case.

The configuration of ALM parameters is carried out using the Peek and Poke tool. Refer to Section 6 for instructions on how to run the use case from this tool.

In Peek and Poke: Run the ALM use case and start by clicking the “Write configuration” button. This will save the default values in the NTAG. Then, configure the parameters using the text boxes and menus available and click on “Write Configuration” so the NTAG can be configured.

In Android App: Go to the ALM use case menu and click on the “Read ALM Configuration”. The parameters will update accordingly to what has been configured in Peek and Poke.

You can use the “Read Configuration” button of the ALM menu in Peek and Poke to check that the configuration read matches the one in the Android App.

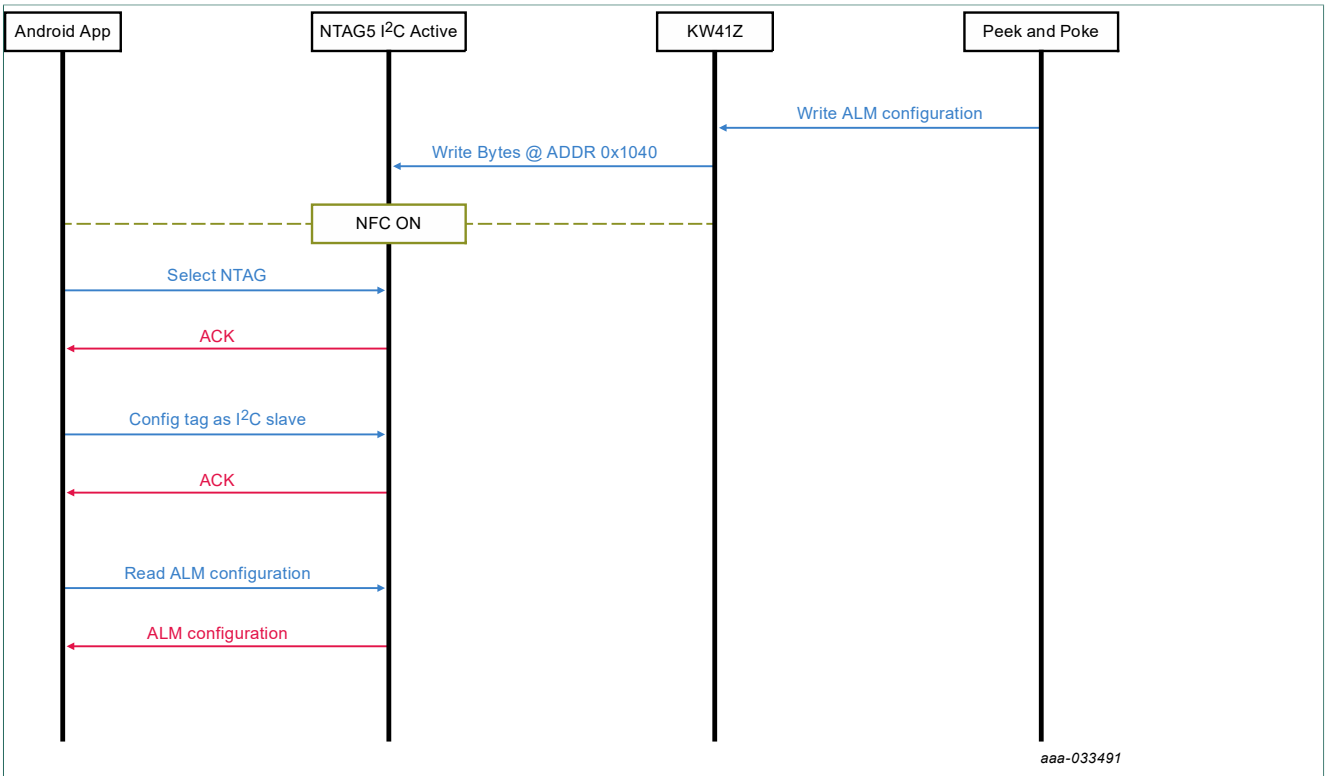

Figure 6. ALM use case sequence diagram.

3.2 User Guide – NTAG 5 configuration

The NTAG 5 is configured in a different way depending on the use case to be demonstrated. All the different configurations can be found in Table 1.

Table 1. NTAG 5 – NTAG 5 configurations

Use case	Configuration	Bits 5 and 4 in TAG_CONFIG_1
NDEF message	I^2C Slave	00b
GPIO	GPIO/PWM	10b
PWM	GPIO/PWM	10b
I^2C Master channel	I^2C Master	01b

Use case	Configuration	Bits 5 and 4 in TAG_CONFIG_1
Pass-Through mode	I^2C Slave	00b

More information about the tag configuration is illustrated in section 8.1.3.13 of the product data sheet [Datasheet].

The Android application includes several options in the action bar regarding the tag configuration. This action bar menu can be observed in Figure 7. There is one option to read the current configuration in the tag and also another three options to configure the tag in I^2C slave, I^2C master and GPIO/PWM modes.

Every time the tag is configured in a different mode it is necessary to reset the IC so that the configuration takes place. **The tag can be reset by removing its actual power source (RF field or wired connection) and then reconnected.**

Figure 7. Android app - Tag configuration

3.3 User Guide – NFC connection with board

Before using the application, it is necessary to activate the NFC capabilities in the smartphone (it is mandatory to have NFC capabilities in the phone).

Once the NFC communication is activated in the phone, the first step is to tap the phone to the NTAG 5 board. When the connection is correctly established, a pop-up will be displayed at the bottom of the screen saying that the connection has been successful and the NTAG 5 board is ready to be used.

More details about how to use each use case can be found in the corresponding use case explanation in Section 3.1.

