AN12415

Hybrid (Dual-Mode) Bluetooth Low Energy and GenFSK Heart Rate Sensor Application

Rev. 0 — June 2019 Application Note

Contents

1 Introduction

This application note is a guide to implement an hybrid model with Bluetooth® Low Energy (LE) and Generic Frequency Shift Keying (GenFSK) based on a Heart Rate Sensor (HRS) application. This application is called as hybrid HRS across the document.

To share the radio, GenFSK is only executed during the Idle states of the Bluetooth LE protocol, this coexistence can be given by the Mobile Wireless System (MWS) coexistence API.

2 Hybrid (dual-mode) definition

A hybrid (dual-mode) model is defined as the coexistence of two protocols that share a common resource in the application, in this case the radio without one protocol interrupting the other.

The application demonstrates the coexistence of GenFSK and Bluetooth LE using the MWS software layer to manage the radio resource between the two protocols. MWS coexistence model can manage priorities that allows the application to preempt one protocol over to the other. However, for Bluetooth LE hybrid applications, the GenFSK protocol must ensure to release the radio before any Bluetooth LE event.

3 MWS coexistence

The MWS coexistence API functionality allows the coexistence of multiple wireless protocols on the same MCU and/or protocols on different MCUs.

This API is intended for use by the link layer of the protocols stacks, but higher layers can also control the access to the resources.

For the on-chip coexistence on dual mode microcontrollers like Kinetis KW36, the Bluetooth LE protocol has a higher priority than the GenFSK protocol, because of HW link layer implementation. For the same reason GenFSK protocol will only be executed during the inactivity periods of Bluetooth LE. Before starting a sequence, GenFSK needs to check the inactivity time. If there is not enough time to complete a sequence, it will be notified that it can't access to the radio.

For detailed information about the MWS coexistence and other functionalities, refer to the documentation of the connectivity framework located inside the SDK common wireless documentation.

For the development of the application, the API primitives that will be used are:

- MWS_Register on page 2
- MWS_GetInactivityDuration on page 2
- MWS_Acquire on page 2
- MWS_Release on page 2

All the API primitives are conditioned by the `gMWS_Enabled_d flag` state. If this is not enabled, they won't be used.

3.1 MWS_Register

This API registers the protocol stack to MWS.

Before using any other MWS API, the application shall register the protocols that are going to form a part of the application. Otherwise, the MWS cannot manage the resources of the protocols.

```
mwsStatus_t MWS_Register    (mwsProtocols_t protocol, pfMwsCallback cb);
```

For the objective of the application, the registered protocols are the GenFSK and the Bluetooth LE. The protocols are registered most of the times, for the GenFSK inside of the `genfsk_ll.c` and the Bluetooth LE in the `ble_controller_task.c`.

3.2 MWS_GetInactivityDuration

This function calculates the inactivity period of the current protocol and returns the minimum time until the highest priority protocol needs to be serviced.

Prototype:

```
uint32_t MWS_GetInactivityDuration (mwsProtocols_t currentProtocol);
```

For this application, the API is used to determine the period that GenFSK can be executed depending on the Bluetooth LE.

3.3 MWS_Acquire

This API tries to acquire the radio for the protocol, you can either force the protocol to be preempted or wait for the radio to be available, this is set by a flag of TRUE or FALSE in where TRUE means to preempt the protocol to use the radio.

Prototype:

```
mwsStatus_t MWS_Acquire     (mwsProtocols_t protocol, uint8_t force)
```

This API helps in the coexistence of both protocols, the radio must be acquired to be used. Before starting any protocol activity, the protocol must ensure that the radio was successfully acquired.

3.4 MWS_Release

This function will release access to the radio, and it will notify other protocols that the resource is idle waiting to be used.

Prototype:

```
mwsStatus_t MWS_Release     (mwsProtocols_t protocol);
```

The protocol can release the radio whenever it finishes its activity. There is no need to release the radio when performing Bluetooth LE activity as it is handled by the Bluetooth LE stack. The application must release the radio when there is no GenFSK activity or before any Bluetooth LE event. Otherwise, this could prevent Bluetooth LE to access the radio.

4 Hybrid application

The hybrid application consists on adding GenFSK operation to a Heart Rate Sensor, hence, this is called hybrid HRS based on a dual-protocol communication, in this case, the KW36 device uses the radio to communicate with a Bluetooth LE Central Device (smartphone) and a GenFSK device (KW36). As previously explained, the GenFSK packets are sent during the Idle states of the Bluetooth LE.

During GenFSK operation, the hybrid device transmits status of the Bluetooth LE communication. Once the packet is transmitted, it waits for an acknowledge packet from the remote GenFSK only device. The GenFSK activity is driven by two timers:

- one interval timer to check if there is enough time for packet transmission.
- one timer, whose value depends on the `MWS_GetInactivityDuration`, aborts GenFSK activity if no acknowledge packet is received.

GenFSK operation begins once the hybrid HRS starts advertising.

Figure 1 on page 3 shows the hybrid application.

