User Manual

DA1468x Software Developer's Guide

UM-B-056

Abstract

This manual intends to assist software developers which implement applications using the DA1468x development platform. A certain degree of reader familiarity with programming environments, debugging tools and software engineering process in general is assumed by the authors.

Contents

Abstract **1**

Contents **2**

Figures **3**

Tables **5**

Codes **5**

1 Terms and definitions **6**

2 References **6**

3 Prerequisites **7**

4 Introduction **8**

5 The Proximity Reporter Application **9**

5.1 Basic Services and Features 9

5.2 User Interface 10

5.3 Importing the project 10

5.4 Project Execution 11

5.4.1 Building 11

5.4.2 Programming the QSPI Flash 12

5.5 Interacting with the application 16

5.5.1 LightBlue iOS application 16

5.5.2 B-BLE Android application 17

5.6 Source code walkthrough 18

6 Peripheral Demo Application **26**

6.1 Basic services and features 26

6.2 User Interface 27

6.3 Importing the project 28

6.3.1 Building the project 28

6.3.2 Programming the QSPI Flash 28

6.4 Interacting with the Application 29

7 Power Measurements Demo Application **33**

7.1 Basic Services and Features 33

7.2 User Interface 36

7.3 Importing the project 36

7.3.1 Building the project 36

7.3.2 Programming the QSPI Flash 37

7.4 Interacting with the Application 38

7.4.1 Controlling via UART2 39

7.4.2 Controlling via GPIO 39

7.4.3 Set advertising interval 39

7.4.4 Set channel map 42

7.4.5 Set recharge period 43

7.4.6 Set connection parameters 45

8 Create a custom application **47**

8.1 Creating a Bluetooth low energy project 47

8.2 Configuring your application 47

 © 2022 Renesas Electronics

8.3 Adding Bluetooth low energy functionality 47
8.3.1 Including BLE header files 47
8.3.2 Adding BLE services 48
8.3.3 Bonding information management 49
8.3.4 Hooks 49
9 Software Upgrade 52
9.1 Software Upgrade Over The Air (SUOTA) 52
9.1.1 Introduction 52
9.1.2 SUOTA service description 52
9.1.3 SUOTA Flow 55
9.1.4 SUOTA Flash memory layout 58
9.1.5 Performing SUOTA upgrade using a mobile phone 59
9.1.6 Performing SUOTA upgrade using two DA1468x 65
9.1.6.1 Building the Bluetooth low energy Central device 66
9.1.6.2 Building the Bluetooth low energy peripheral device 70
9.1.6.3 Running the software upgrade procedure 71
9.1.7 SUOTA in Production and Field deployment 77
9.1.8 Recommendations 78
9.2 Software Upgrade Over USB (SUOUSB) 78
9.2.1 Introduction 78
9.2.2 QSPI based SUOUSB 78
9.2.2.1 Prepare bootloader 78
9.2.2.2 Prepare main image 78
9.2.2.3 Prepare SUOUSB image for test 78
9.2.2.4 Running the SUOUSB process 79
9.2.2.5 Transfer from a Windows host 79
9.2.2.6 Transfer from a Linux host 79
9.2.3 RAM based SUOUSB 80
9.2.3.1 Prepare bootloader 80
9.2.3.2 Prepare main image 80
9.2.3.3 Prepare SUOUSB image for test 80
9.2.3.4 Running the SUOUSB process 81
9.2.4 Use both SUOUSB and SUOTA 81
10 Enabling features on the Proximity Reporter application 82
10.1 Enabling the Charger 82
10.2 Configuration for SUOTA 82
10.2.1 Version header file 82
10.2.2 Code analysis 83
10.2.3 Application start address 85
Revision history 87

Figures

Figure 1: LED D1 on Pro DK DA1468x 9
Figure 2: LED D1 on Basic DK DA1468x 10
Figure 3: Importing “pxp_reporter" into Eclipse 11
Figure 4: Building Project in Release_QSPI mode 12

© 2022 Renesas Electronics

Figure 5: Selecting the build mode 13
Figure 6: External Tool Configurations Menu 14
Figure 7: QSPI programming configuration 14
Figure 8: Changing existing QSPI programming configuration 15
Figure 9: Configuring COM Port 15
Figure 10: Configure and start Debug perspective 16
Figure 11: LightBlue application connected to Proximity Reporter application 17
Figure 12: B-BLE in Play Store 17
Figure 13: B-BLE application connected to Proximity Reporter application 18
Figure 14: pxp_reporter project structure 19
Figure 15: FTDI cable connected to UART on Basic DK with no flow control 26
Figure 16: Connecting USB-UART cable to UART2 27
Figure 17: peripherals_demo – system overview 27
Figure 18: Building Project in Release_QSPI mode 28
Figure 19: Selecting the build mode to be programmed 29
Figure 20: Program the QSPI Flash 29
Figure 21: Configuration of the Serial Terminal 30
Figure 22: Enable local echo in TeraTerm 30
Figure 23: Output in serial terminal 31
Figure 24: Jumper settings for UART2 and GPIO configuration ProDK Virtual COM port with Flow Control 34
Figure 25: Connect USB-UART cable to ProDK Breakout Headers with Flow Control 35
Figure 26: Connect USB-UART cable to BasicDK Breakout Headers with Flow Control 36
Figure 27: Building Project in Release_QSPI mode 37
Figure 28: Selecting the build mode to be programmed 37
Figure 29: External Tools 38
Figure 30: Set advertising interval via CLI 40
Figure 31: Set Advertising Interval via GPIOs 41
Figure 32: Power Profiler output for the second configuration index 41
Figure 33: Set advertising channel map via CLI 42
Figure 34: Three advertising channels 43
Figure 35: Two advertising channels 43
Figure 36: Set recharge period via CLI 44
Figure 37: `SLEEP_TIMER_REGISTER` contents 45
Figure 38: Connection parameters update 46
Figure 39: Sleep current measurement 46
Figure 40: Advertising 51
Figure 41: Connected 51
Figure 42: BLE SUOTA loader 56
Figure 43: BLE SUOTA Service 57
Figure 44: Flash memory partition layout comparison between SUOTA and non-SUOTA build (1Mbyte QSPI Flash) 58
Figure 45: Run mkimage.bat script on Windows 60
Figure 46: Run mkimage.sh on Linux 61
Figure 47: Project directory 61
Figure 48: Scripts 62
Figure 49: Device selection 62
Figure 50: Update device 63
Figure 51: Image file 63
Figure 52: Parameter settings (ignore for DA1468x) 64
Figure 53: Uploading image and reboot 65
Figure 54: When file upload is finished, press "Close" 65
Figure 55: Dual SUOTA architecture 66
Figure 56: Building `ble_suota_client` 67
Figure 57: Flash the `ble_suota_client` binary to the QSPI Flash 67
Figure 58: Selecting the target device 67
Figure 59: Building pxp_reporter for SUOTA 68
Figure 60: Creating image 68
Figure 61: Jumpers only on Rx and Tx for ProDK no Flow Control 69

© 2022 Renesas Electronics

Figure 62: Uploading image to the Client 69
Figure 63 : Building ble_suota_loader project 70
Figure 64: Flashing ble_suota_loader to QSPI Flash 70
Figure 65: Selecting the target device 71
Figure 66: Jumpers on Rx, Tx and CTS 71
Figure 67: Specifying the serial port number 72
Figure 68: Configuring the serial port 72
Figure 69: Information regarding the image stored in the NVMS_BIN_PART partition are displayed during boot 72
Figure 70: Specifying the serial port number 73
Figure 71: Configuring the serial port 73
Figure 72: ble_suota_loader information is displayed during boot 74
Figure 73: Scanning for available devices 74
Figure 74: Stop scanning procedure 75
Figure 75: Connecting to loader device 75
Figure 76: Updating with new image the loader device 76
Figure 77: Transfer complete 76
Figure 78: Rebooting and loading image 77
Figure 79: Verifying that loader is running PX Reporter 77

Tables

Table 1 : Build configuration Pattern 11
Table 2 : Components need for the "pxp_reporter" project 20
Table 3: UART settings 39
Table 4: GPIO configuration type 39
Table 5: GPIO configuration index 39
Table 6: Advertising interval settings 40
Table 7: Channel map settings 42
Table 8: Recharge period settings 44
Table 9: Connection parameters settings 45
Table 10 : Dialog BLE API header files 47
Table 11 : BLE service API header files 48
Table 12: Macros for the configuration of the hook functions 49
Table 13: Notification bit masks 50
Table 14: SUOTA service characteristics 52
Table 15: Product header description 59
Table 16: Image header description 59

Codes

Code 1: Create the Proximity Reporter application task 21
Code 2: Initializing and configuring the BLE 21
Code 3: Configure Device Name 21
Code 4: Immediate Alert and Link Loss Services 22
Code 5: Set up device to start advertising 23
Code 6: Handling events 24
Code 7: Configuration for additional hardware 32
Code 8: Enable UART and/or GPIO configuration 38
Code 9: Flash memory partition table 59
Code 10: Charger configuration 82
Code 11: sw_version.h 83
Code 12: Macro to enable SUOTA 83
Code 13: Defining SUOTA version and L2CAP COC PSM 83
Code 14: Passing L2CAP events to the SUOTA service 83
Code 15: Declare SUOTA variable 84
Code 16: Register SUOTA and DIS 84

 © 2022 Renesas Electronics

Code 17: DIS data 84
Code 18: Header files 84
Code 19: Advertising and scan response data 85
Code 20: Code Base Address 85
Code 21: Set starting address 86

1 Terms and definitions

ADC	Analog to Digital Converter
API	Application Programming Interface
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
BSP	Board Support Package
CCC	Client Characteristic Configuration
COC	Connection Oriented Channels
DIS	Device Information Service
DK	Development Kit
FreeRTOS	Free Real Time Operating System
ISR	Interrupt Service Routine
LED	Light Emitting Diode
L2CAP	Logical Link Control and Adaptation Protocol
MTU	Maximum Transmission Unit
NVMS	Non-Volatile Memory Storage
OS	Operating System
OSAL	Operating System Abstraction Layer
PM	Proximity Monitor
PR	Proximity Reporter
SDK	Software Development Kit
SUOTA	Software Update Over The Air
SW	Software
QSPI	Queued Serial Peripheral Interface
ROM	Read Only Memory
RTOS	Real Time Operating System
GATT	Generic Attribute Profile
GAP	Generic Access Profile
UUID	Universally Unique Identifier

2 References

[1] DA14680, Datasheet, Dialog Semiconductor.
[2] UM-B-057-SmartSnippets™ Studio user guide, User manual, Dialog Semiconductor.
[3] UM-B-047 DA1468x Getting Started, User manual, Dialog Semiconductor.
[4] UM-B-044 DA1468x Software Platform Reference, User manual, Dialog Semiconductor.
[5] UM-B-060-DA1468x_DA1510x Development kit – Pro, User manual, Dialog Semiconductor.
[6] UM-B-066-DA1468x_DA1510x Development kit – Basic, User manual, Dialog Semiconductor.
[7] UM-B-083 SmartSnippets™ Toolbox, User manual, Dialog Semiconductor.

