RM00222

NTAG 5 - Firmware development for KW41Z

Rev. 1.0 — 9 January 2020 Reference manual

531910 COMPANY PROPRIETARY

Document information

Information	Content
Keywords	NTAG 5, FRDM-KW41Z, OM2NTA5332, OM2NTP5332, ISO/IEC 15693, Pass-Through, Bluetooth pairing, GPIO, PWM, NDEF.
Abstract	This application note describes the usage of the FRDM-KW41Z firmware developed to interact with the NTAG 5 IC. 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 FRDM-KW41Z project developed to work with the NTAG 5 IC.

This project is divided in four use cases. This document describes each use case in detail, presenting the most important parts of the code snippets. The proper board configurations to run each example are also explained.

1.2 Target audience

This document is intended for developers who are working with the FRDM-KW41Z board and NTAG 5 customer development boards OM2NTA5332 and OM2NTP5332. NXP has created a reference source code to demonstrate four different usages of the new IC. Thus, it is necessary to have some experience in embedded development to get 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 Device with a tap. With this ability, the IC offers a long-reading range and privacy due to close proximity with mobile devices.

2.2 Features and applications

NTAG 5 family 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 IC 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 IC has several innovative features such as Active Load Modulation, NFC and I^2C Silence, 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 FRDM-KW41Z setup

Figure 1 displays an image from the board; all pins used to communicate with the NTAG 5 are highlighted in yellow.

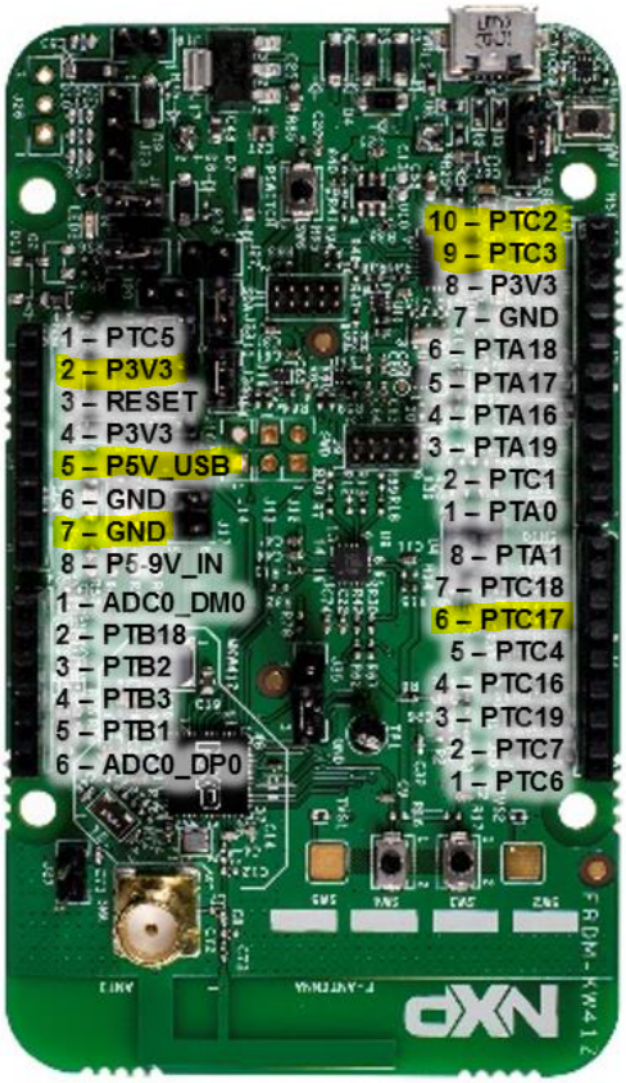

Figure 1. FRDM-KW41Z board

The jumpers configuration for the FRDM-KW41Z board is displayed in Figure 2.

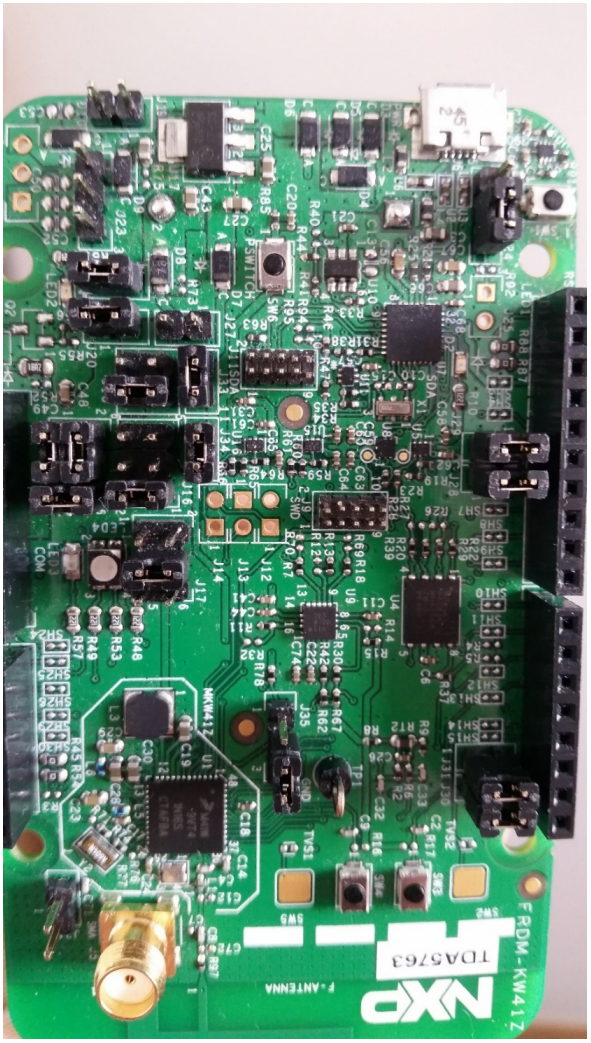

Figure 2. FRDM-KW41Z jumpers configuration.

The NTAG 5 has a particular feature: I^2C pins and GPIO/PWM pins are multiplexed, which means it is not possible to communicate via I^2C at the same time the GPIO/PWM is being used. For this reason, each code snippet has a different configuration for the PTC2 and PTC3 pins. These configurations are explained in detail in the corresponding section.

The connection between the FRDM-KW41Z board and the OM2NTx5332 is shown in Figure 3.

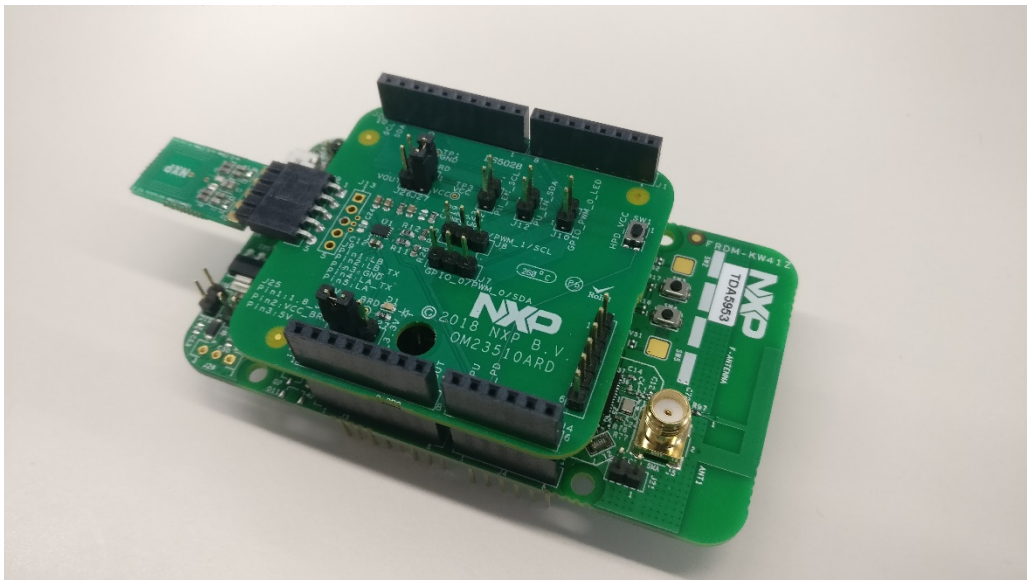

Figure 3. Connection between FRDM-KW41Z board and OM2NTx5332

4 NTAG 5 - Use cases definition in FRDM-KW41Z

4.1 NDEF message use case

This use case aims to show how to write NDEF messages in a Type 5 Tag. There are two different NDEF messages created in the code by default; one URL containing the NXP website and one Bluetooth pairing message to perform a Bluetooth connection with the FRDM-KW41Z board.

In this use case, both buttons (SW3 and SW4) configured in the FRDM-KW41Z board. When buttons are pushed, the FRDM-KW41Z board will randomly write one of the two predefined NDEF messages into the IC. Once the NDEF message has been written in the NTAG, it will be available to be read by a mobile phone with NFC capabilities and FRDM-KW41Z's LED colour will change.

More information about NDEF message generation in a Type 5 Tag can be found in the corresponding specification of the NFC Forum [Specification].

4.2 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 I^2C lines as general-purpose input/output lines. The NTAG 5 has the I^2C 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 4. First of all, the IC needs to be configured in GPIO/PWM mode. This is necessary because the I^2C lines and the GPIO/PWM lines are multiplexed and the IC needs to select one of them. Once the IC is correctly configured, it is possible to read the configuration for the two GPIO channels.

