# AN12872

# Anti-relay Attack Using GFSK

Rev. 1 — 09/2020 Application Note

Contents

## 1 Introduction

This document introduces the anti-relay attack using GFSK, focusing on the multi-link monitoring concepts and applications. It proposes a system implementation using KW36 or KW38 Wireless MCU.

Kinetis MKW36 or KW38 is a wireless MCU that supports Bluetooth LE v5.0 protocol and Generic FSK (GFSK) modulation.

The readers of this document are expected to have a basic knowledge of Arm® MCU architecture and radio communication.

## 2 Use case of anti-relay attack using GFSK

The RSSI-based localization is a simple and effective localization solution. There are two main methods of RSSI-based localization with Bluetooth LE, connectionless method and connection method. For detailed knowledge and implementation about RSSI-based localization, see *KW36 Localization Based on RSSI Ranging Application* (document AN12865) and *KW38 Localization Based on RSSI Ranging Application* ( document AN12977).For the convenience, let's take the car access system as an example, which is a typical system requiring distance estimation.

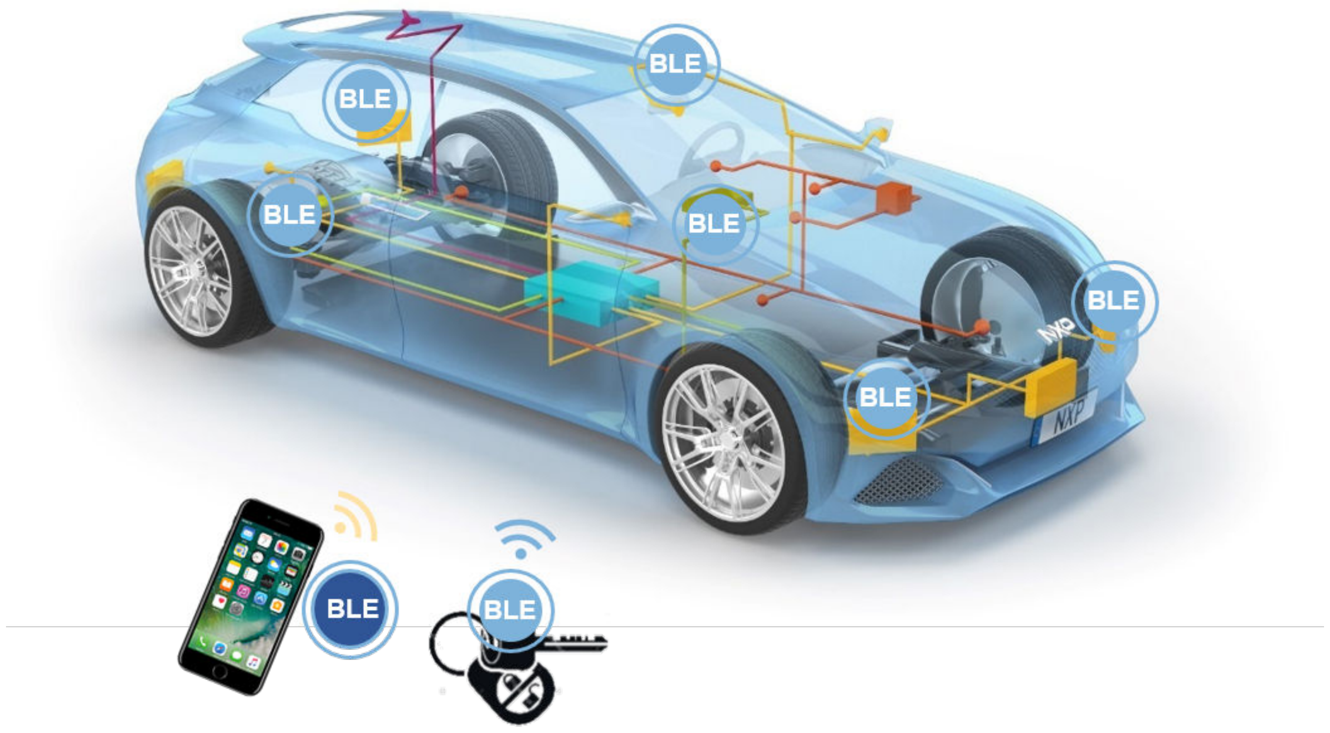


Figure 1. Connectionless method or connection method

The connectionless method is realized through the three advertising channels and there is no connection between the devices. This method is simple to implement and easy to deploy, but cannot prevent replay attacks.

The connection method requires Bluetooth LE connection between devices. If the pairing and bonding are enabled, this method can achieve anti-relay attacks. However, when there are multiple connections, there are problems such as increased power consumption, reduced response speed, and difficulty in deployment.

In order to avoid the problems of the above two methods, we propose to use GFSK to prevent relay attacks. This implementation is valid for a car access system in which there is more than one BLE/GFSK node. In this way, the connection guarantees security and the monitoring avoids the existence of multi-device connections to reduce system complexity.