 © 2022 Renesas Electronics

3 Prerequisites

- SmartSnippets™ Studio package
- Dialog's Semiconductor SmartSnippets™ DA1468x SDK
- Operating System (Windows or Linux)
- Pro DA1468x and accessories or Basic DA1468x Development Kits (DK)
- Android or iOS mobile phone
- SUOTA Dialog Application for mobile
- A PC terminal application, for example Tera Term (download at https://ttssh2.osdn.jp/index.html.en)

© 2022 Renesas Electronics

4 Introduction

This document provides an overview of the SmartSnippets™ Software Development Kit (SDK) used for application development using the DA1468x chipset devices and boards. The SmartSnippets™ DA1468x SDK includes a set of libraries, example projects, drivers and middleware modules which facilitate the creation of complex applications by fully exploiting the provided hardware resources of the connected DA1468x device.

The SmartSnippets™ DA1468x SDK provides all necessary programming tools, libraries, APIs, resources and access to device features that a developer is likely to use for implementing a software application. The main features of the SDK are:

- Preemptive multitasking using the freeRTOS real time operating system (www.freeRTOS.org).
- Access to the on-chip peripherals via Low Level Drivers and Adapters which allow multiple tasks to share peripherals.
- Seamlessly integrating a version 4.2 compliant Bluetooth® low energy stack and radio.
- Firmware updates, including the novel Software Upgrade Over The Air (SUOTA) process.
- Structured access to the Flash memory device via a Non-Volatile Memory Storage (NVMS) adapter that supports wear levelling.
- Support of the on-chip power management facilities enabling sleep and hibernation functionality.
- On-chip charger integration.
- OS-aware watchdog service.

In order to successfully run the applications and examples included in this guide, users must have already completed the installation of all necessary software described in UM-B-057-SmartSnippets TM Studio user guide [2].

The SmartSnippets™ DA1468x SDK supports two Development Kits – the Pro [5] and the Basic [6] DK which share many features:

- 1Mbyte QSPI Flash
- Breakout Headers that support daughter boards such as the Sensor Board.
- Virtual COM port exposed over USB (only works correctly on Pro DK)
- Onboard J-Link debugger

In addition to these shared features the Pro DK also supports charging and power profiling using onboard current measurement circuitry.

In this document they will both be referred to as Development Kit (DK) and if functionality is only available on the Pro DK this will be noted.

 © 2022 Renesas Electronics

5 The Proximity Reporter Application

The Proximity Reporter application is an implementation of the adopted GATT Proximity Profile (PXP). More details can be found at https://www.bluetooth.com/specifications/gatt. It is designed to monitor any change to the physical proximity of two connected Bluetooth low energy devices using the established communication channel between them. The Proximity Profile defines the behavior of any Bluetooth device when it moves relative to a peer node to trigger an alert to the user.

There are two cases that can be identified by the Profile

- Two peers are further apart as the connection has dropped or the signal loss has increased
- Two peers are closer together as the connection has been established or the signal loss has decreased

The Proximity profile defines two roles:

- `Proximity Monitor (PM)` which is a Generic Attribute Profile (GATT) client.
- `Proximity Reporter (PR)` which is a Generic Attribute Profile (GATT) server.

This section shall describe in detail only the Proximity Reporter entity.

5.1 Basic Services and Features

The Proximity Reporter Application supports the following services each marked with an adopted Universally Unique Identifier (`UUID`).

- Immediate Alert service (`UUID 0x1802`).
- Link Loss service (`UUIID 0x1803`).
- Tx Power service (`UUID 0x1804`).
- Battery service (`UUID 0x180F`).

The Proximity Reporter application provides the following set of features:

- Two levels of alert indications, marked as "Low" and "High" which respectively flash white LED1 either slowly or fast. The position of White LED D1 is shown in Figure 1 for Pro DK and Figure 2 for Basic DK.

Figure 1: LED D1 on Pro DK DA1468x

© 2022 Renesas Electronics

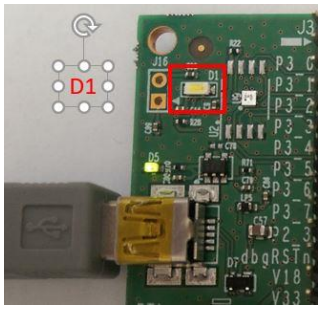


Figure 2: LED D1 on Basic DK DA1468x

- Two levels of advertising interval, a "fast" one (approx. 20-30ms) for the first 45 seconds of operation and a "reduced power" one (approx.1/1.5s) after the first 45 seconds until a connection has been established.
- Extended Sleep mode.
- Pairing/bonding/encryption.

The Proximity Reporter functionality is implemented in the `pxp_reporter_task.c` source file.

5.2 User Interface

For user notification purposes the application shall use the white LED D1 presented in Figure 1 or Figure 2 when either link loss or immediate alerts are triggered. The Alert Notifications are:

- **High level alert:** A fast blinking white LED D1.
- **Low level alert:** A slow blinking white LED D1.

5.3 Importing the project

The first step is importing the project into the Project Explorer of SmartSnippets™ Studio from the folder `<sdk_root_directory>\projects\dk_apps\demos\`

Start SmartSnippets™ Studio by double clicking the icon located in the Desktop.

Go to **File > Import > General > Existing Projects into Workspace** and click **Next**.

1. Find the folder that contains the project and click "OK" (the project location is shown in Figure 3, Reference Point 1).
2. Tick the **pxp_reporter** (Figure 3, Reference Point 2).
3. Click **Finish** (Figure 3, Reference Point 3).

© 2022 Renesas Electronics

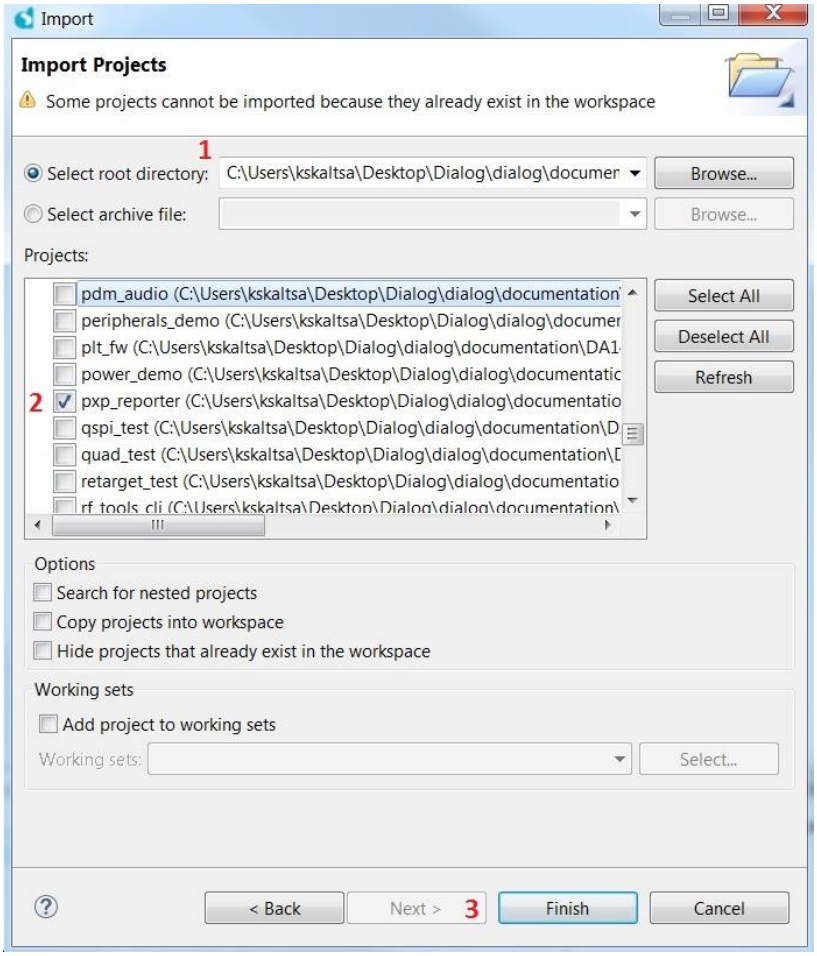


Figure 3: Importing "pxp_reporter" into Eclipse

5.4 Project Execution

5.4.1 Building

The Proximity Reporter Application project, like all other Bluetooth low energy projects in the SmartSnippets™ SDK, comes with a built-in configuration for QSPI cached execution mode. The project can be built with either the `Debug_QSPI` or the `Release_QSPI` configuration to generate the binary which is programmed into the device's QSPI Flash memory. This is done by selecting the project and clicking on the Build button (Hammer icon) on the SmartSnippets™ Studio toolbar as shown in Figure 4 or by right clicking on the project's name and select Build Project from the pop-up menu. The build configuration options are listed in Table 1. At this point recommended build is `DA14681-01-Release_QSPI`.