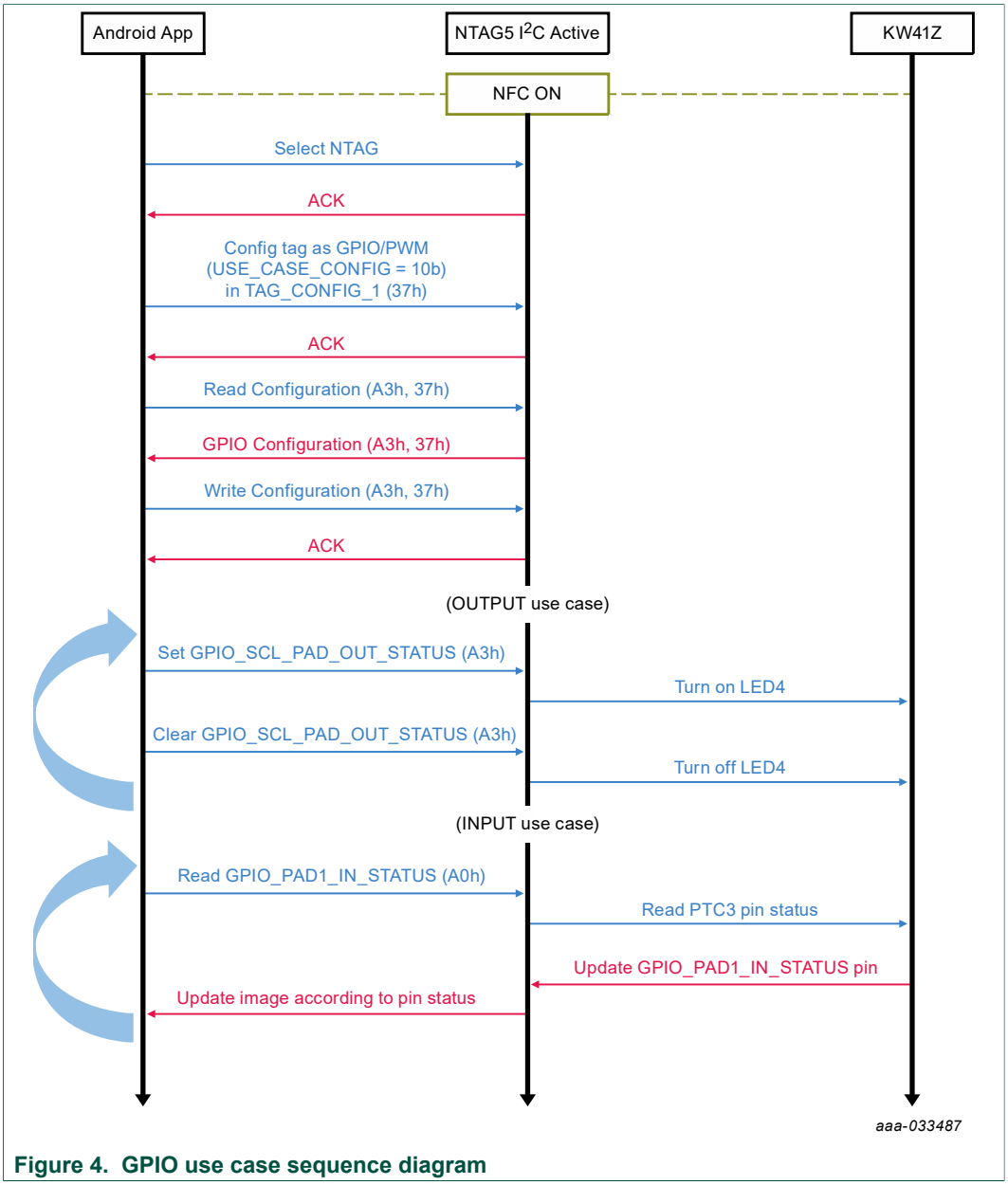

Figure 4. GPIO use case sequence diagram

4.2.1 GPIO output

In the GPIO output, the idea is to use the FRDM-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^2C 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 the 'Clear GPIO output' button, the value in the session register will be cleared and both LED4 in the board and bulb image in the app 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 vice-versa). 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. Thus, it is recommended to configure the IC 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 1b), the LED 3 will be illuminating in the FRDM-KW41Z board. At a falling edge, the register will be set to 0b and the LED 3 will be turned off.

4.2.2 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 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 1b 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 IC 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.

4.3 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 stated, 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 IC 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 about how the NTAG 5 configures a PWM signal can be found 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 configured an interrupt to turn on/off LED3 and LED4 according to the input of the corresponding pin.

A sequence diagram of the use case can be found in Figure 5.

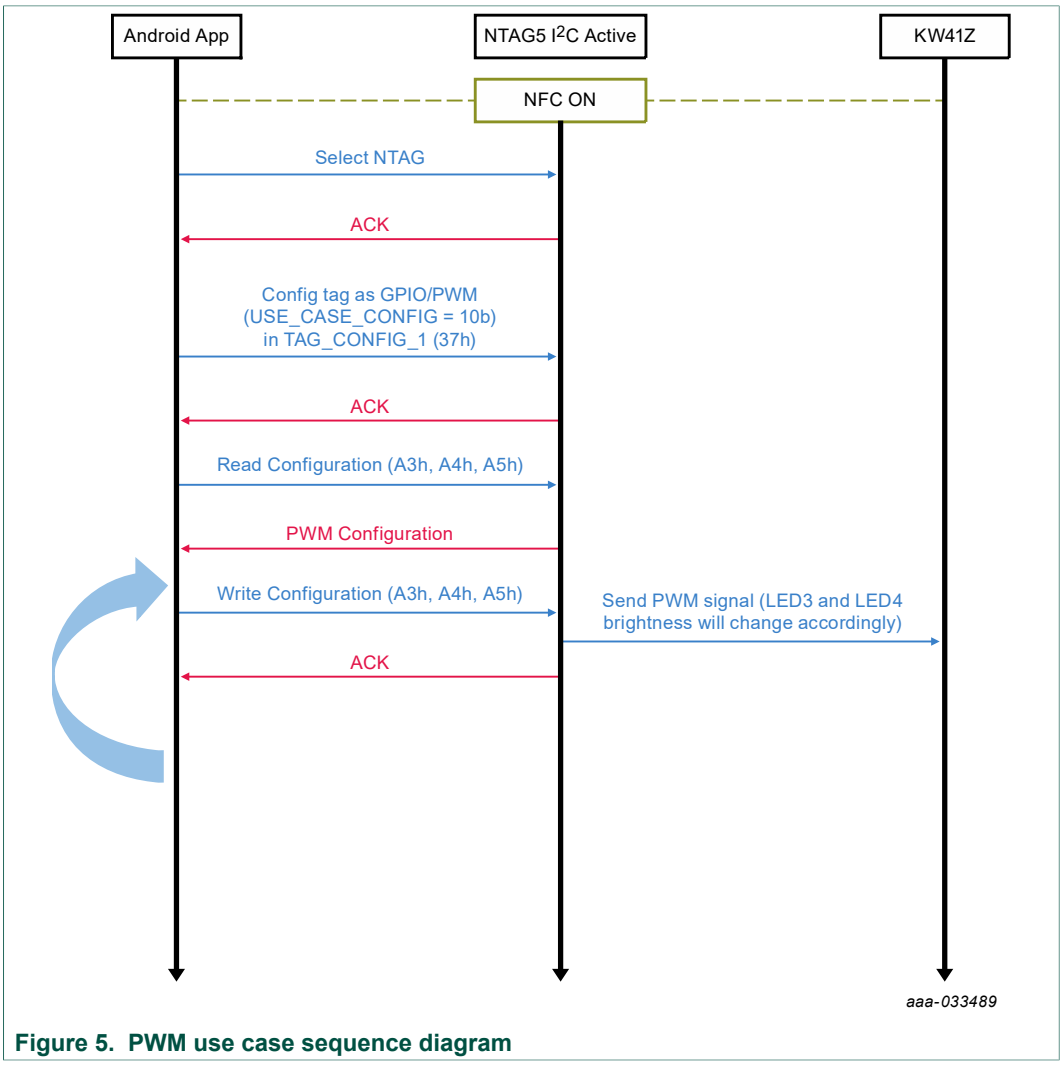

Figure 5. PWM use case sequence diagram

4.4 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 IC 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 [Application note].

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 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 IC 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.

4.4.1 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 IC 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 6.

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 [Datasheet] and [Application note].

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 the SRAM is ready to be read via I^2C.