4 NTAG 5 – Use cases code snippets

4.1 General information and project structure

The main objective of this reference application is to clearly organize each use case to ease code migration to other projects.

The Android Application has been created in Android Studio 3.0.1 with target SDK version 27.

As mentioned above, it is structured in four separated modules. Each module is developed in a separate Activity accessible from the main menu. In Figure 8, you can find the project structure from Android Studio.

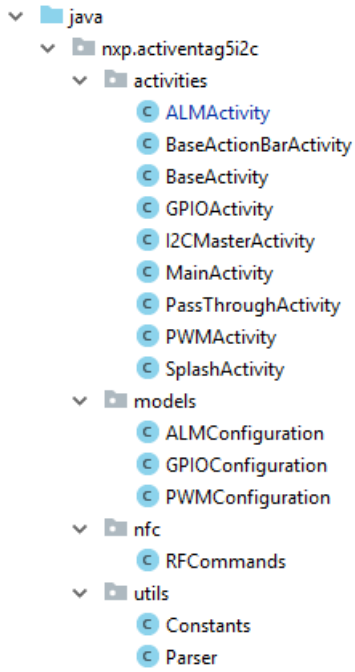

Figure 8. Android Studio project structure

The application has been designed following a layered structure: one layer for the NFC communication (contained in *BaseActivity class*), another layer to operate with the action bar (class *BaseActionBarActivity*) and finally the application logic and UI which are contained in the different activities for each use case (*PassThroughMode*, *GPIOActivity*, *I^2CMasterActivity* and *PWMActivity*).

The class '*RFCommand*' has been created in the package called '*nfc*'. This class contains all the necessary RAW commands (following ISO/IEC 15693 spec) to communicate with the tag. Also it has been necessary to develop a Parser class to get and set all the tag parameters at a bit level. In the future, it is planned to have TapLinx support for the NTAG 5. In the meantime, it is necessary to communicate with the tag by using RAW commands.

4.1.1 BaseActivity

As previously stated, *BaseActivity* is in charge of managing the NFC communication. At first, when the application is opened, it is necessary to check if the NFC is enabled in the phone; otherwise, the application warns the user to turn it on.

The application is restricted to work with ISO/IEC 15693 ICs. Hence, in case another IC type is read, it will detect that it is not a vicinity IC and the application will not be able to operate. In Figure 8 you can check how the IC is detected and selected.

```
@Override
protected void onNewIntent(final Intent intent) {
    super.onNewIntent(intent);

    tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    Log.d(TAG, msg: "Card ID: " + Utils.byteArrayToHex(tag.getId()));

    String[] techList = tag.getTechList();

    //Check that the discovered tag is a vicinity tag
    if (techList[0].equals("android.nfc.tech.NfcV")) {
        byte[] tagUid = tag.getId();

        nfcvTag = NfcV.get(tag);

        //ISO/IEC 15693 tags can be operated in two modes:
        // Select mode and Addressed mode.
        //To work in the select mode it is needed to send a SELECT
        // command at the beginning of communic.
        //In the address mode, the tag UID is sent within each command.
        //This application works in SELECT MODE.
        byte[] select_command = RFCommands.cmd_select;
        System.arraycopy(tagUid, i: 0, select_command, i1: 2, i2: 8);

        if (nfcvTag != null) {
            try {
                nfcvTag.connect();
                byte[] select_respo = nfcvTag.transceive(select_command);
                Log.d(TAG, msg: "Select response: " +
                        Utils.byteArrayToHex(select_respo));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figure 9. Tag detection in BaseActivity

As commented in the code snippet in Figure 9, ISO/IEC 15693 ICs can work in two different modes: addressed and selected.

In the selected mode. it is necessary to first select the IC using the desired UID. Once the IC is selected, it is not necessary to include the UID in the subsequent commands.

In the addressed mode, the IC is not selected at the beginning and each command has to include the UID of the desired IC.

This Android application works in selected mode.

BaseActivity class contains just one method to communicate with the IC using the '*transceive'* method from the NFC API in Android. You can see this method in Figure 10.

Calling the '*transceive*' method will return the IC response to the command. This response is stored and returned each time the '*sendCommand*' method is called.