Table 1 : Build configuration Pattern

Device	Version	Build Configuration Type	Build configuration name
DA14680/1	01	Debug_QSPI	DA14681-01-Debug_QSPI
DA14680/1	01	Debug_QSPI with SUOTA	DA14681-01-Debug_QSPI_SUOTA
DA14680/1	01	Release_QSPI	DA14681-01-Release_QSPI
DA14680/1	01	Release_QSPI with SUOTA	DA14681-01-Release_QSPI_SUOTA
DA14683	00	Debug_QSPI	DA14683-00-Debug_QSPI

© 2022 Renesas Electronics

Device	Version	Build Configuration Type	Build configuration name
DA14683	00	Debug_QSPI with SUOTA	DA14683-00-Debug_QSPI_SUOTA
DA14683	00	Release_QSPI	DA14683-00-Release_QSPI
DA14683	00	Release_QSPI with SUOTA	DA14683-00-Release_QSPI_SUOTA

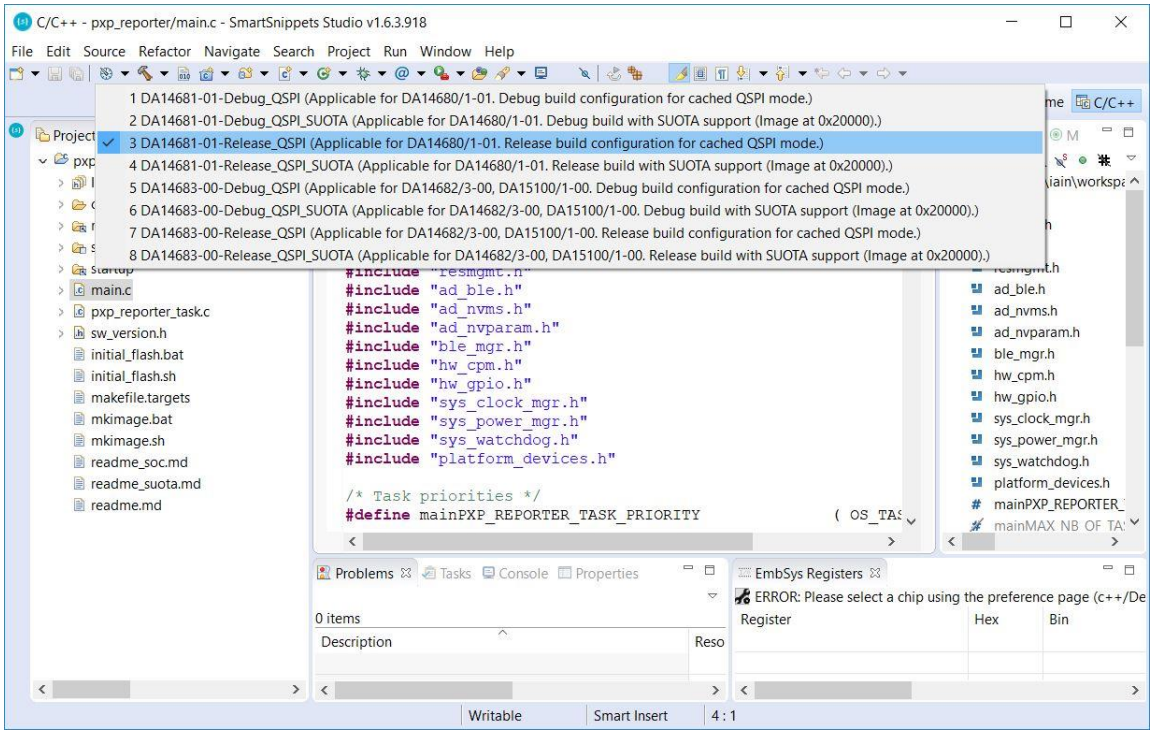


Figure 4: Building Project in Release_QSPI mode

After the build process is completed the generated binary must be written to the QSPI Flash memory. The scripts to write to the QSPI Flash are contained in the project `scripts` which must first be imported. This is done in the same way as the `pxp_reporter` project was previously imported. The scripts project is found at:

`<sdk_root_directory>\utilities\scripts`

If after importing the scripts they are not visible, expand the **Run > External tools** drop-down, click on **Organize Favourites**, click **Add** and then click **Select all > OK > OK**.

The scripts are needed so that the user does not have to manually enter commands through a command line interface. Scripts for preparing the binary image and writing to flash memory are provided.

5.4.2 Programming the QSPI Flash

After the build process is completed, one of the provided `program_QSPI` script files may be used to program the Proximity Reporter binary to the QSPI Flash memory.

There are several `program_QSPI` scripts supplied with the following naming format

`program_QSPI_<transport>_<hostOS>`

- where <transport> can be either
  - `jtag` –SWD link is used to transfer the binary image to the target device which then writes it to QSPI flash. The name JTAG is used as a generic term even although actual link is SWD.

© 2022 Renesas Electronics

- `serial` – serial link is used to transfer the binary image to the target device using the ROM bootloader which then writes it to QSPI flash

- and `<hostOS>` can be either
  - `win` – for running on a Windows host, or
  - `linux` – for running on a Linux host

More details on which script and how to use it can be found in the [2], Section 9. However, the basic steps are as follows for a Windows host (use `_linux` for a Linux host)

4. Select the project folder and click on the icon as shown Figure 5 and make active the desired build configuration to produce the binary to be programmed to the QSPI Flash memory. The programming script uses the binary from the selected build.

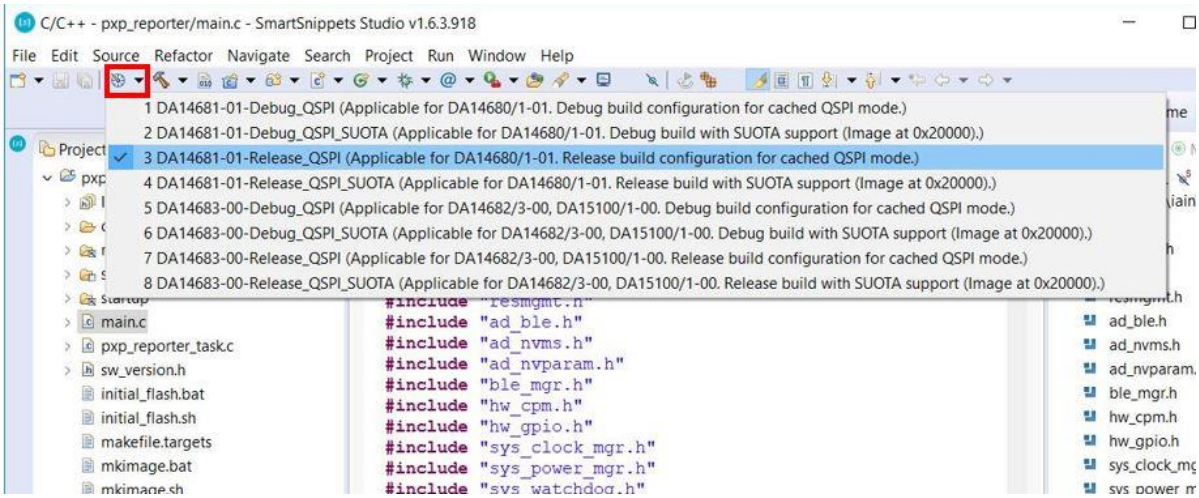


Figure 5: Selecting the build mode

5. Select the External Tool Configurations button (highlighted in Figure 6), choose the appropriate script file (in this example, the `program_qspi_serial_win` script is used) to program the QSPI Flash memory and select Run (alternatively, click **Run > External Tools > program_qspi_serial_win**).

© 2022 Renesas Electronics

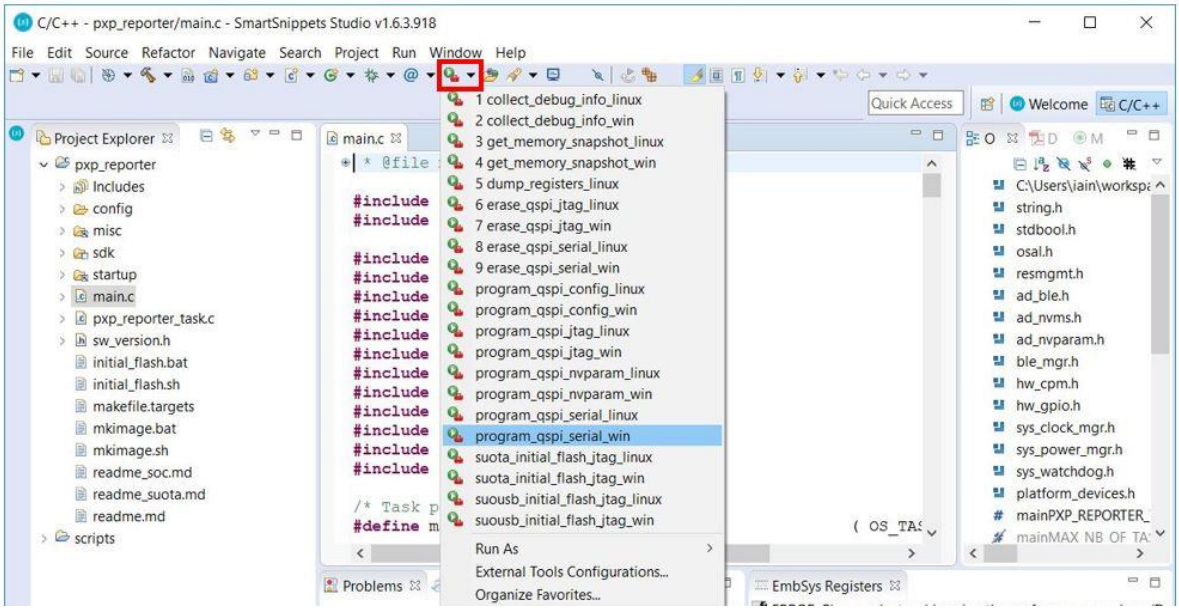


Figure 6: External Tool Configurations Menu

When executed for the first time the user is asked to configure the QSPI header. The configuration dialog is shown in Figure 7. For the DA14681 enter 0.