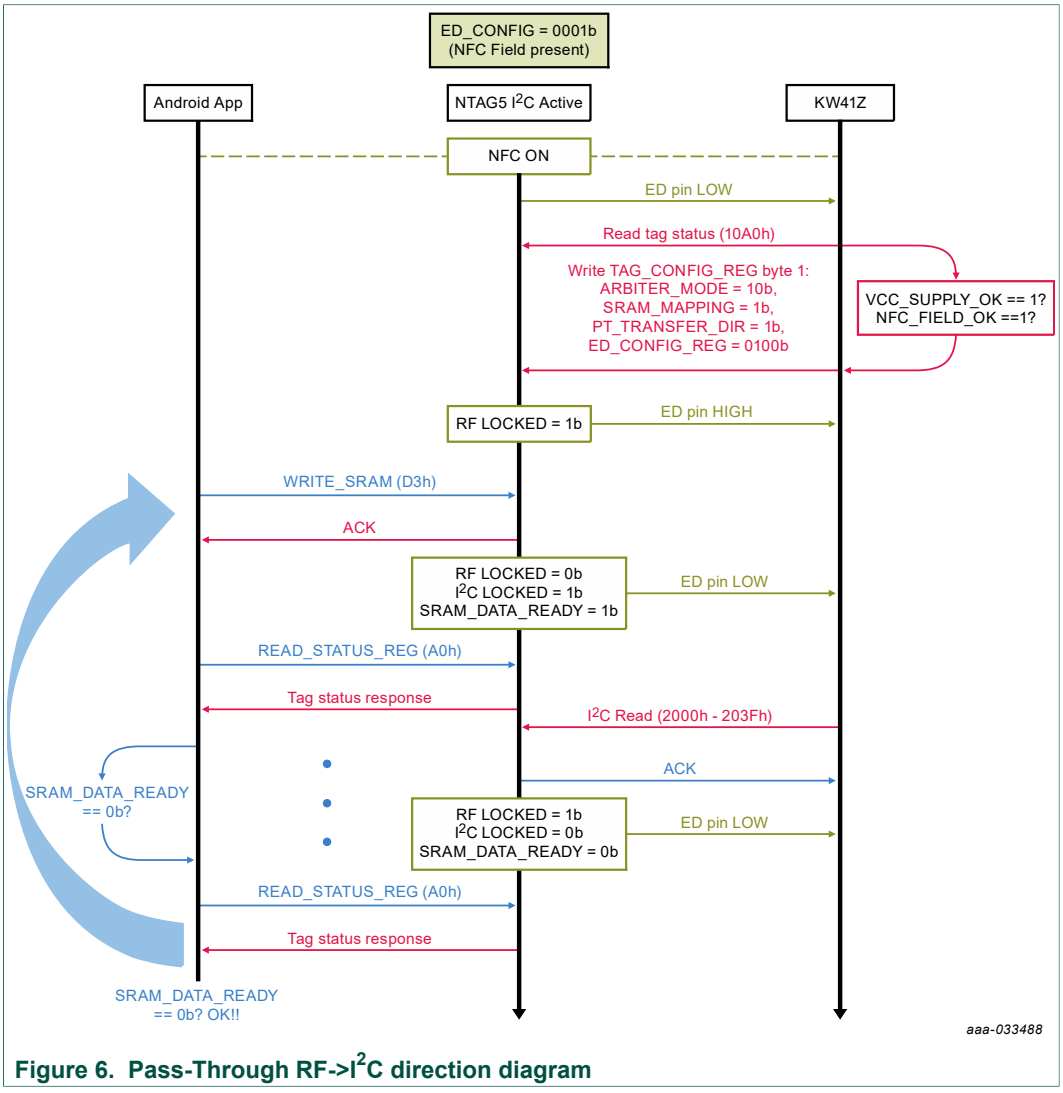

Figure 6. Pass-Through RF->I^2C direction diagram

4.4.2 I²C->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 7.

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 [Datasheet] 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 I²C interface is locked and the SRAM_DATA_READY bit is cleared, this means the SRAM is ready to be written via I²C.

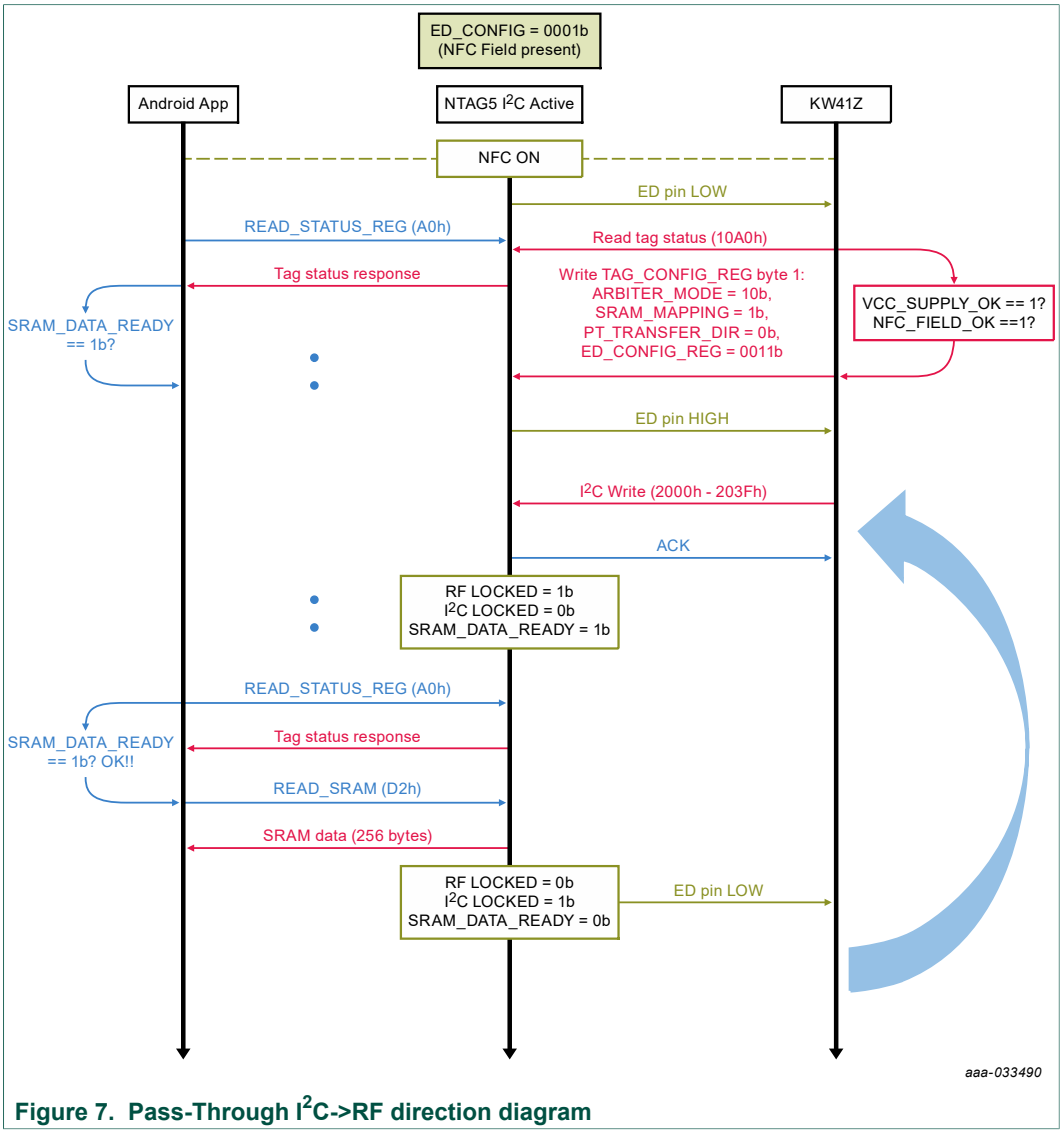

Figure 7. Pass-Through I²C->RF direction diagram

4.5 Active Load Modulation 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 [Datasheet]. Figure 8 shows the diagram of this use case.

The configuration of ALM parameters is carried out using the Peek and Poke 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 “Write Configuration” so the NTAG can be configured.

In Android App: Go to the ALM use case menu and click “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.

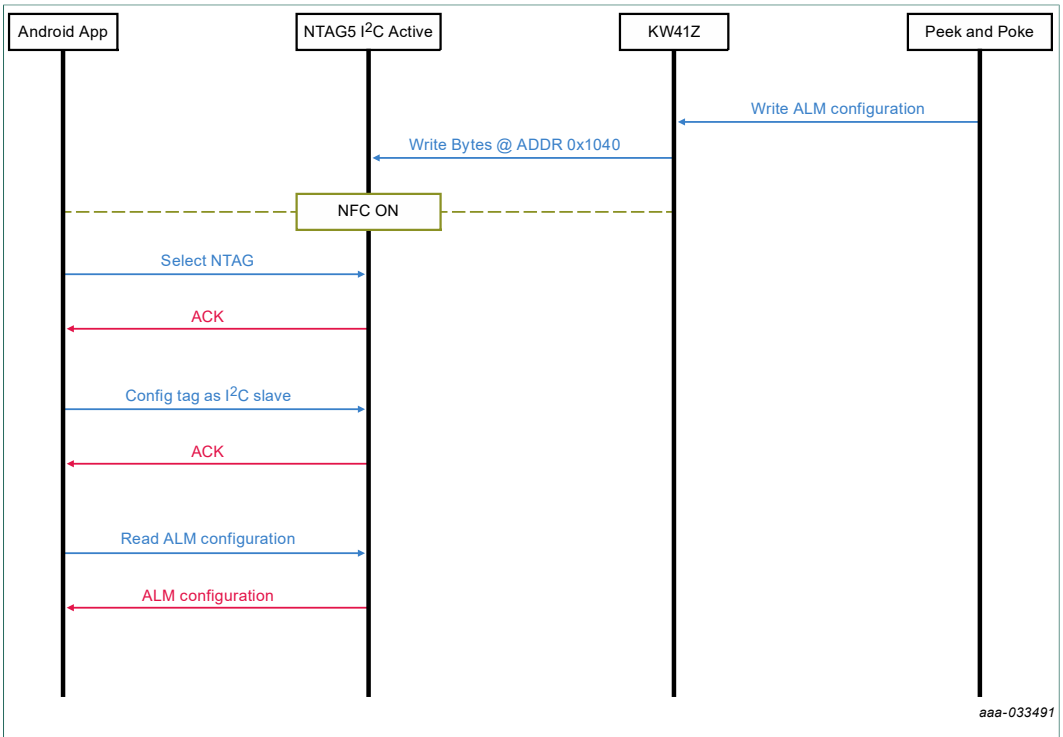

Figure 8. ALM use case sequence diagram.

5 NTAG 5 - Use cases code snippets

5.1 General information

MCUXpresso has been chosen as IDE for the development of the FRDM-KW41Z firmware. For this reason, it is desirable to have some background with this IDE. The MCUXpresso version used to develop this product support package has been v10.1.0 [Build 589] and the version used in the FRDM-KW41Z SDK is 2.2.0.

Each use case has been developed as a separate project, this is because each use case has a different board configuration (for instance, different pin and interruption configurations).

5.2 Use cases

5.2.1 NDEF messages

As explained in the previous section, the purpose of this demo is to write an NDEF message in the IC when pushing the button SW3 in the FRDM-KW41Z board. Each time the button 3 in the FRDM-KW41Z board is pushed an NDEF message is written in the NTAG. The NDEF message to be written is selected randomly each time the button is pushed.

Figure 9 shows the folder structure for this project.