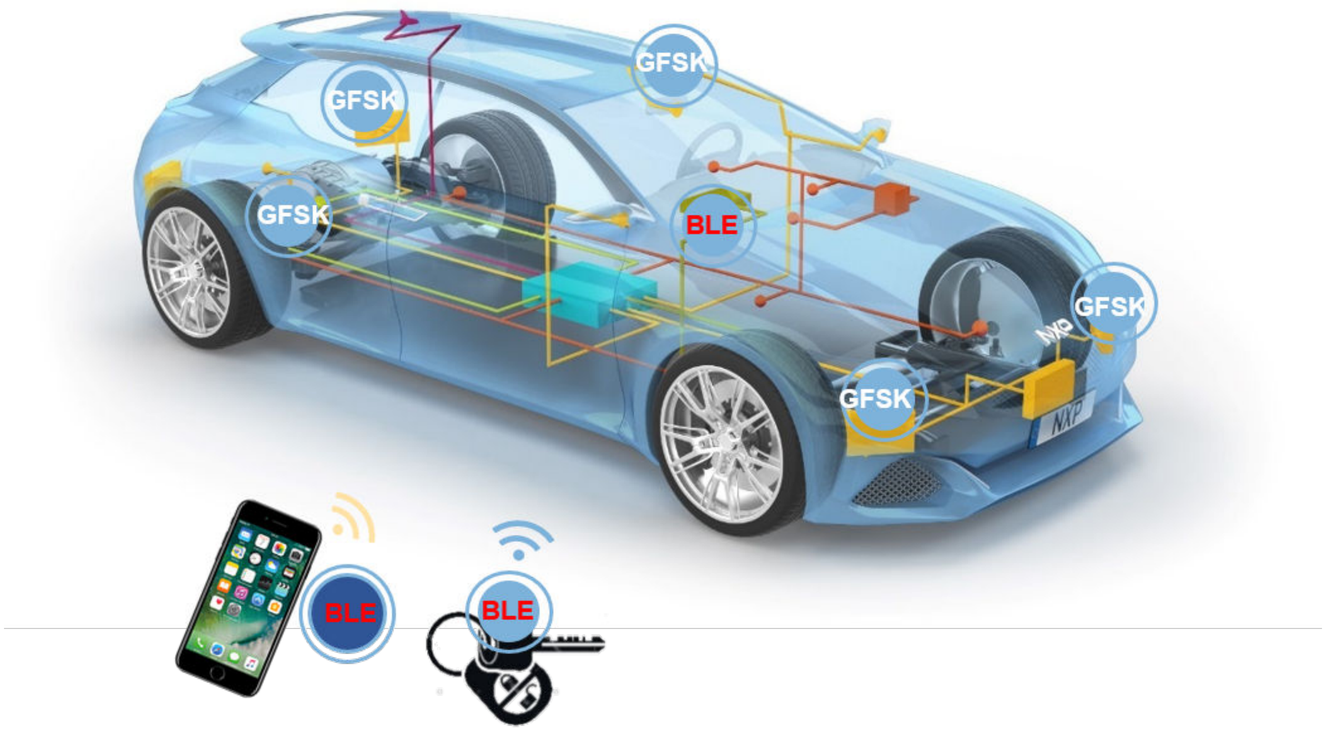


Figure 2. Anti-relay attack using GFSK

## 3 Anti-relay attack using GFSK implementation

The GFSK protocol enables radio operation using a custom GFSK/GMSK or MSK modulation format achieved by programming a set of PHY variables, such as, BT product, modulation index, and modulation filter co-efficients. The GFSK can receive Bluetooth LE packets with proper parameter configurations.

With the use of all channels and the following connection information obtained, we can track the link data of Bluetooth LE according to the channel selection algorithm.

- Access address
- Connection interval
- Channel hop increment
- CRC seed

Figure 3 is a block diagram of single Bluetooth LE connection. It contains three roles, master, slave and monitor. There is a Bluetooth LE connection between master and slave, and CAN bus or LIN bus communication between master and monitor. The Bluetooth LE master connects to the slave and transmits the connection information to the GFSK monitor through the CAN or LIN bus. Then, the GFSK monitor starts to sniff the air packages through these information.

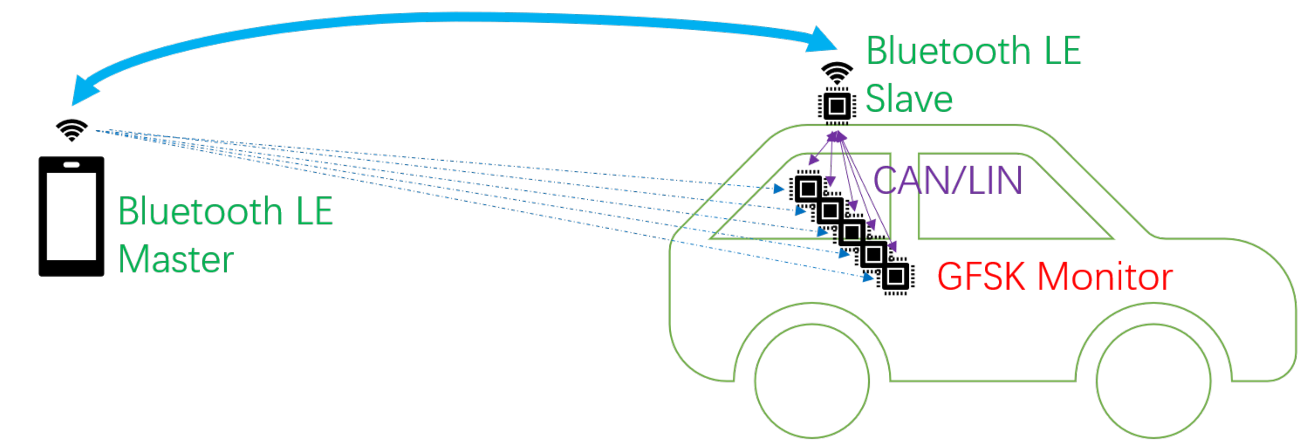


Figure 3. Monitoring one connection

## 3.1 GFSK parameter configuration

Modify the following macro configuration to adapt to the Bluetooth LE parameter requirements.

- `#define gGenFskDefaultLengthFieldSize_c (8)/*!< Number of bits in the LENGTH field. */`
- `#define gGenFskDefaultH1FieldSize_c (0)/*!< Number of bits in the H1 field. */`
- `#define gGenFskDefaultH0Value_c (0x0000)/*!< H0 field value. */`
- `#define gGenFskDefaultH0Mask_c 0/*!< Mask to select which bits of H0 must match the h0_match field. */`
- `#define gGenFskDefaultH1Value_c (0x0000)/*!< H1 field value. */`
- `#define gGenFskDefaultH1Mask_c 0/*!< Mask to select which bits of H1 must match the h1_match field. */`

## 3.2 Connection information

The `CONNECT_REQ` PDU contains the connection information we need above.