Figure 7: QSPI programming configuration

After completing the procedure, the QSPI configuration is saved and subsequently used by the program script. The user may change the configuration at any time by running the script `program_qspi_config_win`. Running this script shows the current configuration in the console window and then provide the option to change it as shown in Figure 8.

© 2022 Renesas Electronics

```
Problems  Tasks  Console  Properties
<terminated> program_qspi_config_win [Program] C:\Users\iain\workspace_SmartSnippets_Studio-doc\utilities\scripts\qspi\program_qspi_config.bat
.....................................................................................................
..
.. QSPI PROGRAMMING CONFIGURATION
..
.....................................................................................................

Existing configuration:
-----------------------------
PRODUCT_ID=DA14681-01
-----------------------------


Change existing configuration? (y/n or [ENTER] for n)
-> y

Product id options:
       0:  DA14680/1-01
       1:  DA14682/3-00, DA15XXX-00 (default)
Product id ?  (0..1 or [ENTER] for 1)
-> 0

PRODUCT_ID=DA14681-01


.....................................................................................................
..
.. CONFIGURATION FINISHED
..
.....................................................................................................
```

Figure 8: Changing existing QSPI programming configuration

The virtual[1]COM port currently assigned to the DA1468x device should be entered for the QSPI Flash programming to run as shown in Figure 9. In addition, a board reset may be necessary. Once programming is completed, the Proximity application may be run by simply resetting the device.

```
.....................................................................................................
..
.. QSPI PROGRAMMING
..
.....................................................................................................

Please enter your COM port number and press enter.
-> 13

COMPORT=COM13

C:\Users\iain\workspace_SmartSnippets_Studio-doc\utilities\scripts\qspi>CALL "..\..\..\binaries\cli_programmer.exe" --prod-i
cli_programmer 1.24
Copyright (c) 2015-2017 Dialog Semiconductor

Using serial port COM13 at baud rate 57600.
bootloader file not specified, using internal uartboot.bin

Connecting to device...
Writing to address: 0x00000000 offset: 0x00000000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00002000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00004000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00006000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00008000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x0000a000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x0000c000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x0000e000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00010000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00012000 chunk size: 0x00002000
Writing to address: 0x00000000 offset: 0x00014000 chunk size: 0x00000298
Writing to address: 0x00000000 offset: 0x00000000 chunk size: 0x00000002
done.

.....................................................................................................
..
.. FINISHED
..
```

Figure 9: Configuring COM Port

[1] For instruction on finding the virtual COM port, please check [3].

© 2022 Renesas Electronics

Now that the QSPI flash is programmed there are two ways to run the application.

1. Press the K2 RESET button on the DK board.
2. Start the debugger and run the application in it. To do this the project must be built in DA14681-01-Debug_QSPI mode. As SmartSnippets TM Studio is an Eclipse based tool the Debug view is shown a different perspective. The Debug perspective is started as follows (Figure 10):

Run > Debug Configurations > Smartbond "SmartSnippets DA1468x SDK" via J-Link GDB Server > QSPI > Debug .

This starts the debug perspective and load the symbols for the selected project from the bin file. The application can then be run using the Run button (F5).

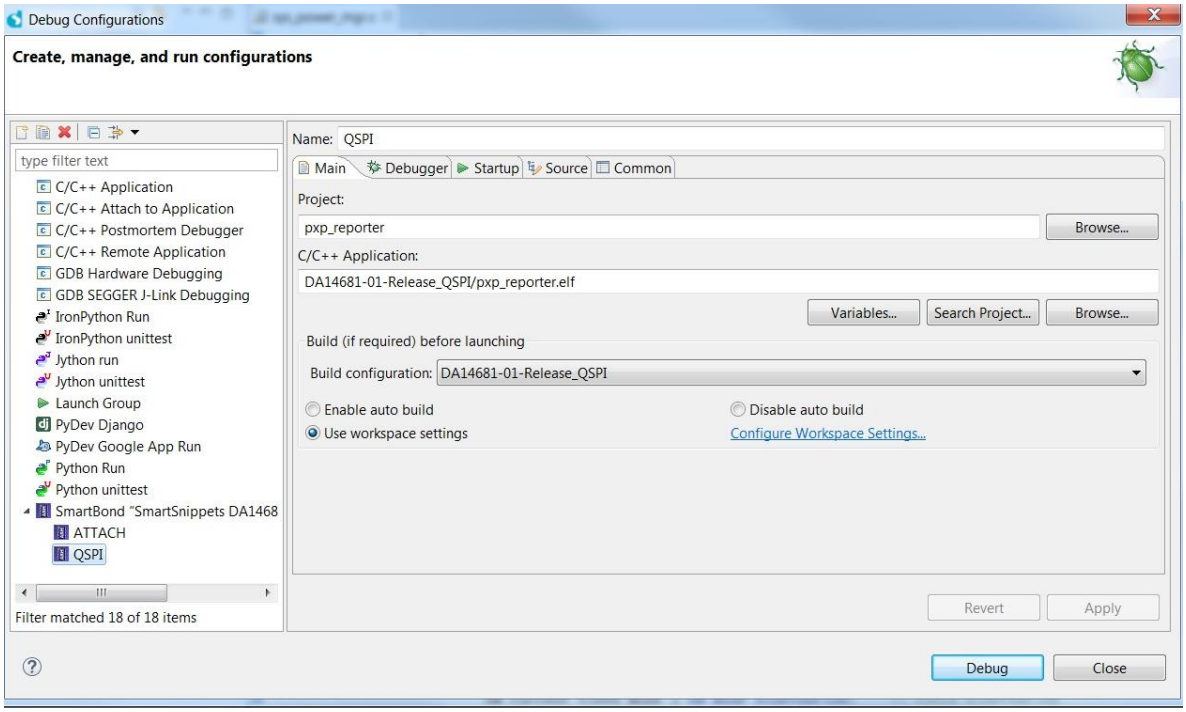


Figure 10: Configure and start Debug perspective

When the application is running the device starts advertising and will be visible to all Bluetooth low energy devices that have scanning capabilities.

5.5 Interacting with the application

Now the project is running on the DA1468x device it is possible to interact with it via a mobile phone (the mobile phone must support Bluetooth low energy). Applications that can communicate with the DA1468x device are presented in the next two sections.

Note 1 There many mobile applications for Android and iOS that can interact with the Proximity Reporter firmware. The two listed are just examples.

5.5.1 LightBlue iOS application

The LightBlue iOS application can be used to connect an iOS device to the Proximity Reporter application. In such a case, the iOS device acts as a Bluetooth low energy central device and the application as a Bluetooth low energy peripheral device. Figure 11 shows LightBlue's display after it has connected to a Dialog PX Reporter device. LightBlue can be downloaded from Apple App Store.

© 2022 Renesas Electronics

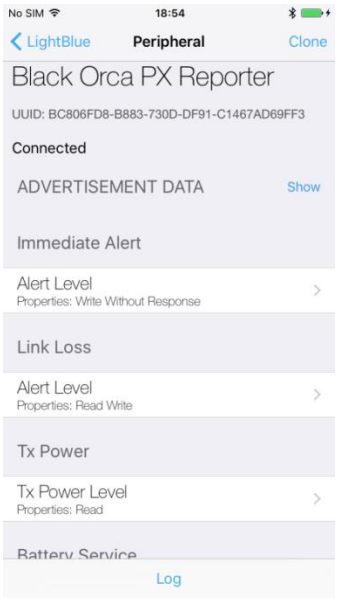


Figure 11: LightBlue application connected to Proximity Reporter application

The LightBlue application can be used to change the values of the Immediate Alert or the Link Loss Alert level attributes. These trigger the appropriate `Proximity Reporter` response.

In the Immediate Alert case, writing a hex value of `0x02` triggers a fast flashing white LED D1 (high alert), while writing a hex value of `0x01` triggers a slow flashing white LED D1 (low alert). To turn off a triggered alert, simply write the hex value of `0x00` in the application.

Writing these values to the Link Loss Alert Level attribute section does not trigger the Link Loss Alert immediately. It only triggers the when a link loss subsequently occurs. A link loss is not triggered when one of the devices disconnects. It is triggered by the devices moving apart and the signal strength deteriorating enough.

5.5.2 B-BLE Android application

The B-BLE Android application is an application similar to LightBlue iOS application described in the previous paragraph, but for the Android mobile OS. The application is free of charge and can be downloaded from Google Play Store.