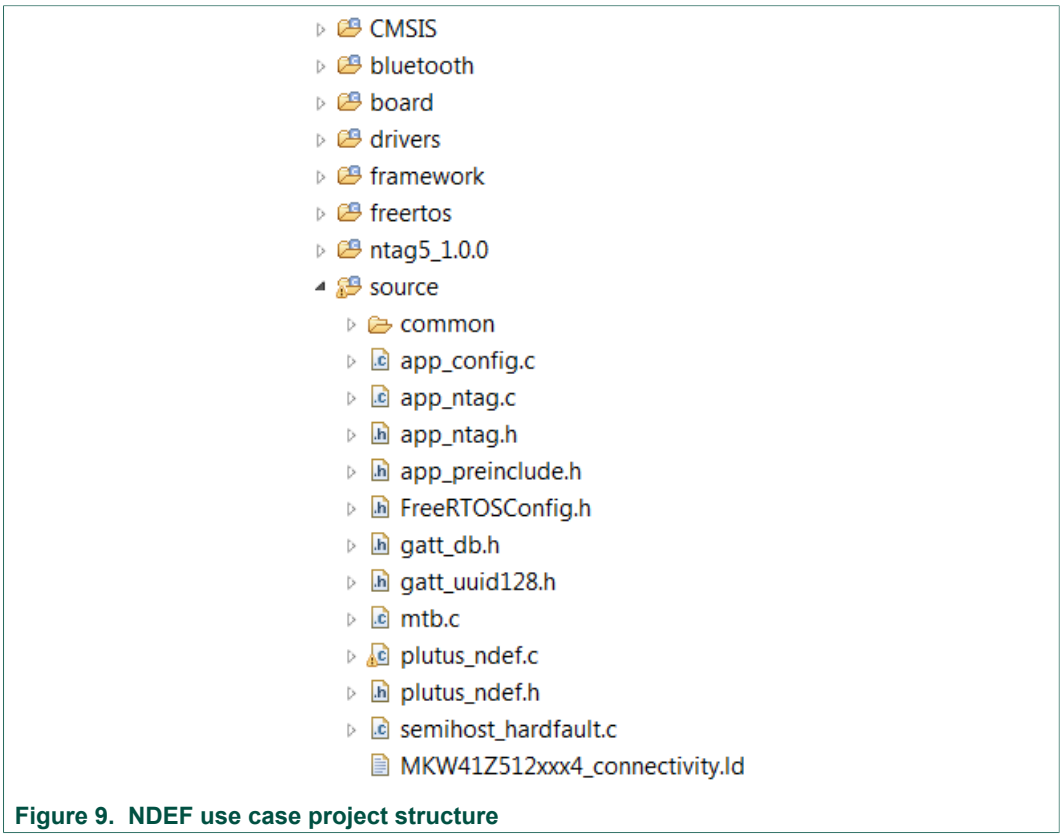

Figure 9. NDEF use case project structure

The project contains the middleware for the NTAG 5 (source folder called '**NTAG 5_1.0.0**'). At the beginning of the application (**main_task** function in **ApplMain.c** file contained in **source/common** folder), it is necessary to initialize the NTAG middleware using **NTAG_InitDevice** function, this function belongs to the **NTAG 5_1.0.0/HAL_NTAG/nIC_driver.c** file.

This project uses FreeRTOS; at the beginning of the code, some initializations are necessary: the event detection task, NTAG 5_1.0.0 middleware, the project framework, the OSAL and the random number generator. This is shown in Figure 10.

```
int main()
{
    ntag = &nDevice;
    phStatus_t status;

    #ifndef PH_OSAL_NULLOS
            phOsal_ThreadObj_t DemoApp;
    #endif /* PH_OSAL_NULLOS */

            phApp_KW41Z_Init();

            /*Initialize random number generator*/
            srand(time(NULL));

            urlNDEFSelected = false;

            blePairingNDEFSelected = false;

            /* Perform OSAL Initialization. */
            (void)phOsal_Init();

            status = phbalReg_Init(&sbalReg, sizeof(sbalReg));

            if (!platformInitialized && (status == PH_DRIVER_SUCCESS))
            {
                uint8_t pseudoRNGSeed[20] = {0};

                platformInitialized = 1;

                /* Framework init */
                MEM_Init();
                TMR_Init();
                LED_Init();
                SecLib_Init();
```

```
#if gKeyBoardSupported_d && (gKBD_KeysCount_c > 0)
        KBD_Init(App_KeyboardCallBack);
#endif

#if gAppUseNvm_d
        /* Initialize NV module */
        NvModuleInit();
#endif

#if !gUseHciTransportDownward_d
        pfBLE_SignalFromISR = BLE_SignalFromISRCallback;
#endif /* !gUseHciTransportDownward_d */

#if (cPWR_UsePowerDownMode || gAppUseNvm_d)
#if (!mAppIdleHook_c)
        AppIdle_TaskInit();
#endif
#endif
#if (cPWR_UsePowerDownMode)
        PWR_Init();
        PWR_DisallowDeviceToSleep()
#endif

    /* Initialize peripheral drivers specific to the application */
    BleApp_Init();

    /* Create application event */
    mAppEvent = OSA_EventCreate(TRUE);
    if( NULL == mAppEvent )
    {
        panic(0,0,0,0);
        return 0;
    }

    /* Prepare application input queue.*/
    MSG_InitQueue(&mHostAppInputQueue);
```

Figure 10. Project initialization

Once the project is initialized, the main loop is waiting for events. At startup, the interrupt to detect when the SW3 button is pushed, needs to be configured inside the '*KBD_Init*' function. This is displayed in Figure 11.

```
963         /* initialize all the GPIO pins for keyboard */
964         BOARD_InitButtons();
965         KbGpioInit();
966 #if gKBD_TsiElectdCount_c
967         KbTsiInit();
968 #if gTMR_Enabled_d
969         TMR_StartLowPowerTimer(mTsiSwTriggerTimerID, gTmrIntervalTimer_c, gKBD_TsiTr
970 #endif
971 #endif
972
973         for( i=0; i<gKBD_KeysCount_c; i++ )
974         {
975             if(kbdSwButtons[i].swType == gKBDTypeGpio_c)
976             {
977                 GpioInstallIsr( Switch_Press_ISR, gGpioIsrPrioLow_c,
978                         gKeyboard_IsrPrio_c, kbdSwButtons[i].config_struct.pSwGpio);
979             }
980         }
981     }

/*! *********************************************************************************
* \brief  Install the Gpio_CommonIsr() ISR for the specified IRQ
*
* \param[in]  irqId    The CMSIS irq Id
* \param[in]  nvicPrio The priority to be set in NVIC
*
* \return  install status
*
********************************************************************************** */
static gpioStatus_t Gpio_InstallPortISR(IRQn_Type irqId, uint32_t nvicPrio)
{
    if( irqId != NotAvail_IRQn )
    {
        OSA_InstallIntHandler(irqId, Gpio_CommonIsr);

        /* Enable IRQ in NVIC and set priority */
        NVIC_ClearPendingIRQ(irqId);
        NVIC_EnableIRQ(irqId);
        NVIC_SetPriority(irqId, nvicPrio >> (8 - __NVIC_PRIO_BITS));
    }
    return gpio_success;
}
```

Figure 11. Interrupt configuration for NDEF use case

After the interrupt configuration, the handler function is shown in Figure 12. This handler function will trigger an event in the main loop according to the random number generated and the selected NDEF message. The keyboard events are managed in the function '*App_HandleKeys*' contained in the '**plutus_ndef.c**' file.

```
void App_HandleKeys(key_event_t events)
{
    int g_randValueGenerated;

    switch (events)
    {
        case gKBD_EventPressPB2_c:
        {
            //Generate a random number
            g_randValueGenerated = rand()&0x01;

            switch(g_randValueGenerated)
            {
                case 0:
                    BleApp_Start();

                    blePairingNDEFSelected = true;
                    OSA_EventSet(mAppEvent, gAppEvtAppCallback_c);

                    break;
                case 1:
                    //Close the bluetooth connection
                    if (mPeerDeviceId != gInvalidDeviceId_c)
                    {
                        Gap_Disconnect(mPeerDeviceId);
                    }

                    urlNDEFSelected = true;
                    OSA_EventSet(mAppEvent, gAppEvtAppCallback_c);
                    break;
            }

            break;
        }

        default:
            break;
    }
}
```

Figure 12. Push button 3 event handler

In Figure 13 the main loop of the application is shown.

```
void App_Thread ()
{
    osaEventFlags_t event;
    BOOL error_initNtag;

    error_initNtag = NTAG_InitDevice(ntag, &sbalReg);
    if(!error_initNtag)
    {
        while(1)
        {
            OSA_EventWait(mAppEvent, osaEventFlagsAll_c, FALSE, osaWaitForever_c , &event);

            if(urlNDEFSelected)
            {
                urlNDEFSelected = false;
                error_initNtag = NTAG_WriteBytes(ntag, 0x0000, (const uint8_t *) &NDEF_NXPlink,
                        NDEF_link_Length);
                if(!error_initNtag)
                {
                    StopLed2Flashing();
                    Led3Flashing();
                }
            }
            else if (blePairingNDEFSelected)
            {
                blePairingNDEFSelected = false;
                error_initNtag = NTAG_WriteBytes(ntag, 0x0000, (const uint8_t *) &BLE_pairing_NDEF_msg,
                        BLE_pairing_NDEF_msg_Length);
                if(!error_initNtag)
                {
                    StopLed3Flashing();
                    Led2Flashing();
                }
            }
```

Figure 13. Main loop of the NDEF use case application

The NDEF messages are written in the NTAG memory using the **'NTAG_WriteBytes'** function contained in the '**nIC_driver.c**' file. The NDEF message in a Type 5 Tag is written at the beginning of the memory (address 0x0000).

The first NDEF message is a Bluetooth pairing with the FRDM-KW41Z board. To indicate that this type of NDEF message has been generated, a red light will be flashing in the FRDM-KW41Z board. When the phone has been tapped and the Bluetooth connection is correctly established, apart from the flashing red light, the blue LED is activated as well.

On the other hand, the second NDEF message is a URL. To indicate that this NDEF has been generated, a green light will be flashing in the board.

5.2.2 General Purpose Input/Output (GPIO)

The project structure for MCUXpresso can be seen in Figure 14.

