

# HOW TO DEVELOP WITH NTAG 5

## NTAG 5 WEBINAR SERIES

PABLO FUENTES

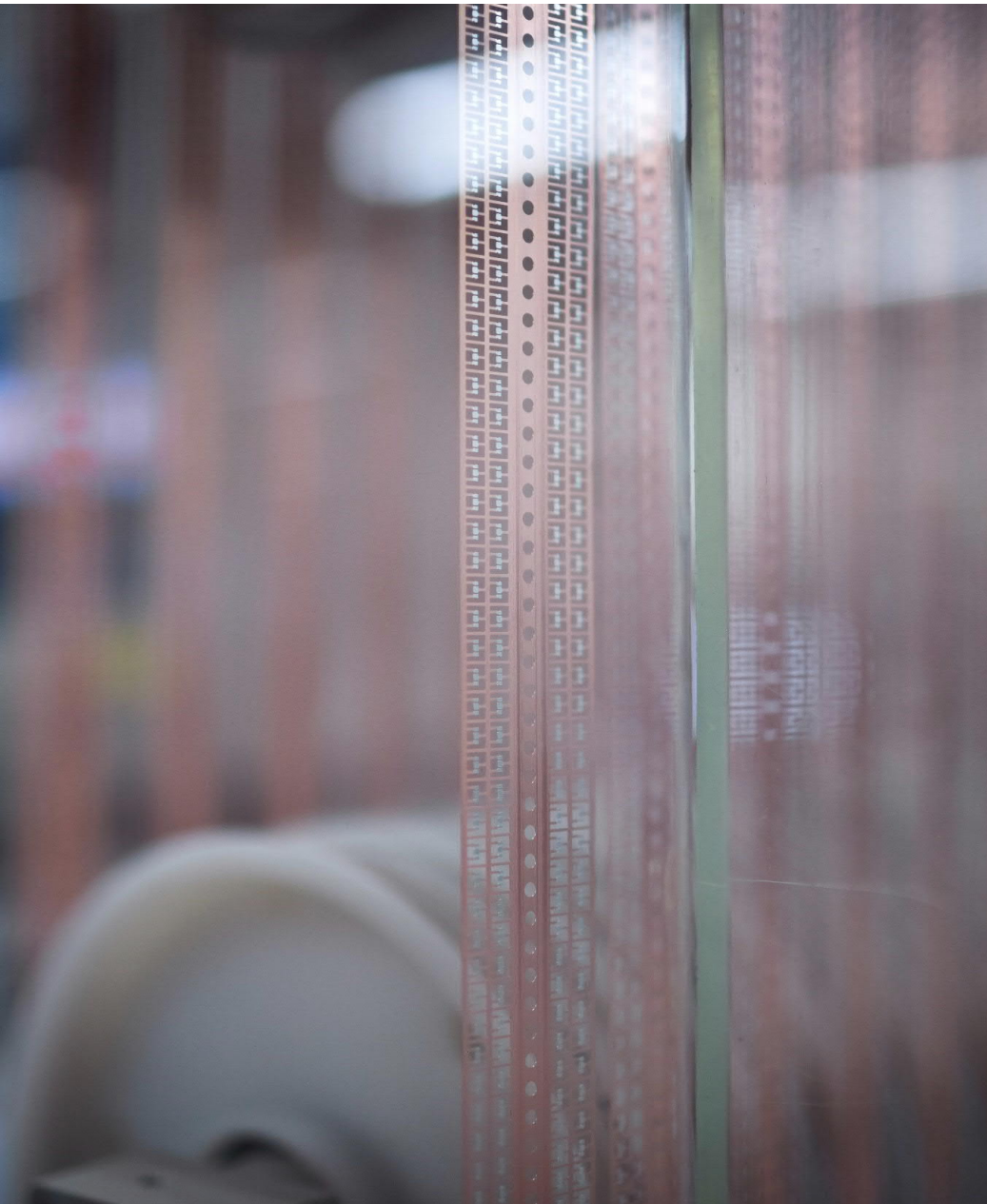
FEBRUARY 2020



PUBLIC



SECURE CONNECTIONS  
FOR A SMARTER WORLD



## Agenda

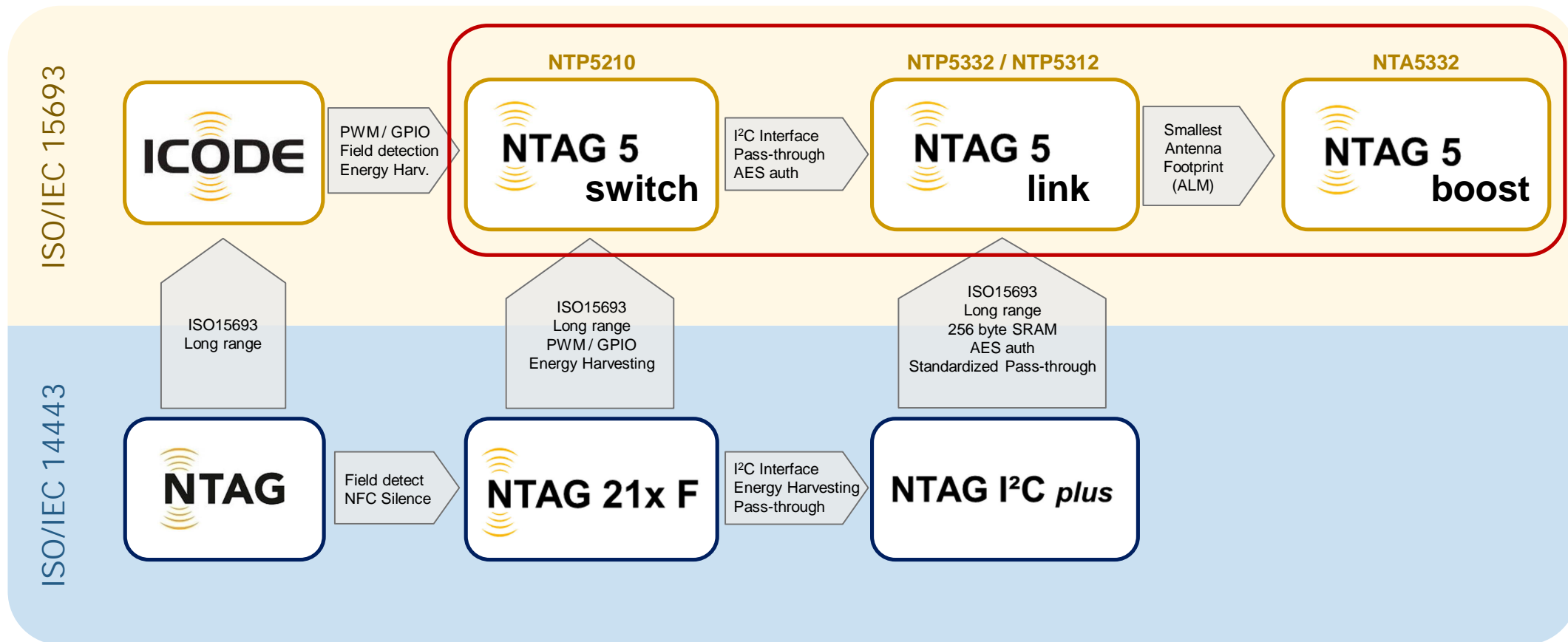
- NTAG 5 Family Overview
- General development considerations
- Using GPIO features
- Using PWM features
- Using Pass-through mode
- Using I<sup>2</sup>C master mode
- More support

# NTAG 5 Family Overview



# NTAG 5 Family Overview

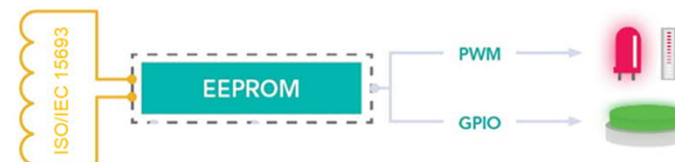
## Positioning



# NTAG 5 Family

## NTAG 5 switch

- Control and dim LEDs
- Calibrate reference current without MCU
- Verify authenticity of the device



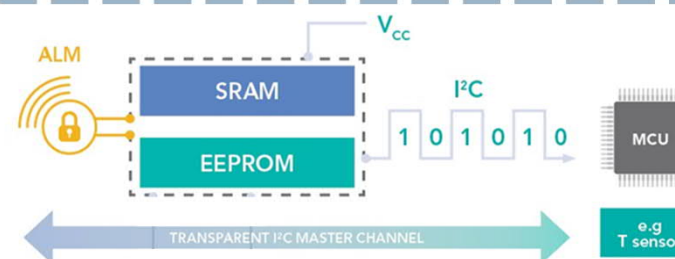
## NTAG 5 link

- Draw power from the NFC reader to supply sensors
- Read out sensor information without an MCU\*
- Secure sensor interaction



## NTAG 5 boost

- Smallest footprint antenna
- Enables NTAG 5 link features for tiny solutions

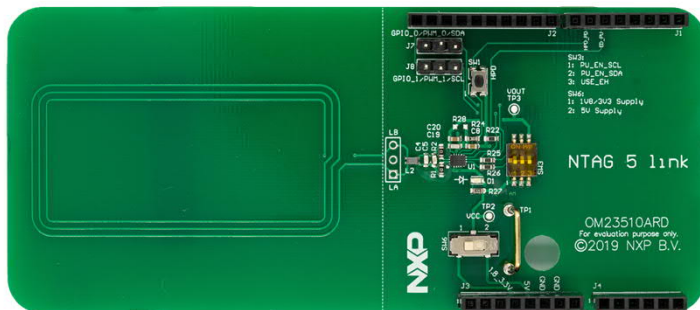


# NTAG 5 Family Overview

## Development kits

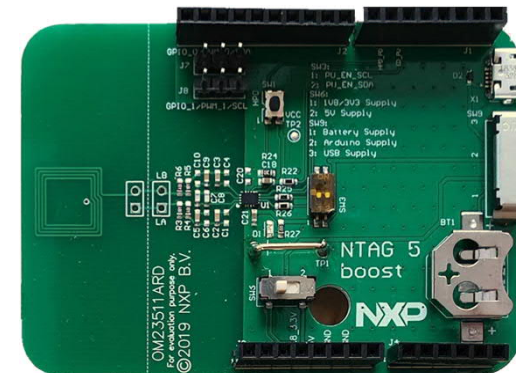
### NTAG 5 link Evaluation board (OM23510ARD)

- Integrating NTAG 5 link (NTP5332)
- 54 x 27 mm Plutus antenna
- Jumper to select between different supply voltages
- Hard-power-down button
- Arduino header
- Easy to access wired interface signals through pins



### NTAG 5 boost Evaluation board (OM23511ARD)

- Integrating NTAG 5 boost (NTA5332)
- 10 x 10 mm Active antenna
- Jumper to select between different supply voltages
- Hard-power-down button
- Arduino header
- Easy to access wired interface signals through pins



# General development considerations





# General development considerations

## Content

- Main supported commands (NFC interface)
- Configuring NTAG 5 wired interface
- Setup used for examples





# General development considerations

## Main commands supported (NFC interface)

WRITE\_CONFIG (Command code C1h)

Flags	WRITE_CONFIG	Manuf. Code	UID (optional)	Block Address	Data	CRC16
8 bits	8 bits	8 bits	64 bits	8 bits	32 bits	16 bits

**Command**



**Response**



Error Code

8 bits / 16 bits

READ\_CONFIG (Command code C0h)

Flags	READ_CONFIG	Manuf. Code	UID (optional)	Block Address	N° of blocks	CRC16
8 bits	8 bits	8 bits	64 bits	8 bits	8 bits	16 bits

**Command**



**Response**



Flags	Data	CRC16
8 bits	N° of block x 32 bits	16 bits

For Read Single Block and Write Single Block (EEPROM access) refer to ISO15693 or NFC Forum Type 5 tag specifications

For more information on Flags and Error code refer to ISO15693 specifications

# General development considerations

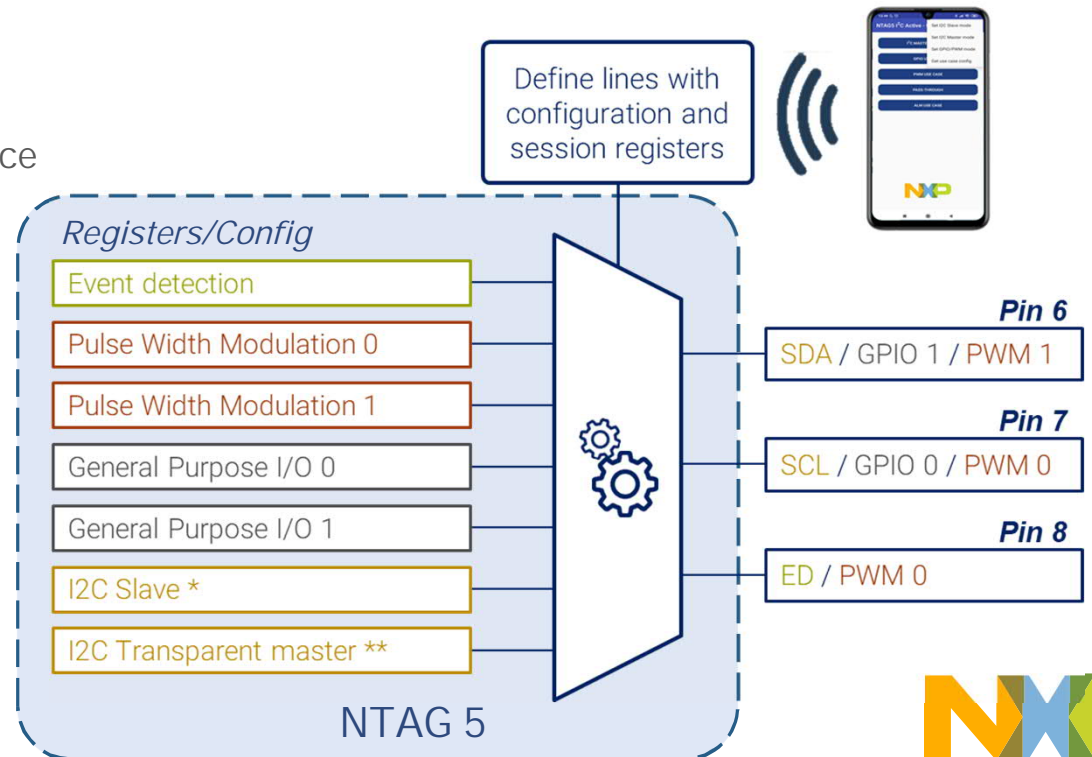
## Configuring wired interface

NTAG 5 wire interface must be configured depending on the application. It can be configured via:

- NFC Interface (Recommended)
  - Always configurable through NFC interface
- I<sup>2</sup>C Interface
  - Only available if preconfigured as I<sup>2</sup>C Slave
  - Configuration not reversible through I<sup>2</sup>C interface

\* I<sup>2</sup>C interface not supported in NTAG 5 switch version

\*\* I<sup>2</sup>C master only supported in NTP5332 and NTA5332



# General development considerations

## Configuring wired interface

- Wired interface is configured through USE\_CASE\_CONF parameter from Configuration bytes block.

Most of the wired interface registers have both configuration and session registers.