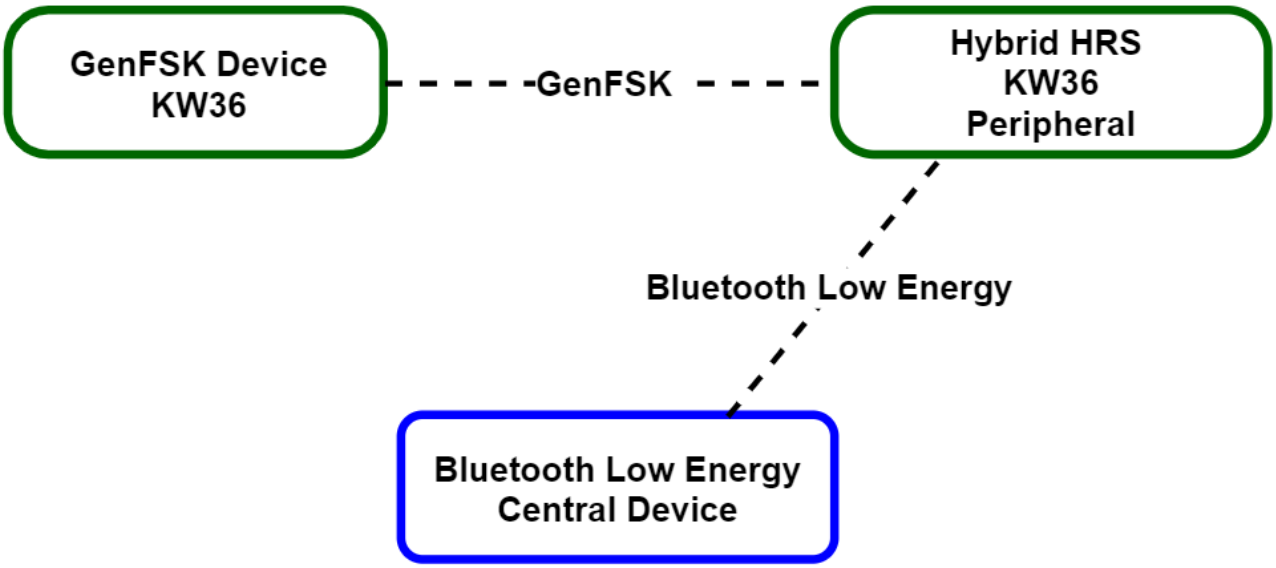

Figure 1. Hybrid HRS application diagram

Figure 2 on page 4 shows the application expected behavior.

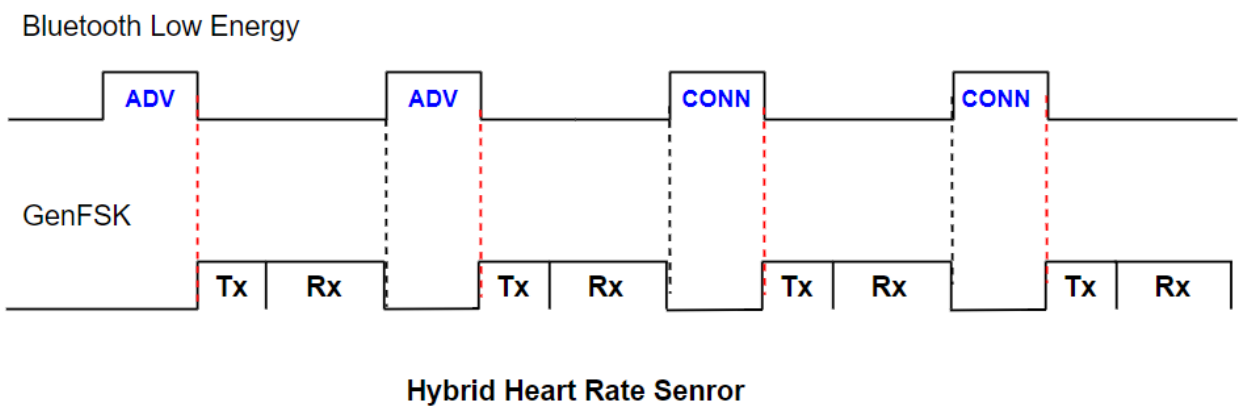

Figure 2. Hybrid application operation

As shown in Figure 2 on page 4, the GenFSK operation is expected during the Bluetooth LE idle state. The GenFSK idle state is during advertising (ADV) or connection (CONN) events.

4.1 Hybrid development

Figure 3 on page 5 shows the expected functionality of the application.

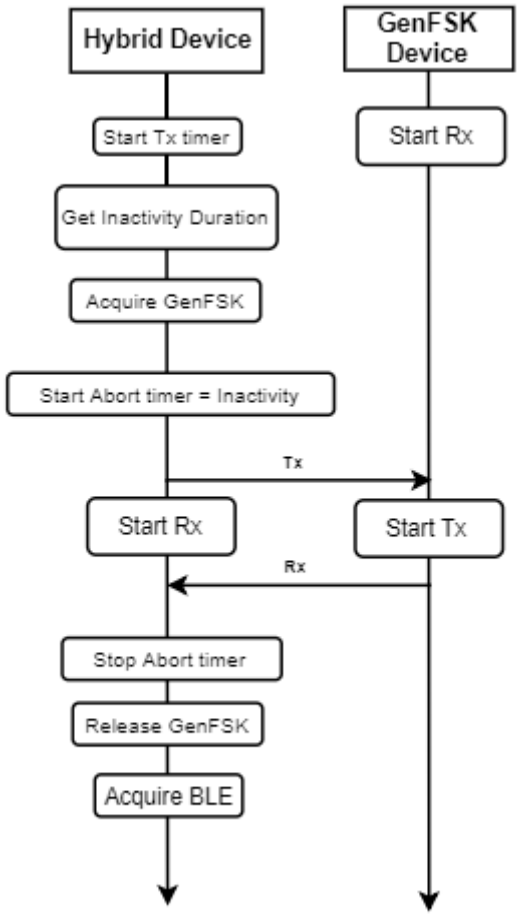

Figure 3. Hybrid expected functionality diagram

In the Hybrid device, the transmission interval timer starts when the Bluetooth LE begins to advertise. When the timer expires, the application callback requests for the inactivity duration time of the other registered protocols excluding GenFSK. If there is enough time (more than 10 miliseconds), the GenFSK protocol requests for the radio. If the radio is acquired, the abort timer is started. It is important to mention that abort timer is calculated using the `MWS_GetInactivityDuration`. In addition to this, 10 milliseconds are subtracted from the inactivity duration to give some time to the Bluetooth LE stack to acquire the radio.

```
MWS_GetInactivityDuration(gMWS_GENFSK_c);
MWS_Acquire(gMWS_GENFSK_c, FALSE);
```

If the GenFSK Tx transmission was successful, application switches to GenFSK Rx mode to begin the reception of the acknowledge packet from the GenFSK only device. If the acknowledge packet is received, the application stops the abort timer, releases GenFSK to send an MWS idle signal to Bluetooth LE stack to be able to acquire the radio.

```
MWS_Release(gMWS_GENFSK_c);
```

Figure 4 on page 6 shows the application behavior in case no acknowledge packet is received.