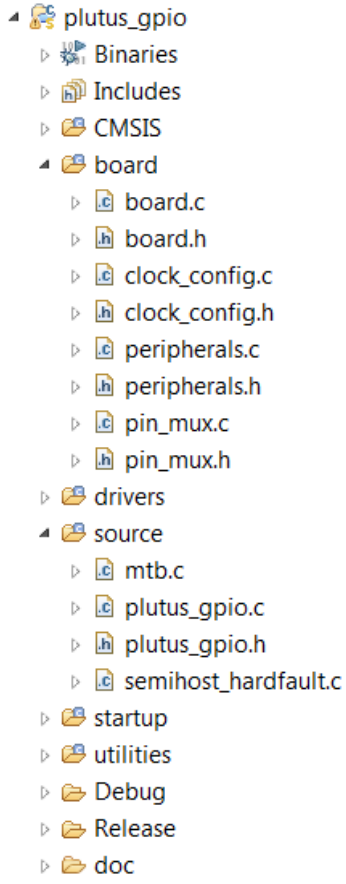

Figure 14. GPIO use case project structure

All the pin configurations are contained in '**pin_mux.c**' file contained in '**board**' folder and the main functionalities of the project are contained in '**plutus_gpio.c**' file contained in the '**source**' folder.

As explained in Section 4.2, this project makes use of the GPIO feature of the NTAG 5. GPIO feature is divided in GPIO Output and GPIO Input:

- For the GPIO Output, the main objective is to configure the PTC2 pin as an input pin in the FRDM-KW41Z and drive an LED using a signal coming from the *GPIO_SCL_PAD_OUT_STATUS* register in the NTAG 5.
- For the GPIO Input, the purpose is to use the PTC3 pin as an output pin in the FRDM-KW41Z board to write the *GPIO_PAD1_IN_STATUS* register in the NTAG 5 using the SW3 button.

Since the GPIO functionality is fixed for both channels, it is necessary to configure the pins PTC2 and PTC3 in the FRDM-KW41Z to work according to this input and output definition.

5.2.2.1 Output use case

PTC2 will be the input port for the channel 0 signal coming from the NTAG 5. It is necessary to set up an interrupt in this port because the board has to turn on/off an LED each time a new command is written in the NFC interface. The configuration for PTC2 can be found in Figure 15.

```
const port_pin_config_t portc2_config = {
    kPORT_PullDown,                          /* Internal pull-up resistor is enabled */
    kPORT_FastSlewRate,                      /* Slow slew rate is configured */
    kPORT_PassiveFilterDisable,              /* Passive filter is disabled */
    kPORT_LowDriveStrength,                  /* Low drive strength is configured */
    kPORT_MuxAsGpio,                         /* Pin is configured as PTC4 */
  };

PORT_SetPinConfig(PORTC, PIN2_IDX, &portc2_config);    /* PORTC2 is configured as PTC2 */

        /* Define the init structure for the input switch pin */
        gpio_pin_config_t sw_input_config = {
            kGPIO_DigitalInput, 0,
        };

/* Enable PTC2 as GPIO input for the signal sent from NTAG5 when channel 0 is activated. */
PORT_SetPinInterruptConfig(PORTC, PIN2_IDX, kPORT_InterruptEitherEdge);
GPIO_PinInit(GPIOC, PIN2_IDX, &sw_input_config);

 /* Enable interruptions in Port C for switchs 3, 4 and PTC2 input. */
 EnableIRQ(PORTB_PORTC_IRQn);

/* Init LED pin. */
GPIO_PinInit(BOARD_LED_RED_GPIO, BOARD_LED_RED_GPIO_PIN, &sw_output_config);
LED_RED_INIT(0);
LED_RED_OFF();
```

Figure 15. PTC2 configuration in FRDM-KW41Z for GPIO output

In Figure 15, the EnableIRQ function is also valid for the SW3 button (used for the input use case) since they also belong to the PORTB_PORTC interrupt group. The code displayed in Figure 17 can be found in files '**pin_mux.c**' for the pin configuration, '**plutus_gpio.h**' for the '*sw_input_config*' definition and '**plutus_gpio.c**' to initialize the port and set the interruption.

Finally, when an interrupt arrives to PORTC, it is necessary to handle it in the corresponding function. In Figure 16, you can find the way the interrupt is handled for the PTC2 port.

```
else if(interruptStatus & (1U << PIN2_IDX)) /*PTC2 interruption*/
{
    /* Clear external interrupt flag. */
    GPIO_ClearPinsInterruptFlags(GPIOC, 1U << PIN2_IDX);

    uint32_t input_value = GPIO_ReadPinInput(GPIOC, PIN2_IDX);

    if(input_value == 1) /* If the bulb image is pushed in
                         the Android app then I turn on the LED. */
    {
        LED_RED_ON();
    }
    else if(input_value == 0) /* If the clear button is pushed
                              in the Android app then I turn off the LED. */
    {
        LED_RED_OFF();
    }
}
```

Figure 16. Interruption handler for PTC2 for GPIO output

5.2.2.2 Input use case

For the input use case, button SW3 together with port PTC3 is used. Then it is necessary to configure these elements. When SW3 button is pushed, PTC3 pin (which is configured as output pin) has a high level. By releasing SW3, PTC3 pin will have a low level.

The configuration for the button SW3 is shown in Figure 17

```
const port_pin_config_t portC_config = {
  kPORT_PullUp,                            /* Internal pull-up resistor is enabled */
  kPORT_FastSlewRate,                      /* Slow slew rate is configured */
  kPORT_PassiveFilterDisable,              /* Passive filter is disabled */
  kPORT_LowDriveStrength,                  /* Low drive strength is configured */
  kPORT_MuxAsGpio,                         /* Pin is configured as PTC4 */
};

    /* Define the init structure for the input switch pin */
    gpio_pin_config_t sw_input_config = {
        kGPIO_DigitalInput, 0,
    };

/* Init input switch 3. */
PORT_SetPinInterruptConfig(BOARD_SW3_PORT, BOARD_SW3_GPIO_PIN, kPORT_InterruptEitherEdge);
GPIO_PinInit(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN, &sw_input_config);

/* Enable interruptions in Port C for switchs 3, 4 and PTC2 input. */
EnableIRQ(PORTB_PORTC_IRQn);
```

Figure 17. SW3 configuration for GPIO input

Once SW3 button is configured, it is necessary to configure the PTC3 pin. This configuration can be found in Figure 18.

```
PORT_SetPinConfig(PORTC, PIN3_IDX, &portC_config);     /* PORTC3 is configured as PTC3 */

    /* Define the init structure for the output led pin */
    gpio_pin_config_t sw_output_config = {
        kGPIO_DigitalOutput, 0,
    };

        /* Enable PTC3 as GPIO output when pushing the button
         *  and send a signal to NTAG5 in channel 1. */
        GPIO_PinInit(GPIOC, PIN3_IDX, &sw_output_config);
```

Figure 18. PTC3 configuration for GPIO input

Finally, the main loop of the application and the IRQ handler for the GPIO input use case is shown in Figure 19.

```
void PORTB_PORTC_IRQHandler(void)
{
    uint32_t interruptStatus = GPIO_GetPinsInterruptFlags(GPIOC);

    if(interruptStatus & (1U << PIN4_IDX)) /*Clear flag of Button 3 and set push variable*/
    {
        /* Clear external interrupt flag. */
        GPIO_ClearPinsInterruptFlags(BOARD_SW3_GPIO, 1U << BOARD_SW3_GPIO_PIN);

        uint32_t input_value_button = GPIO_ReadPinInput(GPIOC, PIN4_IDX);

        if(input_value_button == 0) /* If the button is pushed,
                                    set the GPIO input register in the NTAG. */
        {
            /* Change state of buttons. */
            g_Button3Pressed = true;
            g_Button3Released = false;
        }
        else if(input_value_button == 1) /* If button is released,
                                         clean the GPIO input register in the NTAG. */
        {
            /* Change state of buttons. */
            g_Button3Pressed = false;
            g_Button3Released = true;
        }
    }
```

```
while(1) {
    if (g_Button3Pressed)
    {
        printf(" Button 3 has been pressed \r\n");

        /* Set PTC3 pin. */
        GPIO_WritePinOutput(GPIOC, PIN3_IDX, 1);

        /* Reset state of button. */
        g_Button3Pressed = false;
    }
    else if (g_Button3Released)
    {
        printf(" Button 3 has been released \r\n");

        /* Clear PTC3 pin. */
        GPIO_WritePinOutput(GPIOC, PIN3_IDX, 0);

        /* Reset state of button. */
        g_Button3Released = false;
    }
}
```

Figure 19. Main loop for GPIO input use case

5.2.3 Pulse Width Modulation (PWM)

As previously explained, the idea of the PWM use case is to configure two PWM signals in the NTAG 5 using the Android app. Then the PWM signals are used to modulate two LEDs in the FRDM-KW41Z board.

The project structure can be seen in Figure 20.

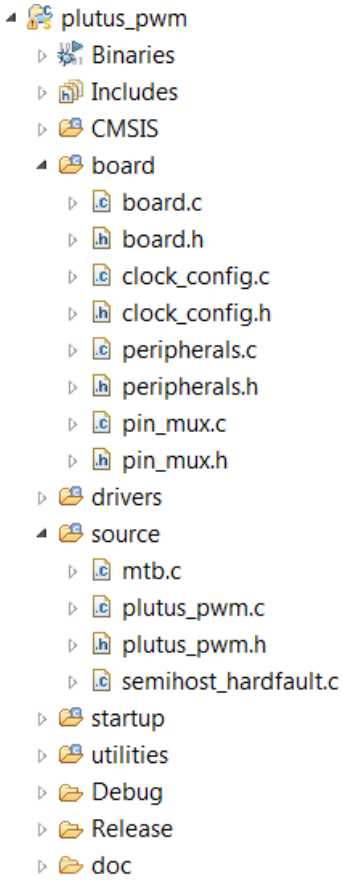

Figure 20. PWM use case project structure

The FRDM-KW41Z configures the pins PTC2 and PTC3 to be used as input ports. These PWM signals will trigger interrupts on these ports to turn on and turn off LEDs 3 and 4. The configuration for these pins can be found in Figure 20. These configurations can be found in the files **'pin_mux.c'** in the **'board'** folder and **'plutus_pwm.c'** file in the **'source'** folder.

```
const port_pin_config_t portInput_config = {
    kPORT_PullDown,                                  /* Internal pull-up resistor is enabled */
    kPORT_FastSlewRate,                             /* Slow slew rate is configured */
    kPORT_PassiveFilterDisable,                     /* Passive filter is disabled */
    kPORT_LowDriveStrength,                         /* Low drive strength is configured */
    kPORT_MuxAsGpio,                                /* Pin is configured as PTC4 */
  };
```

```
/* Enable PTC2 as GPIO input for the signal sent from NTAG5 when PWM comes through channel 0. */
PORT_SetPinInterruptConfig(PORTC, PIN2_IDX, kPORT_InterruptEitherEdge);
GPIO_PinInit(GPIOC, PIN2_IDX, &sw_input_config);

/* Enable PTC3 as GPIO input for the signal sent from NTAG5 when PWM comes through channel 1. */
PORT_SetPinInterruptConfig(PORTC, PIN3_IDX, kPORT_InterruptEitherEdge);
GPIO_PinInit(GPIOC, PIN3_IDX, &sw_input_config);

/* Enable interruptions in Port C for PTC2 and PTC3 inputs. */
EnableIRQ(PORTB_PORTC_IRQn);
```

Figure 21. PTC2 and PTC3 configuration for PWM

It is necessary to configure both pins with '*kPort_InterruptEitherEdge*' to trigger the interrupts at rising and falling edges of the signal.