### Session registers:

- ✓ Changes take effect immediately
- x Not persistent after reset

### Configuration settings:

- ✓ Value remains valid after chip reset.
- x No immediate effect

  Session register address

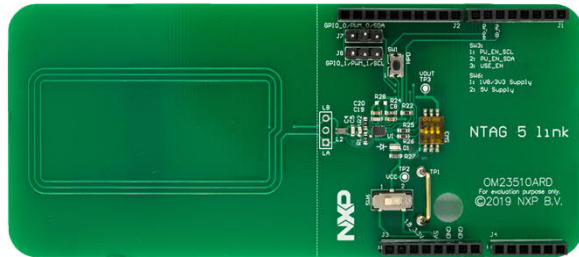
Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
37h	1037h	CONFIG_0	CONFIG_1	CONFIG_2	RFU
A1h	10A1h				

Bit	Name	Value	Description
7	EH_ARBITER_MODE_EN	0b	ARBITER_MODE needs to be set after startup
		1b	ARBITER_MODE is set automatically in any case after startup
6	ALM_PLM	0b	PLM
		1b	ALM mode when supplied by Vcc else PLM (default)
4-5	USE_CASE_CONF	00b	I <sup>2</sup> C slave (default)
		01b	I <sup>2</sup> C master
		10b	GPIO / PWM
		11b	All host interface functionality disabled
2-3	ARBITER_MODE	00b	Normal mode (default)
		01b	SRAM mirror mode
		10b	SRAM pass-through mode
		11b	SRAM PHDC mode
1	SRAM_ENABLE	0b	SRAM not accessible (default)
		1b	SRAM is available (when Vcc supplied)
0	PT_TRANSFER_DIR	0b	Data transfer direction is I <sup>2</sup> C to NFC (default)
		1b	Data transfer direction is NFC to I <sup>2</sup> C

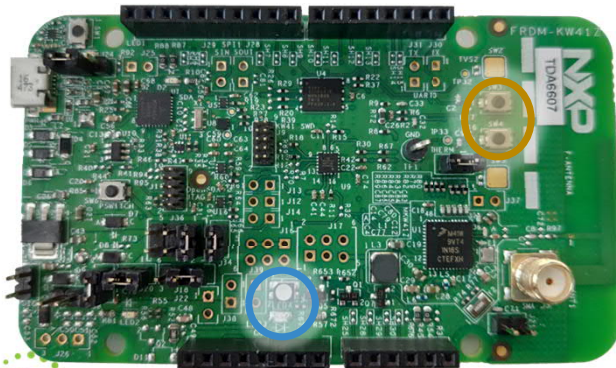
# General development considerations

## Setup used for examples

NTAG 5 link evaluation board



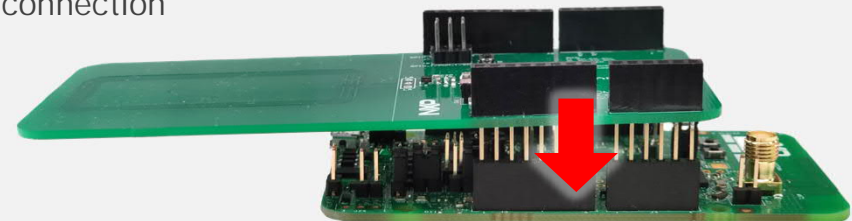
FRDM-KW41Z development board



KW41Z development board (FRDM-KW41Z)

- NXP's ultra-low-power KW41Z Wireless MCU
- Fully compliant Bluetooth v4.2 Low Energy
- 4-Mbit external serial flash memory for OTAP support
- Two LED indicator (One RGB and one red)
- Two push-button switches
- Two TSI buttons
- Arduino compatible header

Arduino header connection



# Using GPIO features



# General development considerations

## Configuring wired interface

- Wired interface is configured through USE\_CASE\_CONF parameter from Configuration bytes block.

Most of the wired interface registers have both configuration and session registers.

### Session registers:

- ✓ Changes take effect immediately
- x Not persistent after reset

### Configuration settings:

- ✓ Value remains valid after chip reset.
- x No immediate effect

  Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
37h	1037h	CONFIG_0	CONFIG_1	CONFIG_2	RFU
A1h	10A1h				

Bit	Name	Value	Description
7	EH_ARBITER_MODE_EN	0b	ARBITER_MODE needs to be set after startup
		1b	ARBITER_MODE is set automatically in any case after startup
6	ALM_PLM	0b	PLM
		1b	ALM mode when supplied by Vcc else PLM (default)
4-5	USE_CASE_CONF	00b	I <sup>2</sup> C slave (default)
		01b	I <sup>2</sup> C master
		10b	<b>GPIO / PWM</b>
		11b	All host interface functionality disabled
2-3	ARBITER_MODE	00b	Normal mode (default)
		01b	SRAM mirror mode
		10b	SRAM pass-through mode
		11b	SRAM PHDC mode
1	SRAM_ENABLE	0b	SRAM not accessible (default)
		1b	SRAM is available (when Vcc supplied)
0	PT_TRANSFER_DIR	0b	Data transfer direction is I <sup>2</sup> C to NFC (default)
		1b	Data transfer direction is NFC to I <sup>2</sup> C

# Using GPIO features

## Configuring wired interface

### Step 2

- Define if pads are used as GPIO or PWM
- For GPIO pads, we should also define if they are destined as output or input pads

Wired interface registers have both configuration and session registers.

#### Session registers:

- ✓ Changes take effect immediately
- x Not persistent after reset

#### Configuration settings:

- ✓ Value remains valid after chip reset.
- x No immediate effect

  Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
39h	1039h	PWM_GPIO_CONFIG_0_REG	PWM_GPIO_CONFIG_1_REG	RFU	
A3h	10A3h				

Bit	Name	Value	Description
7	GPIO1_SDA_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
6	GPIO0_SCL_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
5	GPIO1_SDA_PAD_IN_STATUS	0b	Input status
		1b	
4	GPIO0_SCL_PAD_IN_STATUS	0b	Input status
		1b	
3	GPIO1_SDA_PAD	0b	Output
		1b	Input
2	GPIO0_SCL_PAD	0b	Output
		1b	Input
1	GPIO1_PWM1_SDA_PAD	0b	GPIO
		1b	PWM
0	GPIO0_PWM0_SCL_PAD	0b	GPIO
		1b	PWM



# Using GPIO features

## Changing GPIO line state (output)

### Setting up line state

Write to PWM\_GPIO\_CONFIG\_REG on bit 6 or bit 7 depending on the line chosen

- Write 0b to set line to LOW state
- Write 1b to set line to HIGH state

Wired interface registers have both configuration and session registers.

#### Session registers:

- ✓ Changes take effect immediately
- x Not persistent after reset

#### Configuration settings:

- ✓ Value remains valid after chip reset.
- x No immediate effect

  Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
39h	1039h	PWM_GPIO_CONFIG_0_REG	PWM_GPIO_CONFIG_1_REG	RFU	
A3h	10A3h				

	Bit	Name	Value	Description
GPIO 1	7	GPIO1_SDA_PAD_OUT_STATUS	0b	Output status on pad is LOW
			1b	Output status on pad is HIGH
GPIO 0	6	GPIO0_SCL_PAD_OUT_STATUS	0b	Output status on pad is LOW
			1b	Output status on pad is HIGH
	5	GPIO1_SDA_PAD_IN_STATUS	0b	Input status
			1b	
	4	GPIO0_SCL_PAD_IN_STATUS	0b	Input status
			1b	
	3	GPIO1_SDA_PAD	0b	Output
			1b	Input
	2	GPIO0_SCL_PAD	0b	Output
			1b	Input
	1	GPIO1_PWM1_SDA_PAD	0b	GPIO
			1b	PWM
	0	GPIO0_PWM0_SCL_PAD	0b	GPIO
			1b	PWM

# Using GPIO features

## Reading GPIO line state (input)

### Monitoring line state

Read STATUS1\_REG bit 3 or bit 4 depending on the line chosen

- 0b indicates LOW level in the pad
- 1b indicates HIGH level in the pad

Wired interface registers have both configuration and session registers.

#### Session registers:

- ✓ Changes take effect immediately
- x Not persistent after reset

#### Configuration settings:

- ✓ Value remains valid after chip reset.
- x No immediate effect

  Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I2C				
A0h	10A0h	STATUS0_REG	STATUS1_REG	RFU	

GPIO 1

GPIO 0

Bit	Name	Value	Description
7	VCC_BOOT_OK	0b	VCC boot not done
		1b	VCC boot done
6	NFC_BOOT_OK	0b	NFC boot not done
		1b	NFC boot done
5	ACTIVE_NFC_OK	0b	ALM RF not OK
		1b	AKN RF OK
4	GPIO_PAD1_IN_STATUS	0b	GPIO_1 input is LOW
		1b	GPIO_1 input is HIGH
3	GPIO_PAD0_IN_STATUS	0b	GPIO_0 input is LOW
		1b	GPIO_0 input is HIGH
2	ALM_PLM	0b	Only Passive Load Modulation supported
		1b	Active Load Modulation supported
1	I2C_IF_LOCKED	0b	I2C interface not locked by arbiter
		1b	Arbiter locked to I2C
0	NFC_IF_LOCKED	0b	NFC interface not locked by arbiter
		1b	Arbiter locked to NFC

# LED example (output)

*using FRDM-KW41Z and NTAG 5 Demo app*

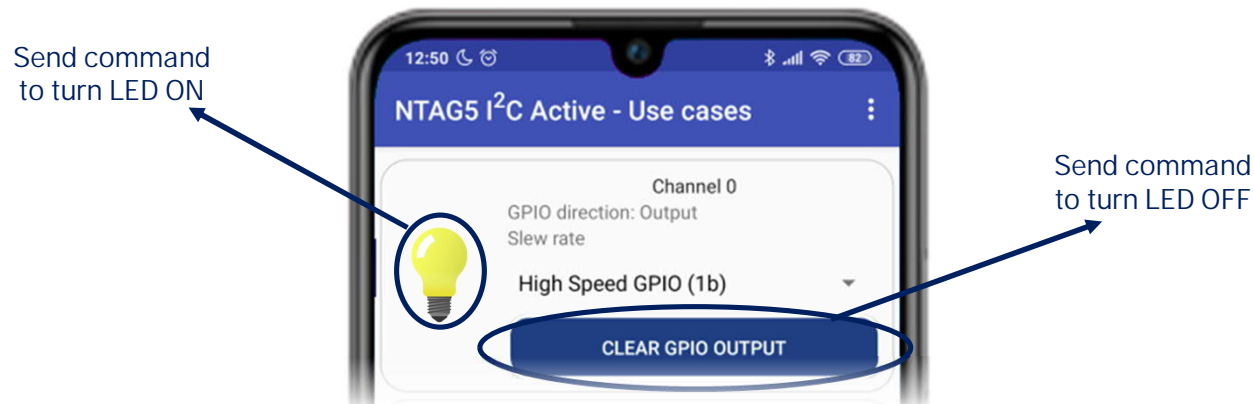


# Using GPIO features

## LED example

### Description

- Using GPIO signal configured as output to switch ON / OFF LED present in FRDM-KW41Z board
- KW41Z shall be flashed so MCU dumps input signal to LED red channel\*
- Example available in NTAG 5 Demo app for mobiles
- OM2351OARD shall be connected to FRDM-KW41Z



# Using GPIO features

## LED example

### Step 1

- WRITE\_CONFIG command (C1h) over Configuration Bytes block (37h):

```
@Override
public void onClick(DialogInterface dialog,
                    int which) {
    byte[] response = sendCommand(cmd_activateGPIOPWM);
    if (response != null)
        showConfirmationSnackbar();
}
```

```
public static final byte[] cmd_activateGPIOPWM = new byte[]{
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0x37,
    (byte) 0x00,
    (byte) 0x22,
    (byte) 0x0F,
    (byte) 0x00
};
```

Flags	WRITE_CONFIG	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	37h	00220F00	Auto

Config Bytes block address

WRITE\_CONFIG command code

GPIO/PWM mode

Non-address mode & High data rate



# Using GPIO features

## LED example

### Step 2

- Change line state using PWM\_GPIO\_CONFIG\_REG

#### Turn LED ON

```
public static final byte[] cmd_gpioSetSessionOutput = new byte[]{
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0xA3,
    (byte) 0x48,
    (byte) 0x00,
    (byte) 0x00,
    (byte) 0x00
};
```

Register A3 → PWM\_GPIO\_CONFIG\_REG

Both PADS as GPIO  
GPIO\_SDA\_PAD as input  
GPIO\_SCL\_PAD as output  
GPIO\_SCL\_PAD\_OUT\_STATUS set to **HIGH**

#### Turn LED OFF

```
public static final byte[] cmd_gpioClearSessionOutput = new byte[]{
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0xA3,
    (byte) 0x08,
    (byte) 0x00,
    (byte) 0x00,
    (byte) 0x00
};
```

Register A3 → PWM\_GPIO\_CONFIG\_REG

Both PADS as GPIO  
GPIO\_SDA\_PAD as input  
GPIO\_SCL\_PAD as output  
GPIO\_SCL\_PAD\_OUT\_STATUS set to **LOW**

Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
A3h	10A3h	PWM_GPIO_CONFIG_0_REG	PWM_GPIO_CONFIG_1_REG	RFU	

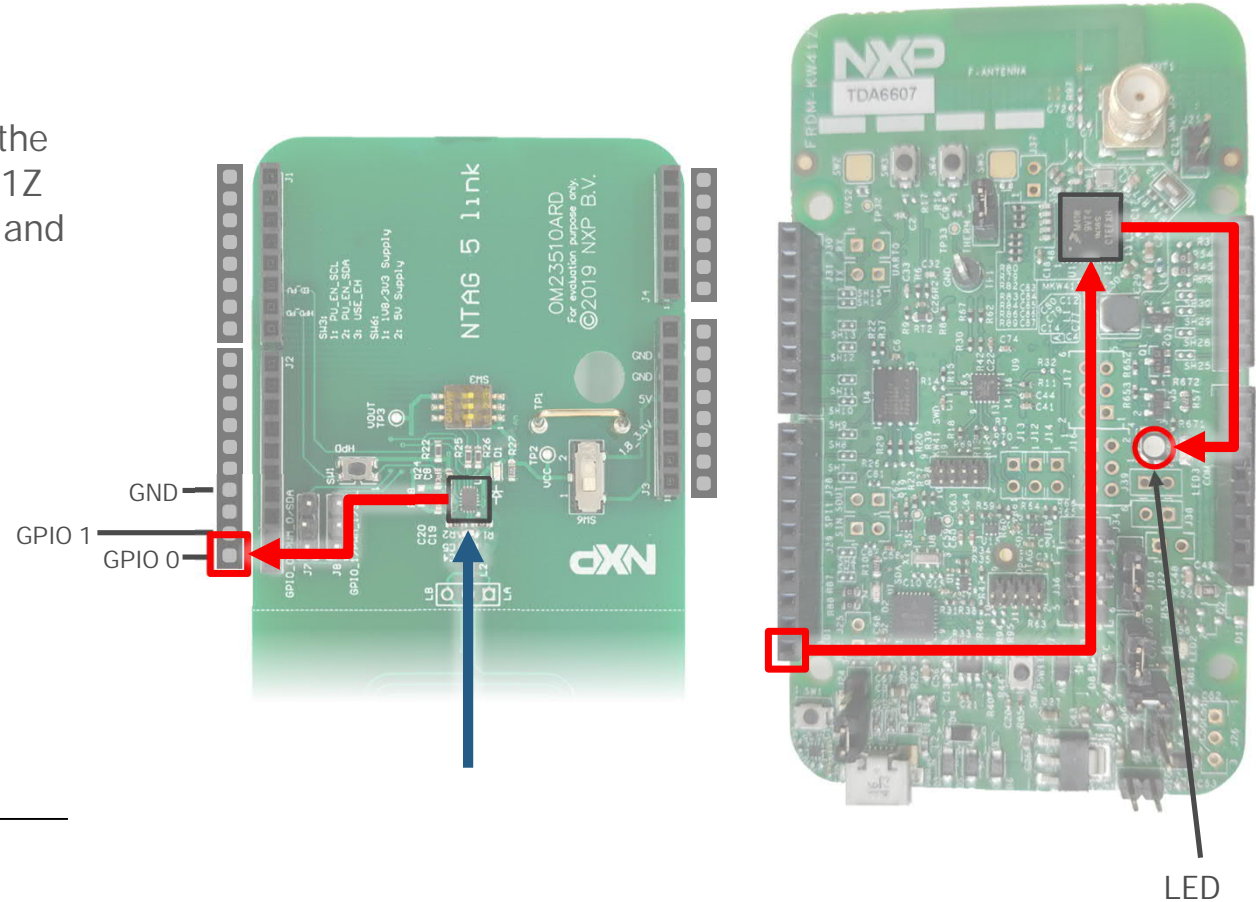
Bit	Name	Value	Description
7	GPIO_SDA_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
6	GPIO_SCL_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
5	GPIO_SDA_PAD_IN_STATUS	0b	Input status
		1b	
4	GPIO_SCL_PAD_IN_STATUS	0b	Input status
		1b	
3	GPIO_SDA_PAD	0b	Output
		1b	Input
2	GPIO_SCL_PAD	0b	Output
		1b	Input
1	GPIO_PWM1_SDA_PAD	0b	GPIO
		1b	PWM
0	GPIO_PWM0_SCL_PAD	0b	GPIO
		1b	PWM

# Using GPIO features

## LED example: Signal generation

1. Signal is generated by NTAG 5 depending on the register dedicated to control the GPIO 1. KW41Z monitors the signal generated by the NTAG 5 and dumps its value to turn ON/OFF LED 3.