```
/**
 * This method sends RF commands to the connected NFC-V tag,
 * the command is included as parameter
 * the operation will be useful in MainActivity to distinguish
 * the operations in different fragments
 *
 * @param command
 */
public byte[] sendCommand(byte[] command) {
    byte[] response;
    try {
        response = nfcvTag.transceive(command);
        Log.d(TAG, msg: "command response: " + Utils.byteArrayToHex(response));
    } catch (Exception e) {
        e.printStackTrace();
        response = null;
    }
    return response;
}
```

Figure 10. Send command method in BaseActivity

4.1.2 BaseActionBarActivity

This activity will extend from *BaseActivity* and will only take care of the action bar.

In this application, the action bar has several options to configure the tag. It can be configured to the different use cases: I^2C Slave, I^2C Master and GPIO/PWM. It can also retrieve the current configuration in case the user does not remember the configuration that was previously set.

4.1.3 MainActivity

This activity is very simple; it will contain a menu view to navigate through the different use cases. As this activity extends from *BaseActionBarActivity* (which extends from *BaseActivity*) it will be the link between the UI and NFC communication. Therefore, all the particular activities for each use case will extend from this activity to have the possibility to send NFC commands and update the UI in a simple way.

Figure 11 shows a screenshot of the main menu.

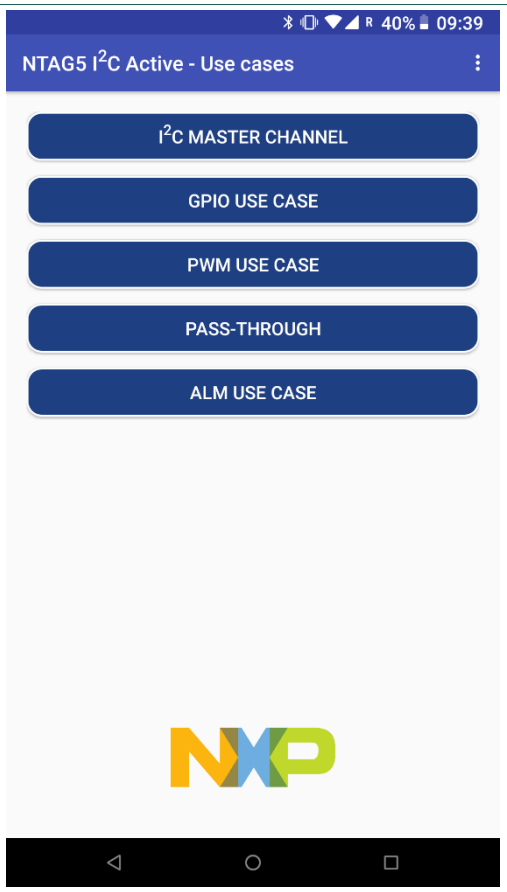

Figure 11. Android App Main menu

4.2 Use cases

4.2.1 I²C Master Channel

The I²C Master Channel screen view is displayed in Figure 12. To use this feature, it is necessary to configure the IC in I²C Master Mode. This configuration can be easily done in the Android app by using the corresponding option in the Action bar.

Once the tag is correctly configured, there are two different ways to send I²C commands to the tag: using a pre-defined command or writing a custom command. Each option will be explained in a special section.

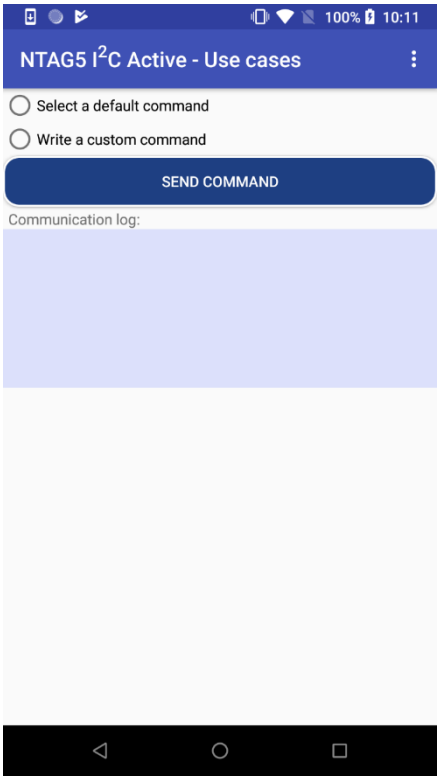

Figure 12. I²C Master Configuration view

4.2.1.1 I²C Master channel communication sequence

NTAG 5 has two commands to communicate in this mode: Write I²C and Read I²C. These commands will be explored in more detail in section 8.2.4.9 in the product [Data sheet].

The I²C Address of the FXOS8700CQ combo sensor is 0x1F. This information will be needed to form the Write and Read I²C Commands in the application.

The I²C Master communication has to follow the sequence explained below:

1. Send Write I²C Command (D4h): in the '*I²C param*' byte. It is necessary to set the '*Disable Stop condition*' because right after a write I²C command, a read I²C command will be sent. Finally, the register to be read from the combo sensor (Section 14 in the FXOS8700CQ [Data sheet]) has to be included in the '*Data*' byte. More information about this command can be found in section 8.2.4.9.1 in the NTAG 5 [Data sheet].

As an example, a write I²C command to get the temperature from the sensor is shown in Figure 13.

```
public static final byte[] cmd_GetTempI2CMasterCommand = new byte[]{
        (byte) 0x12, // FLAGS
        (byte) 0xD4, //WRITE I2C
        (byte) 0x04,
        (byte) 0x9F, //Generate Stop condition
                     //I2C address = Stop condition bit | 0x1F = 0x9F
        (byte) 0x00,
        (byte) 0x51  // Temperature register in combosensor
};
```

Figure 13. I²C Master Write I²C sample

1. Send Read I²C Command (D5h): in this case, in the '*I²C param*' byte, it is necessary to generate a stop condition because any other I²C command will follow this Read I²C Command. Thus, in this case the '*Disable Stop condition*' parameter will be 0.

Figure 14 shows a Read I²C command.

```
public static final byte[] cmd_readI2CMasterCommand = new byte[]{
        (byte) 0x12, // FLAGS
        (byte) 0xD5, // READ I2C
        (byte) 0x04,
        (byte) 0x1F, //Don't generate Stop condition
                     //I2C address = 0x1F
        (byte) 0x00
};
```

Figure 14. I²C Master Read I²C sample

1. When the Read I²C command is correctly sent, the data retrieved from the sensor is stored in the SRAM. Then it is necessary to send a Read SRAM command (D2h) to get the data from the sensor.

Figure 15 shows a full I²C Master communication, including the commands sent in the sequence shown above.