Table 1. LLData field structure in CONNECT_REQ PDU's payload

| LLData | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| AA (4 octets) | CRCInit (3 octets) | WinSize (1 octet) | WinOffset (2 octets) | Interval (2 octets) | Latency (2 octets) | Timeout (2 octets) | ChM (5 octets) | Hop (5 bits) | SCA (3 bits) |

Table 1 is the information obtained from the Air Interface Packets. This information exists in the Data Channel Registers in KW3x. The master obtains this information and then transmits to the monitor via CAN or LIN.

| Addr 16-bit | Addr 32-bit | Register name | Description |
|---|---|---|---|
| 0x80 | 0x100 | CONN_INTERVAL | — |
| 0x88 | 0x110 | PDU_ACCESS_ADDR_L_REGISTER | — |
| 0x8A | 0x114 | PDU_ACCESS_ADDR_H_REGISTER | — |
| 0xF4 | 0x1E8 | CONN_PARAM1 | Connection parameters, such as, `sca`, `hop_increment`, and `crc_init`, exchanged in the `connect_request` of this connection. |
| 0xF6 | 0x1EC | CONN_PARAM2 | Connection parameter, `crc_init` bits `24:7`, exchanged in the `connect_request` of this connection. |

## 3.3 Channel selection algorithm

Bluetooth LE uses a frequency hopping transceiver, so to track the air interface package of Bluetooth LE, the monitoring device should also implement the same frequency hopping algorithm.

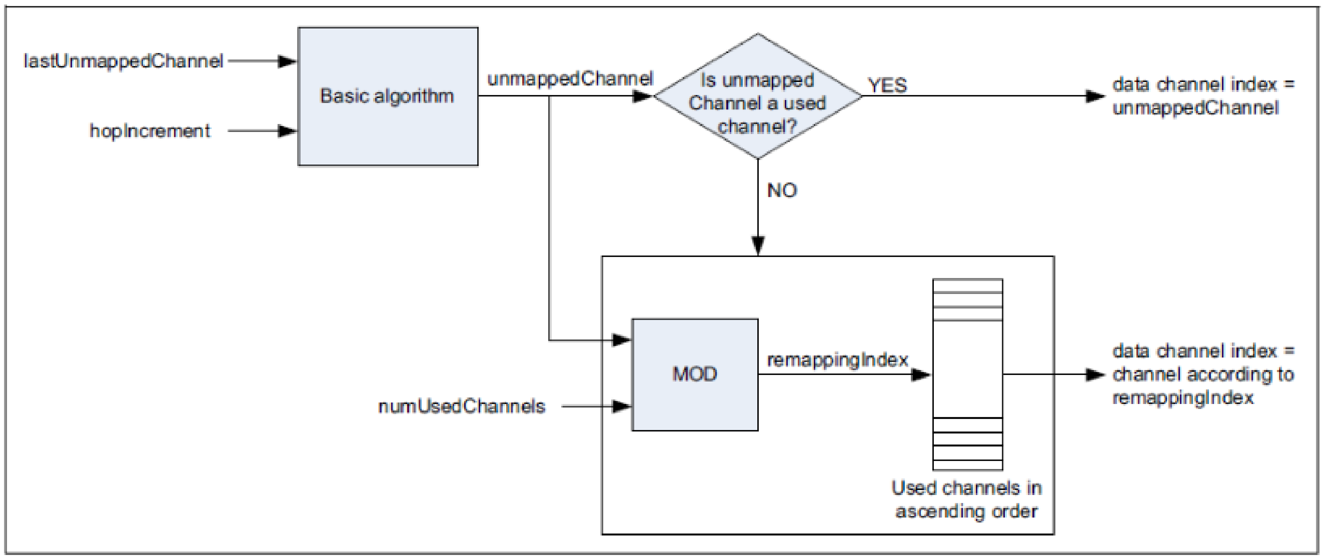

lastUnmappedChannel
hopIncrement
Basic algorithm
unmappedChannel
Is unmapped Channel a used channel?
YES
data channel index = unmappedChannel
NO
MOD
numUsedChannels
remappingIndex
Used channels in ascending order
data channel index = channel according to remappingIndex

Figure 4. Block diagram of data channel selection algorithm

## 3.4 Interrupt handling

As each connection event contains the master and slave packets, the GFSK interrupt handler needs to be modified to retrieve these two sets of data. After receiving the master data packet, the timeout timer is no longer cleared, and the Rx is restarted. After the slave data is received, the timeout timer is cleared and the Rx completion event is set.

```
#if LINK_MONITOR

#else
            GENFSK_TimeCancelEvent(&rxTimeoutTimer);
            GENFSK->T1_CMP &= ~GENFSK_T1_CMP_T1_CMP_EN_MASK;
```

```
#endif
            if (genfskLocal[mGenfskActiveInstance].packetReceivedCallbackIsr != NULL)
            {
                /* Examine RX buffer in ISR context if CallbackIsr is set */
                GENFSK_RxIsrContext();
            }

            if (genfskLocal[mGenfskActiveInstance].enabledEvents & gGenfskRxEvent)
            {
#if LINK_MONITOR
                uint16_t byteCount = ((GENFSK->RX_WATERMARK & GENFSK_RX_WATERMARK_BYTE_COUNTER_MASK)
>> GENFSK_RX_WATERMARK_BYTE_COUNTER_SHIFT);
                if(packetFromMaster == 1)          //master data
                {
                    GENFSK->XCVR_CTRL = 5;

                    packetFromMaster = 0;
                    masterPacketLen = byteCount;
                    FLib_MemCpy(masterPacket, (uint8_t*)PACKET_BUFFER_BASE_ADDR, masterPacketLen);
                    masterRSSI = (int8_t)((GENFSK->XCVR_STS & GENFSK_XCVR_STS_RSSI_MASK) >>
GENFSK_XCVR_STS_RSSI_SHIFT);
                }
                else                                //slave data
                {
                    slaveRSSI = (int8_t)((GENFSK->XCVR_STS & GENFSK_XCVR_STS_RSSI_MASK) >>
GENFSK_XCVR_STS_RSSI_SHIFT);
                    packetFromMaster = 1;
                    eventFlags |= gGenfskRxEventFlag_c;

                    GENFSK_TimeCancelEvent(&rxTimeoutTimer);
                    GENFSK->T1_CMP &= ~GENFSK_T1_CMP_T1_CMP_EN_MASK;
                }
#else
                eventFlags |= gGenfskRxEventFlag_c;
#endif
            }
            else
            {
                /* No notification enabled */
                genfskLocal[mGenfskActiveInstance].genfskState = gGENFSK_LL_Idle;
            }
```