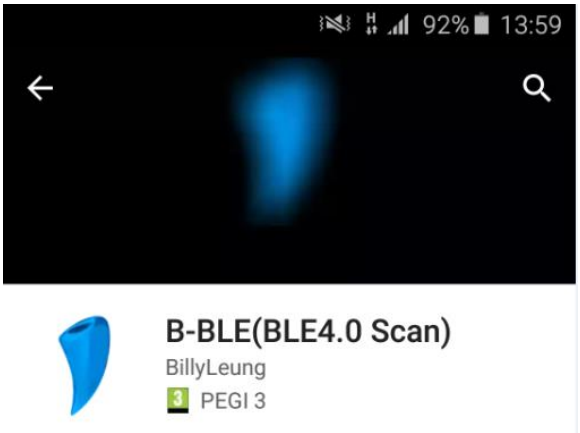


Figure 12: B-BLE in Play Store

Figure 13 shows B-BLE display after it has established connection with a Dialog PX Reporter device.

© 2022 Renesas Electronics

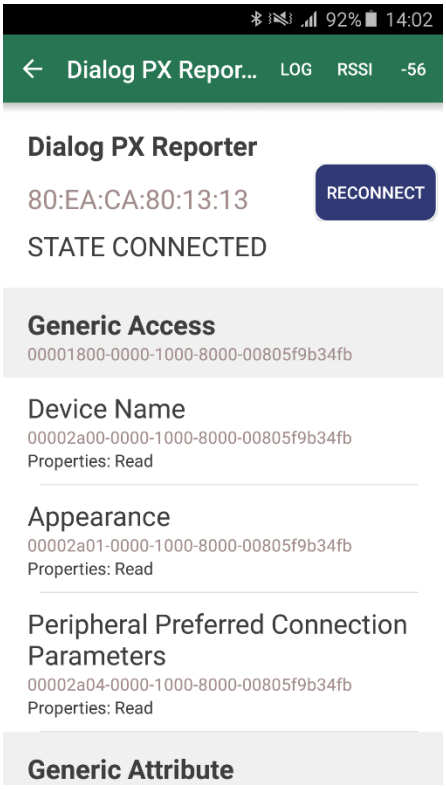


Figure 13: B-BLE application connected to Proximity Reporter application

The application can be used to read or write the available service attributes in the same manner as LightBlue application or any other Bluetooth low energy app.

5.6 Source code walkthrough

Figure 14 shows the `pxp_reporter` folder structure with an emphasis on the project configuration files. More info about the SmartSnippets™ DA1468x SDK structure can be found in the Software Platform Reference manual [4].

© 2022 Renesas Electronics

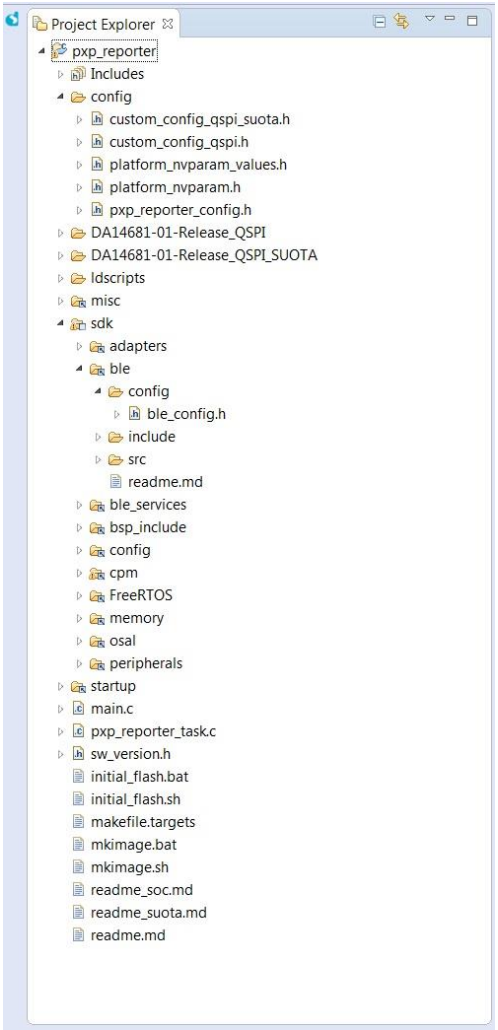


Figure 14: pxp_reporter project structure

To customize the project configuration the developer needs to change the parameters in just the following three files:

- `config/custom_config_qspi.h`: All project specific configuration options are defined in this file. They cover the system clocks, the execution mode, the minimum sleep time, the charging functionality, the total heap size, etc.
- `config/pxp_reporter_config.h`: Can be used to configure the default name of the device.

Note 2 If the user has modified the `BD_ADDR` of the device via `platform_nvparam_values.h`, then the device shall use as its advertising name the one which is configured in `platform_nvparam_values.h`

- `ldscripts/mem.ld.h`: Can be used to configure the sizes of the different memory sections. This file is used when the project is built to generate the `mem.ld` file needed by the linker.

The `misc` folder contains the ROM symbol definition file (`da14681_01_rom.symbols`) for the Dialog software; it is included in all projects that require linking to ROM functions and variables.

The project's `sdk` linked folder contains the SmartSnippets™ DA1468x SDK components needed for the `pxp_reporter` project. The components included are the following:

© 2022 Renesas Electronics

Table 2 : Components need for the "pxp_reporter" project

Components	Description
Adapters	Peripheral adapters used by the system and the Bluetooth low energy software for access to the Non-Volatile Memory Storage (NVMS), the Flash memory and the radio.
Ble	The Bluetooth low energy framework used by the pxp_reporter application. This includes the BLE API, the BLE manager and the BLE adapter header and source code files, as well as the BLE stack API header files.
ble_services	The BLE services framework used to implement the BLE services used by the Proximity Reporter application.
bsp_include	BSP-related files. Board Support Package (BSP).
Config	BSP default configuration files – not to be modified.
Cpm	Clock and Power Manager.
FreeRTOS	The FreeRTOS operating system.
Memory	The Non-Volatile Memory Storage (NVMS) implementation.
Osal	The Operating System Abstraction Layer.
peripherals	The peripherals low-level drivers.

The linked folder `startup` contains the necessary files for the initialization of the system.

- The main application logic is implemented in the following files, `main.c` and the `pxp_reporter_task.c`. The system's initialization is implemented in `main.c`, which contains both `main()` and `system_init()` functions, while the main Proximity Reporter application functionality is implemented in `pxp_reporter_task.c`, which contains the Proximity Reporter task function, called `pxp_reporter_task()`.

The `main()` function in most projects only accomplishes the following basic actions:

1. Performs a basic initialization of the system's clock.
2. Creates the system initialization task.
3. Starts the FreeRTOS scheduler.

The system initialization task is implemented by function `system_init()` in file `main.c`. When the scheduler is started, this is the only task created and it is run by the scheduler.

The `system_init()` function initializes all other required components and creates all other required tasks, including the application task that implements the Proximity Reporter. In particular, `system_init()` does the following:

1. Initializes the system clocks that were not initialized by the `cm_clk_init_low_level()`.
2. Initializes the watchdog service (which is the mechanism that is used to detect and recover from an unexpected execution sequence).
3. Initializes the power manager and configures the sleep mode used by the project.
4. Configures the system clock (`cm_sys_clk_set(sysclk_XTAL16M)`).
5. Initializes adapter and manager components, as needed:
   a. The NVMS adapter (`ad_nvms_init()`).
   b. The BLE adapter task, which implements the interface to the BLE stack (`ad_ble_init()`).
   c. The BLE manager task, which implements the BLE framework (`ble_mgr_init()`).
6. Creates the Proximity Reporter application task, using OSAL macro `OS_TASK_CREATE()`:

© 2022 Renesas Electronics

```
/* Start the PXP reporter application task. */
OS_TASK_CREATE("PXP Reporter",    /* The text name assigned to the task, for
                                  debug only; not used by the kernel. */
          pxp_reporter_task,      /* The function that implements the task. */
                       NULL,      /* The parameter passed to the task. */
   #if (dg_configDISABLE_BACKGROUND_FLASH_OPS == 1)
                            512,   /* The number of bytes to allocate to the
   stack of the task. */
   #else
                            756,   /* The number of bytes to allocate to the
   stack of the task. */
   #endif
mainPXP_REPORTER_TASK_PRIORITY, /* The priority assigned to the task. */
                        handle);
```

Code 1: Create the Proximity Reporter application task

7. Finally, after having completed its operation, the system initialization task deletes itself (`OS_TASK_DELETE(OS_GET_CURRENT_TASK())`). From this point on, the scheduler schedules all tasks created by the system initialization task, which was given the highest priority to prevent it being interrupted during the initialization process.

The next sections focus on the Proximity Reporter application task and the way it interfaces with the BLE manager using the Dialog BLE API and the BLE services framework.

The function that implements the Proximity Reporter task is `pxp_reporter_task()` which is located in `pxp_reporter_task.c`. The following Dialog API calls are used to initialize and configure the BLE subsystem, just before the main application infinite loop.

```
ble_peripheral_start();

ble_register_app();
```

Code 2: Initializing and configuring the BLE

The `ble_peripheral_start()` function enables the Bluetooth low energy functionality and configures the device with the peripheral role. The application registers itself for task notifications from the BLE manager when an event is posted on the BLE event queue using the `ble_register_app()` function. This is only necessary when the application task relies on the RTOS task notifications mechanism to unblock, otherwise it can just block on the BLE manager's event queue.

The Device Name for the particular project is configured in `pxp_reporter_config.h`. This header file is located in the config folder. Other projects may use different files to store the Device Name configuration.

```
/* Name can not be longer than 29 bytes (BLE_SCAN_RSP_LEN_MAX - 2 bytes)*/

#define PX_REPORTER_DEFAULT_NAME        "Dialog PX Reporter"
```

Code 3: Configure Device Name

The application uses the BLE services API to add the services needed for the Proximity Reporter role. There are two steps when adding a service to an application:

1. The service has to be initialized and added to the BLE stack database. This is done using the service's initialization function. If the service requires specific initialization information, like callback functions for certain events the application should respond to, or values that should have an initial value, these are passed as arguments to the initialization function.
2. If the service defines certain events that need to be processed in its context (for example, writing of an attribute whose handle belongs to the service handle range), the service has to be registered to the BLE service framework. This is done using `ble_service_add()` and the service handle is passed as an argument (in SDK release 1.0.10 and later this is seamlessly done by the service initialization function and there is no need to use `ble_service_add()`).