Apart from the PTC2 and PTC3 pins, it is necessary to configure LED3 and LED4 to operate when the interrupts are triggered. Figure 22 indicates the configuration for both LEDS in the PWM use case. These configurations can also be found in the files **'pin_mux.c'** in the **'board'** folder and **'plutus_pwm.c'** file in the **'source'** folder.

```
PORT_SetPinMux(PORTC, PIN1_IDX, kPORT_MuxAsGpio);          /* PORTC1 is configured as PTC1 (LED4) */
PORT_SetPinMux(PORTB, PIN0_IDX, kPORT_MuxAsGpio);          /* PORTB0 is configured as PTB0 (LED3 */

/* Init LED 4 pin. */
GPIO_PinInit(BOARD_LED4_GPIO, BOARD_LED4_GPIO_PIN, &sw_output_config);
GPIO_TogglePinsOutput(BOARD_LED4_GPIO, 1U << BOARD_LED4_GPIO_PIN);

/* Init LED 3 pin. */
GPIO_PinInit(BOARD_LED3_GPIO, BOARD_LED3_GPIO_PIN, &sw_output_config);
GPIO_TogglePinsOutput(BOARD_LED3_GPIO, 1U << BOARD_LED3_GPIO_PIN);
```

Figure 22. LED3 and LED4 configuration for PWM

Finally, in Figure 23 the interrupt handler function shows how the LEDs are toggled each time a rise or falling edge arrives to the port. The IRQ handler function can be found in **'plutus_pwm.c'** file.

```
/*!
 * @brief Interrupt service function of switch.
 *
 */
void PORTB_PORTC_IRQHandler(void)
{
    uint32_t interruptStatus = GPIO_GetPinsInterruptFlags(GPIOC);

    if(interruptStatus & (1U << PIN3_IDX)) /*PTC3 interruption*/
    {
        /* Clear external interrupt flag. */
        GPIO_ClearPinsInterruptFlags(GPIOC, 1U << PIN3_IDX);

        /* Change state of led. */
        GPIO_TogglePinsOutput(BOARD_LED3_GPIO, 1U << BOARD_LED3_GPIO_PIN);
    } else if(interruptStatus & (1U << PIN2_IDX)) /*PTC2 interruption*/
    {
        /* Clear external interrupt flag. */
        GPIO_ClearPinsInterruptFlags(GPIOC, 1U << PIN2_IDX);

        /* Change state of led. */
        GPIO_TogglePinsOutput(BOARD_LED_RED_GPIO, 1U << BOARD_LED_RED_GPIO_PIN);
    }
}
```

Figure 23. IRQ Handler function for PWM use case

5.2.4 Pass-Through mode

The project structure of the Pass-Through mode can be found in Figure 24.

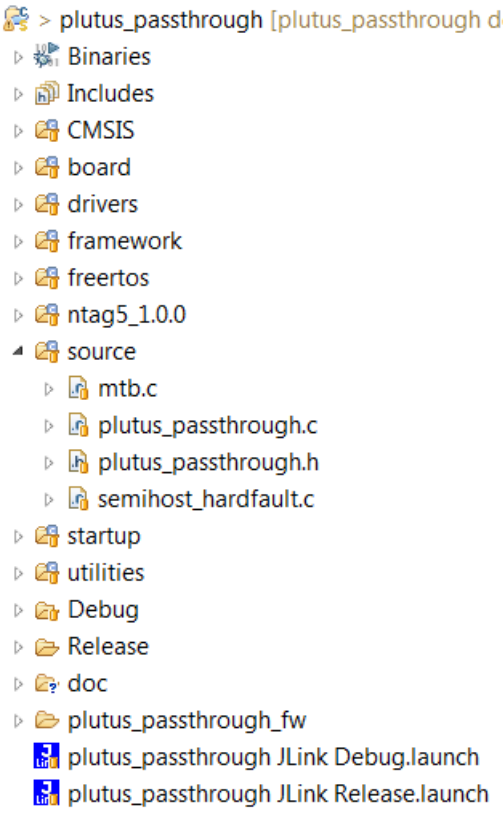

Figure 24. Pass-Through mode project structure

At application startup (in '**plutus_passthrough.c**' file), it is necessary to initialize some elements in the board (button SW3 to switch direction, event detection pin, etc.), the LED framework and NTAG 5 middleware. These initializations are displayed in Figure 25.

Important: Please remember to configure the IC in I^2C Slave mode to use this functionality.

```
/* Init input switch 3. Interrupt for portC is enabled later, in phApp_Configure_IRQ() */
PORT_SetPinInterruptConfig(BOARD_SW3_PORT, BOARD_SW3_GPIO_PIN, kPORT_InterruptFallingEdge);
GPIO_PinInit(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN, &sw_input_config);

#ifndef PH_OSAL_NULLOS
        phOsal_ThreadObj_t DemoApp;
#endif /* PH_OSAL_NULLOS */

        /* Initialize KW41Z board. */
        phApp_KW41Z_Init();

        /* Initialize LED framework. */
        LED_Init();

        /* RF->I2C is the default direction for pass-through, set Blue LED to indicate it. */
        Led4On();

        read_loop_counter = 0;
        write_loop_counter = 0;

        rf_i2c_interrupt = false;
        i2c_rf_interrupt = false;

        /* Perform OSAL Initialization. */
        (void)phOsal_Init();

        /* Initialize BAL register */
        phbalReg_Init(&sbalReg, sizeof(sbalReg));

        /* Configure IRQ for pin 17 (ED pin) */
        (void)phApp_Configure_IRQ();
```

Figure 25. Pass-Through mode project initialization

The Event Detection pin is configured in the '*phApp_Configure_IRQ()*' function inside the project initialization. This pin configuration is shown in Figure 26.

```
static phStatus_t phApp_Configure_IRQ()
{
    phDriver_Pin_Config_t pinCfg;

    pinCfg.bOutputLogic = PH_DRIVER_SET_LOW;
    pinCfg.bPullSelect = PHDRIVER_PIN_IRQ_PULL_CFG;

    pinCfg.eInterruptConfig = PIN_IRQ_TRIGGER_TYPE;
    phDriver_PinConfig(PHDRIVER_PIN_IRQ, PH_DRIVER_PINFUNC_INTERRUPT, &pinCfg);

    NVIC_SetPriority(EINT_IRQn, EINT_PRIORITY);
    NVIC_ClearPendingIRQ(EINT_IRQn);
    EnableIRQ(EINT_IRQn);

    return 0;
}
```

Figure 26. Pass-Through mode Event Detection pin configuration

Finally, the main loop can be seen in Figure 27 and Figure 28. Figure 27 shows the main loop for the RF->I^{2}C direction and Figure 28 for the I^{2}C->RF direction.

For both directions, first it is necessary to configure the Pass-Through direction in the NTAG by writing the TAG_CONFIG_1 register (more information about Pass-Through configuration can be found in the product data sheet [Datasheet]). Once the IC is configured, the next loop iteration goes directly to the Pass-Through functionality.

It is also important to see that the Event Detection pin functionality is configured each time the IC reads or writes the SRAM. This is necessary to configure to trigger the interrupt in the ED pin each time the last byte of the SRAM has been read or write via RF.

```
if (g_Direction == rf_i2c)
{
    /* Check if the tag is already configured in this mode. */
    if(!tag_configured)
    {
        rf_sram_written = false; /* SRAM hasn't been written via RF yet */

        /* Check if VCC and NFC are OK. Also Check that SRAM_DATA_READY == 0,
         *  this indicates SRAM is ready to be written via RF */
        error_status =  NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
        if(!error_status && ((value & 0x0003) == 0x0003) && !((value & 0x0020) == 0x0020))
        {
            /*Enable the Pass through mode in session registers. set the direction as RF to I2C
             *  and activate SRAM_MAPPING*/
            error_status = NTAG_WriteRegister(ntag, NTAG_MEM_OFFSET_TAG_CONFIG0_REG,0xFF00, 0x0200);
            if(error_status) continue;
            error_status = NTAG_SetPthruOnOff(ntag, 0x01);
            if(error_status) continue;
            error_status = NTAG_SetTransferDir(ntag, RF_TO_I2C);
            if(error_status) continue;

            /* Check if PT direction is correctly set in the TAG STATUS REG */
            error_status = NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
            if(!error_status && ((value & 0x0007) == 0x0007)) tag_configured = true;
            else continue;

            Led1Off(); /* Operation LED in KW41Z is off, still no data has been read via I2C */

            rf_i2c_interrupt = false;

            /* Configure ED pin in NTAG5 to detect when the last SRAM byte has been written via RF */
            NTAG_ClearED(ntag);
            phDriver_PinClearIntStatus(PHDRIVER_PIN_IRQ);
            error_status = NTAG_WaitForEvent( ntag, NTAG_EVENT_LAST_BYTE_WRITTEN_BY_RF, 5, 1);
        }
    }
    else
    {
        if(rf_i2c_interrupt)
        {
            /* Check if NFC and VCC are OK. Also check that SRAM_DATA_READY == 1 so the data is
             *  ready to be read via I2C */
            error_status = NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
            if(!error_status && ((value & 0x0027) == 0x0027))
```

```
        {
            rf_sram_written = true; /* This confirms that the SRAM has been written via RF */
        }
        else{
            rf_sram_written = false;
            Led1Off();
        }

        if(rf_sram_written && (read_loop_counter<5))
        {
            /* Check if SRAM_DATA_READY is set and start reading the SRAM, also check if VCC and NFC are OK*/
            error_status =  NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
            if(!error_status && ((value & 0x0023) == 0x0023))
            {
                /* Read SRAM */
                error_status = NTAG_ReadBytes( ntag, NTAG_MEM_BLOCK_START_SRAM, &gSRamData[0], 256);
                if(error_status) continue;

                read_loop_counter++;

                if(read_loop_counter<5)
                {
                    rf_i2c_interrupt = false;

                    NTAG_ClearED(ntag);
                    phDriver_PinClearIntStatus(PHDRIVER_PIN_IRQ);
                    error_status = NTAG_WaitForEvent( ntag, NTAG_EVENT_LAST_BYTE_WRITTEN_BY_RF, 5, 1);
                }

                /* SRAM read via I2C is indicated with LED red ON*/
                Led1On();
            }
        }
        else if(read_loop_counter == 5) /* When the application loops for 5th time automatically change PT direction */
        {
            g_Direction = i2c_rf;
            Led3On();
            Led4Off();

            tag_configured = false;

            /* Reset all counters and interrupt indicators */
            Led1Off();

            read_loop_counter = 0;
            write_loop_counter = 0;
```

```
                rf_i2c_interrupt = false;
                i2c_rf_interrupt = false;
            }
        }
}
```