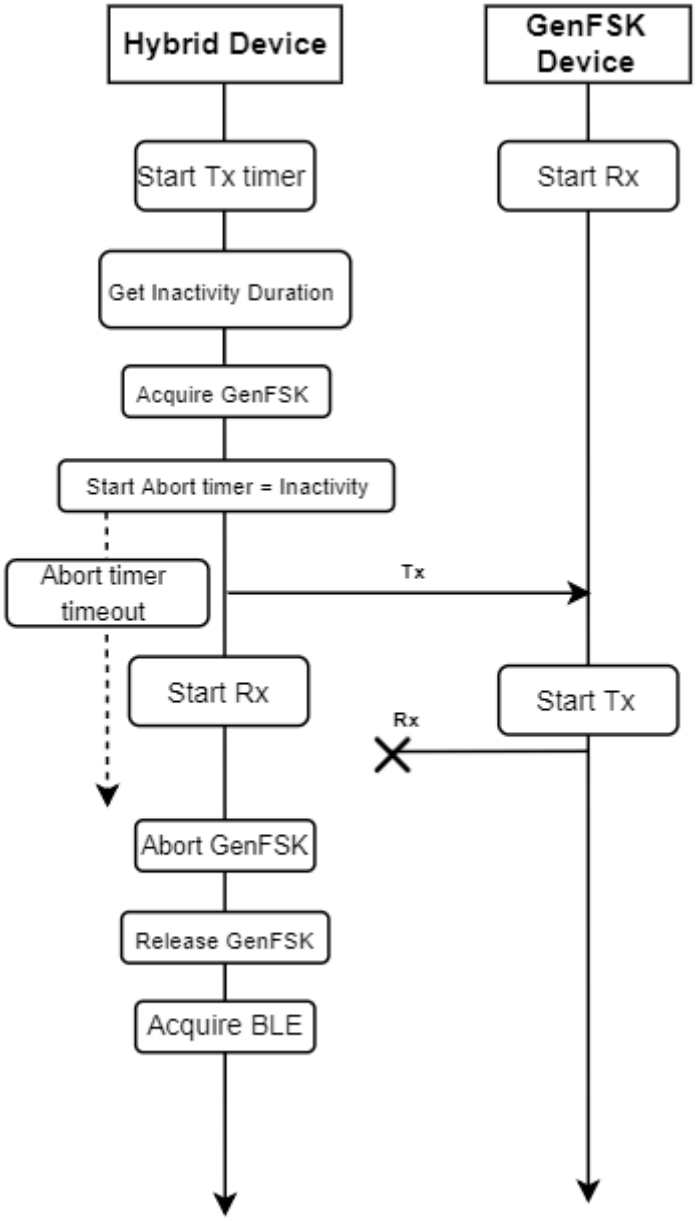

Figure 4. Application behavior when acknowledge packet is not received

If the GenFSK Tx is successful but the Rx package has never been received during the abort timer, the abort Rx timer callback is triggered and it aborts any GenFSK activity to release the radio. Then, Bluetooth LE can acquire the radio.

```
GENFSK_AbortAll();
MWS_Release(gMWS_GENFSK_c);
```

Figure 5 on page 7 shows the Hybrid GenFSK transmission behavior.