GPIO 0 (output)



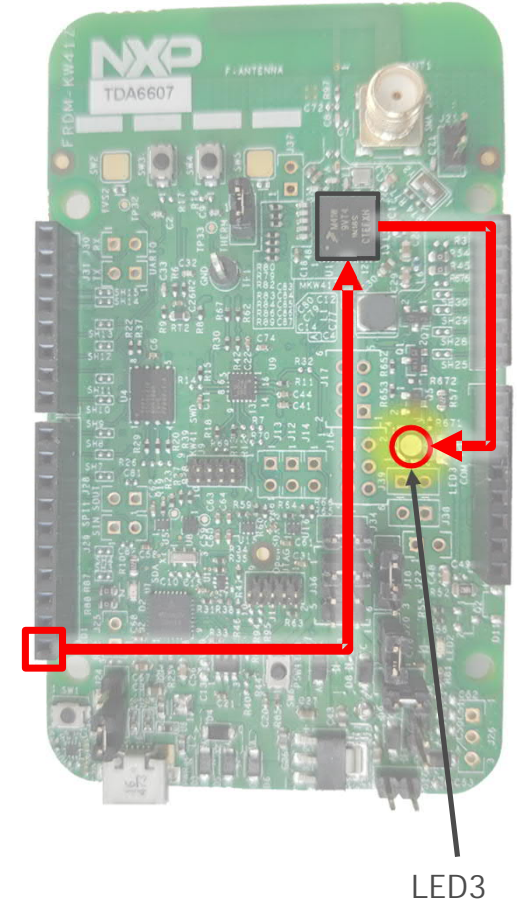
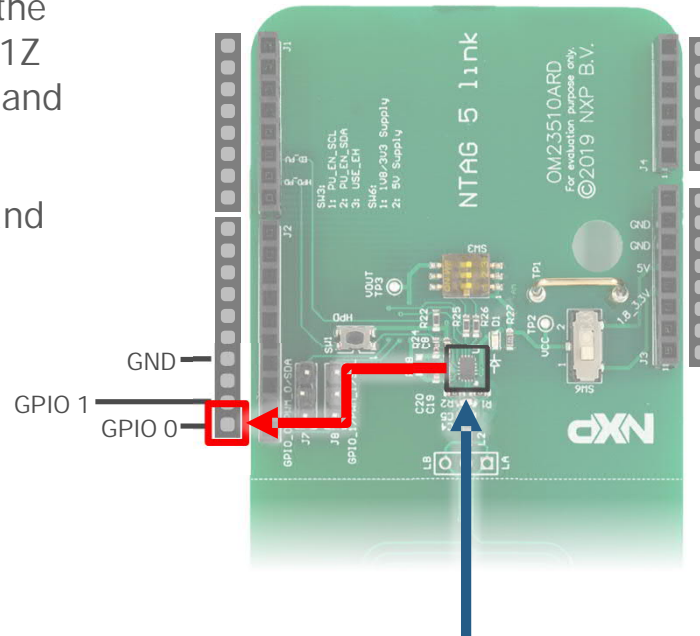
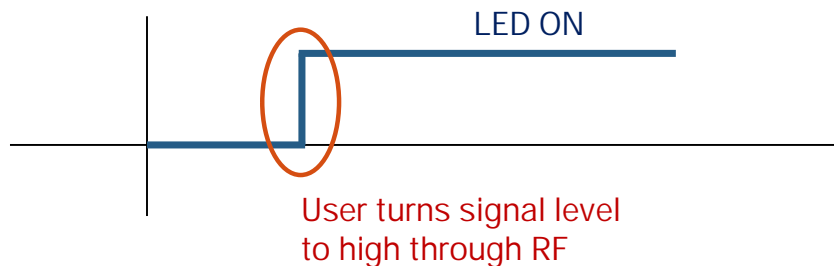


# Using GPIO features

## LED example: Signal generation

1. Signal is generated by NTAG 5 depending on the register dedicated to control the GPIO 1. KW41Z monitors the signal generated by the NTAG 5 and dumps its value to turn ON/OFF LED 3
2. User can control the level state of the signal and therefore the LED by writing to the specific register in NTAG 5 memory.

GPIO 0 (output)



Evaluation board image  
is **NOT** the final one

# Toggle button example (input)

*using FRDM-KW41Z and NTAG 5 Demo app*



# Using GPIO features

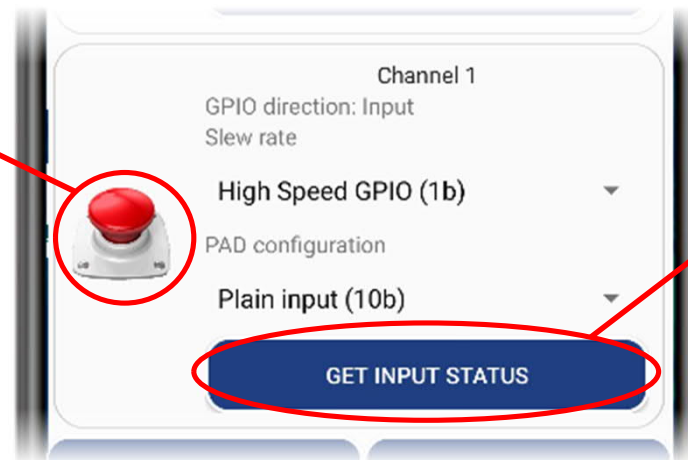
## Toggle button example

### Description

- Using GPIO signal configured as input to monitor button state
- Toggle button emulated using SW3 and SW4 buttons from FRDM-KW41Z
- KW41Z shall be flashed so MCU changes signal state depending on button clicked
- Example available in NTAG 5 Demo app for mobiles
- OM23510ARD shall be connected to FRDM-KW41Z

### Button state image:

- Pressed when signal is LOW
- Not pressed when signal is HIGH



Updates button  
state image



# Using GPIO features

## Toggle button example

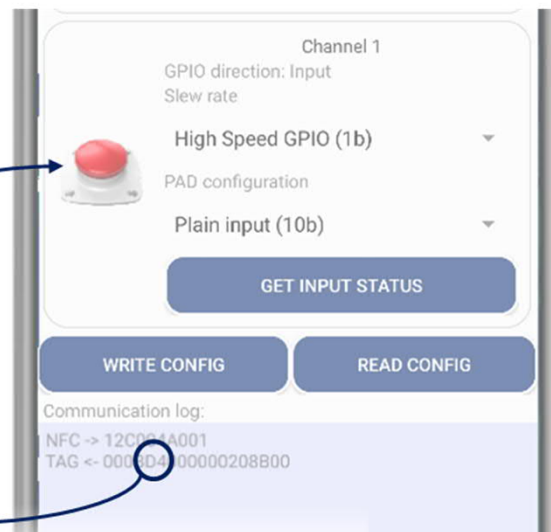
Check button state

```
public static final byte[] cmd_readTagStatus = new byte[]{
    (byte) 0x12,
    (byte) 0xC0,
    (byte) 0x04,
    (byte) 0xA0,
    (byte) 0x01
};
```

Register A0 → STATUS1\_REG

Update image

Response:  
**STATUS1\_REG = D4**  
Input is LOW (**Pressed**)



Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I2C				
A0h	10A0h	STATUS0_REG	STATUS1_REG	RFU	

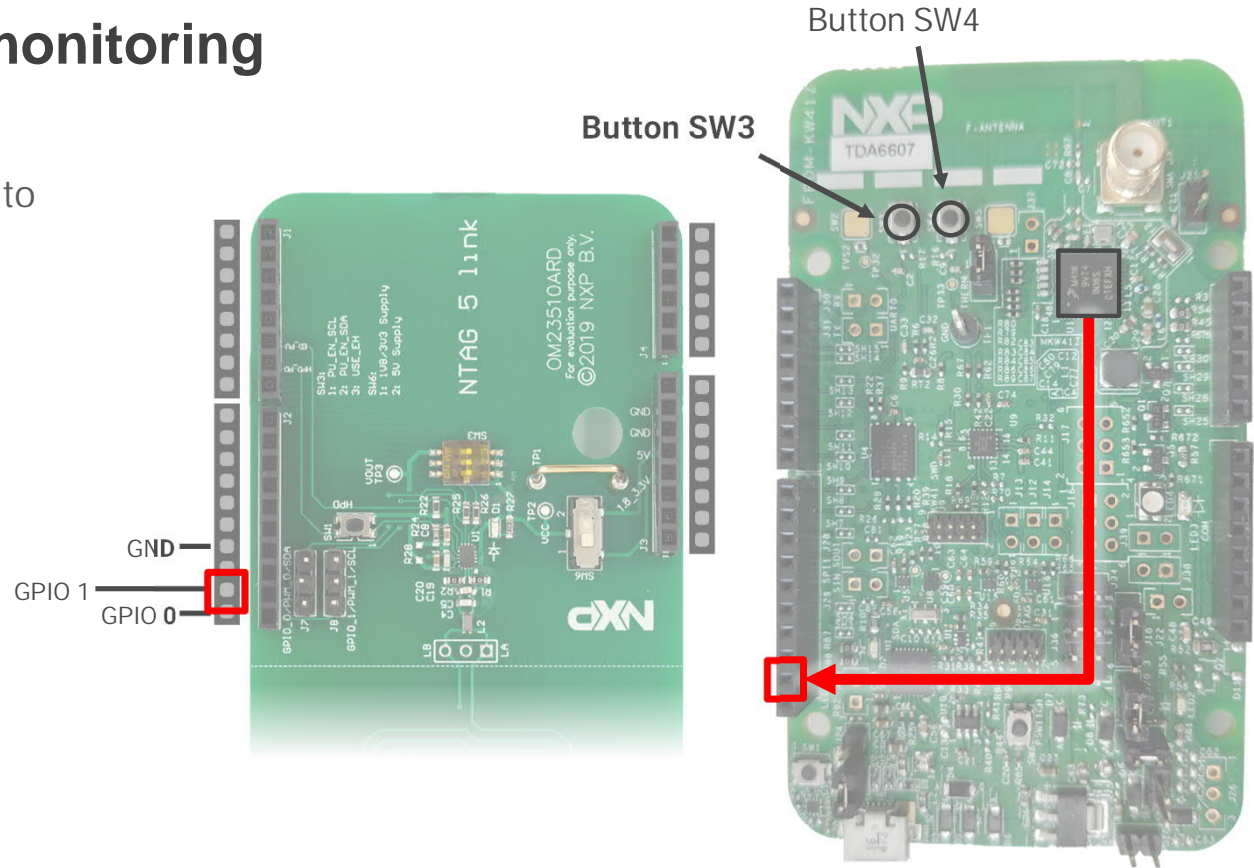
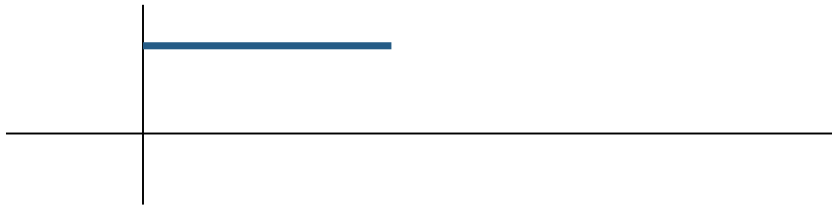
Bit	Name	Value	Description
7	VCC_BOOT_OK	0b	VCC boot not done
		1b	VCC boot done
6	NFC_BOOT_OK	0b	NFC boot not done
		1b	NFC boot done
5	ACTIVE_NFC_OK	0b	ALM RF not OK
		1b	ALM RF OK
4	GPIO_PAD1_IN_STATUS	0b	GPIO_1 input is LOW
		1b	GPIO_1 input is HIGH
3	GPIO_PAD0_IN_STATUS	0b	GPIO_0 input is LOW
		1b	GPIO_0 input is HIGH
2	ALM_PLM	0b	Only Passive Load Modulation supported
		1b	Active Load Modulation supported
1	I2C_IF_LOCKED	0b	I2C Interface not locked by arbiter
		1b	Arbiter locked to I2C
0	NFC_IF_LOCKED	0b	NFC interface not locked by arbiter
		1b	Arbiter locked to NFC

# Using GPIO features

## Toggle button example: Signal monitoring

1. Signal is generated by the KW41Z and rooted to the GPIO 1 pin of the NTAG 5 Eval board

GPIO 1 (input)



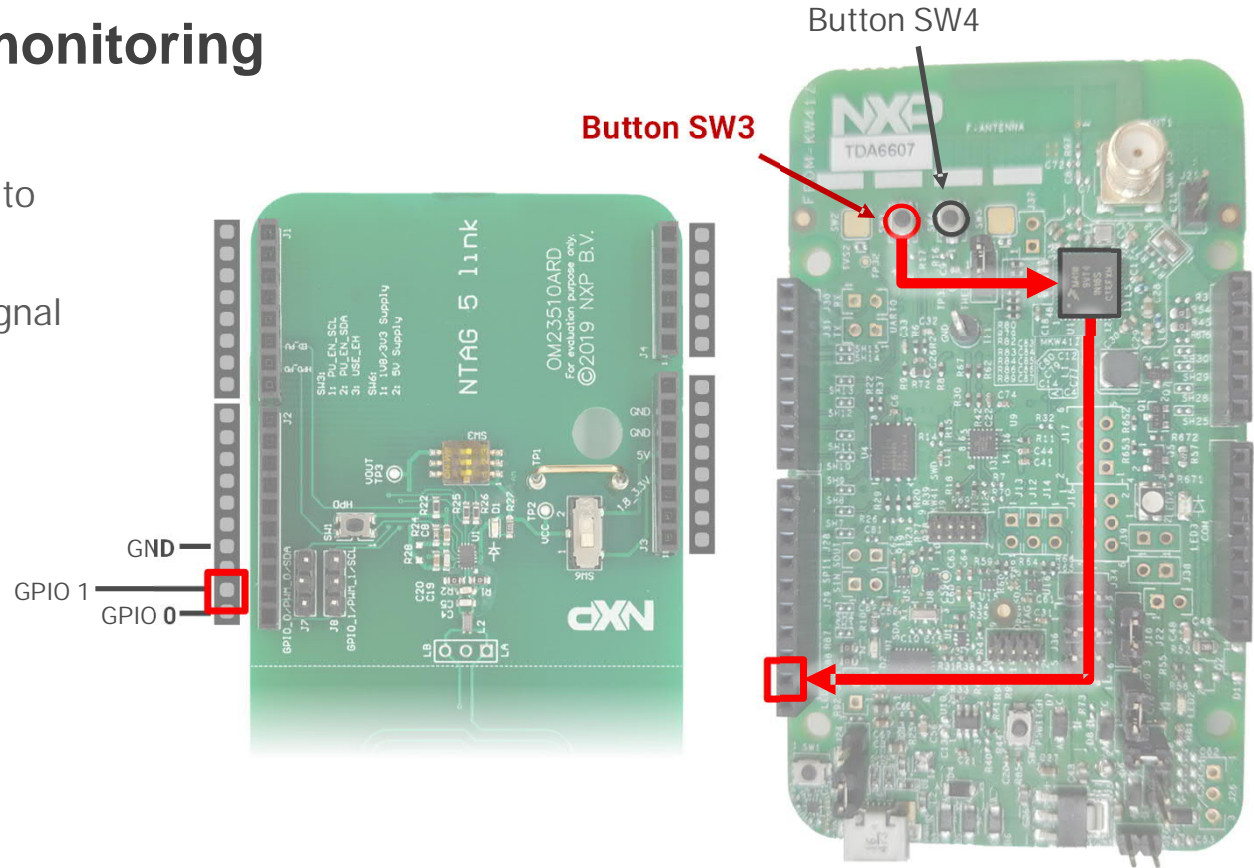
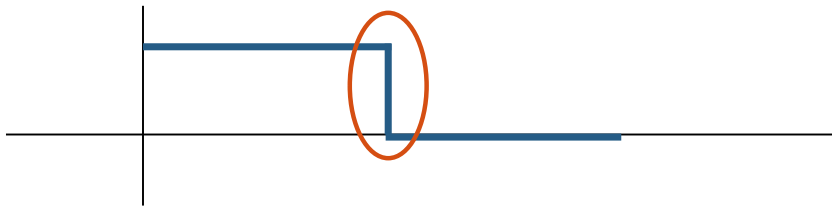


# Using GPIO features

## Toggle button example: Signal monitoring

1. Signal is generated by the KW41Z and rooted to the GPIO 1 pin of the NTAG 5 Eval board
2. If user clicks SW3 button, KW41Z turns the signal level to low state.