Figure 27. Pass-Through RF->I²C main loop

In both Figures can be observed how the Pass-Through performs a 5 time loop before automatically switch to the other direction. When the direction is changed everything is reset, the LED3 and LED4 toggle their status and the IC is configured with the new Pass-Through direction.

```
        else if (g_Direction == i2c_rf)
        {
            /* Check if the tag is already configured in this mode*/
            if(!tag_configured)
            {
                rf_sram_read = false;

                /* Check if VCC and NFC are OK*/
                error_status =  NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
                if(!error_status && ((value & 0x0003) == 0x0003))
                {
                    /*Enable the Pass through mode in session registers. set the direction
                     *  as I2C to RF and activate SRAM_MAPPING*/
                    error_status = NTAG_WriteRegister(ntag, NTAG_MEM_OFFSET_TAG_CONFIG0_REG,0xFF00, 0x0200);
                    if(error_status) continue;
                    error_status = NTAG_SetPthruOnOff(ntag,0x01);
                    if(error_status) continue;
                    error_status = NTAG_SetTransferDir(ntag, I2C_TO_RF);
                    if(error_status) continue;

                    /* Check that NFC and VCC are OK. Check that PT mode is correctly configured */
                    error_status = NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
                    if(!error_status && ((value & 0x0003) == 0x0003)) tag_configured = true;
                    else continue;

                    /* Write the SRAM for the first time when PT mode is configured*/
                    error_status = NTAG_WriteBytes(ntag,  NTAG_MEM_BLOCK_START_SRAM, &gSRamDataFixed[0], 256);
                    if(error_status) continue;
                    else write_loop_counter++;

                    /* SRAM written via I2C indicated with LED red ON*/
                    Led1On();

                    /* Configure the ED pin to trigger the interrupt when the last byte has been read via RF.*/
                    i2c_rf_interrupt = false;
                    NTAG_ClearED(ntag);
                    phDriver_PinClearIntStatus(PHDRIVER_PIN_IRQ);
                    error_status = NTAG_WaitForEvent( ntag, NTAG_EVENT_LAST_BYTE_READ_FROM_RF, 5, 1);
                }
            }
            else
            {
                if(i2c_rf_interrupt)
                {
```

```
    /* Before start writing check that SRAM_DATA_REAY == 0, which means
     *  the data has been read via RF and is ready
     * to be written via I2C. Also check that NFC and VCC are ok.*/
    error_status = NTAG_ReadRegister( ntag, NTAG_MEM_OFFSET_TAG_STATUS_REG, &value);
    if(!error_status && ((value & 0x0003) == 0x0003) && !((value & 0x0020) == 0x0020)) rf_sram_read = true;
    else{
        rf_sram_read = false;
        Led1Off();
    }

    if(rf_sram_read && write_loop_counter < 5)
    {
        /* Write SRAM via I2C.*/
        error_status = NTAG_WriteBytes(ntag,  NTAG_MEM_BLOCK_START_SRAM, &gSRamDataFixed[0], 256);
        if(error_status) continue;

        write_loop_counter++;

        if(write_loop_counter < 5)
        {
            /* Configure the ED pin to trigger the interrupt when the last byte has been read via RF.*/
            i2c_rf_interrupt = false;
            NTAG_ClearED(ntag);
            phDriver_PinClearIntStatus(PHDRIVER_PIN_IRQ);
            error_status = NTAG_WaitForEvent( ntag, NTAG_EVENT_LAST_BYTE_READ_FROM_RF, 5, 1);
        }

        /* SRAM written via I2C indicated with LED red ON*/
        Led1On();
    }
    else if(write_loop_counter == 5) /* When the application loops for 5th time automatically
     change PT direction */
    {
        g_Direction = rf_i2c;
        Led3Off();
        Led4On();

        tag_configured = false;

        /* Reset all counters and interrupt indicators */
        Led1Off();

        read_loop_counter = 0;
        write_loop_counter = 0;

        i2c_rf_interrupt = false;
        rf_i2c_interrupt = false;
```

Figure 28. Pass-Through I²C->RF main loop

5.2.5 ALM use case

Figure 29 shows the project structure of the Active Load Modulation use case. As explained in the previous section, this use case allows controlling the configuration parameters of the Active Load Modulation mode.

More information on which these parameters and their possible values you find in [Datasheet].

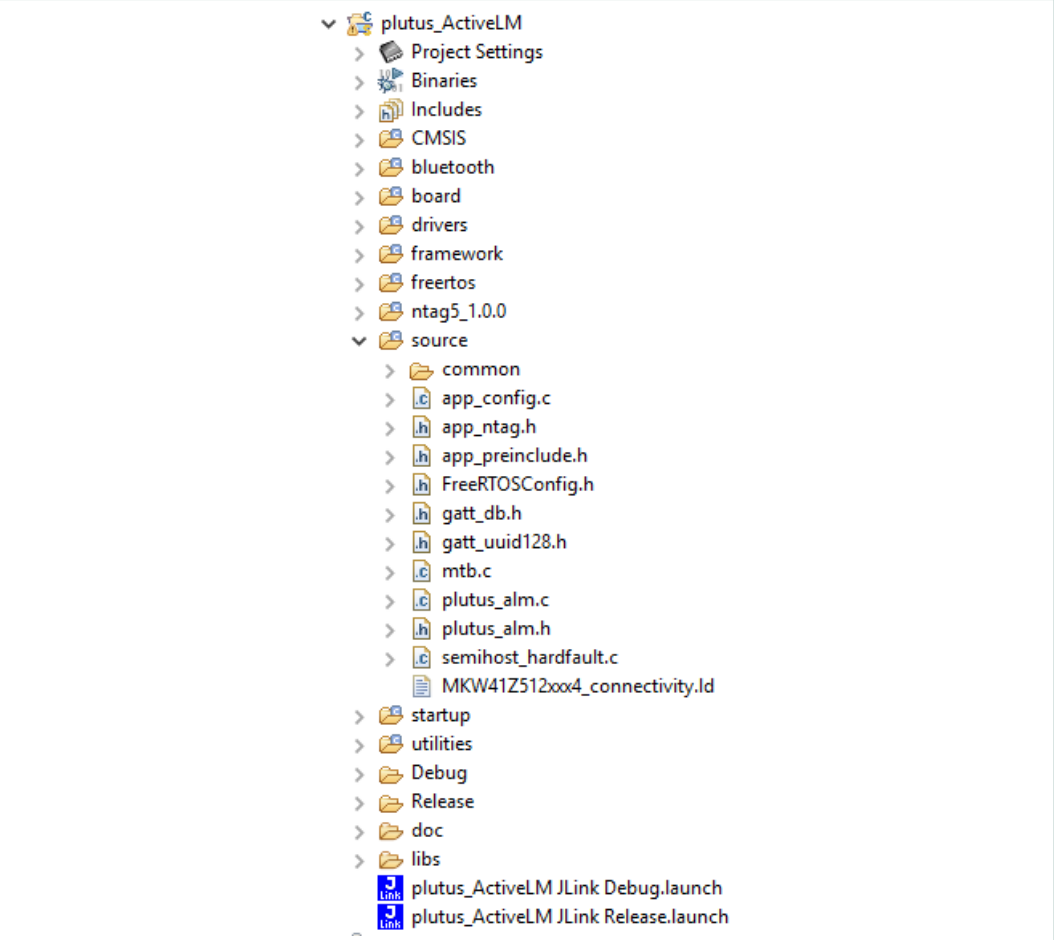

Figure 29. ALM use case folder structure.

The project contains the middleware for the NTAG 5 (source folder called '**NTAG 5_1.0.0**'). At the beginning of the application (**App_Thread** function in **ApplMain.c** file contained in **source/common** folder), it is necessary to initialize the NTAG middleware using **NTAG_InitDevice** function, this function belongs to the **NTAG 5_1.0.0/HAL_NTAG/nIC_driver.c** file.