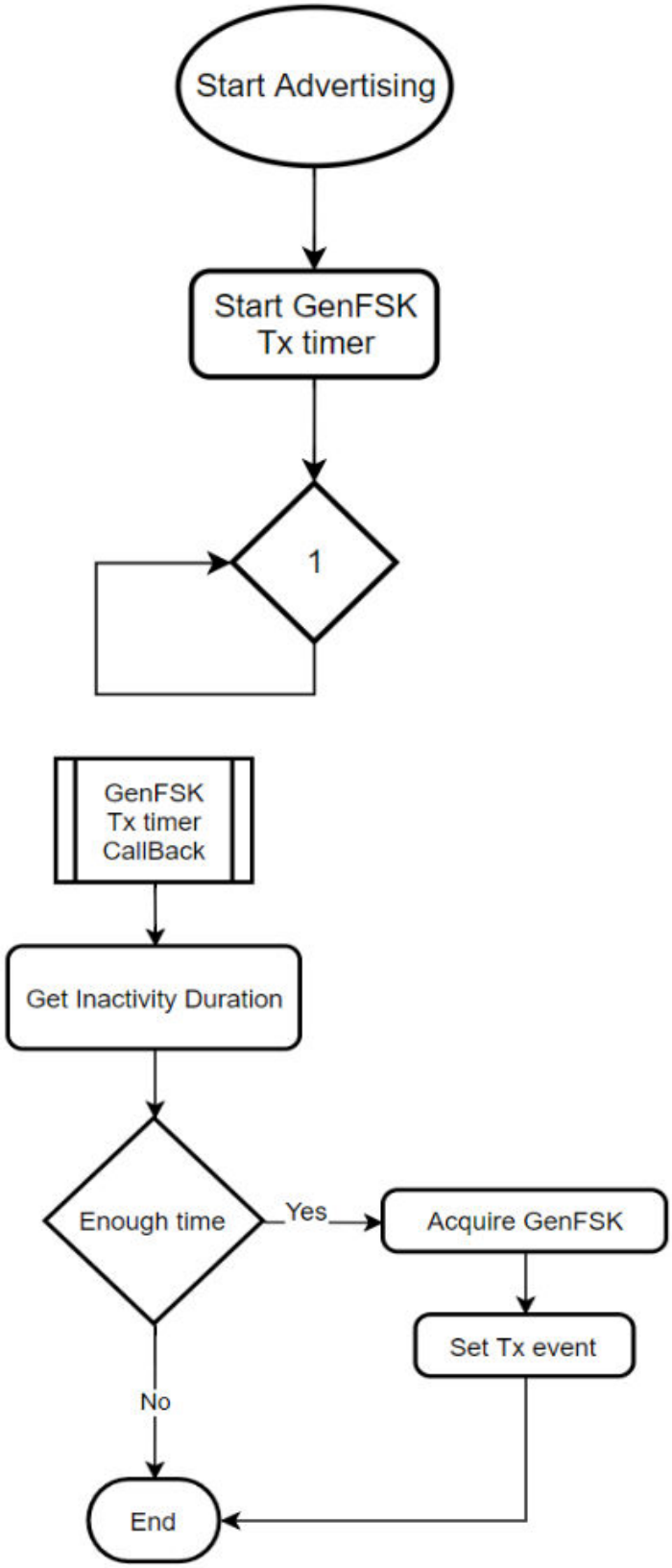

Figure 5. Hybrid GenFSK transmission behavior

If GenFSK has enough time to execute, a Tx event is set, and a transmission packet is sent. The content of the packet depends on the Bluetooth LE status (connected, advertising or disconnected).

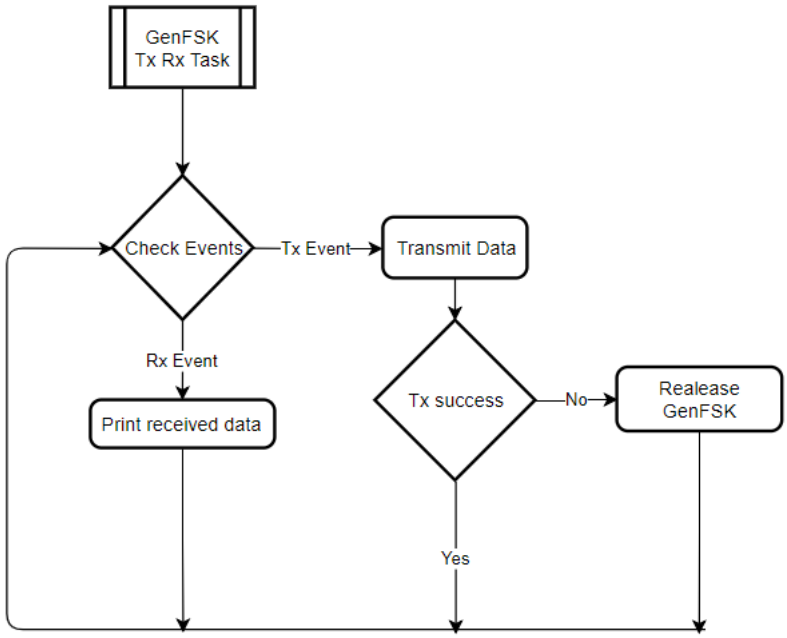

Figure 6. Hybrid GenFSK task flow chart

If a packet wasn't transmitted properly, GenFSK releases the radio and waits until the next TX timer interval to check the events to transmit a new packet.

5 Hybrid HRS demo application

This application works with two different protocols, Bluetooth LE and GenFSK. The application sends GenFSK packets during the Idle state of Bluetooth LE. The objective of this application is to communicate with both protocols and send packets with different messages depending on the HRS Bluetooth LE states: advertising, connected or disconnected.

The GenFSK only device communicates with the hybrid device which at the same time is connected via Bluetooth LE to a smartphone running the NXP IoT Toolbox application

Prerequisites for the application:

- Two FRDM-KW36 boards. One acting as a Hybrid HRS (Bluetooth LE peripheral) and other as GenFSK only device.
- MCUXpresso IDE 10.2 or later.
- Smartphone running NXP IoT Toolbox (iOS or Android).

5.1 Programming and running the application

This application note includes two compressed zip folders, one including the hybrid project and the other including the GenFSK-only project.

Perform the following steps to import, compile and program the projects.

5.1.1 Importing a project to MCUXpresso

MCUXpresso IDE supports importing project directly from the zipped package.

Perform the following steps to import a zipped package:

1. Click the **import projects from file system…** command in the quick start panel.

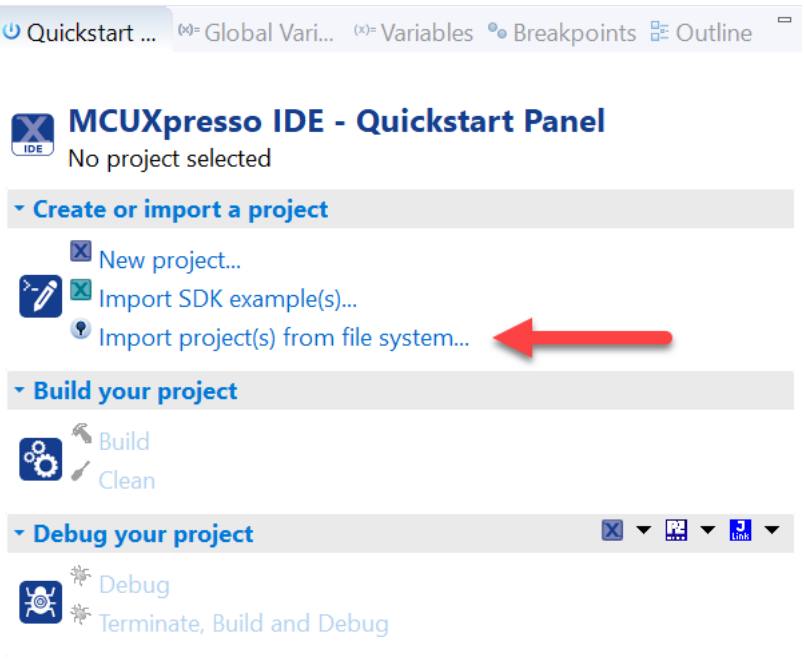

Figure 7. Importing a zipped file (step 1)