```
private void getTempI2CMaster() {
    try {
        writeSendLog(cmd_GetTempI2CMasterCommand);
        byte[] response = sendCommand(cmd_GetTempI2CMasterCommand);
        writeReceiveLog(response);

        writeSendLog(cmd_readI2CMasterCommand);
        byte[] response2 = sendCommand(cmd_readI2CMasterCommand);
        writeReceiveLog(response2);

        writeSendLog(cmd_readSRAM);
        byte[] responseSRAM = sendCommand(cmd_readSRAM);
        writeReceiveLog(responseSRAM);

    }catch (Exception e) {
        e.printStackTrace();
        Snackbar.make(findViewById(android.R.id.content), text: "Operation interrupted! Please try again",
                .show();
    }
}
```

Figure 15. Get Temperature I²C Master channel

4.2.1.2 I²C Master predefined commands

Predefined commands can be observed in Figure 16; all of these commands can be found in the '*RF_Commands*' class:

- Config sensor: It is mandatory to send this command first. It will set the sensor in the FRDM-KW41Z to the most appropriate configuration. It sets the sensor in hybrid mode (so the accelerometer and magnetometer are active at the same time).
- Get Temperature: This command retrieves the room temperature.
- Get X, Y and Z axis from the accelerometer.
- Get X, Y and Z axis from the magnetometer.

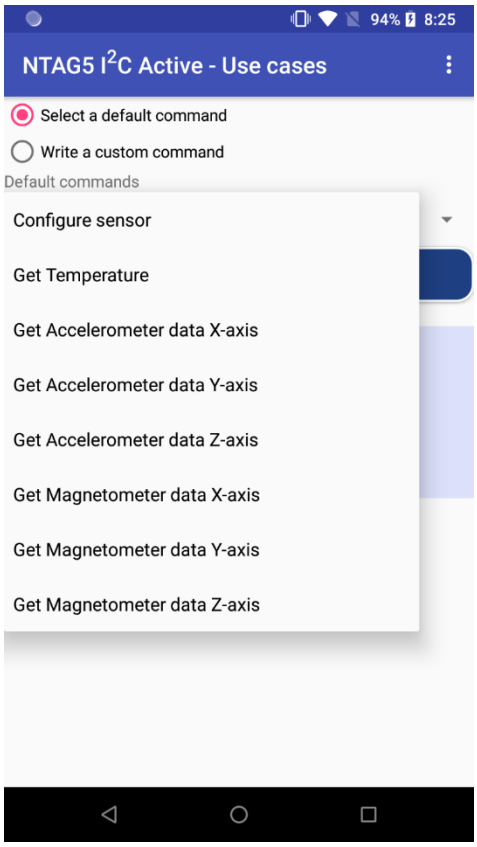

Figure 16. I²C default commands list

4.2.1.3 I²C Master custom commands

Custom I²C Master Command view is shown in Figure 17.

To send a custom command, the user needs to write in the EditText box the Write I²C command according to the NTAG 5 [Datasheet]. The way a Write I²C command is formed was explained in section 4.2.4.1. For example, to get the temperature from the sensor, the value '12D4049F0051' has to be written in the EditText box. This is the same command shown in section 4.2.4.1.

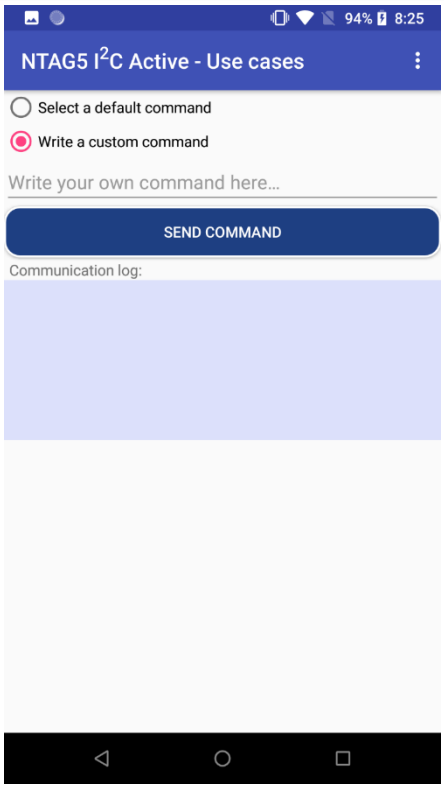

Figure 17. I²C Master custom command

4.2.2 General Purpose Input/Output (GPIO)

This code snippet works in conjunction with the respective binary for the FRDM-KW41Z board. The Android application has two channels to configure (like in the PWM use case). In this case, channel 0 is fixed as a GPIO output and channel 1 is fixed as GPIO input. For further explanation of how this use case works, please go to section 3.1.1

Figure 18 shows the screen view of the GPIO use case.

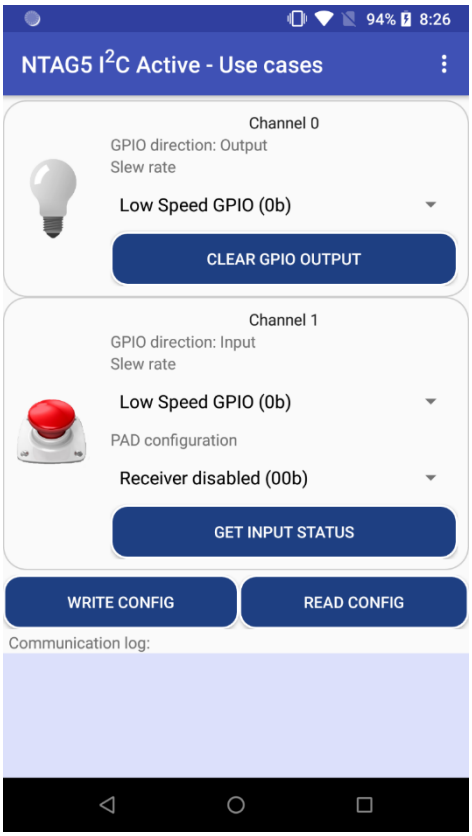

Figure 18. GPIO configuration fragment view

Before operating with the GPIO channels, it is necessary to read the configuration of the GPIO.

It is also important to remember that GPIO/PWM mode has to be configured in the IC. There is an option in the action bar to check which use case is currently configured in the tag.

Figure 19 shows the code snippet for the Read GPIO registers method.