## 3.5 Disabling DCOC calibration

For processing data packets received from master and slave, the DC Offset Calibration time is relatively long, so DC Offset Calibration needs to be disabled.

```
static void BleGenfskDisableDcocCal(void)
{
    static uint8_t dcoc_cal_enabled = TRUE;
    if(TRUE == dcoc_cal_enabled)
    {
        dcoc_cal_enabled = FALSE;

        XCVR_RX_DIG->RX_DIG_CTRL &= ~XCVR_RX_DIG_RX_DIG_CTRL_RX_DCOC_CAL_EN_MASK;
        XCVR_TSM->TIMING36   = 0x3431FFFFU;
        XCVR_TSM->TIMING37   = 0x3231FFFFU;
        XCVR_TSM->TIMING39   = 0x3431FFFFU;
        XCVR_TSM->TIMING14   = 0x34316863U;
```

```
            XCVR_TSM->END_OF_SEQ = 0x34336E67U;
        }
    }
```

## 3.6 Multi-link monitoring

Figure 5 shows a block diagram of monitoring multiple Bluetooth LE connections. It contains three roles, master, slave and monitor. There are four Bluetooth LE connections between master and slaves, and CAN bus or LIN bus communication between master and monitors to transfer connection information.

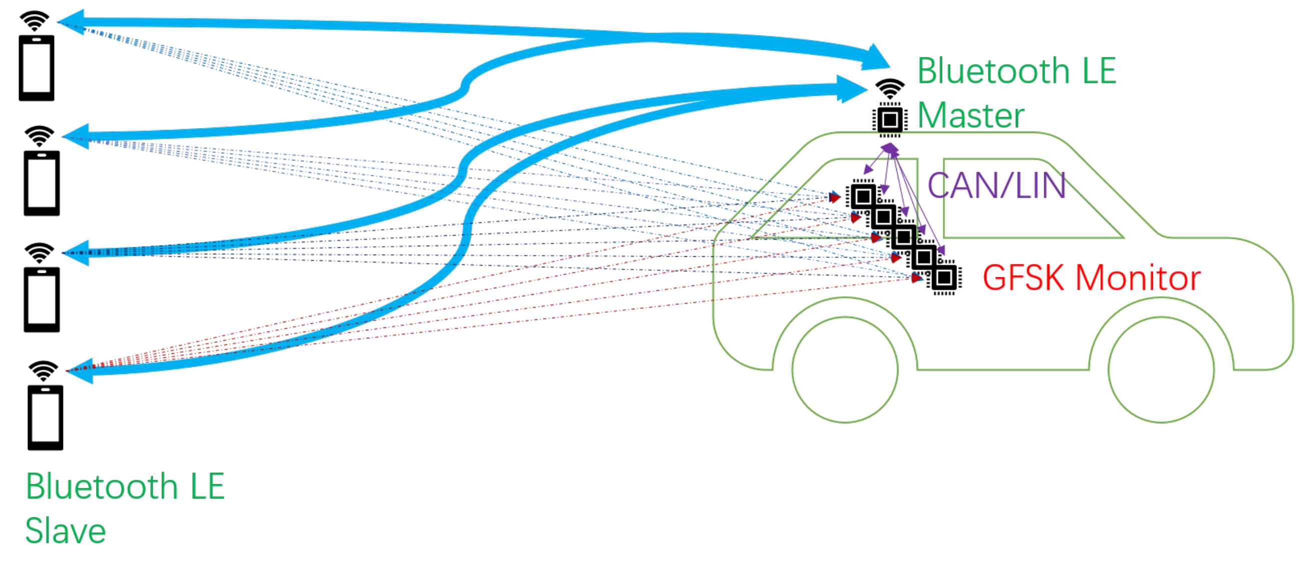


Figure 5. Monitoring multiple connections

For the monitoring of a single link, we can wait for a valid data packet on a fixed channel. Once the data is received, the next data can be tracked according to the frequency hopping algorithm. The connection interval is simple but the multi-link monitoring is relatively complicated because it involves link management.

Figure 6 is a instant timing diagram of a master connected to four slaves. With S1 as the anchor point, the interval between the device and the next device is fixed to 33 slots (about 20 ms).

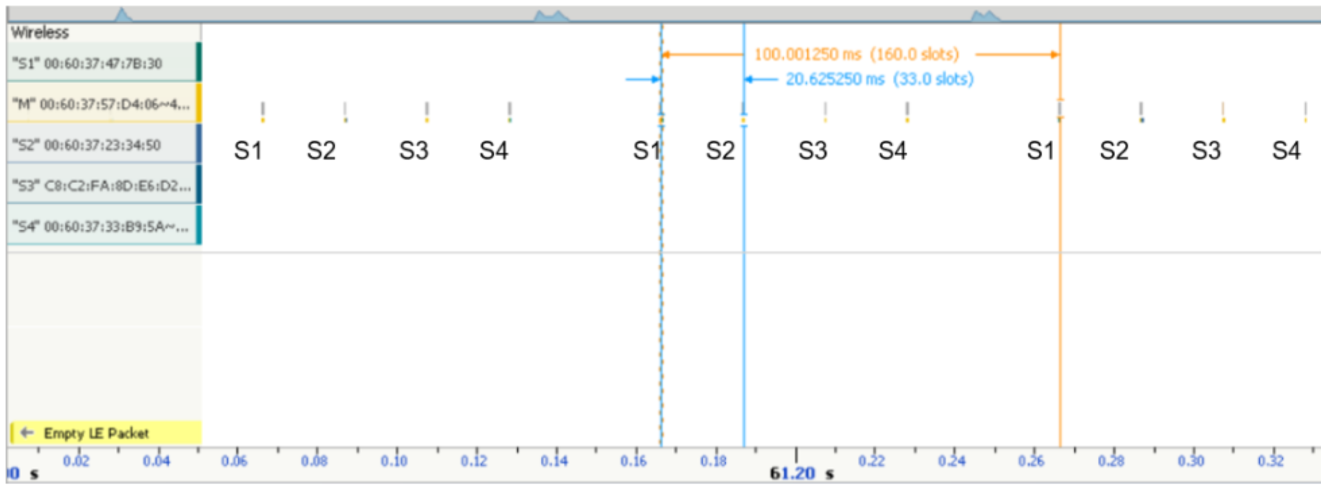


Figure 6. 4-link connection instant timing

We can synchronize the S1 device by monitoring the fixed channel and then use S1 as the anchor point and extend 20 ms in order to synchronize other devices. Figure 7 is an operation flowchart.