2. Click **browse…** and chose the zipped driver or application package.

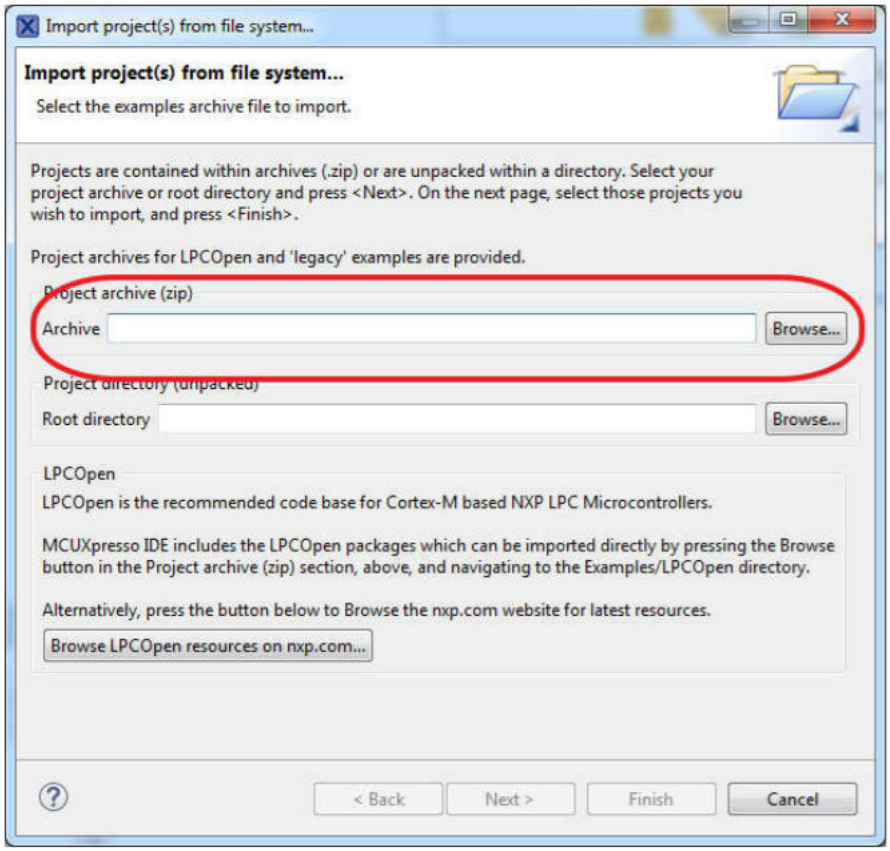

Figure 8. Importing a zipped file (step 2)

3. Click **Next** and the project included in the package is displayed.

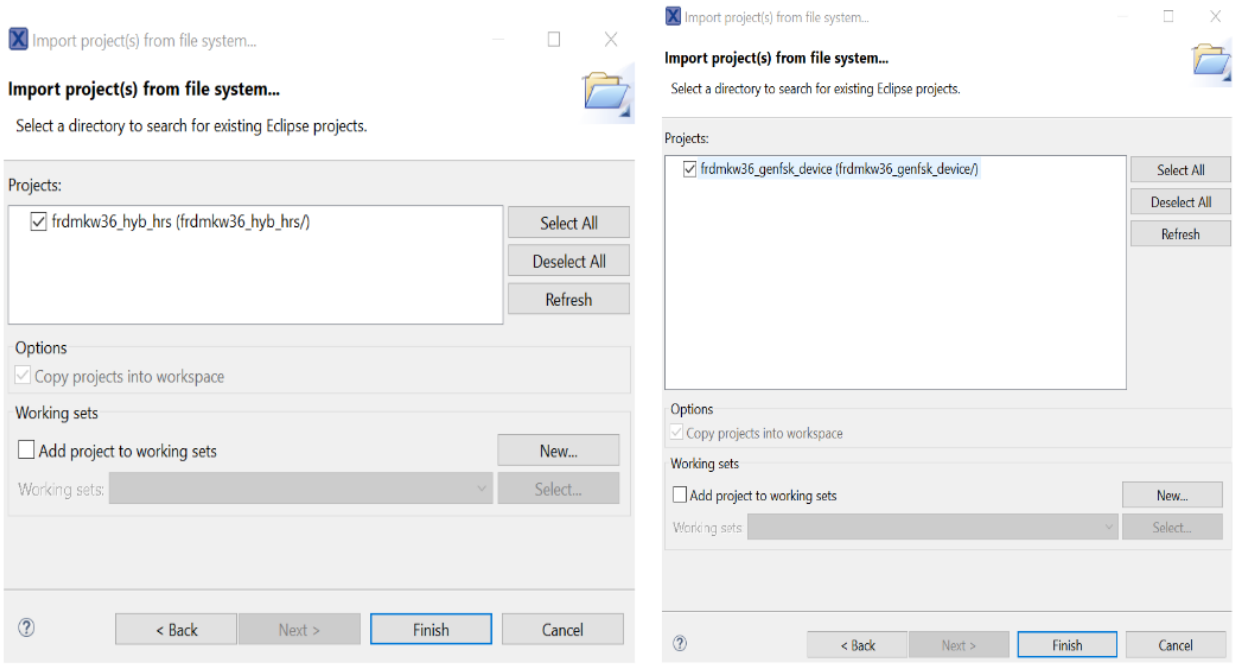

Figure 9. Importing a zipped file (step 3)