```
private void readGPIO() {
    try {
        writeSendLog(cmd_readGPIOPWMConfig);
        byte[] response = sendCommand(cmd_readGPIOPWMConfig);
        writeReceiveLog(response);

        writeSendLog(cmd_readTagConfig);
        byte[] response2 = sendCommand(cmd_readTagConfig);
        writeReceiveLog(response2);

        Snackbar.make(findViewById(android.R.id.content), text: "Tag correctly read",
                .show();

        gpioConfigurationList = Parser.parseGPIORead(gpioConfigurationList,
                response, response2);

        imageGPIO_1.setImageResource(R.drawable.light_off);

        setUIElements();

        textLog.setText(logTextGPIO.toString());
```

Figure 19. Read GPIO

As explained in the use case definition section, it is recommended to configure the IC in ‘High speed’ slew rate for both channels. For the PAD configuration, the best option depends on if there is an external resistor connected to the board. In this case, the most common configuration is ‘weak pull-up’. Once the configuration has been set in the UI, you can click in ‘Write config’ button and it will be written in the EEPROM. This write command is shown in Figure 20.

```
private void writeGPIO() {
    byte[] tagConfig = Parser.getTagConfig(gpioConfigurationList);
    byte[] tagSession = Parser.getTagSession(gpioConfigurationList);

    try {
        writeSendLog(cmd_writeGPIOConfig);
        byte[] response = sendCommand(cmd_writeGPIOConfig);
        writeReceiveLog(response);

        writeSendLog(cmd_writeGPIOSession);
        byte[] response2 = sendCommand(cmd_writeGPIOSession);
        writeReceiveLog(response2);

        writeSendLog(tagConfig);
        byte[] response3 = sendCommand(tagConfig);
        writeReceiveLog(response3);

        writeSendLog(tagSession);
        byte[] response4 = sendCommand(tagSession);
        writeReceiveLog(response4);

        Snackbar.make(findViewById(android.R.id.content),
                text: "Tag correctly written", Snackbar.LENGTH_SHORT)
                .show();

        textLog.setText(logTextGPIO.toString());
```

Figure 20. Write GPIO config

For the output channel, each time the bulb image is pushed, the Android app will set the '*GPIO_SCL_PAD_OUT_STATUS'* bit in the 'PWM_GPIO_CONFIG_REG' register. This is shown in the method 'pushGPIOOutput' in Figure 21.

```
private void pushGPIOOutput() {
    try {
        writeSendLog(cmd_gpioSetSessionOutput);
        byte[] response = sendCommand(cmd_gpioSetSessionOutput);
        writeReceiveLog(response);

        Snackbar.make(findViewById(android.R.id.content),
                text: "Register correctly set", Snackbar.LENGTH_SHORT)
                .show();

        imageGPIO_0.setImageResource(R.drawable.light_on);

        textLog.setText(logTextGPIO.toString());
```

Figure 21. Set GPIO output channel 0 register

After calling the 'pushGPIOOutput' method, LED 4 in the FRDM-KW41Z is turned on. To turn off this LED and clean the '*GPIO_SCL_PAD_OUT_STATUS'* bit in the register, it is necessary to click on the 'Clean GPIO output' button, which is also contained in the channel 0 options. This method, shown in Figure 22, will clean up this bit.

```
private void clearGPIOOutput() {
    try {
        writeSendLog(cmd_gpioClearSessionOutput);
        byte[] response = sendCommand(cmd_gpioClearSessionOutput);
        writeReceiveLog(response);

        Snackbar.make(findViewById(android.R.id.content),
                text: "Register correctly cleared", Snackbar.LENGTH_SHORT)
                .show();

        imageGPIO_0.setImageResource(R.drawable.light_off);
        textLog.setText(logTextGPIO.toString());
```

Figure 22. Clear GPIO output channel 0 register

For the input channel, the behavior is the opposite. Instead of pushing a button in the Android app and turning on an LED in the FRDM-KW41Z, the button SW3 in the FRDM-KW41Z has to be pushed to set the '*GPIO_PAD1_IN_STATUS*' bit in '*STATUS1_REG*' session register. The status of this register is checked each time the user clicks on 'Get input status' in the channel 1 configuration, when the '*GPIO_PAD1_IN_STATUS'* bit is set, the button in the Android app will be shown as pushed and when the bit is cleared, the button in the image will be released. Figure 23 shows the method 'getGPIOInput' to get the status of the tag.

```
private void getGPIOInput() {
    try {
        writeSendLog(cmd_readTagStatus);
        byte[] response = sendCommand(cmd_readTagStatus);
        writeReceiveLog(response);

        Snackbar.make(findViewById(android.R.id.content),
                text: "Tag correctly read", Snackbar.LENGTH_SHORT)
                .show();

        boolean gpioInput = Parser.parseGPIOInput(response);