© 2022 Renesas Electronics

The following code snippet from the Proximity Reporter task is used to register the required services to the BLE service framework.

```
/*
 * Register IAS and LLS
 *
 * Both IAS and LLS instances should be registered in ble_service framework in
order
 * for events inside service to be processed properly.
 */
        svc = ias_init(ias_alert_cb);
        svc = lls_init(lls_alert_cb);


/*
 * Register TPS
 *
 * TPS doesn't contain any dynamic data thus it doesn't need to be registered
in
 * ble_service framework (but it's not an error to do so). For now we have
output
 * power set to 0 dBm.
 */
        tx_power_level = 0;
        tps_init(tx_power_level);
/*
 * Register BAS
 *
 * BAS is not included in PXP, but it can be so PXP monitor can also monitor
out
 * battery level. This service should also be registered in ble_service
framework.
 */
        bas = bas_init(NULL, NULL);
```

Code 4: Immediate Alert and Link Loss Services

For the Immediate Alert and Link Loss Services (IAS and LLS respectively), the corresponding callback functions are passed as arguments. These callbacks shall be called respectively by the BLE service framework if a peer device modifies the Immediate Alert or the Link Loss Alert attributes. These callback functions are defined in the same file, `pxp_reporter_task.c`.

After been initialized using `ias_init()` and `lls_init()`, the services are registered to the BLE service framework using the `ble_service_add()` function (in SDK release 1.0.10 and later this is seamlessly done by the service initialization function and there is no need to explicitly call `ble_service_add()`). The Tx Power service is initialized using `tps_init()` with the argument setting the Tx power level used by the system's RF. The Battery Service is initialized using `bas_init()` and

© 2022 Renesas Electronics

added to the service framework. Then, `bas_set_level()` is called to set the default battery level exposed by the service attribute.

The application then creates two software timers using the `OS_TIMER_START()` function, one to disable the Link Loss Alert, 15 seconds after the actual link loss event, and a second one to trigger the switch from "Fast" to "Slow" advertising interval.

The last operations before entering the `for(;;)` loop set up the device to start advertising:

```
/*
   * Set advertising data and scan response, then start advertising.
   *
   * By default, interval values are set to "fast connect" and timer is started
to
   * change them after
   */
   set_advertising_interval(ADV_INTERVAL_FAST);
   ble_gap_adv_data_set(sizeof(adv_data), adv_data, name_len + 2, scan_rsp);
   ble_gap_adv_start(GAP_CONN_MODE_UNDIRECTED);
   OS_TIMER_START(adv_tim, OS_TIMER_FOREVER);
```

Code 5: Set up device to start advertising

The advertising interval is configured using `set_advertising_interval()`, which is also defined in `pxp_reporter_task.c` and eventually calls `ble_gap_adv_intv_set()` to set the desired interval. The advertising data is defined in `adv_data` array and set using `ble_gap_adv_data_set()`. Finally, `ble_gap_adv_start()` is called to start an undirected connectable advertising air operation.

At this point the device is properly configured as a peripheral, having an initialized attribute database and has started advertising. After starting the timer needed to switch to the "reduced power" advertising after 45 seconds, the task now enters the main loop that defines its lifetime behavior and responsiveness to BLE events.

Immediately after entering the `for(;;)` loop, the task notifies the system watchdog, suspends it and then blocks on its task notification value. Task notification is an integral FreeRTOS mechanism (http://www.freertos.org/RTOS-task-notification-API.html) which allows tasks to handle multiple events, such as queues, interrupts, semaphores, etc. Calling `OS_TASK_NOTIFY_WAIT()` results in the calling task being blocked as it waits to receive a task notification on one of the specified notification bits. Each bit of the 32-bit task notification value represents a possible cause to unblock; the least significant bit is always assigned to notifications from the BLE manager's event queue. For the `pxp_reporter` application, two additional notification bits are defined at the start of the `pxp_reporter_task.c` file and they are assigned to the time-out interrupts of the two software timers used by the application.

The task is unblocked when the notification value of the task has been modified to be other than zero. The rest of the `for(;;)` loop handles the event that caused the task to unblock. The most common source would be a BLE event posted at the BLE manager's event queue. The following code snippet presents a method of handling similar events.

```
/*
 * First, application needs to try pass event through ble_framework.
 * Then it can handle it itself and finally pass to default event handler.
 */
      if (!ble_service_handle_event(hdr)) {
```

 © 2022 Renesas Electronics

```
            switch (hdr->evt_code) {

            case BLE_EVT_GAP_CONNECTED:
handle_evt_gap_connected((ble_evt_gap_connected_t *) hdr);

                  break;

            case BLE_EVT_GAP_DISCONNECTED:
handle_evt_gap_disconnected((ble_evt_gap_disconnected_t *) hdr);

                  break;

            case BLE_EVT_GAP_ADV_COMPLETED:
handle_evt_gap_adv_completed((ble_evt_gap_adv_completed_t *) hdr);

                  break;

             case BLE_EVT_GAP_PAIR_REQ:
handle_evt_gap_pair_req((ble_evt_gap_pair_req_t *) hdr);

                  break;


#if dg_configSUOTA_SUPPORT && defined (SUOTA_PSM)
            case BLE_EVT_L2CAP_CONNECTED:
            case BLE_EVT_L2CAP_DISCONNECTED:
            case BLE_EVT_L2CAP_DATA_IND:
                  suota_l2cap_event(suota, hdr);
                  break;
#endif
            default:
                   ble_handle_event_default(hdr);
                   break;
                              }
                        }
```

Code 6: Handling events

After unblocking and verifying that the notification source is related to the BLE manager's event queue, the application task utilizes `ble_get_event()` function to retrieve the actual BLE event from that queue. If the event header is valid then the event is handled appropriately.

The Proximity Reporter application which is used as a functional example contains three ways of handling events from the BLE Manager:

1. The BLE service framework Handling: The event is checked against the added BLE services using `ble_service_handle_event`(). This function checks if any of the added services has defined a specific behavior related to the received event. This may be a write request to an attribute whose handle belongs to a service's handle range, a received notification, etc.
2. Application Specific Handling: If the event is not handled by `ble_service_handle_event()`, then the application can define a default behavior for it. The Proximity Reporter application defines event handlers for connection, completion of advertising and pair request events.
3. Default Handling: If the event is not handled by the BLE service framework and a specific handler has not been defined by the application, then `ble_handle_event_default`() is called. This service certain events, like a connection parameter request and GATT server confirmation requests.

During initialization of the BLE services, the `pxp_reporter_task()` function is passed two specific callback functions to be used when certain service events occurred. These events are the following:

- *Immediate Alert*: An Immediate alert event is triggered when a peer device writes the Immediate Alert attribute of the Immediate Alert service. The callback function defined by the application to be fired at such an event is `ias_alert_cb()`, which simply calls the `do_alert()` function with the level that has been set, to trigger an immediate alert using the board's breath LED (D1 LED).

 © 2022 Renesas Electronics

- *Link Loss Alert*: A Link Loss alert event is triggered when the link with a peer device, that has previously written the Link Loss Alert attribute of the Link Loss service, is lost. The callback function defined by the application to be fired at such an event is `lls_alert_cb()`. This callback function adds the device, with which the link has been lost, to a reconnection list, starts the alert timeout timer and restarts "fast" advertising (with a 20/30ms advertising interval).

In addition to the service callbacks defined, the Proximity Reporter application also defines three event handlers, as seen in the code segment Code 6:

- `handle_evt_gap_connected()`: The application checks the reconnection list and if the connected device is listed there, it removes it, stops the alert and the reconnection timeout timer, and re-sets the advertising interval to "reduced power" (1/1.5s).
- `handle_evt_gap_adv_completed()`: Upon advertising completion, the application restarts advertising using `ble_gap_adv_start()`.
- `handle_evt_gap_pair_req()`: The application's pair request handler by default accepts all pair requests using `ble_gap_pair_reply()`. The pairing procedure shall then be completed by the BLE manager, which also initiates bonding if requested by the peer device.

More events can be handled by the application by adding more cases to the `switch()` statement in the main loop with appropriate handlers or direct calls.

 © 2022 Renesas Electronics

6 Peripheral Demo Application

This sample application shows how to use the drivers for the main peripherals on the DA1468x family of devices.

6.1 Basic services and features

This application demonstrates using device peripherals on the Dialog DA1468x Platform.

- The application is controlled using text-based menu via UART1 which is accessed through the virtual COM port over USB on Pro DK (ensure jumpers are fitted on J15.1-2 and 3-4). **On the Basic DK it must be accessed via an RS232-TTL level converter cable such as an FTDI TTL-232R-RPi. This demo does not use flow control and so the FTDI cable must be connected as follows. The orange wire is connected to P1_3 (Rx), the yellow wire is connected to P2_3 (Tx) and the black wire is connected to GND (J4.2). There must also be no jumpers on J13.** This is shown in Figure 15.