4. Then, the project can be seen in the IDE workspace and root directory.

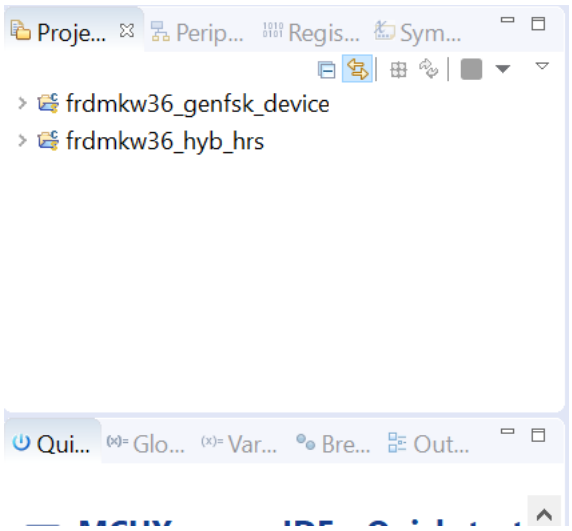

Figure 10. Importing a zipped file (step 4)

Perform the following steps to compile the projects and program the boards.

1. Open the hybrid project and compile.

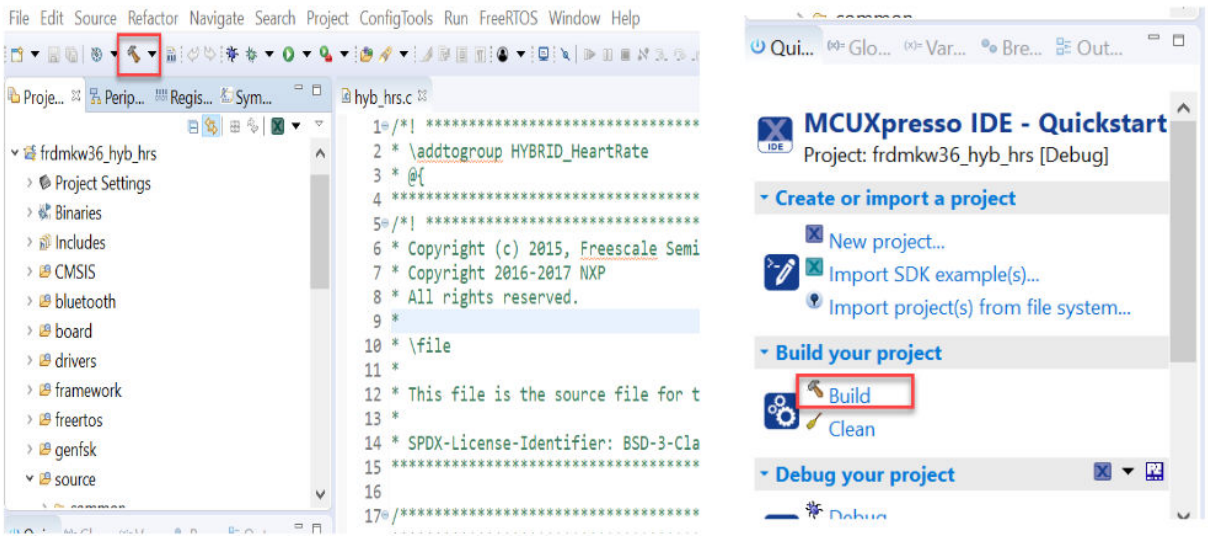

Figure 11. Compiling the projects and programming the boards (step 1)

2. Program the hybrid project in the board.

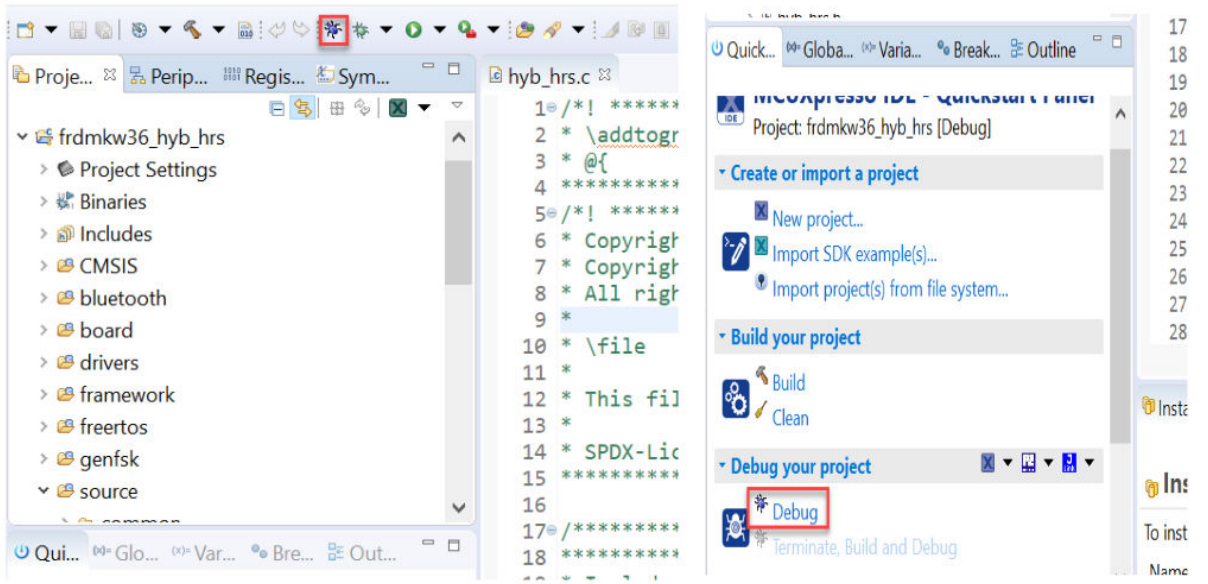

Figure 12. Compiling the projects and programming the boards (step 2)