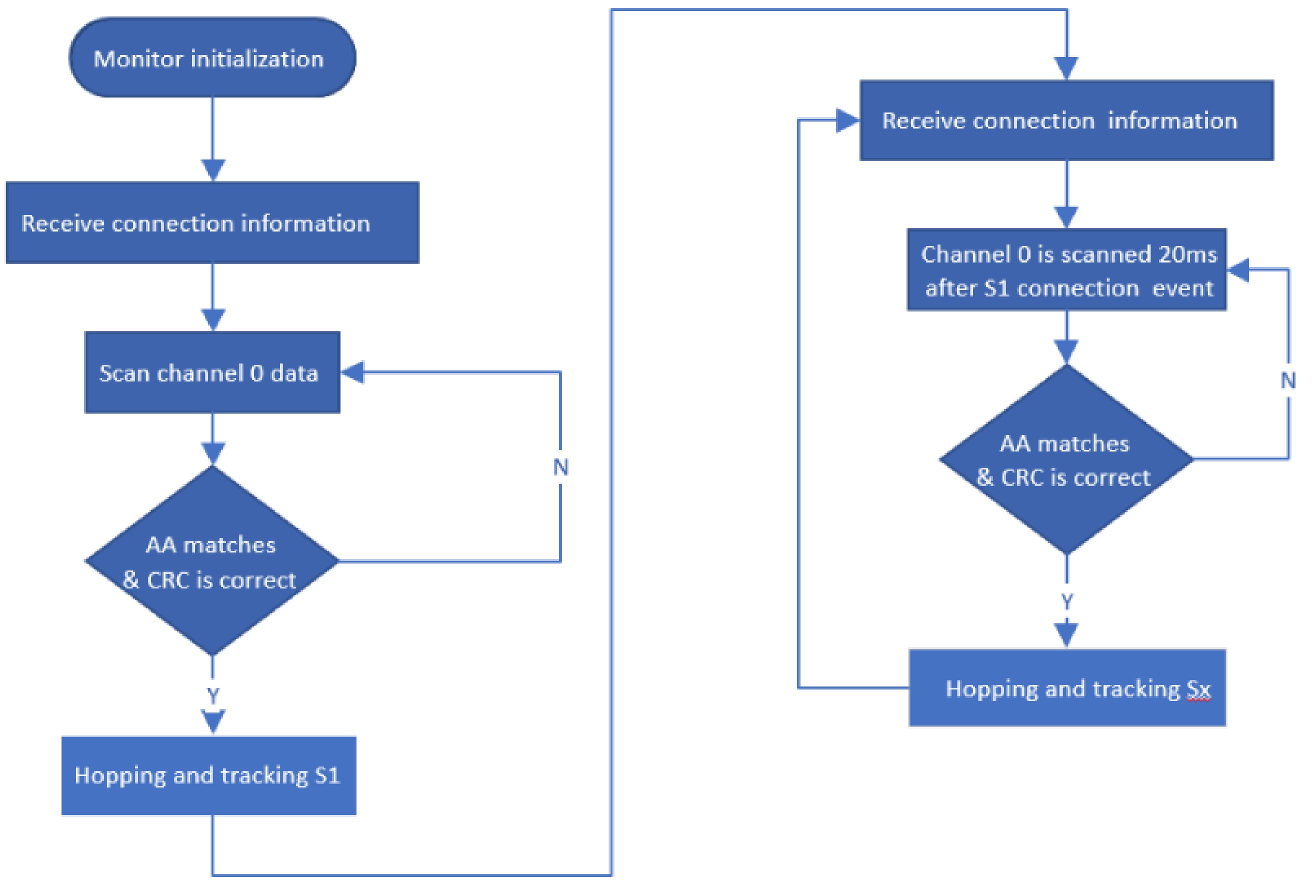


Figure 7. Multi-link monitoring – flow chart

## 4 Setup and results

As shown in Figure 8, in order to simplify the setup, we reduce the number of monitors to one. The demo contains one master, one monitor, and four slaves. The master and slaves are connected through Bluetooth LE. The master and monitor are connected through CAN or LIN bus.