        //Get value of the gpio status passed as bundle and load image of
        // the corresponding bulb (on/off)
        if (gpioInput) {
            imageGPIO_1.setImageResource(R.drawable.pressed_button);
        } else {
            imageGPIO_1.setImageResource(R.drawable.unpressed_button);
        }
        textLog.setText(logTextGPIO.toString());
```

Figure 23. Get GPIO input channel 1 status

4.2.3 Pulse Width Modulation (PWM)

In Figure 24, you can see the Pulse Width Modulation (PWM) view. It is formed by two channels; each channel is configured with the same parameters: Resolution, Start time, Duty Cycle and Prescalar configuration. With the Prescalar configuration and Resolution, the user can set different frequencies for the PWM signal. For more information about PWM signals and its parameters, please check section 8.3.4 in the [Datasheet].

As a reminder, to use the PWM functionality, it is necessary to flash the FRDM-KW41Z board with the appropriate binary file.

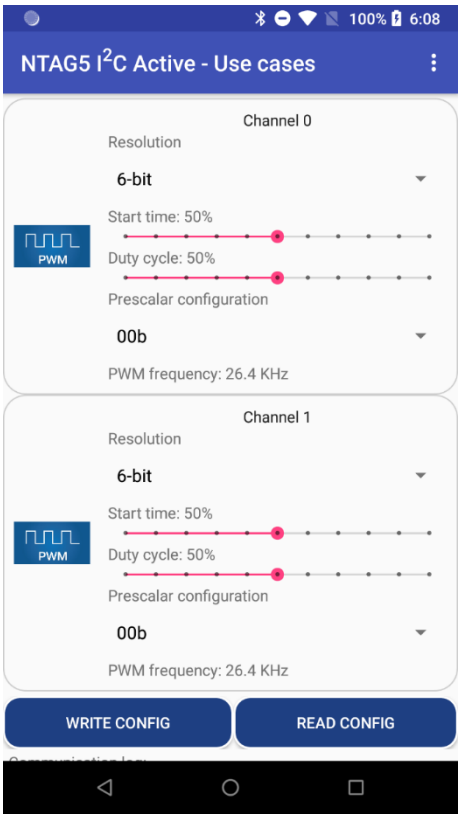

Figure 24. PWM configuration activity view

```
private void readPWM() {
    try {
        writeSendLog(cmd_readGPIOPWMConfig);
        byte[] response = sendCommand(cmd_readGPIOPWMConfig);
        writeReceiveLog(response);

        writeSendLog(cmd_readPWM0Reg);
        byte[] response2 = sendCommand(cmd_readPWM0Reg);
        writeReceiveLog(response2);

        writeSendLog(cmd_readPWM1Reg);
        byte[] response3 = sendCommand(cmd_readPWM1Reg);
        writeReceiveLog(response3);

        Snackbar.make(findViewById(android.R.id.content), text: "Tag correctly read",
                .show();

        pwmConfigurationList = Parser.parsePWMRead(pwmConfigurationList, response,
                response2, response3);

        setUIElements();
        textLog.setText(logTextPWM.toString());
```

Figure 25. Read PWM method

All the commands sent in the '*readPWM*' method are defined in the file '*RF_Commands*'. More information about the PWM registers being read here can be found in sections 8.1.3.15 and 8.1.3.16 of the [Datasheet]:

- '*cmd_readGPIOPWMConfig*' is reading the session register '*PWM_GPIO_CONFIG_REG*' to read the '*PWM_GPIO_CONFIG_0_REG'* and '*PWM_GPIO_CONFIG_1_REG*' bytes.
- '*cmd_readPWM0Reg*' will read session registers '*PWM0_ON_REG'* and '*PWM0_OFF_REG'*.
- '*cmd_readPWM1Reg*' will read session registers '*PWM1_ON_REG*' and '*PWM1_OFF_REG'*.

Finally, when getting the information from the tag, it is necessary to parse it and update the UI.

Figure 26 displays the method to write the PWM data in the IC's EEPROM. In this case, it is necessary to translate the data from the UI to form a ISO/IEC 15693 command. Once the byte array with the raw commands is formed, it is sent via NFC to the IC.

```
private void writePWM() {
    byte[] writeSessionPWM = Parser.getSessionPWMCommand(pwmConfigurationList);
    byte[] pwm0Reg = Parser.getPWM0Command(pwmConfigurationList);
    byte[] pwm1Reg = Parser.getPWM1Command(pwmConfigurationList);

    try {
        writeSendLog(writeSessionPWM);
        byte[] response = sendCommand(writeSessionPWM);
        writeReceiveLog(response);

        writeSendLog(pwm0Reg);
        byte[] response2 = sendCommand(pwm0Reg);
        writeReceiveLog(response2);

        writeSendLog(pwm1Reg);
        byte[] response3 = sendCommand(pwm1Reg);
        writeReceiveLog(response3);

        Snackbar.make(findViewById(android.R.id.content),
                text: "Tag correctly written", Snackbar.LENGTH_SHORT)
                .show();

        textLog.setText(logTextPWM.toString());
```

Figure 26. Write PWM method

4.2.4 Pass-Through mode

The main view of this use case can be found in Figure 27.

Both Pass-Through directions are included in the same view.

As previously stated, the default Pass-Through direction is RF->I^2C. When the button 'Start Demo' is clicked, the Android app writes five times in the SRAM. Then, the FRDM-KW41Z switches the Pass-Through direction to I^2C->RF.

When the I^2C->RF direction is configured, the Android app waits to read the SRAM after the FRDM-KW41Z board writes it. This process is also performed five times.

Once the SRAM has been write/read five times in each direction, the process starts again until the 'Stop Demo' button is pushed in the Android app.

The SRAM data is fixed in both use cases. In the Android app, the main loop is implemented inside an Async task. This Android feature allows it to run heavy operations in a separate thread keeping the UI thread safe from heavy computing tasks.

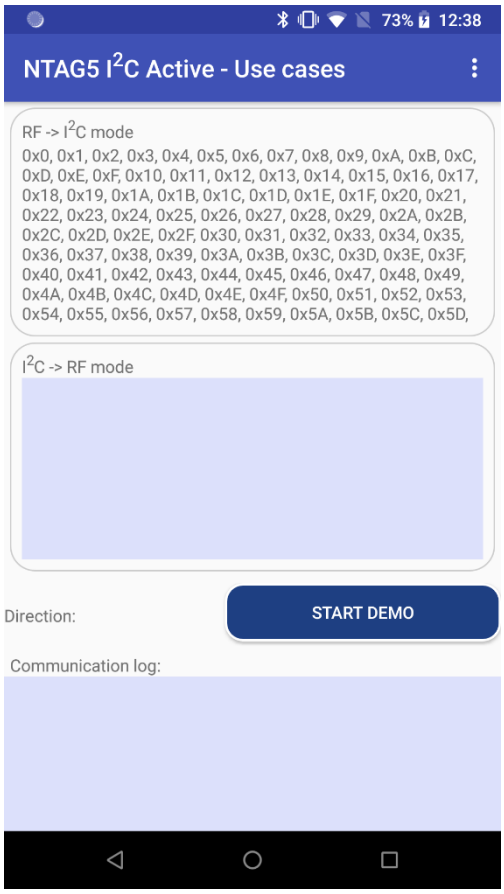

Figure 27. Pass-Through mode main view

The logic behavior of the toggle button is displayed in Figure 28.