3. Open the GenFSK device project and compile.

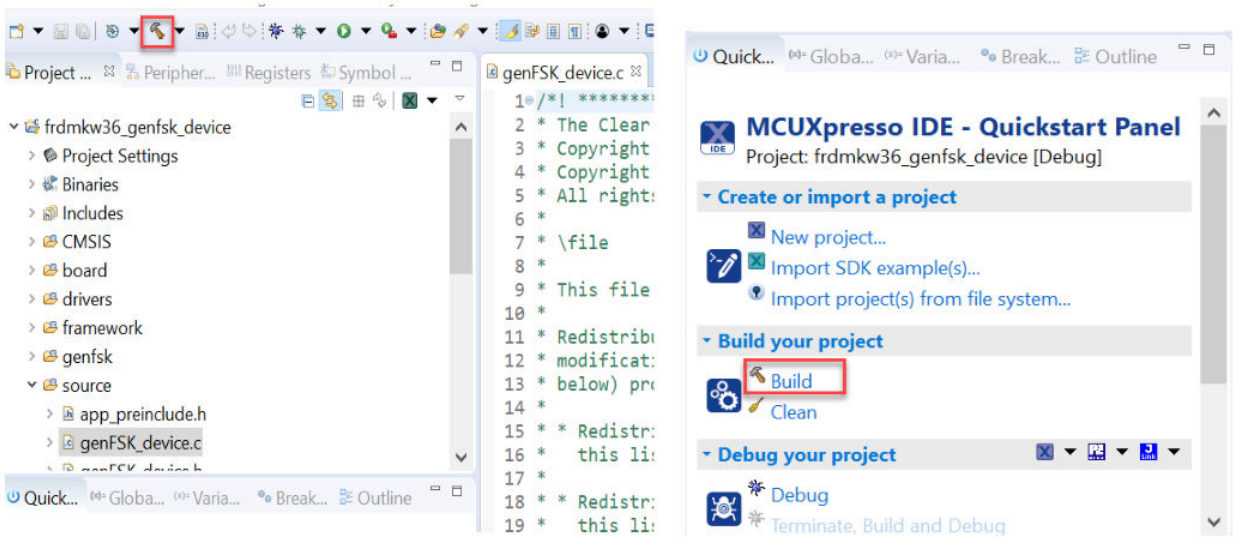

Figure 13. Compiling the projects and programming the boards (step 3)

4. Program the GenFSK device project in the board.

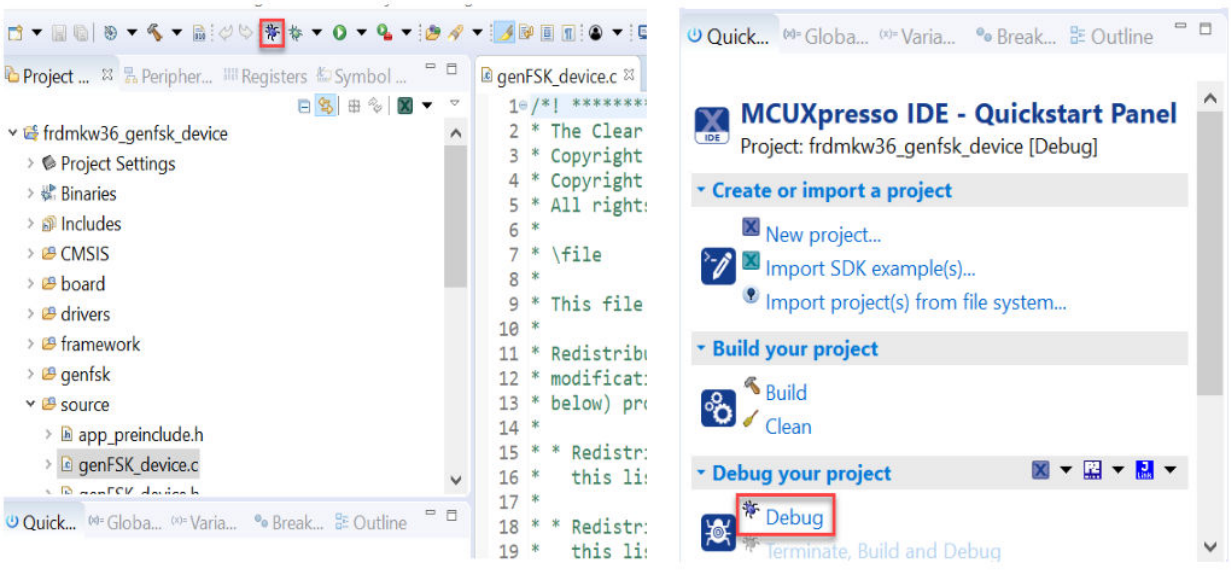

Figure 14. Compiling the projects and programming the boards (step 4)

5. Once both projects are flashed, open two terminals, one for each project, with the following parameters.
   - Baud rate: 115200
   - Data: 8 bit
   - Parity: none
   - Stop bit: 1 bit

5.1.2 Running the application (hybrid device perspective)

Perform the following steps to to start the Hybrid HRS demo:

1. On the hybrid HRS, press the **SW2** to begin advertising.

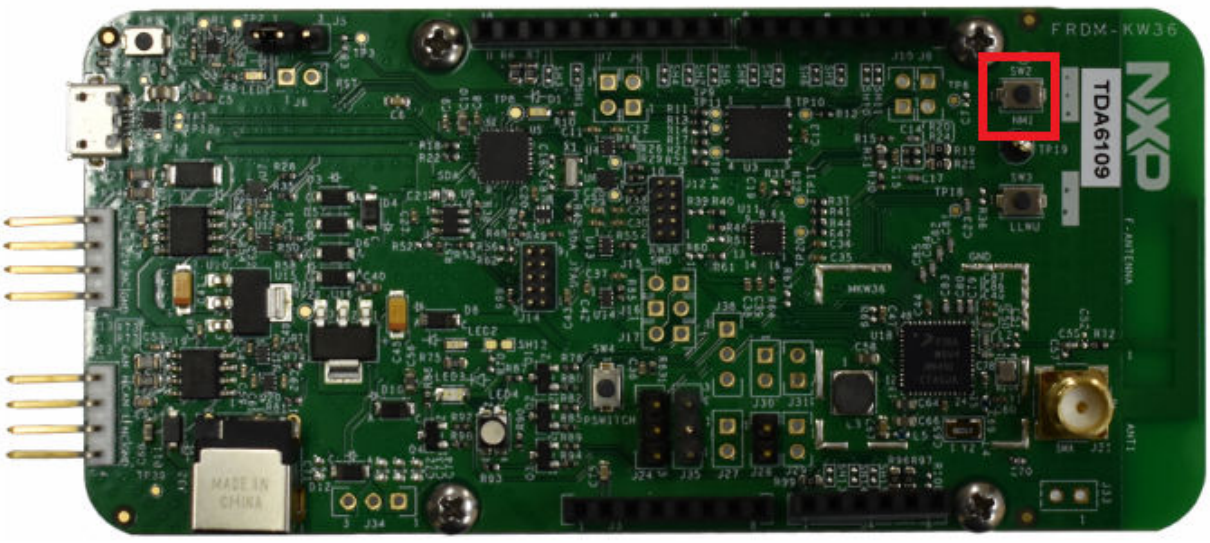

Figure 15. Running the application (step 1)

2. Connect via Bluetooth LE with NXP IoT Toolbox HRS application.

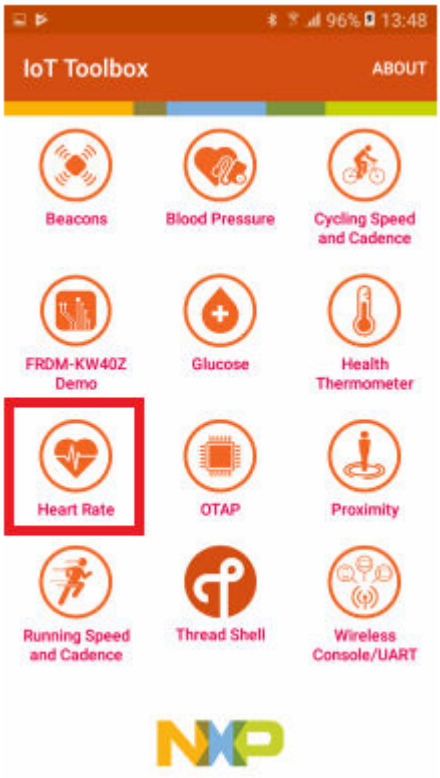

Figure 16. Running the application (step 2)

3. In the terminal screen of the hybrid device a connected Bluetooth LE appears. A disconnected message appears if the advertising restarts.

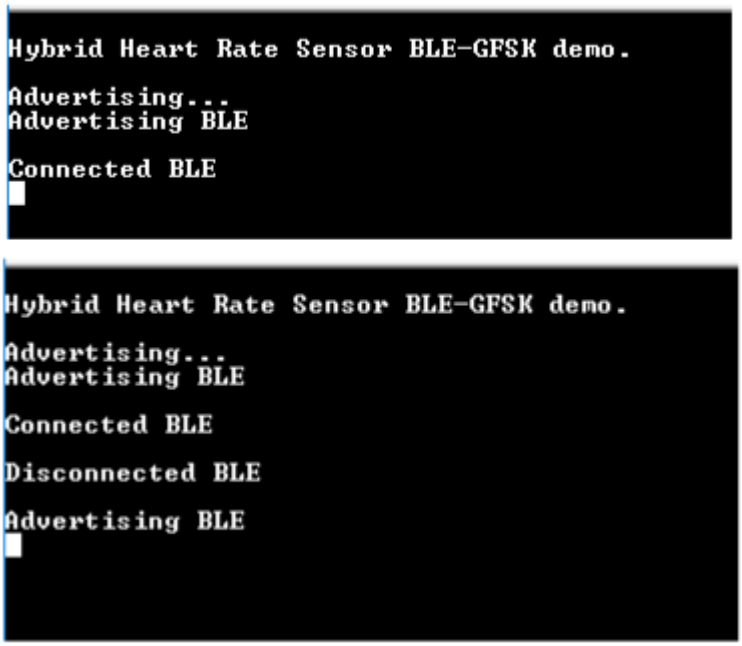

Figure 17. Running the application (step 3)

4. When a GenFSK packet is received, a message as shown in Figure 18 on page 16 appears.

Figure 18. Running the application (step 4)

5.1.3 GenFSK-only device persepective

- Once the GenFSK-only or GenFSK device project is programmed, it starts running, as shown in Figure 19 on page 16.

Figure 19. GenFSK-only device perspective (step 1)

- When a packet is received, it prints a message and then transmits a response packet.

Figure 20. GenFSK-only device perspective (step 2)

This keeps going on until one of the devices stops the GenFSK communication.

At this point, the hybrid feature of the KW36 is shown. The KW36 is sending notifications to a smartphone using Bluetooth LE protocol and it is sending packets through GenFSK protocol. Finally, it is important to mention that the application should always

verify that radio is acquired by the desired protocol to perform its activity. It is common that application could be broken if at some point and the application does not release the radio in the right time. In general, the Bluetooth LE the protocol uses has higher priority due to its link layer which tracks all the timings of the Bluetooth LE events. If the radio is not available for the Bluetooth LE link layer, Bluetooth LE disconnection could be reached as well as an unexpected behavior.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP 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 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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

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

Date of release: June 2019
Document identifier: AN12415

arm