GPIO 1 (input) SW3 clicked

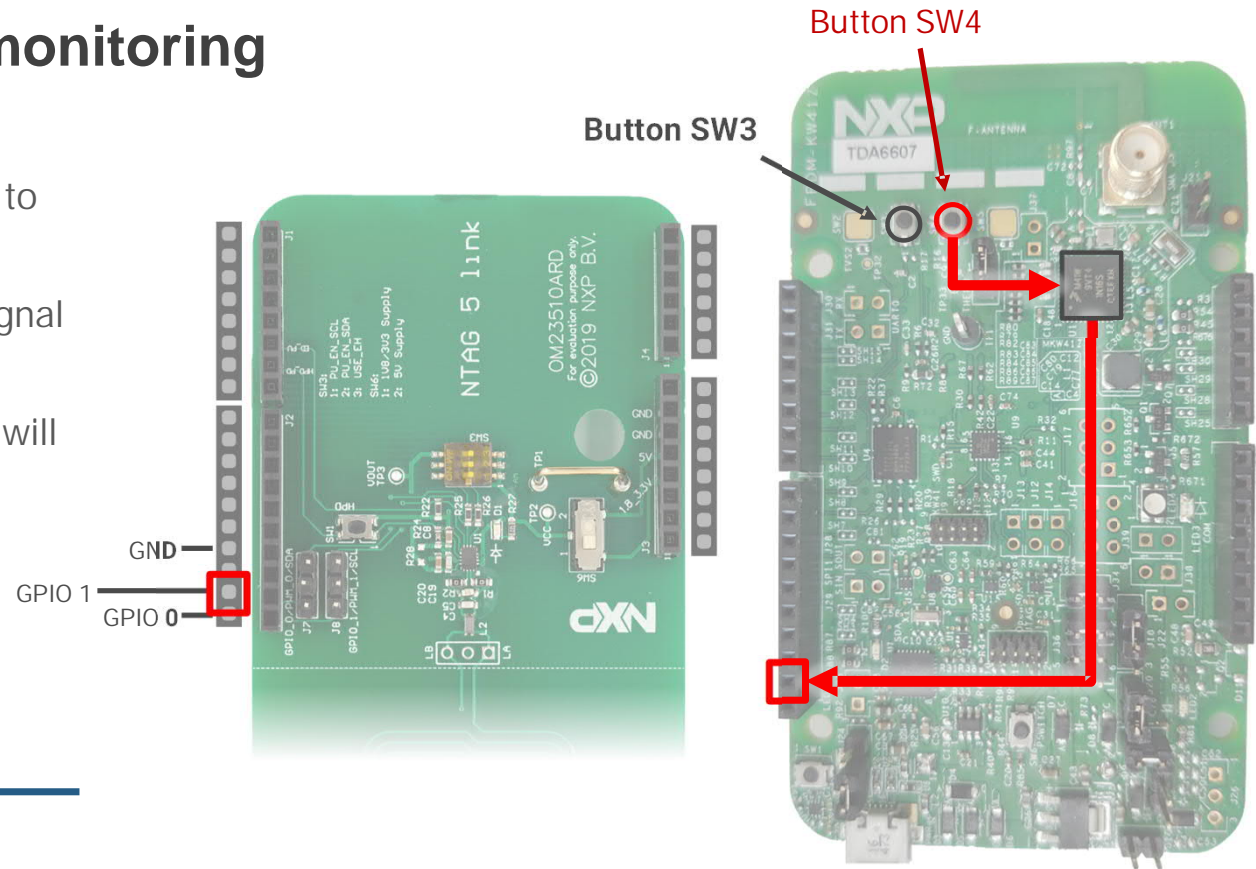
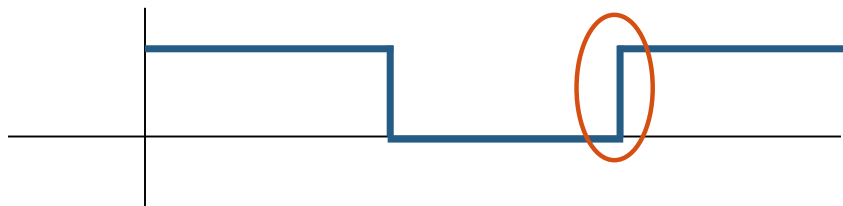


# Using GPIO features

## Toggle button example: Signal monitoring

1. Signal is generated by the KW41Z and rooted to the GPIO 1 pin of the NTAG 5 Eval board
2. If user clicks SW3 button, KW41Z turns the signal level to low state.
3. If user clicks SW4 button, the microcontroller will turn the signal level back to high

GPIO 1 (input)



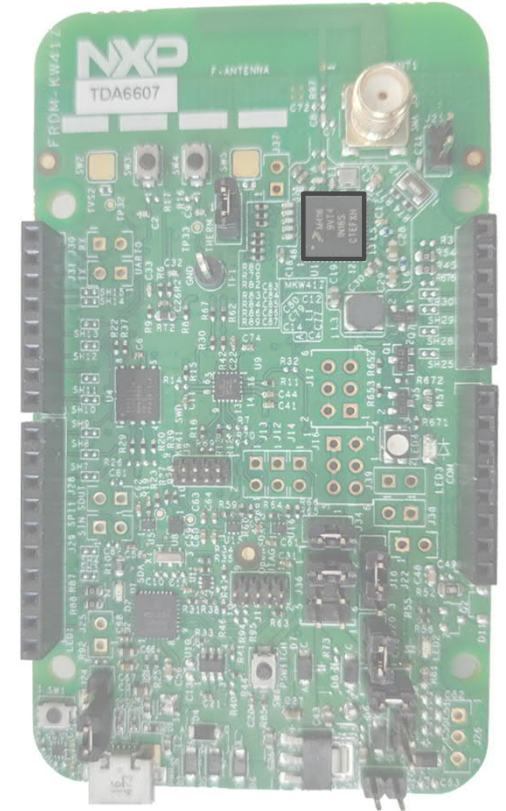
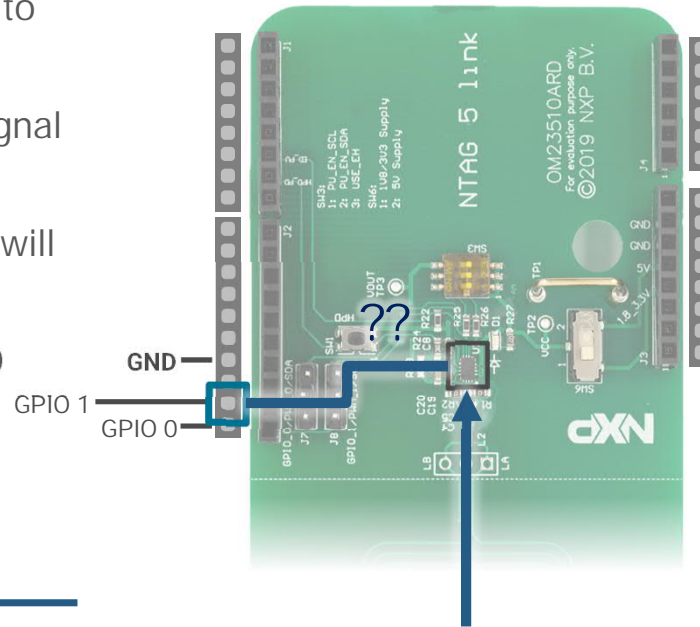
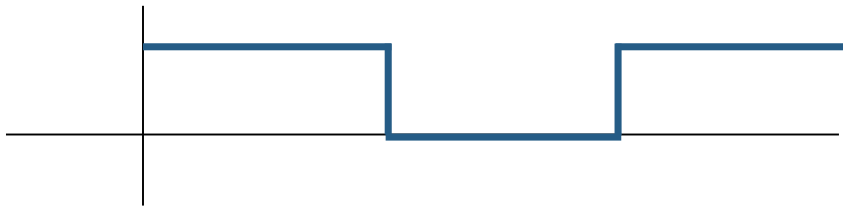


# Using GPIO features

## Toggle button example: Signal monitoring

1. Signal is generated by the KW41Z and rooted to the GPIO 1 pin of the NTAG 5 Eval board
2. If user clicks SW3 button, KW41Z turns the signal level to low state.
3. If user clicks SW4 button, the microcontroller will turn the signal level back to high
4. User can sense and monitor the state of GPIO input pad at any moment through NFC

GPIO 1 (input)



# Using PWM features



# Using PWM features

## Configuring wired interface

### Step 1

- WRITE\_CONFIG command (C1h) over Configuration Bytes block (37h):

```
@Override
public void onClick(DialogInterface dialog,
                    int which) {
    byte[] response = sendCommand(cmd_activateGPIOPWM);
    if (response != null)
        showConfirmationSnackbar();
}
```

```
public static final byte[] cmd_activateGPIOPWM = new byte[]{
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0x37,
    (byte) 0x00,
    (byte) 0x22,
    (byte) 0x0F,
    (byte) 0x00
};
```

Flags	WRITE_CONFIG	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	37h	00220F00	Auto

Non-address mode & High data rate

WRITE\_CONFIG command code

Config Bytes block address

GPIO/PWM mode



# Using PWM features

## Defining pads purposes and PWM parameters

Step 2

- Configure pads as PWM
- Configure pre-scale and resolution for PWM signals

  Session register address

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I <sup>2</sup> C				
39h	1039h	PWM_GPIO_CONFIG_0_REG	PWM_GPIO_CONFIG_1_REG	RFU	
A3h	10A3h				

Bit	Name	Value	Description
7	GPIO_SDA_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
6	GPIO_SCL_PAD_OUT_STATUS	0b	Output status on pad is LOW
		1b	Output status on pad is HIGH
5	GPIO_SDA_PAD_IN_STATUS	0b	Input status
		1b	Input status
4	GPIO_SCL_PAD_IN_STATUS	0b	Input status
		1b	Input status
3	GPIO_SDA_PAD	0b	Output
		1b	Input
2	GPIO_SCL_PAD	0b	Output
		1b	Input
1	GPIO_PWM1_SDA_PAD	0b	GPIO
		1b	PWM
0	GPIO_PWM0_SCL_PAD	0b	GPIO
		1b	PWM

Bit	Name	Value	Description
6-7	PWM1_PRESCALE	00b	Pre-scalar configuration for PWM1 channel (default 00h)
4-5	PWM0_PRESCALE	00b	Pre-scalar configuration for PWM0 channel (default 00h)
2-3	PWM1_RESOLUTION_CONF	00b	6-bit resolution (default)
		01b	8-bit resolution
		10b	10-bit resolution
		11b	12-bit resolution
0-1	PWM0_RESOLUTION_CONF	00b	6-bit resolution
		01b	8-bit resolution
		10b	10-bit resolution
		11b	12-bit resolution

For more information about PWM signal parameters and generation, please refer to application note AN11203



# Using PWM features

## Changing start time and duty cycle

### Step 3

- Configure time for rising and falling edge. This can be calculated out of the start time and duty cycle parameters:

PWMx\_ON:

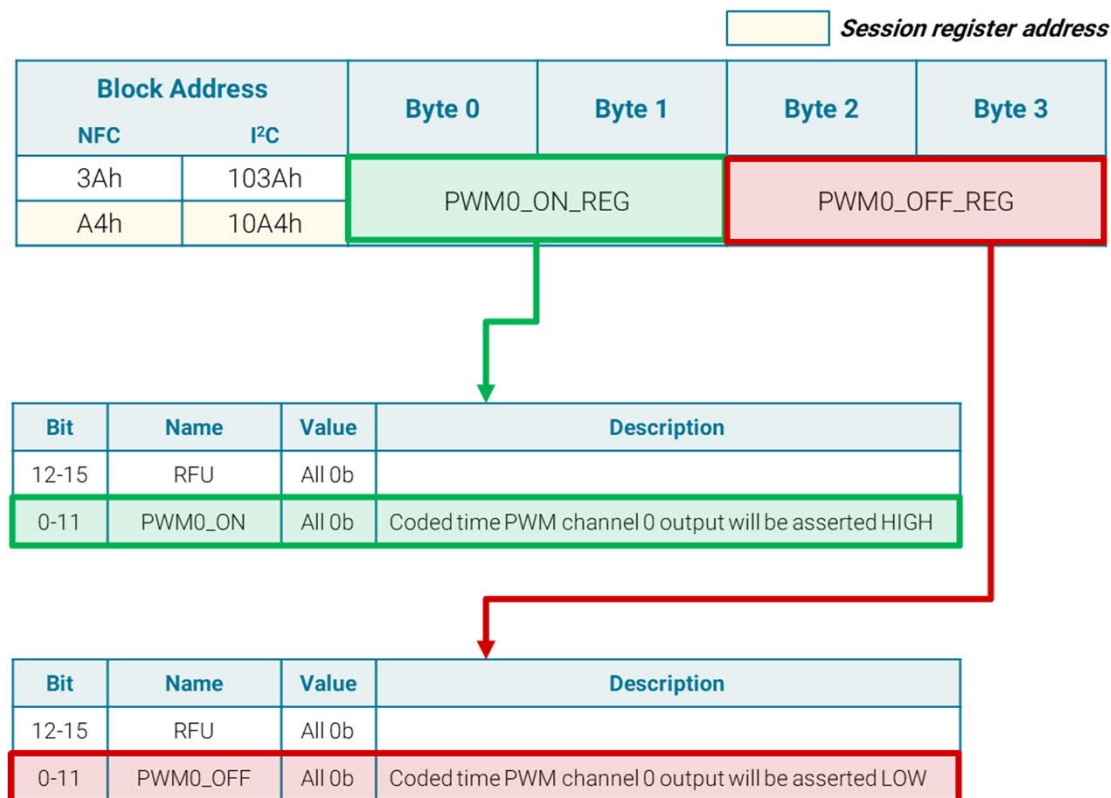
$$\text{Start time} = 2^{\text{Resolution}} \times \text{Start time}_{\%}$$

PWMx\_OFF:

$$\text{End time} = 2^{\text{Resolution}} \times (\text{Start time}_{\%} + \text{Duty cycle})$$

Equivalent register for Channel 1 is found in addresses:

- 3Bh (configuration settings)
- A5h (session register)



# LED intensity example

*using FRDM-KW41Z and NTAG 5 Demo app*



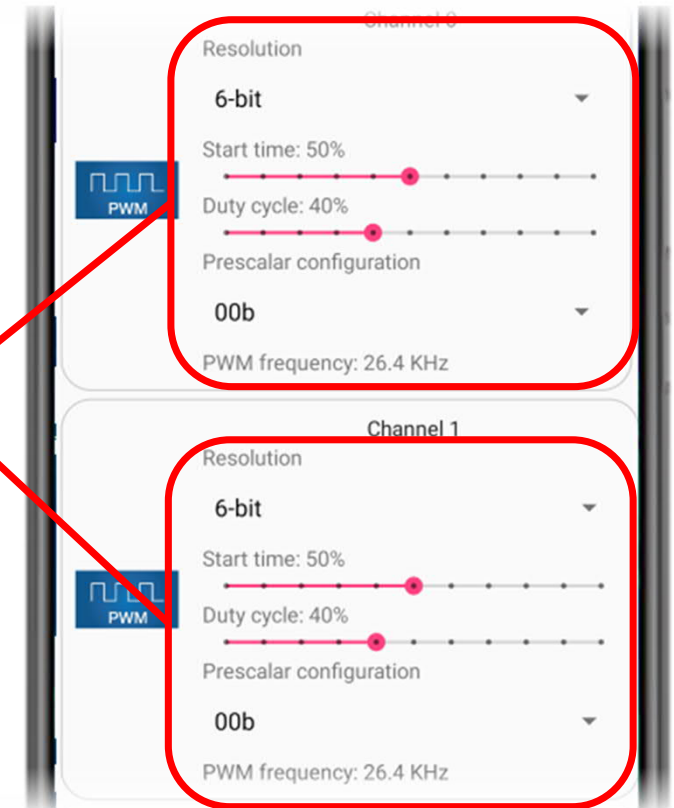
# Using PWM features

## LED intensity example

### Description

- Uses PWM signal to control intensity of two LEDs present in FRDM-KW41Z board using signal duty cycle
- Signal is generated using four different parameters
- Example available in NTAG 5 Demo app for mobiles
- KW41Z shall be flashed so microcontroller roots signal to LEDs input.
- OM23510ARD shall be connected to FRDM-KW41Z