```
buttonStartDemo.setOnClickListener((v) → {
        if (buttonStartDemo.isChecked()) {
            // When pushing the toggle button start the asynctask to continuously
            // perform the demo
            stopLoop = false;
            new SRAMLoop().execute();
            buttonStartDemo.setBackgroundResource(R.drawable.button_pushed_passthrough);
            buttonStartDemo.setTextColor(getResources().getColor((R.color.buttonBlue)));
            buttonStartDemo.setPadding( left: 20,  top: 0,  right: 20,  bottom: 0);
        } else {
            stopLoop = true;
            buttonStartDemo.setBackgroundResource(R.drawable.button_shape);
            buttonStartDemo.setTextColor(getResources().getColor((R.color.buttonWhite)));
            buttonStartDemo.setPadding( left: 20,  top: 0,  right: 20,  bottom: 0);
        }
});
```

Figure 28. Pass-Through mode, button configuration

Finally, in Figure 29 the main loop inside the Async task is shown.

```
while (!stopLoop) {
    byte[] responseTagStatus = sendCommand(cmd_readTagStatus);

    //Check if PT_TRANSFER_DIR == RF->I2C direction
    if (Parser.IsBitSet(responseTagStatus[1], index: 2)) {
        readCounter = 0;

        //In FW and Android app the pass-through mode will loop 5 times in each
        //direction in a continuous loop
        if (writeCounter < SRAM_LOOP_SIZE) {
            //Check if SRAM_DATA_READY != 1, means that the I2C has read the SRAM
            if (!Parser.IsBitSet(responseTagStatus[1], index: 5)) {
                responseWriteSRAM = sendCommand(finalCommandWriteSRAM);
                publishProgress(Constants.PassThroughDirection.RF_I2C);
                writeCounter++;
            }
        }
    } else {
        writeCounter = 0;

        //In FW and Android app the pass-through mode will loop 5 times in each
        //direction in a continuous loop
        if (readCounter < SRAM_LOOP_SIZE) {
            // Check if SRAM_DATA_READY == 1, this means that the I2C interface
            // has written the SRAM and it is ready to be read via RF.
            if (Parser.IsBitSet(responseTagStatus[1], index: 5)) {
                responseRead = sendCommand(cmd_readSRAM);
                publishProgress(Constants.PassThroughDirection.I2C_RF);
                readCounter++;
            }
        }
    }
}
```

Figure 29. Pass-Through mode Android Async task

4.2.5 Active Load Modulation

The main view of this use case can be found in Figure 30. As it can be observed, it contains four different configuration registers (ALM_CONF_xx), which correspond to the four bytes that control the Active Load Modulation configuration.

More information on the ALM parameters can be found in Section 8.1.3.25 of the NTAG 5 [Datasheet].

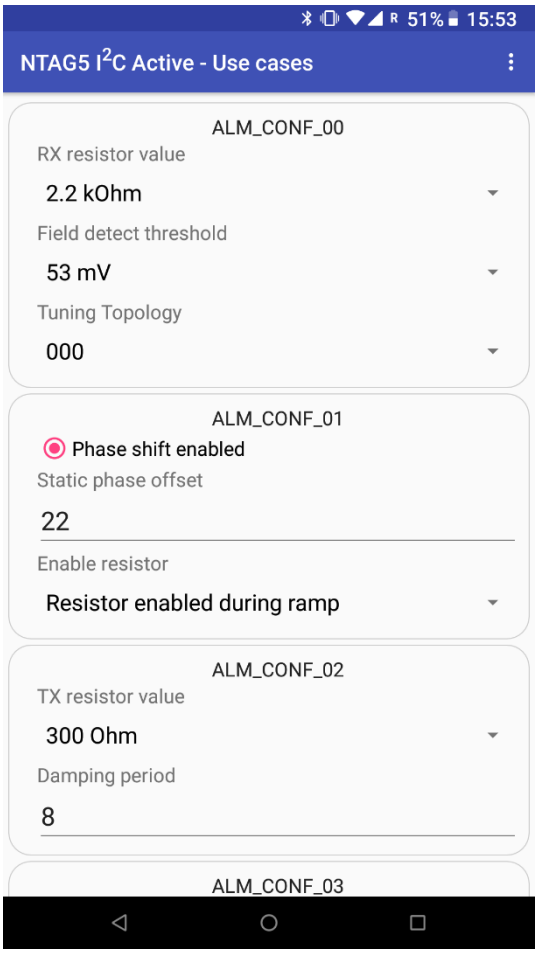

Figure 30. ALM mode main view

The ALM configuration can only be read from the Android App. Click on the “Read ALM Configuration” button and the field of each ALM configuration register will be updated with the values read through NFC interface.

Figure 31 shows the code snippet that reads the ALM parameters. The command *“cmd_readALMConfiguration*” is sent to read the four bytes that control the ALM parameters. Then, the response sent by the NTAG is parsed and the UI controls are updated.