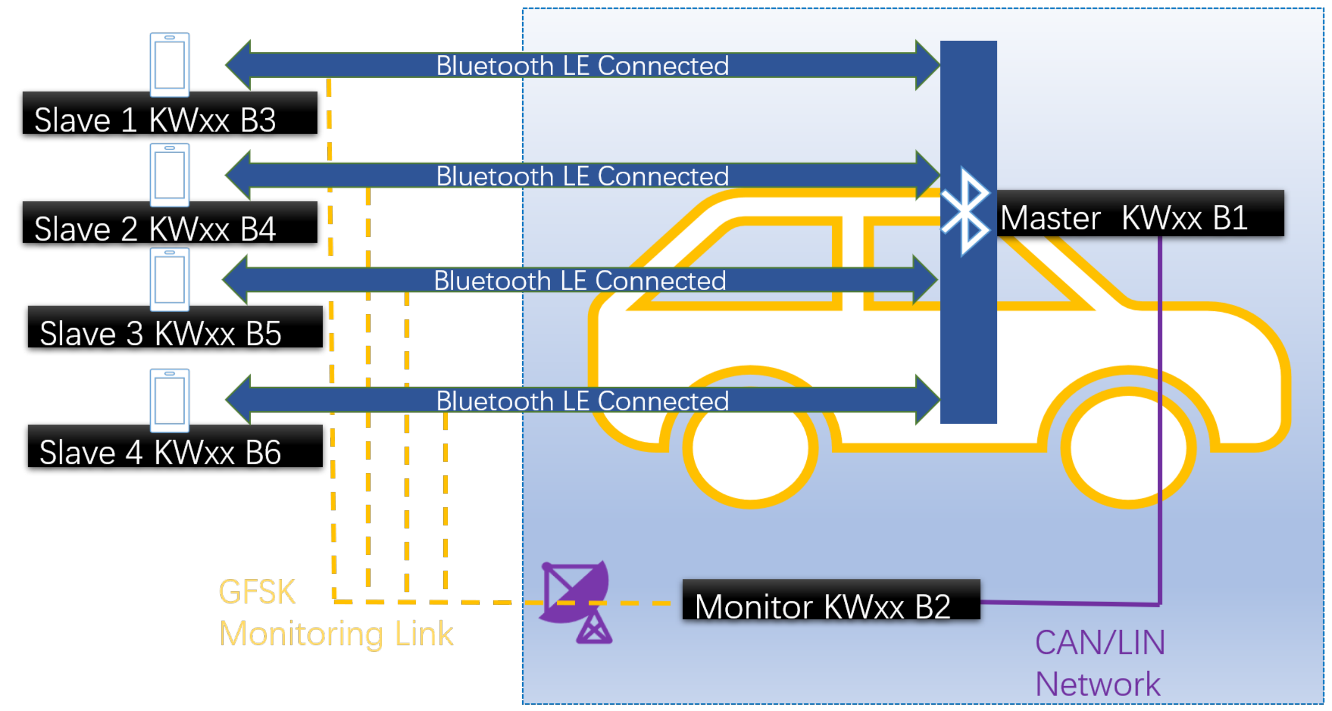


Figure 8. Setup

### 4.1 Hardware prerequisites

- Six Mini/micro USB cables.
- Six FRDM-KW36 or FRDM-KW38 boards. Board is simplified as B.
  - B1 as master.
  - B2 as monitor.
  - B3-B6 as slave.
- Personal computer.
- Power adapter 12 V.
- Three Dupont female-to-female wire.

### 4.2 Board settings

- Connect 12 V adapter to J32 of board B1 or B2.
- Unmount R34 and R27 resistors of board B2.
- Connect J13-1 of the board B1 and B2.
- Connect J13-2 of the board B1 and B2.
- Connect J13-4 of the board B1 and B2.

NOTE

When using the auto baudrate feature, connect J1-5 and J2-9.

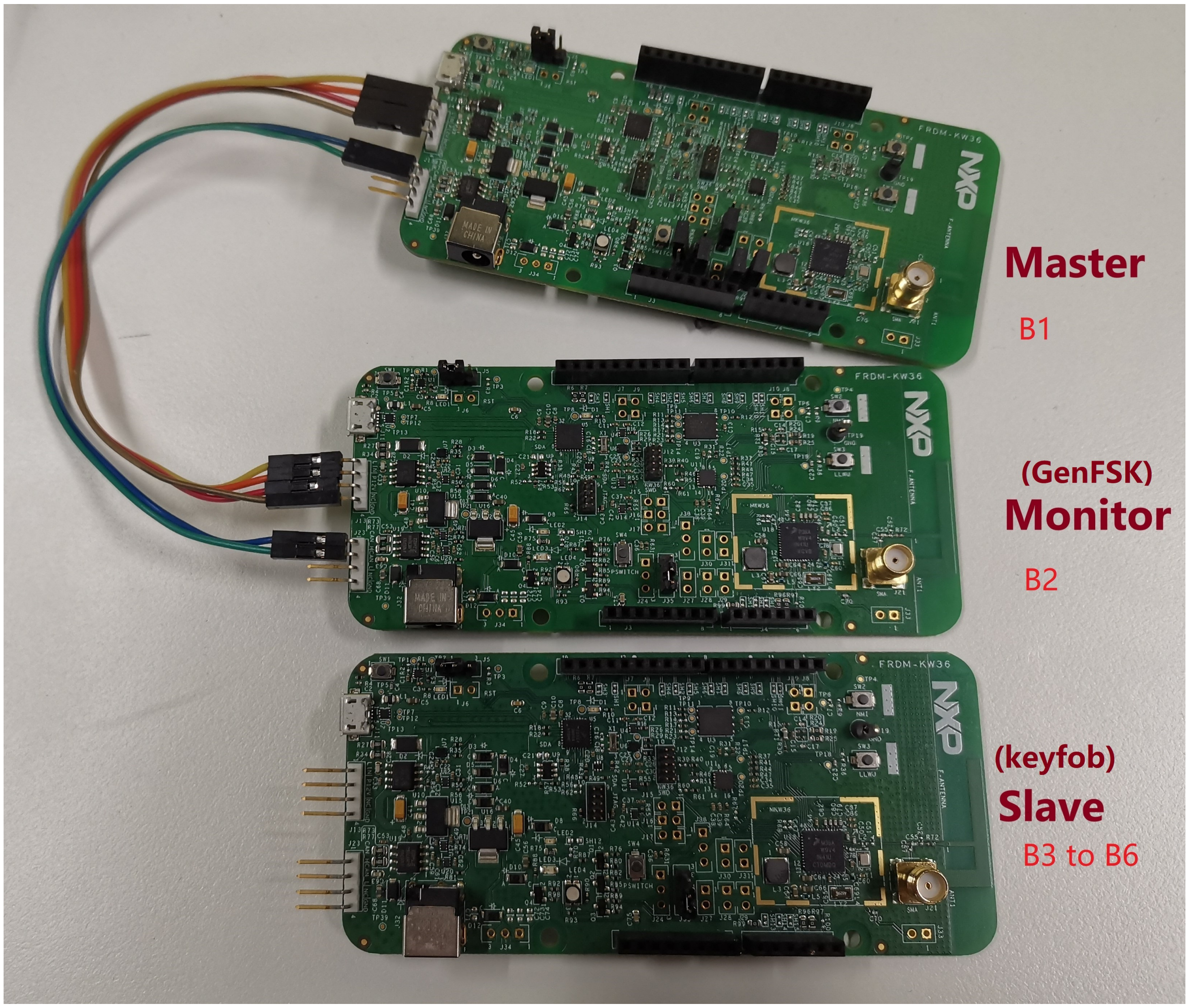


Figure 9. Link monitoring setup with four slaves

## 4.3 Toolchain supported

- IAR Embedded Workbench 8.40.1

## 4.4 Software prerequisites

Link monitoring demo is based on the Wireless UART demo and genfsk demo for FRDM-KW36 or FRDM-KW38, which can be found in AN12872SW. In the application, there are three roles, master, slave and monitor.