Change duty cycle of signals generated for both channels





# Using PWM features

## LED intensity example

1. User changes PWM signal parameters
2. When user clicks on 'Write config' application gathers all information and sends commands to re-configure PWM signal:

Flags	WRITE_CONF	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	<b>A3h</b>	-----	Auto

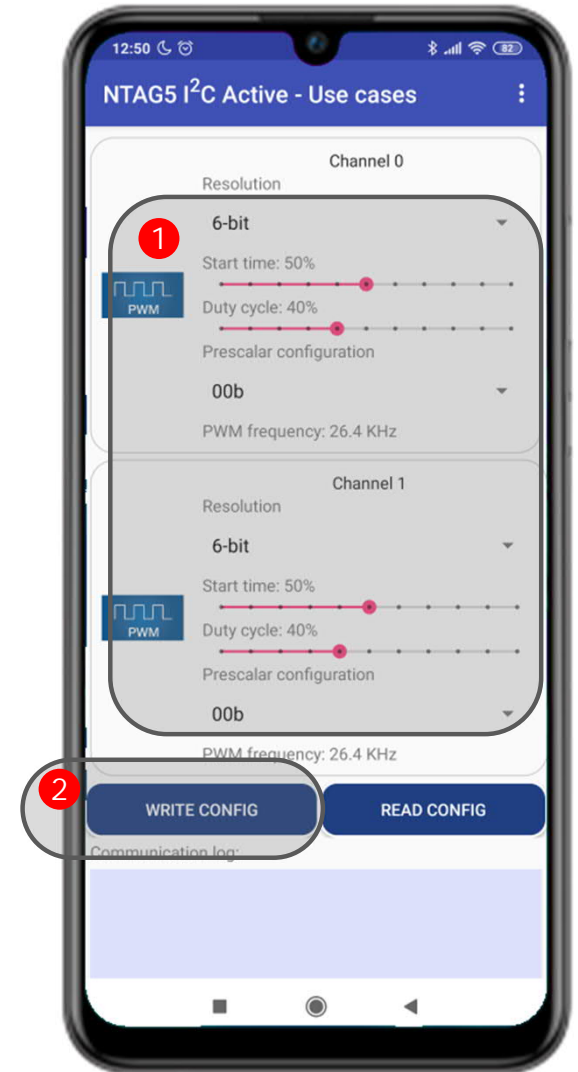
Flags	WRITE_CONF	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	<b>A4h</b>	-----	Auto

Flags	WRITE_CONF	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	<b>A5h</b>	-----	Auto

Configure pads as PWM  
Define pre-scale and resolution

Configure rising/falling edge  
for Channel 0 (LED 3)

Configure rising/falling edge  
for Channel 1 (LED 4)

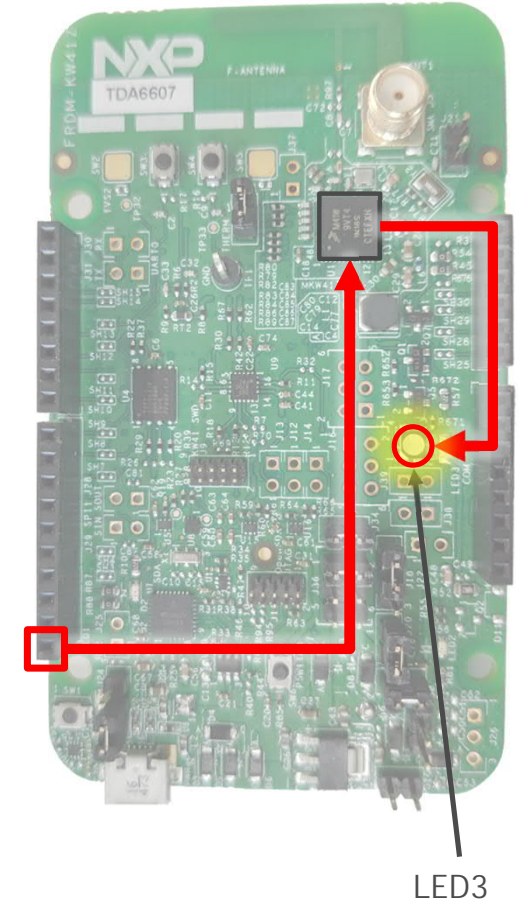
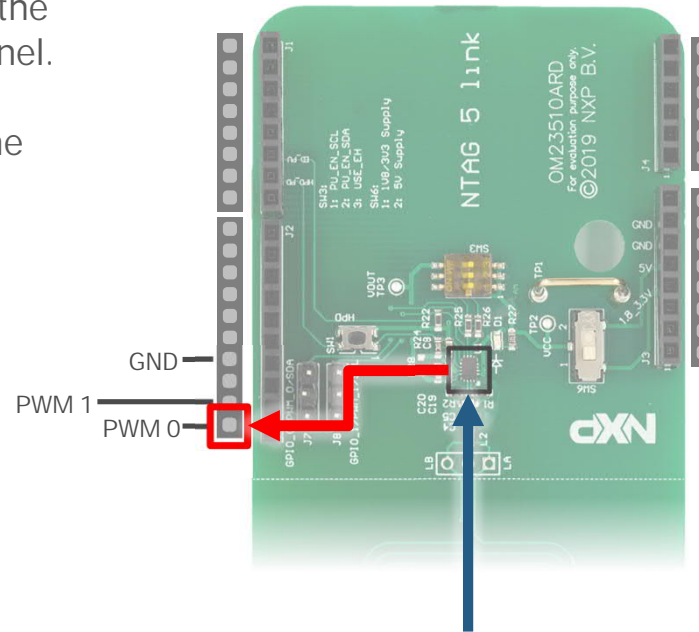
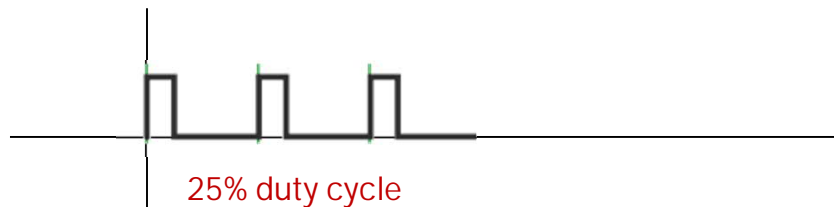


# Using PWM features

## LED intensity example: Signal generation

1. Signal is generated by NTAG 5 depending on the register dedicated to control each PWM channel. KW41Z monitors the signal generated by the NTAG 5 and dumps its value to the input of the respective LED.

PWM 0

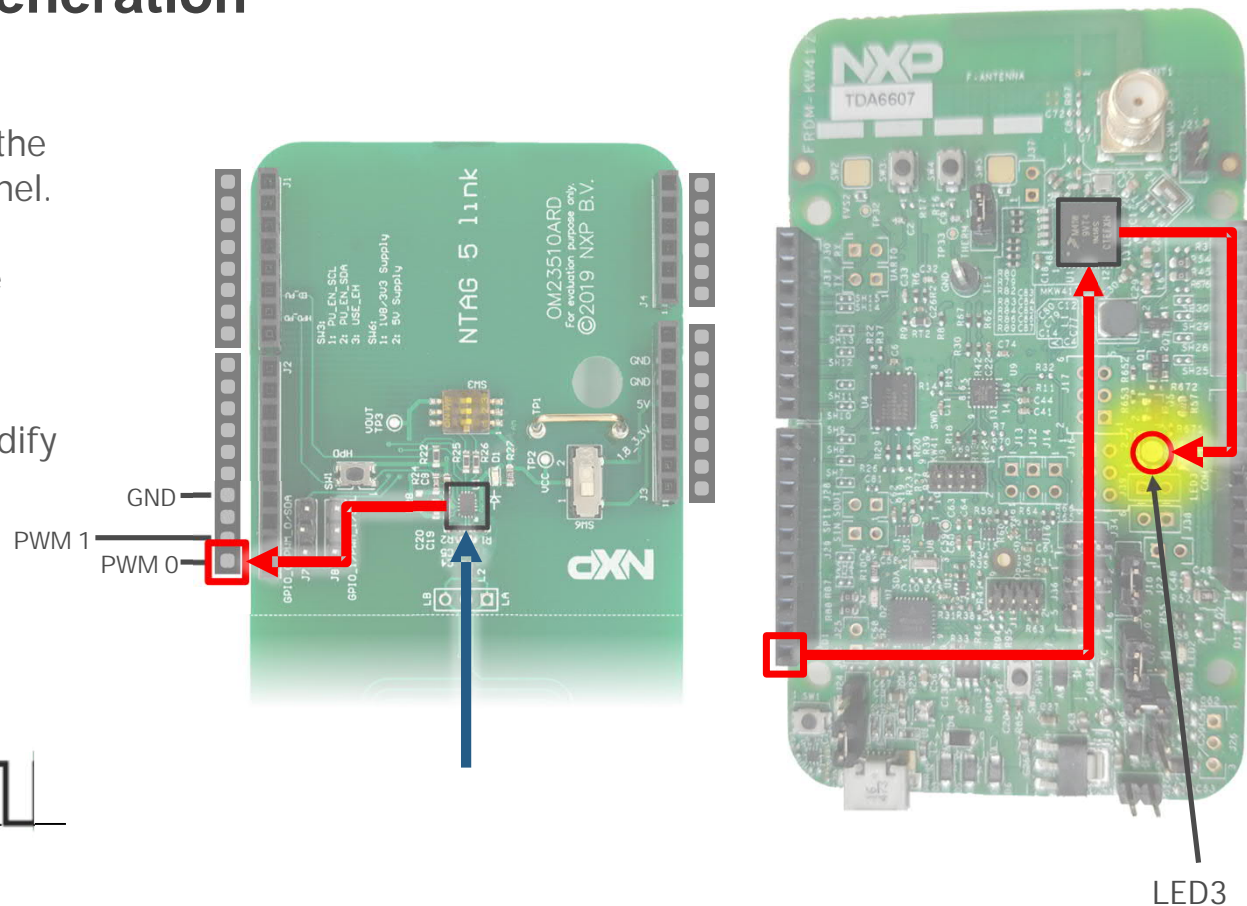
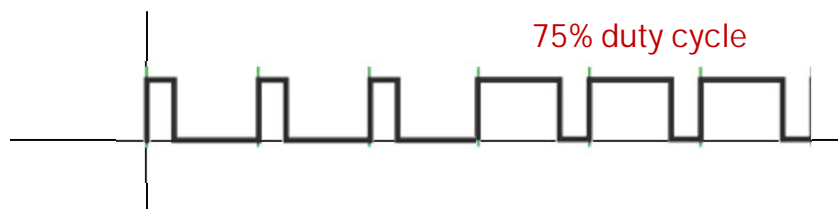


# Using PWM features

## LED intensity example: Signal generation

1. Signal is generated by NTAG 5 depending on the register dedicated to control each PWM channel. KW41Z monitors the signal generated by the NTAG 5 and dumps it value to the input of the respective LED.
2. User can change the intensity of the LED by writing to the related session register and modify the duty cycle of the generated PWM signal

PWM 0



# Using pass-through mode



# Using pass-through mode

## Introduction

- Pass through mode transfers data from RF to I2C interface and vice versa using the 256-byte SRAM saving EEPROM cycles. Available for NTAG 5 link and boost models.
- Data flow from one side to the other is synchronized using interruption signal and register settings.

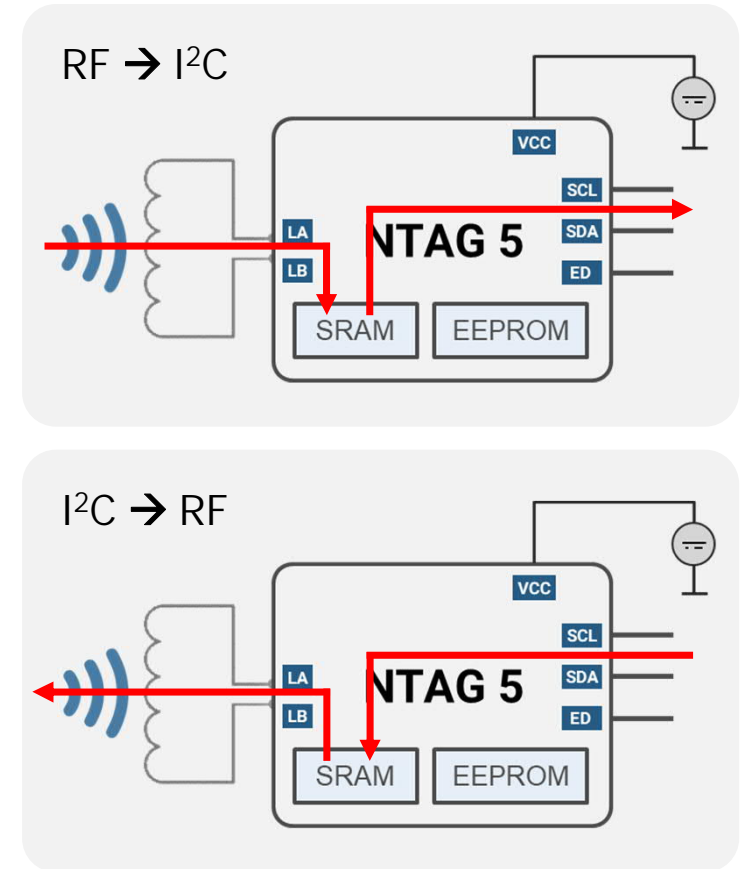
### Use cases:

#### RF → I2C data exchange:

- Mobile device writes data into the microcontroller
- Update microcontroller FW from NFC interface

#### I2C → RF data exchange:

- Download of data into mobile device (e.g., large amount of logging data, error descriptions...)