This use case uses the LPUART of the FRDM-KW41Z to send and receive the values of the parameters that are read from the NTAG 5 though the serial port. Figure 30 shows how the LPUART is initialized in the **ApplMain.c** file.

```
LPUART_GetDefaultConfig(&lpuartConfig);
lpuartConfig.baudRate_Bps = BOARD_DEBUG_UART_BAUDRATE;
lpuartConfig.enableTx = true;
lpuartConfig.enableRx = true;

LPUART_Init(DEMO_LPUART, &lpuartConfig, DEMO_LPUART_CLK_FREQ);

/* Init DMAMUX */
DMAMUX_Init(EXAMPLE_LPUART_DMAMUX_BASEADDR);
/* Set channel for LPUART */
DMAMUX_SetSource(EXAMPLE_LPUART_DMAMUX_BASEADDR, LPUART_TX_DMA_CHANNEL, LPUART_TX_DMA_REQUEST);
DMAMUX_SetSource(EXAMPLE_LPUART_DMAMUX_BASEADDR, LPUART_RX_DMA_CHANNEL, LPUART_RX_DMA_REQUEST);
DMAMUX_EnableChannel(EXAMPLE_LPUART_DMAMUX_BASEADDR, LPUART_TX_DMA_CHANNEL);
DMAMUX_EnableChannel(EXAMPLE_LPUART_DMAMUX_BASEADDR, LPUART_RX_DMA_CHANNEL);

/* Init the EDMA module */
EDMA_GetDefaultConfig(&config);
EDMA_Init(EXAMPLE_LPUART_DMA_BASEADDR, &config);
EDMA_CreateHandle(&g_lpuartTxEdmaHandle, EXAMPLE_LPUART_DMA_BASEADDR, LPUART_TX_DMA_CHANNEL);
EDMA_CreateHandle(&g_lpuartRxEdmaHandle, EXAMPLE_LPUART_DMA_BASEADDR, LPUART_RX_DMA_CHANNEL);

/* Create LPUART DMA handle. */
LPUART_TransferCreateHandleEDMA(DEMO_LPUART, &g_lpuartEdmaHandle, LPUART_UserCallback, NULL, &g_lpuartTxEdmaHandle,
        &g_lpuartRxEdmaHandle);

/* Start to echo. */
sendXfer.data = g_txBuffer;
sendXfer.dataSize = ECHO_BUFFER_LENGTH_TX;
receiveXfer.data = g_rxBuffer;
receiveXfer.dataSize = ECHO_BUFFER_LENGTH;
```

Figure 30. LPUART initialization.

After initializing the LPUART, the MCU enters in a loop which is constantly listening to the USB port. When something is received, the interrupt shown in Figure 31 is triggered and the **g_rxBufferFull** variable is set to true, indicating that there is data to be read in the reception buffer.

```
void LPUART_UserCallback(LPUART_Type *base, lpuart_edma_handle_t *handle, status_t status, void *userData
{
    userData = userData;

    if (kStatus_LPUART_TxIdle == status)
    {
        txBufferFull = false;
        txOnGoing = false;
    }

    if (kStatus_LPUART_RxIdle == status)
    {
        rxBufferEmpty = false;
        rxOnGoing = false;
    }
}
```

Figure 31. LPUART callback.

Figure 32 shows the loop which controls the LPUART channel. The received message will be stored in g_rxBuffer.

If a WRITE request is received (**g_rxBuffer[0] = 0x01**), the memory address where the ALM parameters are stored (0x1040) is extracted and the four bytes of data are written in the NTAG using the **NTAG_WriteBytes** function.

On the other hand, if a READ request is received (**g_rxBuffer[0] = 0x02**), the address to be read is extracted and the **NTAG_ReadBytes** function is used to get the bytes of information and store them in read_buffer. After a small delay, read_buffer is copied into the transmission buffer g_txBuffer and the variable txBufferFull, which indicates there is something to be sent through the serial port, is set to true,

```
/* If RX is idle and g_rxBuffer is empty, start to read data to g_rxBuffer. */
if ((!rxOnGoing) && rxBufferEmpty)
{
    rxOnGoing = true;
    LPUART_ReceiveEDMA(DEMO_LPUART, &g_lpuartEdmaHandle, &receiveXfer);
}

/* If TX is idle and g_txBuffer is full, start to send data. */
if ((!txOnGoing) && txBufferFull)
{
    txOnGoing = true;
    LPUART_SendEDMA(DEMO_LPUART, &g_lpuartEdmaHandle, &sendXfer);
}

/* If g_txBuffer is empty and g_rxBuffer is full, copy g_rxBuffer to g_txBuffer. */
if ((!rxBufferEmpty) && (!txBufferFull))
{
    //Once a TLV has been received and stored in g_rxBuffer, analyze if it is R/W
    //and perform necessary actions. Store the answer in g_txBuffer.

    if (g_rxBuffer[0] == 0x01){
        mem_addr = ((uint16_t)g_rxBuffer[2] << 8) | g_rxBuffer[3];
        for (i = 0; i < 4; i++){
            write_buffer[i] = g_rxBuffer[i+4];
        }
        trans_status = NTAG_WriteBytes(ntag, mem_addr, write_buffer, NTAG_I2C_BLOCK_SIZE);

        rxBufferEmpty = true;
        txBufferFull = true;
    }else if (g_rxBuffer[0] == 0x02){
        mem_addr = ((uint16_t)g_rxBuffer[2] << 8) | g_rxBuffer[3];
        trans_status = NTAG_ReadBytes(ntag, mem_addr, read_buffer, NTAG_I2C_BLOCK_SIZE);
        activeDelay();
        memcpy(g_txBuffer, read_buffer, NTAG_I2C_BLOCK_SIZE);

        rxBufferEmpty = true;
        txBufferFull = true;
    }else{
        //Non-applicable command
    }
}
```

Figure 32. Control of the LPUART channel.

6 Flash NTAG use cases in FRDM-KW41Z using Peek and Poke

The use cases that are described in this document are meant to present the main functionalities of the NTAG 5. They can be flashed into the FRDM-KW41Z board using Peek and Poke PC Application [PEEK&POKE_UM], which can be downloaded from [PEEK&POKE_INSTALLER].

Once the Peek and Poke application has been downloaded and installed, run it by double-clicking the icon. Using a micro USB cable, connect the FRDM-KW41Z development board to your PC. A pop-up menu will appear, which allows you to select the NTAG you are going to use. When the NTAG 5 option is chosen, a new menu will appear, as shown in Figure 33.

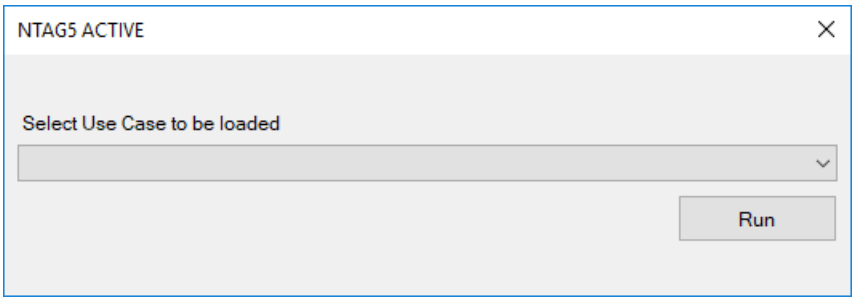

Figure 33. NTAG 5 use case selection menu.

To run one of the uses cases, simply choose the use case in the drop-down menu and click Run. After the use case is successfully flashed, a message indicating that the FRDM-KW41Z must be reconnected will appear. Do so and the use case will be executed.

In the case that the ALM use case if flashed, a new menu will appear. This menu allows configuring the parameters that control the Active NFC behavior.

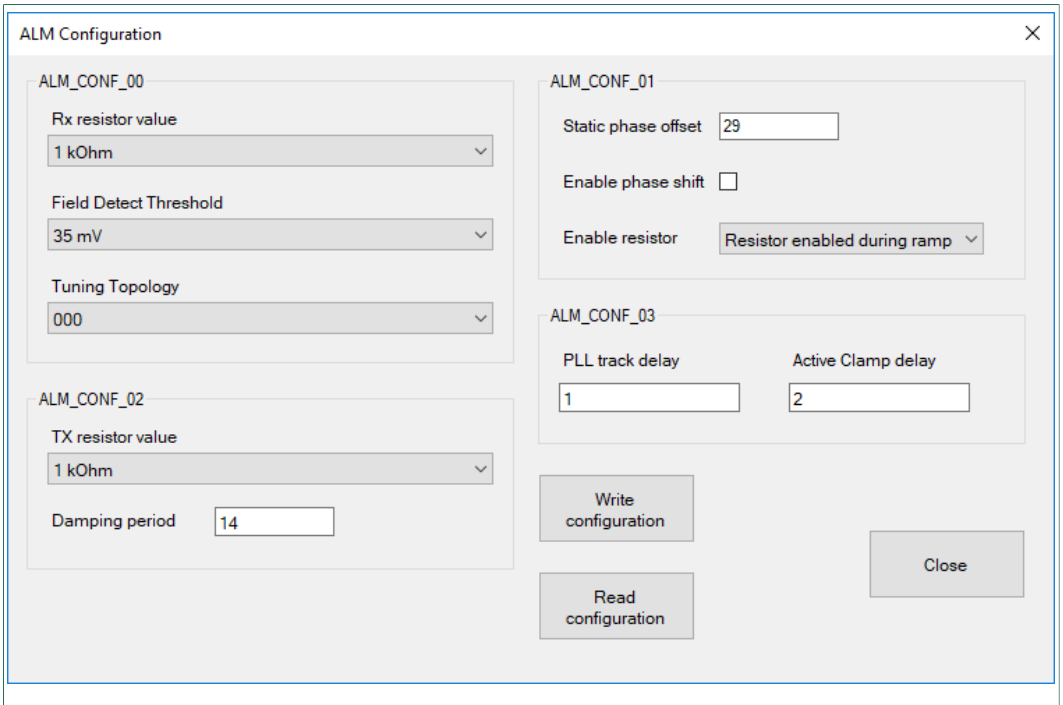

Figure 34. ALM Configuration control menu.

7 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] 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

[3] NFC Forum specification, Type 5 Tag - Technical Specification Version 1.0 2018-04-27 [T5T] NFC Forum™
https://nfc-forum.org/product-category/specification/

[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

8 Legal information

8.1 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.

8.2 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.

8.3 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.

8.4 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.

Bluetooth — The Bluetooth word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Figures

Contents

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 number: 531910