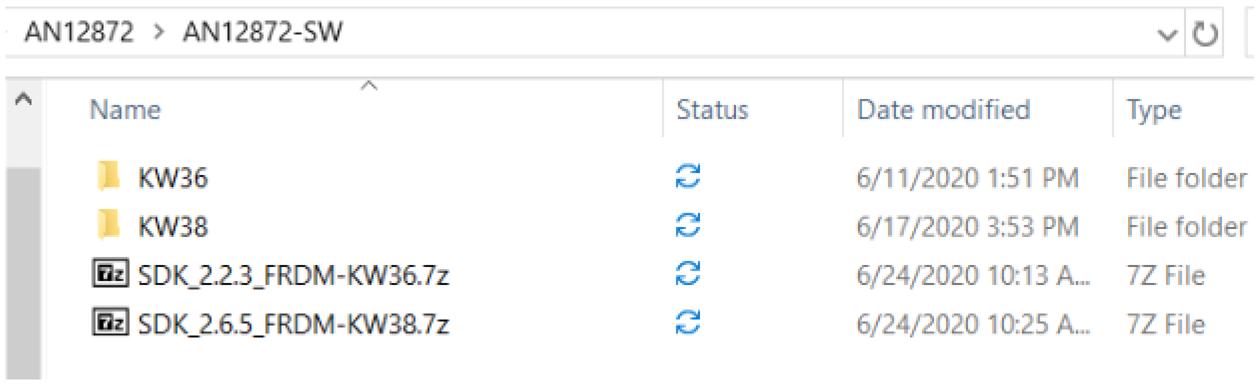
- The project for master is modified from the `wireless_uart` project.
- The project for monitor is modified from the `genfsk` project.
- The slave project is the `wireless_uart` project.

In the master and Monitor projects, we add the functions of CAN and LIN to realize the transmission of connection information.

There are two ways to evaluate the application, described as below.

- Use the ported SDK directly.

  For quick evaluation, AN12872SW includes the SDK that have been ported, SDK_2.2.3_FRDM-KW36 and SDK_2.6.5_FRDM-KW38, as show in Figure 10.



AN12872 > AN12872-SW

| Name | Status | Date modified | Type |
|---|---|---|---|
| KW36 | | 6/11/2020 1:51 PM | File folder |
| KW38 | | 6/17/2020 3:53 PM | File folder |
| SDK_2.2.3_FRDM-KW36.7z | | 6/24/2020 10:13 A... | 7Z File |
| SDK_2.6.5_FRDM-KW38.7z | | 6/24/2020 10:25 A... | 7Z File |

Figure 10. Porting the code using proted SDK

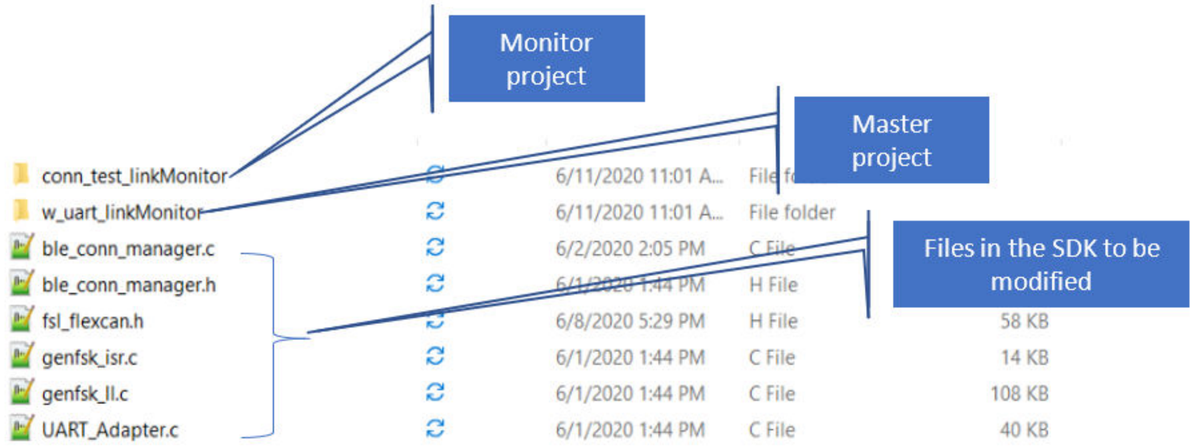
  — The `w_uart_linkMonitor` project – For B1 as master, located at *SDK\boards\frdmkw36\wireless_examples\bluetooth\w_uart_linkMonitor\freertos*.

  — The `conn_test_linkMonitor` project - For B2 as monitor, located at *SDK\boards\frdmkw36\wireless_examples\genfsk\conn_test_linkMonitor\freertos*.

  — The `w_uart` project – For B3-B6 as slave, located at *SDK\boards\frdmkw36\wireless_examples\bluetooth\w_uart\freertos*.

- Port the code to the SDK.

  The AN12872 software package also contains source files and header files, as shown in Figure 11.



| | | | |
|---|---|---|---|
| conn_test_linkMonitor | 6/11/2020 11:01 A... | File fo | |
| w_uart_linkMonitor | 6/11/2020 11:01 A... | File folder | |
| ble_conn_manager.c | 6/2/2020 2:05 PM | C File | |
| ble_conn_manager.h | 6/1/2020 1:44 PM | H File | |
| fsl_flexcan.h | 6/8/2020 5:29 PM | H File | 58 KB |
| genfsk_isr.c | 6/1/2020 1:44 PM | C File | 14 KB |
| genfsk_ll.c | 6/1/2020 1:44 PM | C File | 108 KB |
| UART_Adapter.c | 6/1/2020 1:44 PM | C File | 40 KB |

Figure 11. Porting the code by yourself

  Perform the following steps to create new demo projects in IAR Embedded Workbench and replace the source and header files modified for the demo.