# Using pass-through mode

## Configuration (I)

WRITE\_CONFIG command (C1h) over Configuration Bytes block (37h):

```
@Override
public void onClick(DialogInterface dialog,
                    int which) {
    byte[] response = sendCommand(cmd_activateI2CSlave);
    if (response != null)
        showConfirmationSnackbar();
}
```

```
public static final byte[] cmd_activateI2CSlave = new byte[] {
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0x37,
    (byte) 0x00,
    (byte) 0x02,
    (byte) 0x0F,
    (byte) 0x00
};
```

Flags	WRITE_CONFIG	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	37h	00020F00	Auto

Config Bytes block address

WRITE\_CONFIG command code

I2C slave mode

Non-address mode & High data rate



# Using pass-through mode

## Configuration (II)

### Requirements

1. NFC\_FIELD\_OK = 1b → bit0 of STATUS0\_REG
2. VCC\_SUPPLY\_OK = 1b → bit1 of STATUS0\_REG
3. SRAM\_ENABLE = 1b → bit1 of CONFIG\_1

### Data flow direction

- PT\_TRANSFER\_DIR → bit2 of STATUS0\_REG
- ED pin to notify when last SRAM page was read/written

### Accessing SRAM

- RF perspective  
SRAM\_READ and SRAM\_WRITE over addresses 00h-3Fh
- I2C perspective  
READ / WRITE over addresses 2000h-203Fh

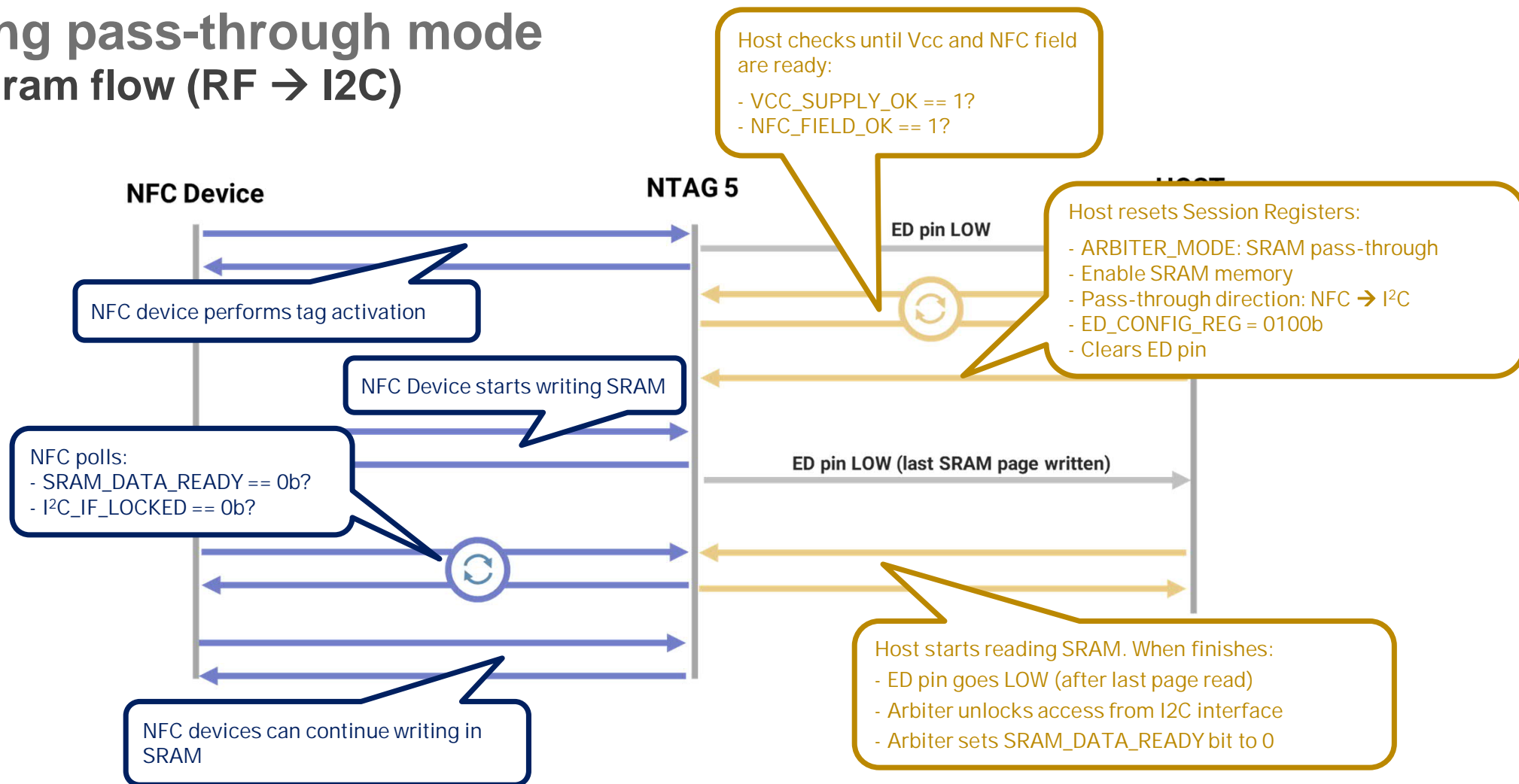
Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I2C				
A0h	10A0h	STATUS0_REG	STATUS1_REG	RFU	

Block Address		Byte 0	Byte 1	Byte 2	Byte 3
NFC	I2C				
37h	1037h	CONFIG_0	CONFIG_1	CONFIG_2	RFU



# Using pass-through mode

## Diagram flow (RF → I2C)





# Pass-through example

*using FRDM-KW41Z and NTAG 5 Demo app*

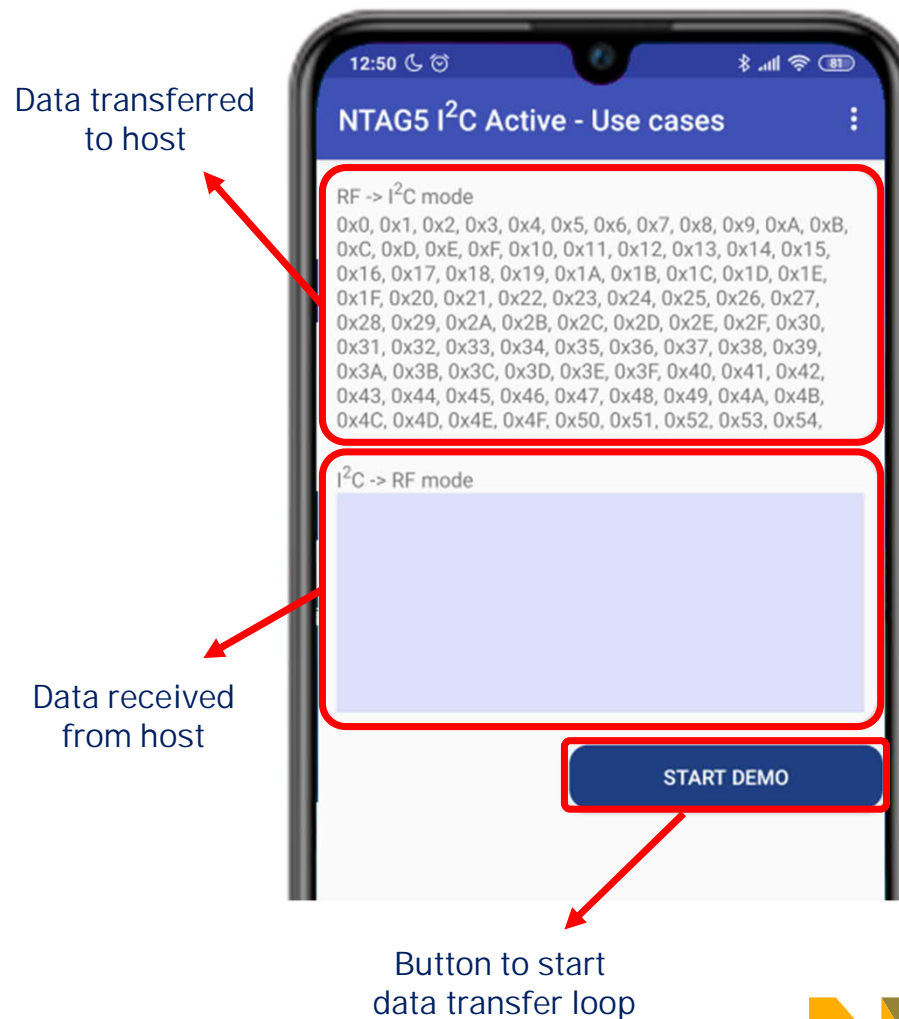


# Using pass-through mode

## Pass-through example

### Description

- Uses pass-through feature to exchange pre-defined data with KW41Z
- Wired interface must be configured in I<sup>2</sup>C slave mode to communicate with KW41Z using I<sup>2</sup>C interface.
- Example available in NTAG 5 Demo app for mobiles
- KW41Z shall be flashed so microcontroller roots signal to LEDs input.
- OM2351OARD shall be connected to FRDM-KW41Z



# Using pass-through mode

## Pass-through example

Block diagram

```
while(continueLoop){
```

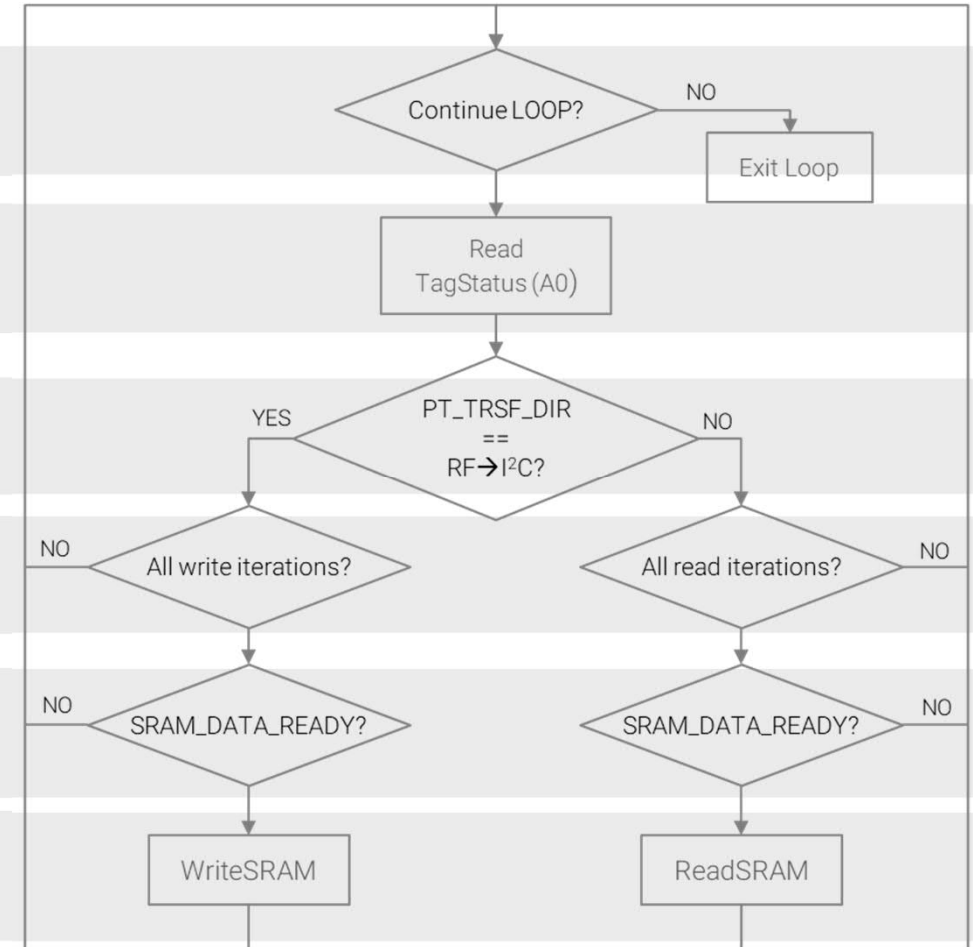
```
byte[] responseTagStatus = sendCommand(cmd_readTagStatus);
```

```
if (Parser.IsBitSet(responseTagStatus[1], 2)) {
```

```
if (writeCounter < SRAM_LOOP_SIZE) {
```

```
if (!Parser.IsBitSet(responseTagStatus[1], 5)) {
```

```
responseWritesSRAM = sendCommand(finalCommandWriteSRAM);
```

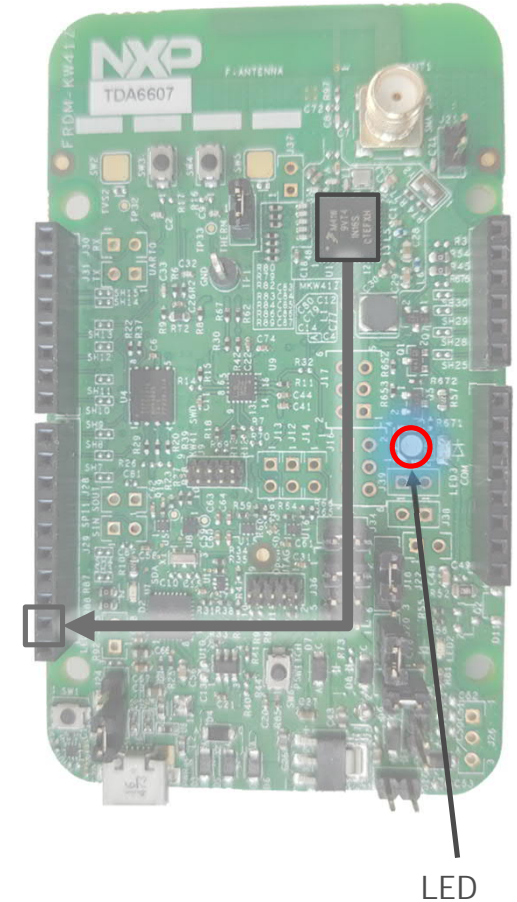
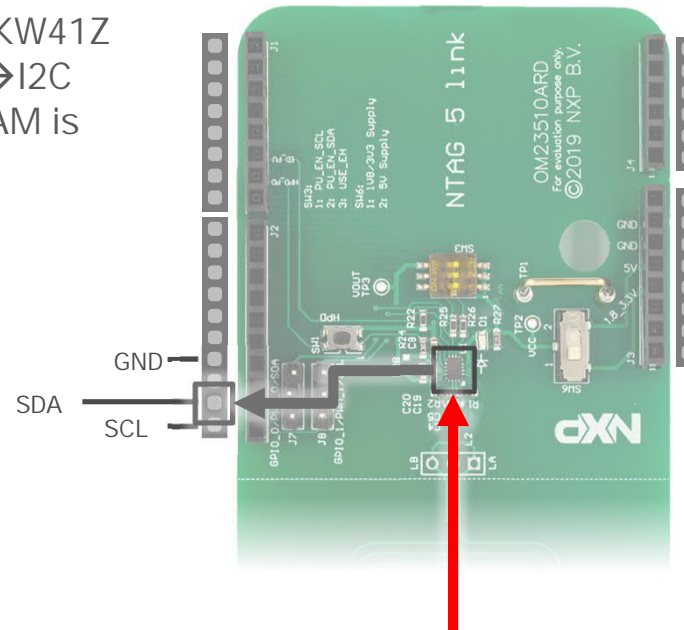
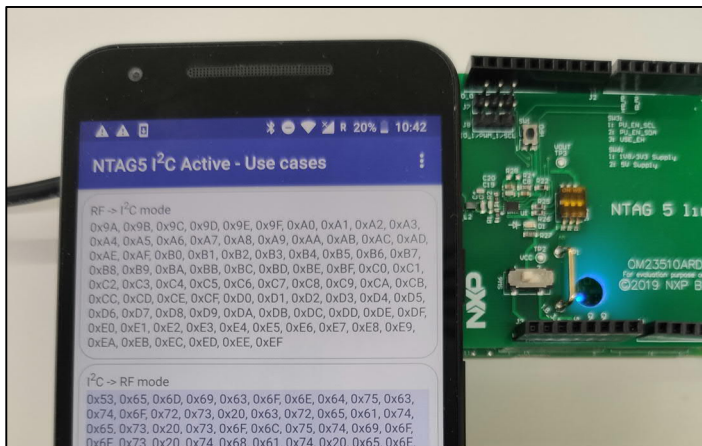


# Using pass-through mode

## Pass-through example

RF  $\rightarrow$  I<sup>2</sup>C

1. NFC device writes in NTAG 5 SRAM memory. KW41Z detects that PT\_TRANSFER\_DIR indicates RF $\rightarrow$ I<sup>2</sup>C direction, turns LED in blue and waits until SRAM is available to be read.