```
private void readALM() {
    try {
        byte[] response = sendCommand(cmd_readALMConfiguration);

        almConfiguration = Parser.parseALMRead(almConfiguration, response);

        updateUIElements();
    } catch (Exception e) {
        e.printStackTrace();
        Snackbar.make(findViewById(android.R.id.content), text: "Operation interrupted! " +
                "Please try again", Snackbar.LENGTH_SHORT)
                .show();
    }
}
```

Figure 31. Read ALM Configuration.

5 References

[1] NTP5210 - NTAG 5 switch, NFC Forum-compliant PWM and GPIO bridge, doc.no. 5477xx
https://www.nxp.com/docs/en/data-sheet/NTP5210.pdf

[2] NTP53x2 - NTAG 5 link, NFC Forum-compliant I^2C bridge, doc.no. 5476xx
https://www.nxp.com/docs/en/data-sheet/NTP53x2.pdf

[3] NTA5332 - NTAG 5 boost, NFC Forum-compliant I^2C bridge for tiny devices, doc.no. 5475xx
https://www.nxp.com/docs/en/data-sheet/NTA5332.pdf

[4] AN11203 - NTAG 5 Use of PWM, GPIO and Event detection, doc.no. 5302xx
https://www.nxp.com/docs/en/application-note/AN11203.pdf

[5] AN12364 - NTAG 5 Bidirectional data exchange, doc.no. 5303xx
https://www.nxp.com/docs/en/application-note/AN12364.pdf

[6] FXOS8700CQ - 6-axis sensor with integrated linear accelerometer and magnetometer, Rev. 8 — 25 April 2017
https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf

6 Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — While NXP Semiconductors has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP Semiconductors accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

6.3 Licenses

Purchase of NXP ICs with NFC technology

Purchase of an NXP Semiconductors IC that complies with one of the Near Field Communication (NFC) standards ISO/IEC 18092 and ISO/IEC 21481 does not convey an implied license under any patent right infringed by implementation of any of those standards. Purchase of NXP Semiconductors IC does not include a license to any NXP patent (or other IP right) covering combinations of those products with other products, whether hardware or software.

6.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I²C-bus — logo is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Tables

Tab. 1. NTAG 5 – NTAG 5 configurations 13

Figures

Fig. 1. I2C Master channel sequence diagram 6
Fig. 2. GPIO use case sequence diagram 7
Fig. 3. PWM use case sequence diagram 9
Fig. 4. Pass-Through RF→ I2C direction diagram 11
Fig. 5. Pass-Through I2C → RF direction diagram 12
Fig. 6. ALM use case sequence diagram. 13
Fig. 7. Android app - Tag configuration 14
Fig. 8. Android Studio project structure 16
Fig. 9. Tag detection in BaseActivity 17
Fig. 10. Send command method in BaseActivity 18
Fig. 11. Android App Main menu 19
Fig. 12. I2C Master Configuration view 20
Fig. 13. I2C Master Write I2C sample 21
Fig. 14. I2C Master Read I2C sample 21
Fig. 15. Get Temperature I2C Master channel 22
Fig. 16. I2C default commands list 23
Fig. 17. I2C Master custom command 24
Fig. 18. GPIO configuration fragment view 25
Fig. 19. Read GPIO 26
Fig. 20. Write GPIO config 27
Fig. 21. Set GPIO output channel 0 register 27
Fig. 22. Clear GPIO output channel 0 register 28
Fig. 23. Get GPIO input channel 1 status 28
Fig. 24. PWM configuration activity view 29
Fig. 25. Read PWM method 30
Fig. 26. Write PWM method 31
Fig. 27. Pass-Through mode main view 32
Fig. 28. Pass-Through mode, button configuration 32
Fig. 29. Pass-Through mode Android Async task 33
Fig. 30. ALM mode main view 34
Fig. 31. Read ALM Configuration. 35

Contents

1 Introduction 3
1.1 Objective 3
1.2 Target audience 3
2 NTAG 5 overview 4
2.1 General description 4
2.2 Features and applications 4
3 NTAG 5 – Use cases definition in Android App 5
3.1 Use cases definition 5
3.1.1 I2C master channel use case 5
3.1.2 GPIO use case 6
3.1.2.1 GPIO output 7
3.1.2.2 GPIO input 8
3.1.3 PWM use case 8
3.1.4 Pass-Through mode use case 10
3.1.4.1 RF→ I2C direction 10
3.1.4.2 I2C → RF direction 11
3.1.5 ALM use case 12
3.2 User Guide – NTAG 5 configuration 13
3.3 User Guide – NFC connection with board 14
4 NTAG 5 – Use cases code snippets 16
4.1 General information and project structure 16
4.1.1 BaseActivity 17
4.1.2 BaseActionBarActivity 19
4.1.3 MainActivity 19
4.2 Use cases 20
4.2.1 I2C Master Channel 20
4.2.1.1 I2C Master channel communication sequence 20
4.2.1.2 I2C Master predefined commands 22
4.2.1.3 I2C Master custom commands 24
4.2.2 General Purpose Input/Output (GPIO) 24
4.2.3 Pulse Width Modulation (PWM) 29
4.2.4 Pass-Through mode 31
4.2.5 Active Load Modulation 33
5 References 36
6 Legal information 37

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 9 January 2020
Document identifier: RM00221
Document number: 531810