  1. Place `conn_test_linkMonitor` to the directory of *SDK\boards\frdmkw3x\wireless_examples\genfsk*.
  2. Place `w_uart_linkMonitor` to the directory of *SDK\boards\frdmkw3x\wireless_examples\bluetooth*.
  3. Replace five files in the following directory:

— *middleware/wireless/bluetooth_x.x.x/application/common/ble_conn_manager.c*

— *middleware/wireless/bluetooth_x.x.x/application/common/ble_conn_manager.h*

— *middleware/wireless/framework_x.x.x/SerialManager/Source/UART_Adapter.c*

— *middleware/wireless/genfsk_x.x.x/source/genfsk_isr.c*

— *middleware/wireless/genfsk_x.x.x/source/genfsk_ll.c*

NOTE

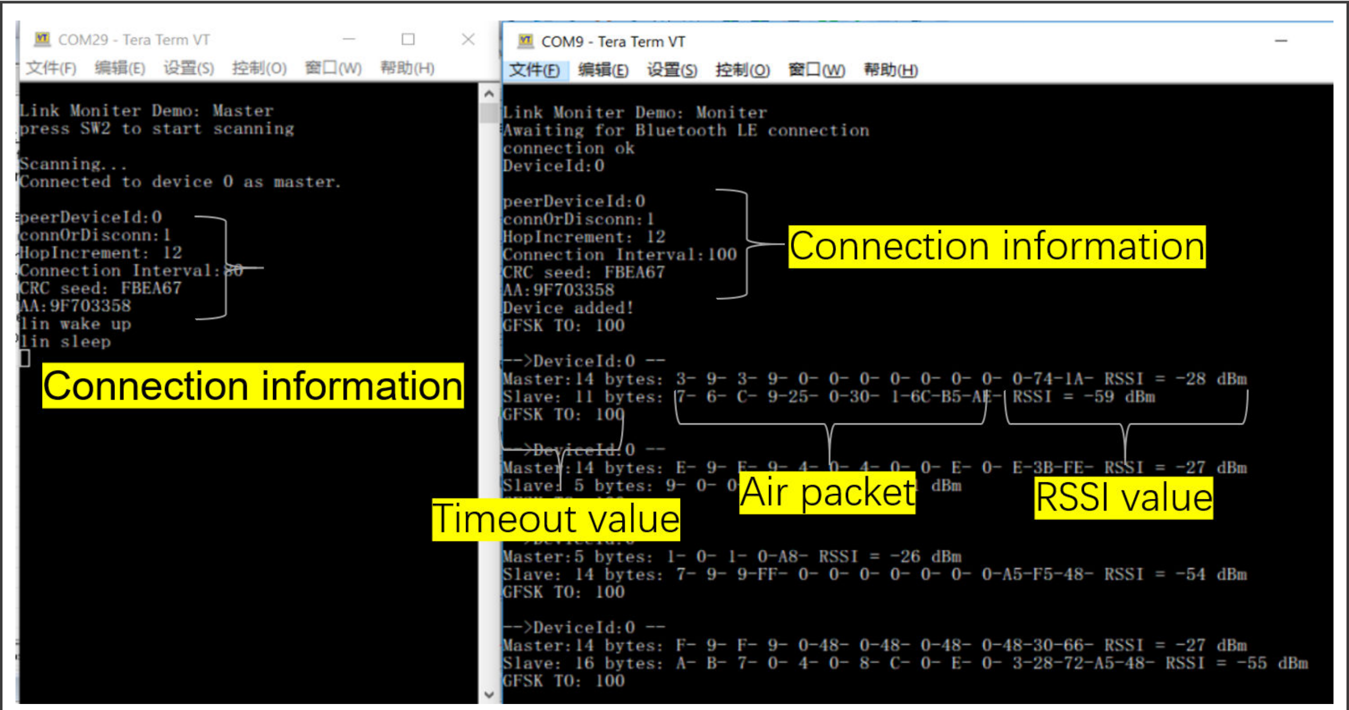
The project created above is based on SDK 2.2.3 (released 2020-04-30) for KW36, SDK 2.6.5 (released 2020-05-07) for KW38. If you are using other versions of the SDK, please modify the project files accordingly.

## 4.5 Testing method

1. Download the program to the target boards.
2. Connect a micro USB cable between the PC and the OpenSDA USB port on the board B1 and B2.
3. Open a serial terminal on PC for OpenSDA serial device with these settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
4. Power on B3-B6, switch roles, and start advertising.
5. Press **SW2** on board B1 to start scanning, connection, and monitoring.

## 4.6 Testing results

Figure 12 shows the test result, including link data and RSSI value from master and slave. As expected, all the information is found due to the GFSK monitoring.



```
Link Moniter Demo: Master
press SW2 to start scanning

Scanning...
Connected to device 0 as master.

peerDeviceId:0
connOrDisconn:1
HopIncrement: 12
Connection Interval:80
CRC seed: FBEA67
AA:9F703358
lin wake up
lin sleep
```

```
Link Moniter Demo: Moniter
Awaiting for Bluetooth LE connection
connection ok
DeviceId:0

peerDeviceId:0
connOrDisconn:1
HopIncrement: 12
Connection Interval:100
CRC seed: FBEA67
AA:9F703358
Device added!
GFSK TO: 100

-->DeviceId:0 --
Master:14 bytes: 3- 9- 3- 9- 0- 0- 0- 0- 0- 0- 0- 0-74-1A- RSSI = -28 dBm
Slave: 11 bytes: 7- 6- C- 9-25- 0-30- 1-6C-B5-AE- RSSI = -59 dBm
GFSK TO: 100

-->DeviceId:0 --
Master:14 bytes: E- 9- E- 9- 4- 0- 4- 0- 0- E- 0- E-3B-FE- RSSI = -27 dBm
Slave: 5 bytes: 9- 0- 0                          dBm

Master:5 bytes: 1- 0- 1- 0-A8- RSSI = -26 dBm
Slave: 14 bytes: 7- 9- 9-FF- 0- 0- 0- 0- 0- 0- 0-A5-F5-48- RSSI = -54 dBm
GFSK TO: 100

-->DeviceId:0 --
Master:14 bytes: F- 9- F- 9- 0-48- 0-48- 0-48- 0-48-30-66- RSSI = -27 dBm
Slave: 16 bytes: A- B- 7- 0- 4- 0- 8- C- 0- E- 0- 3-28-72-A5-48- RSSI = -55 dBm
GFSK TO: 100
```

Figure 12. Test results

## 5 Conclusion

Through GFSK, we can effectively monitor the connection of Bluetooth LE and combine with the high-precision RSSI detection capability of KW36/38. It is feasible and effective to realize the ranging with anti-relay attack through KW36 or KW38.

## 6 Revision history

Table 2. Revision history

| Rev. | Date | Description |
|---|---|---|
| 0 | 07/2020 | Initial release |
| 1 | 09/2020 | • Updated Figure 1, Figure 2, Figure 3, Figure 9, and Figure 12<br>• Added Figure 5 and Figure 8 |

Date of release: 09/2020
Document identifier: AN12872