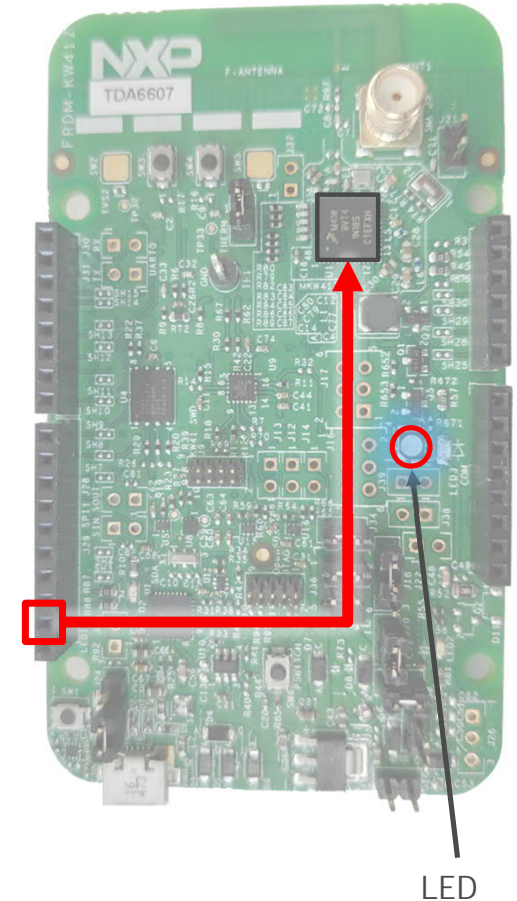
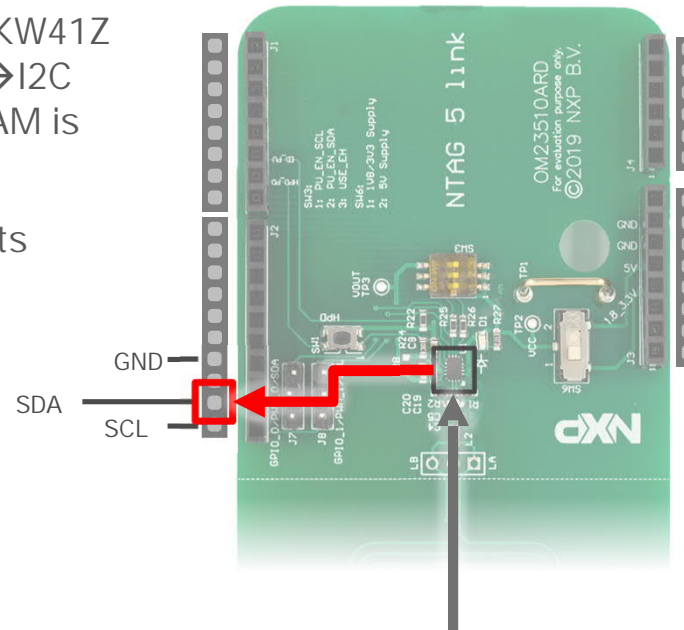
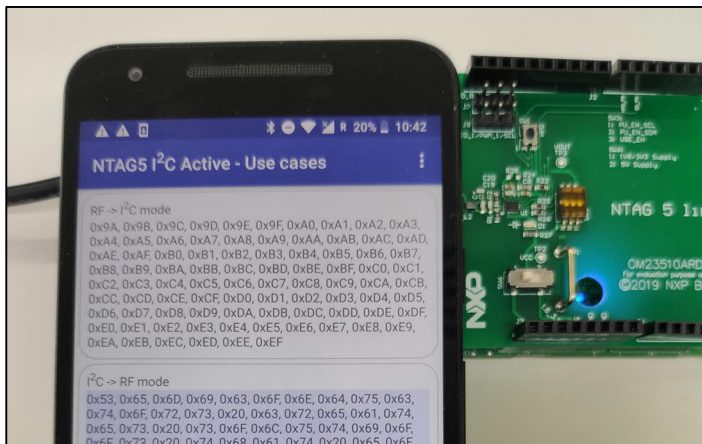
Evaluation board image  
is NOT the final one

# Using pass-through mode

## Pass-through example

RF  $\rightarrow$  I<sup>2</sup>C

1. NFC device writes in NTAG 5 SRAM memory. KW41Z detects that PT\_TRANSFER\_DIR indicates RF $\rightarrow$ I<sup>2</sup>C direction, turns LED in blue and waits until SRAM is available to be read.
2. When NFC device finishes writing KW41Z starts reading from SRAM. LED remains in blue until PT\_TRANSFER\_DIR changes to I<sup>2</sup>C $\rightarrow$ RF.



Evaluation board image  
is NOT the final one

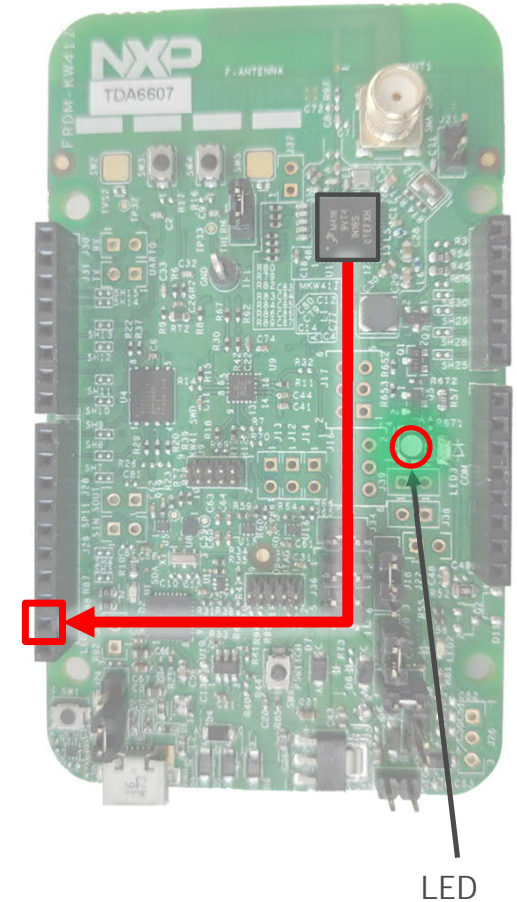
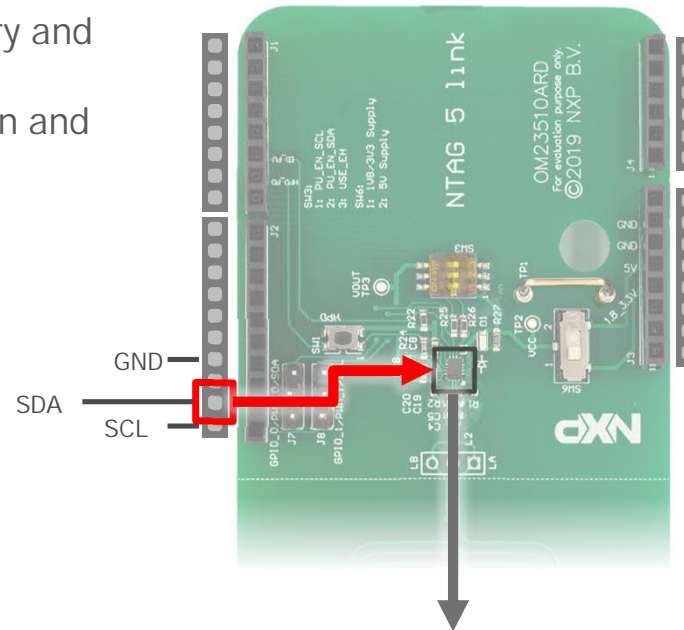


# Using pass-through mode

## Pass-through example

I<sup>2</sup>C → RF

1. KW41Z starts writing in NTAG 5 SRAM memory and turns LED in green. NFC device detects that PT\_TRANSFER\_DIR indicates I2C→RF direction and waits until SRAM is available to be read.



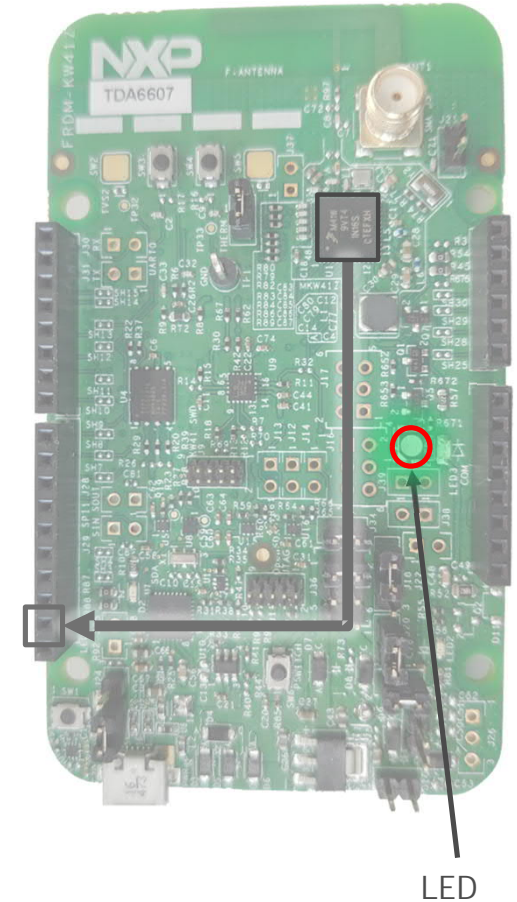
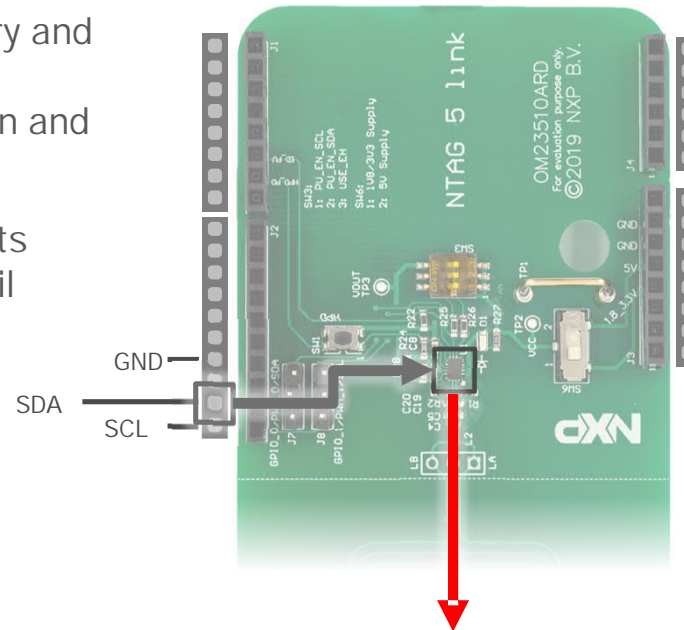
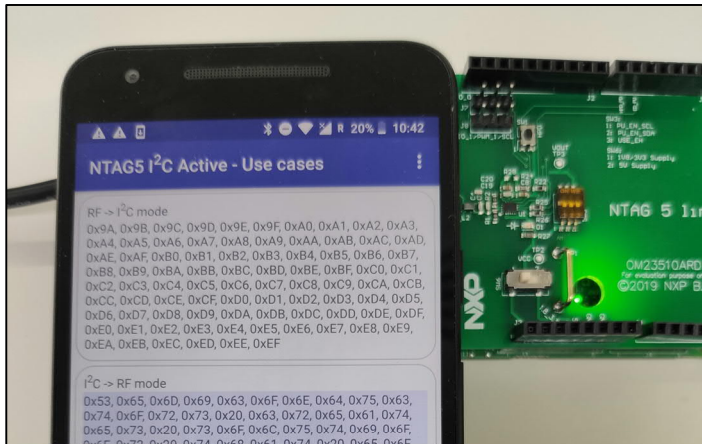
Evaluation board image  
is NOT the final one

# Using pass-through mode

## Pass-through example

I<sup>2</sup>C → RF

1. KW41Z starts writing in NTAG 5 SRAM memory and turns LED in green. NFC device detects that PT\_TRANSFER\_DIR indicates I2C→RF direction and waits until SRAM is available to be read.
2. When KW41Z finishes writing NFC device starts reading from SRAM. LED remains in green until PT\_TRANSFER\_DIR changes to I<sup>2</sup>C→RF.





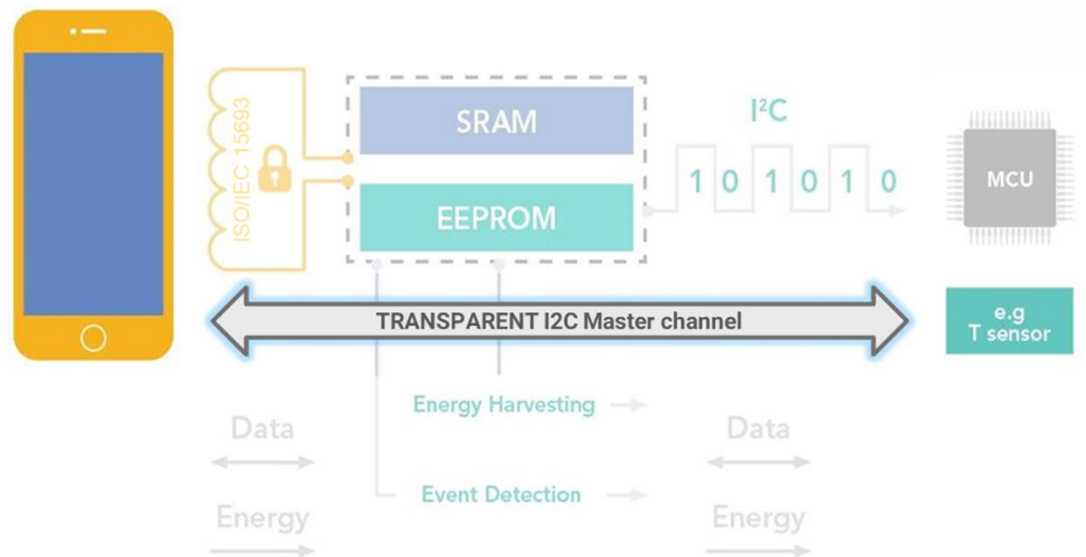
# Using I<sup>2</sup>C Master mode



# Using I<sup>2</sup>C Master mode

## Introduction

- I2C Master mode allows users to execute I2C commands directly from an NFC device by creating a transparent I2C channel with devices working as I2C slave.
- Working in I2C master mode, different I2C slave devices (e.g., sensors) can be connected without a microcontroller.
- Needed power for the sensors can be provided with NTAG 5 energy harvesting capabilities.



# Using I<sup>2</sup>C Master mode Configuration

WRITE\_CONFIG command (C1h) over Configuration Bytes block (37h):

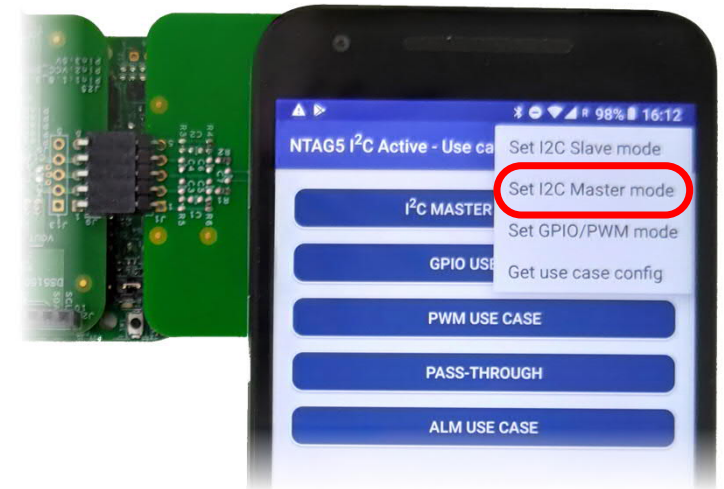
```
@Override
public void onClick(DialogInterface dialog,
                    int which) {
    byte[] response = sendCommand(cmd_activateI2CMaster);
    if (response != null)
        showConfirmationSnackbar();
}
```

```
public static final byte[] cmd_activateI2CMaster = new byte[] {
    (byte) 0x12,
    (byte) 0xC1,
    (byte) 0x04,
    (byte) 0x37,
    (byte) 0x00,
    (byte) 0x12,
    (byte) 0x0F,
    (byte) 0x00
};
```

Flags	WRITE_CONFIG	Manuf. Code	UID (optional)	Block Address	Data	CRC16
12h	C1h	04h	----	37h	00120F00	Auto

Config Bytes block address  
WRITE\_CONFIG command code

I<sup>2</sup>C Master mode



# Using I<sup>2</sup>C Master mode

## Sending Read/Write I2C commands

Writing to I<sup>2</sup>C interface

- Write\_I2C command (D4h) → Writes command into I2C line. Includes I2C address of the target slave

Flags	WRITE I <sup>2</sup> C	Manuf. Code	UID (optional)	I <sup>2</sup> C Param	Data length N	Data	CRC16
8 bits	D4	8 bits	64 bits	8 bits	8 bits	(N+1) x 8 bits	16 bits

Reading from I<sup>2</sup>C interface

- Read\_I2C command (D5h) → Reads data from I2C line and transfer it into NTAG 5 SRAM

Flags	READ I <sup>2</sup> C	Manuf. Code	UID (optional)	I <sup>2</sup> C Param	Data Length N	CRC16
8 bits	D5	8 bits	64 bits	8 bits	8 bits	16 bits

Reading from SRAM memory

- Read SRAM command (D2h) → Reads data from NTAG 5 SRAM memory

Flags	Read SRAM	Manuf. Code	UID (optional)	Block Address	Number of blocks	CRC16
8 bits	D2	8 bits	64 bits	8 bits	8 bits	16 bits