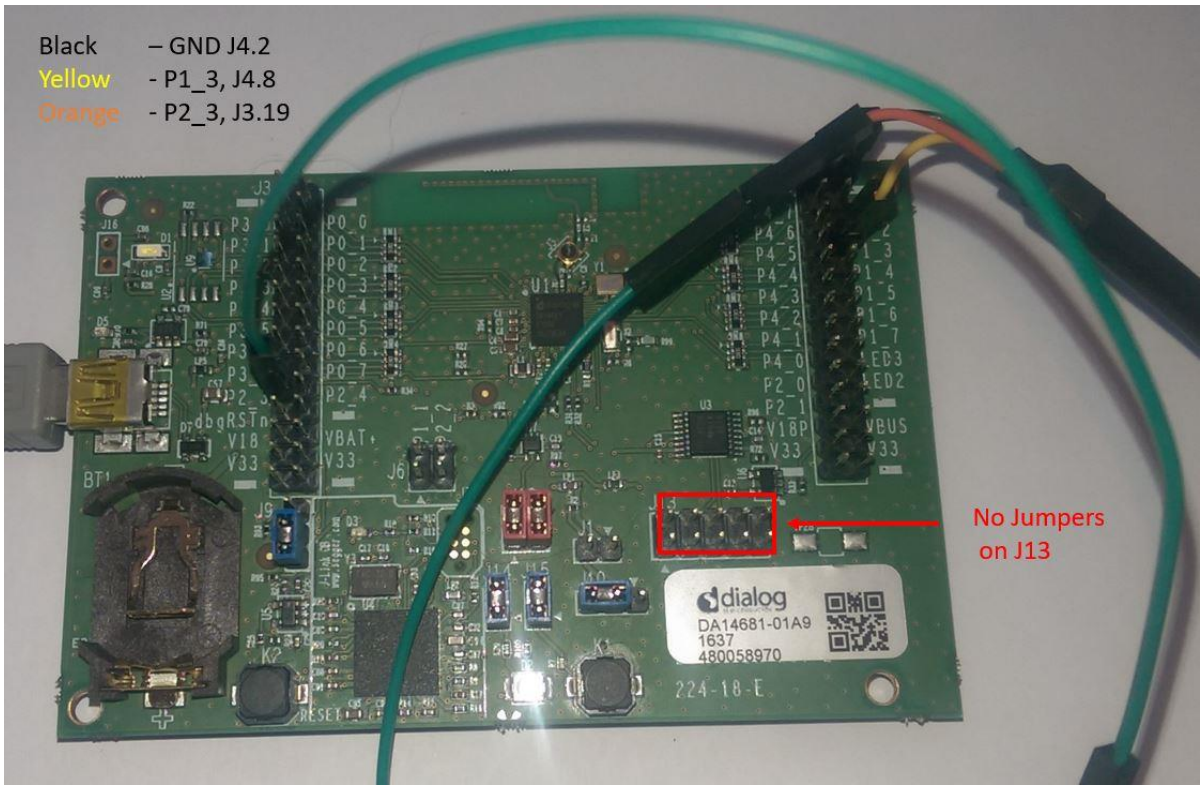


Figure 15: FTDI cable connected to UART on Basic DK with no flow control

- Some demos emit debug messages that are only visible only via UART2,
- Almost all peripherals are exercised by this application.

On both Pro and Basic DK `UART2` must be accessed using an `RS232-TTL` level converter cable such as an FTDI TTL-232R-RPi or a UART-USB conversion module to connect to the host PC.

Figure 16 shows a FTDI TTL-232R-RPI cable connected to UART2. The orange wire is connected to P4_1 (Rx), the yellow wire is connected to P4_2 (Tx) and the black wire is connected to GND (J4.1).

The actual function of these pins (P4_1 and P4_2) is controlled by the function `periph_setup()` which sets the pin multiplexing.

© 2022 Renesas Electronics

Figure 16: Connecting USB-UART cable to UART2

6.2 User Interface

The DK is connected to a PC running windows using a USB cable which is used to power the development kit. On the Pro DK it is also used to provide a virtual COM port to transfer data between a serial terminal running on the PC and the `peripherals_demo` application running on the DA1468x using UART1. On the Basic board UART1 is accessed directly from the Breakout header

When the application starts a menu appears on the terminal application on the host that allows the user to interact with the `peripherals_demo` executable running on the DK.

The `peripherals_demo` was written to run on the DK development kit for the DA1468x family of devices. Figure 17 illustrates the general setup of the project.

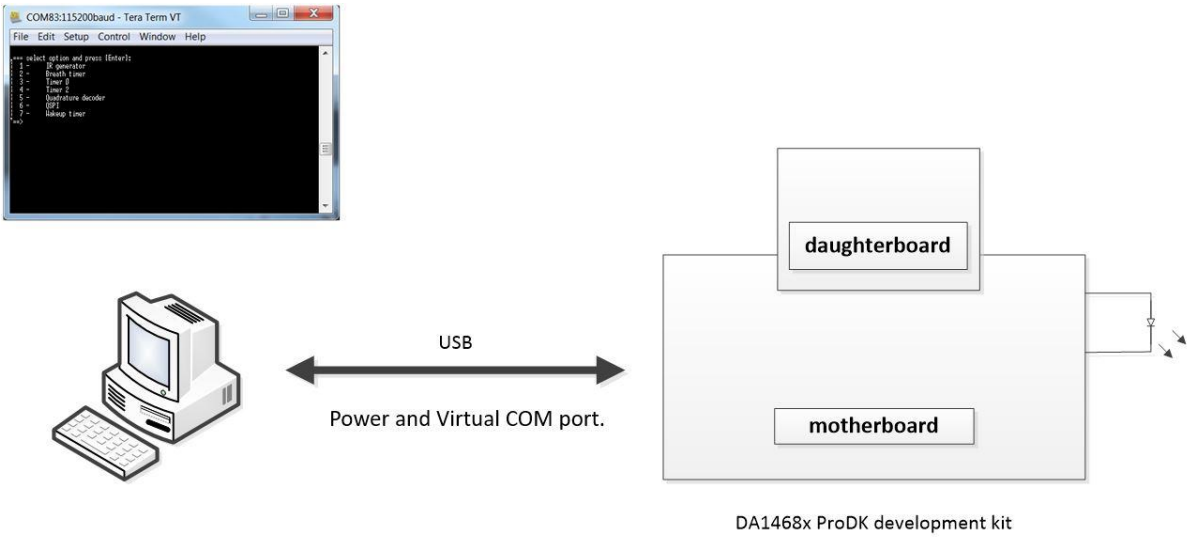


Figure 17: peripherals_demo – system overview

© 2022 Renesas Electronics

6.3 Importing the project

The user must import the project into the Project Explorer of SmartSnippets™ Studio. To import the project located in `<sdk_root_directory>\projects\dk_apps\demos\peripherals_demo` folder do the following steps:

1. Start SmartSnippets™ Studio by double clicking to the icon located in the Desktop.
2. Go to **File > Import > General > Existing Projects into Workspace** and click **Next**.
3. Find the folder containing the project is located: `<sdk_root_directory>\projects\dk_apps\demos\peripherals_demo` and click **OK**.
4. Tick `peripherals_demo`.
5. Click **Finish**.

6.3.1 Building the project

The Peripheral Demo Application project, like all other Bluetooth low energy projects, comes with a built-in configuration for QSPI cached execution mode. The project can be built using either the `DA14681-01-Debug_QSPI` or the `DA14681-01-Release_QSPI` configuration. This is done by selecting the project and clicking on the **Build** button on the SmartSnippets™ Studio toolbar as shown in Figure 18 or right click on the project's name and select **Build Project** from the pop-up menu.

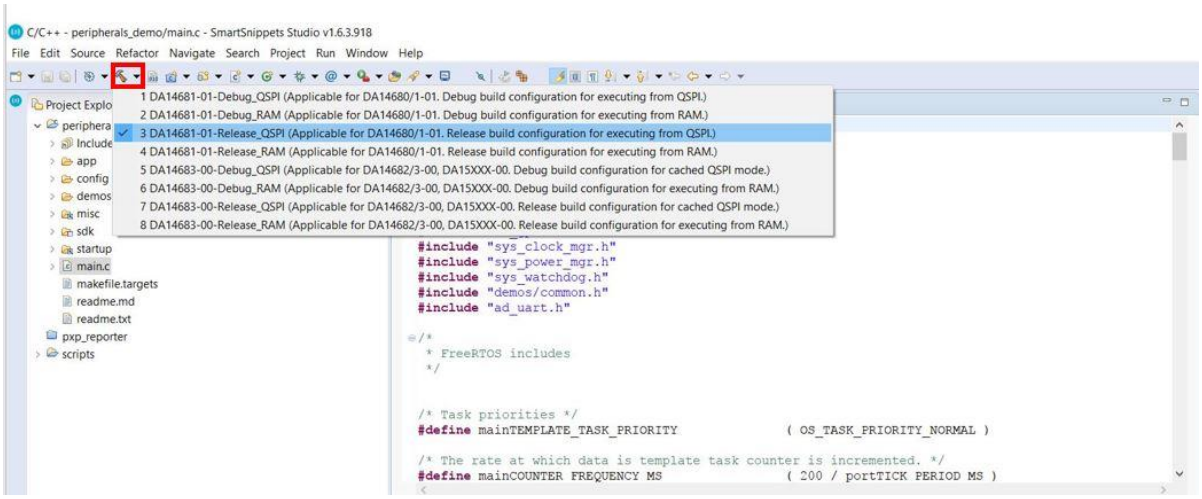


Figure 18: Building Project in Release_QSPI mode

6.3.2 Programming the QSPI Flash

1. Select the project folder and click on the icon as shown Figure 19 and make active the build mode that you want to be programmed to the QSPI Flash memory.