# I<sup>2</sup>C Master mode example

*using FRDM-KW41Z and NTAG 5 Demo app*

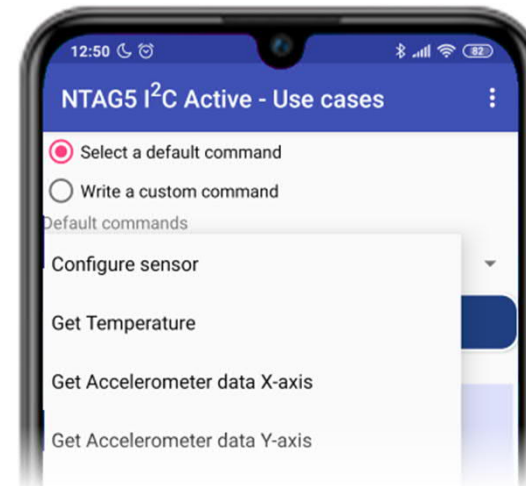
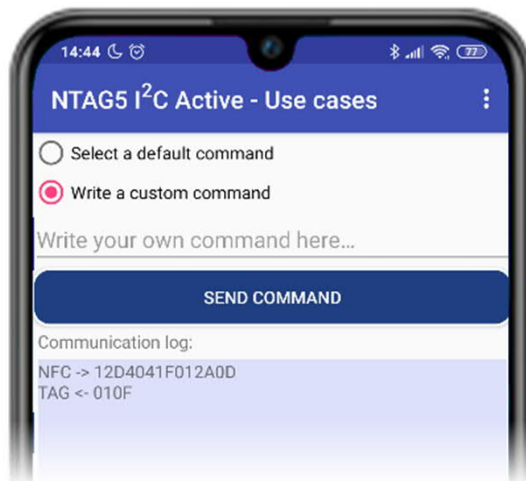


# Using I<sup>2</sup>C Master mode

## I<sup>2</sup>C Master mode example

### Description

- Allows user to directly send I2C commands to FXOS8700CQ accelerometer and magnetometer sensor present in FRDM-KW41Z board.
- User can send a default command from a list or introduce a customized command



# Using I<sup>2</sup>C Master mode

## I<sup>2</sup>C Master mode example

Example: Get temperature from sensor

- Step 1: Sending command to Read temperature:

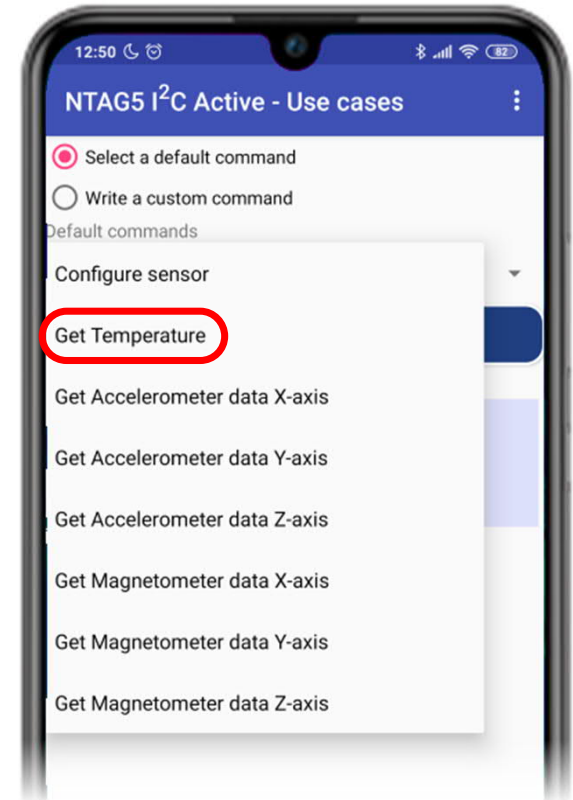
```
byte[] response = sendCommand(cmd_GetTempI2CMasterCommand);  
writeReceiveLog(response);  
readData();
```

```
public static final byte[] cmd_GetTempI2CMasterCommand = new byte[]{  
    (byte) 0x12, // FLAGS, CHECK ISO/IEC 15693  
    (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 composensor  
};
```

- Step 2: Reading response from I<sup>2</sup>C line and put it in SRAM:

```
private void readData(){  
    try {  
        byte[] response2 = sendCommand(cmd_readI2CMasterCommand);  
        writeReceiveLog(response2);
```

```
public static final byte[] cmd_readI2CMasterCommand = new byte[]{  
    (byte) 0x12, // FLAGS, CHECK ISO/IEC 15693  
    (byte) 0xD5, // READ I2C  
    (byte) 0x04,  
    (byte) 0x1F, //Don't generate Stop condition  
                //I2C address = 0x1F  
  
    (byte) 0x00  
};
```





# Using I<sup>2</sup>C Master mode

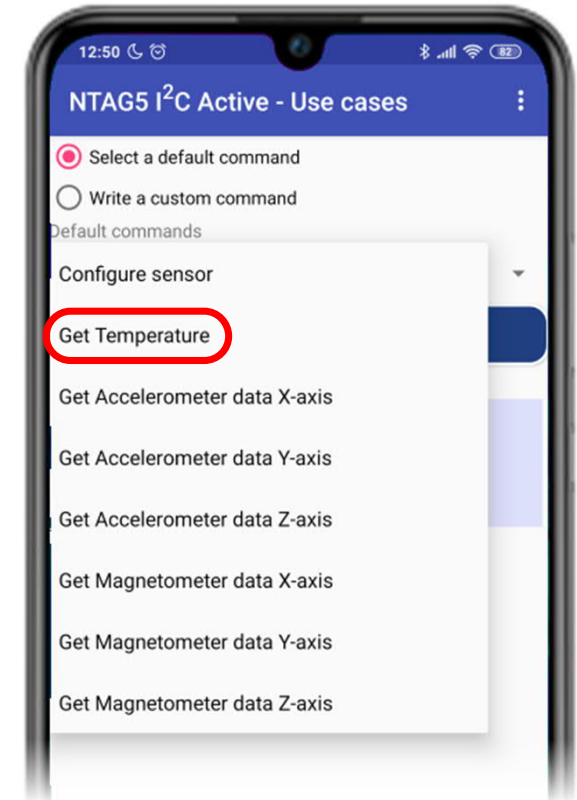
## I<sup>2</sup>C Master mode example

Example: Get temperature from sensor

- Step 3: Reading content from NTAG 5 SRAM:

```
private void readData(){  
    try {  
        byte[] response2 = sendCommand(cmd_readI2CMasterCommand);  
        writeReceiveLog(response2);  
  
        byte[] responseSRAM = sendCommand(cmd_readSRAMI2CMaster);  
        writeReceiveLog(responseSRAM);  
    }  
}
```

```
public static final byte[] cmd_readSRAMI2CMaster = new byte[]{  
    (byte) 0x12,  
    (byte) 0xD2,  
    (byte) 0x04,  
    (byte) 0x00,  
    (byte) 0x06  
};
```



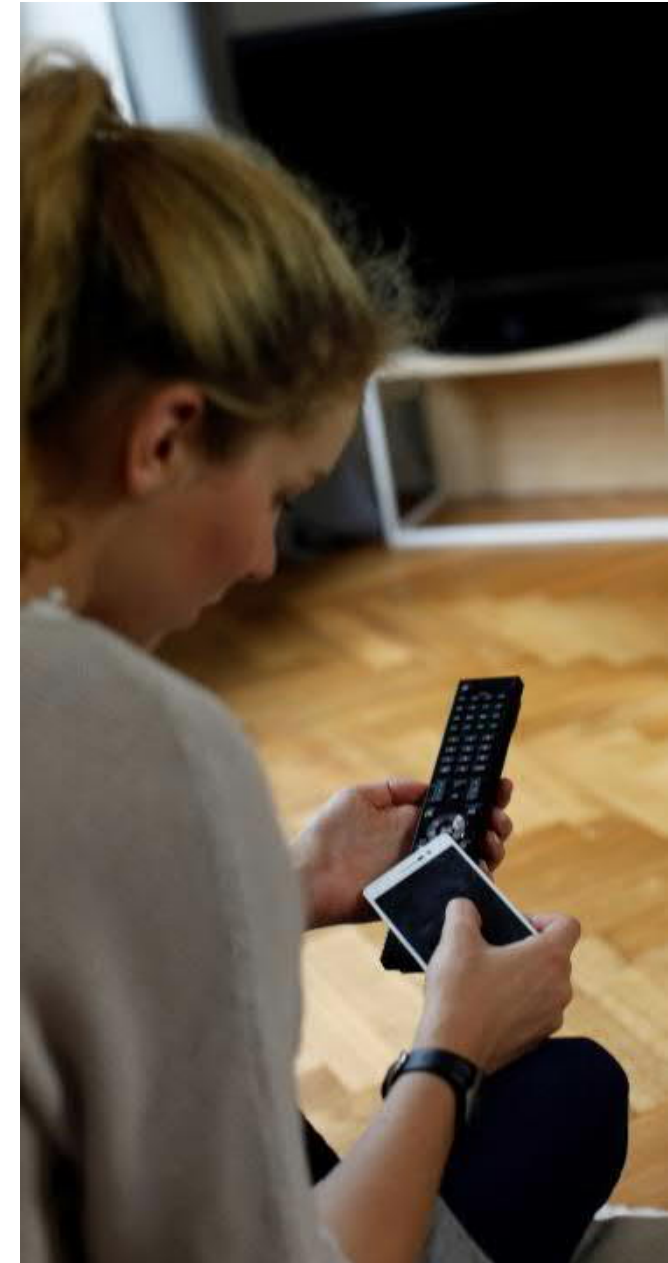
**More support**

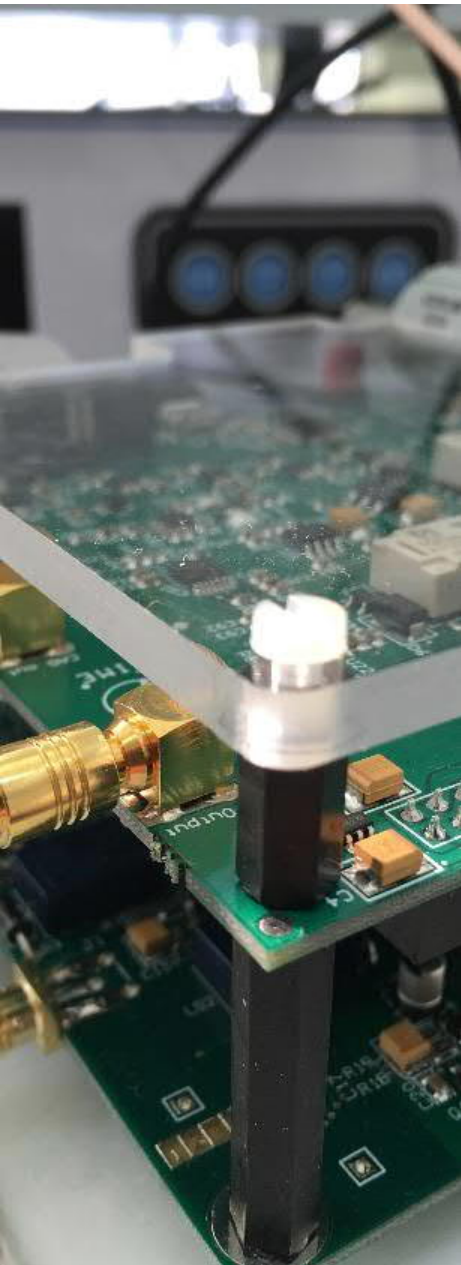


# More support

## Relevant resources regarding NTAG 5 family

- NTAG 5 switch website  
<https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag/nfc-tags-for-electronics/ntag-5-switch-nfc-forum-compliant-pwm-gpio-bridge-for-lighting-and-gaming:NTAG5-SWITCH>
- NTAG 5 link website  
<https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag/nfc-tags-for-electronics/ntag-5-link-nfc-forum-compliant-ic-bridge-for-iot-on-demand:NTAG5-LINK>
- NTAG 5 boost website  
<https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag/nfc-tags-for-electronics/ntag-5-boost-nfc-forum-compliant-ic-bridge-for-tiny-devices:NTAG5-BOOST>
- NTAG 5 development kit  
<http://www.nxp.com/products/rfid-nfc/nfc-hf/ntag/ntag-5-development-kit:OM23510ARD>
- NXP Tech community  
<https://www.nxp.com/support/support:SUPPORTHOME>





## MobileKnowledge

MobileKnowledge is a team of HW, SW and system engineers, experts in **smart, connected and secure** technologies for the IoT world. We are your ideal **engineering consultant** for any specific support in connection with your **IoT** and **NFC** developments. We design and develop secure HW systems, embedded FW, mobile phone and secure cloud applications.

Our services include:

- **Secure hardware design**
- **Embedded software development**
- **NFC antenna design and evaluation**
- **NFC Wearable**
- **EMV L1 pre-certification support**
- **Mobile and cloud application development**
- **Secure e2e system design**

[www.themobileknowledge.com](http://www.themobileknowledge.com)

[mk@themobileknowledge.com](mailto:mk@themobileknowledge.com)



We help companies leverage  
the secure IoT revolution





# NTAG 5 Webinar series – Product Support Package

Pablo Fuentes (Speaker)

Angela Gemio (Host)

Time for  
**Q & A**







## NTAG 5 Webinar series – Product Support Package

**Thank you for your kind attention!**

Please remember to fill out our **evaluation survey** (pop-up)

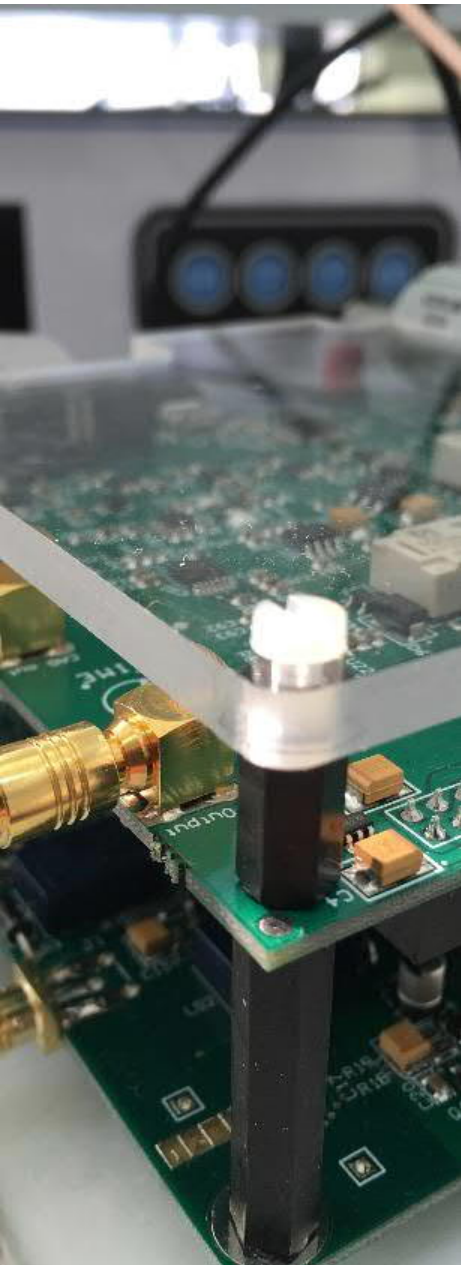
Check your email for **material download** and on-demand **video** addresses

Please check NXP and MobileKnowledge websites for **upcoming webinars** and **training sessions**

<http://www.nxp.com/support/classroom-training-events:CLASSROOM-TRAINING-EVENTS>

[www.themobileknowledge.com/content/knowledge-catalog-0](http://www.themobileknowledge.com/content/knowledge-catalog-0)





## MobileKnowledge

MobileKnowledge is a team of HW, SW and system engineers, experts in **smart, connected and secure** technologies for the IoT world. We are your ideal **engineering consultant** for any specific support in connection with your **IoT** and **NFC** developments. We design and develop secure HW systems, embedded FW, mobile phone and secure cloud applications.

Our services include:

- **Secure hardware design**
- **Embedded software development**
- **NFC antenna design and evaluation**
- **NFC Wearable**
- **EMV L1 pre-certification support**
- **Mobile and cloud application development**
- **Secure e2e system design**

[www.themobileknowledge.com](http://www.themobileknowledge.com)

[mk@themobileknowledge.com](mailto:mk@themobileknowledge.com)



We help companies leverage  
the secure IoT revolution