© 2022 Renesas Electronics

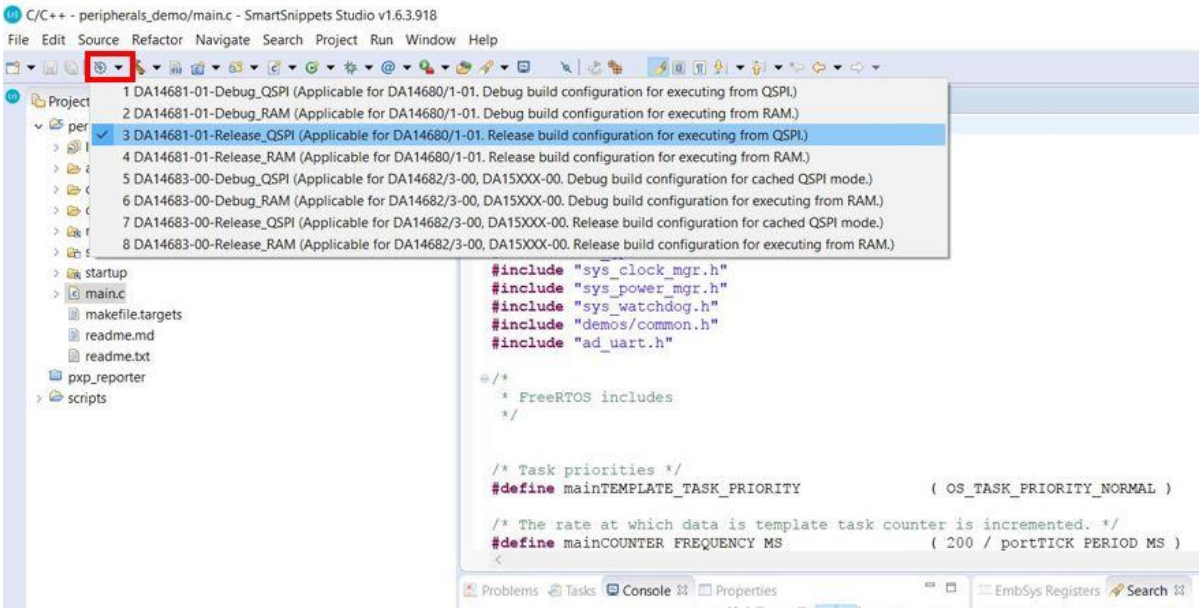


Figure 19: Selecting the build mode to be programmed

8. Under the **External Tool Configurations** menu, click on the `program_qspi_serial_win` option as shown in Figure 20 (Or **Run > External Tools > program_qspi_serial_win**).

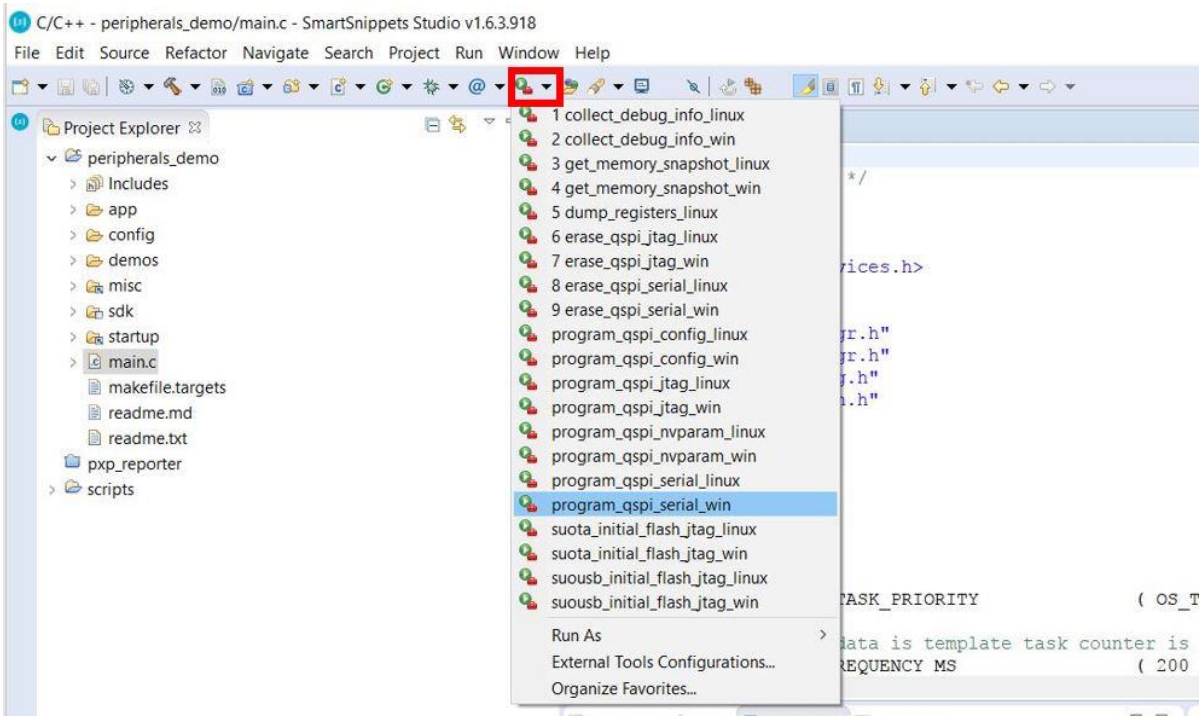


Figure 20: Program the QSPI Flash

6.4 Interacting with the Application

1. Open the Terminal program on the host PC. These instructions are for TeraTerm on Windows. Configure TeraTerm by selecting 'serial', then navigate to 'Setup' and select 'Serial port..' and configure it with the parameters as shown in**Error! Reference source not found.**.

© 2022 Renesas Electronics

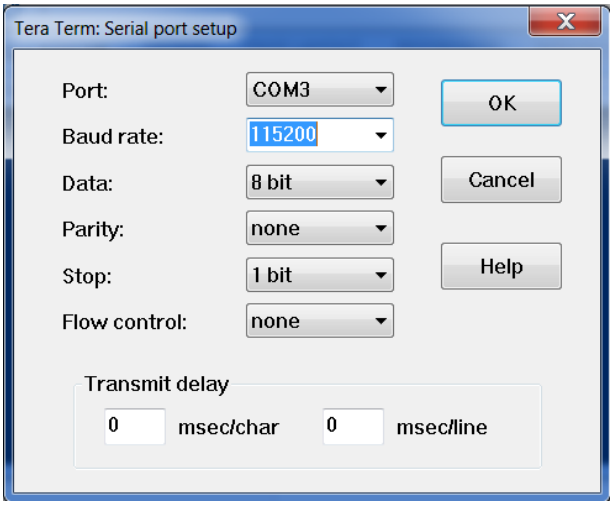


Figure 21: Configuration of the Serial Terminal

2. On a Linux Host connect the terminal program to `/dev/ttyUSB0` and use the same serial port configuration options shown in Figure 21.
3. In TeraTerm it is necessary to enable local echo so that the commands typed on the Host PC appear in the terminal. Configure TeraTerm using **Setup > Terminal** and then tick local echo as shown in Figure 22.

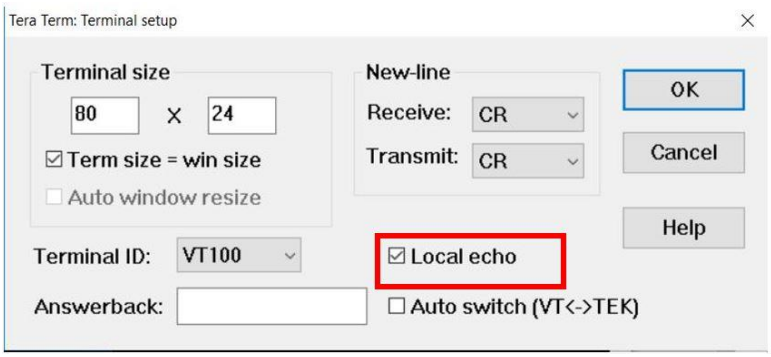


Figure 22: Enable local echo in TeraTerm

4. Press `K2` Reset button on the DK board to start the application.

When the serial terminal adapter has been correctly set up and the `peripherals_demo` has been started, the following screen (Figure 23) should appear in the terminal:

© 2022 Renesas Electronics

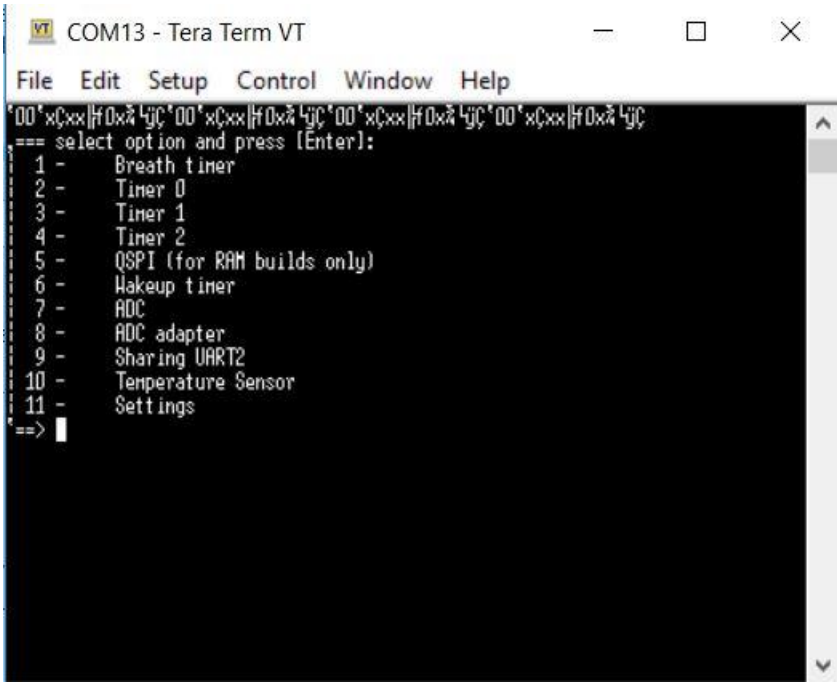


Figure 23: Output in serial terminal

User is now able to select an option by inserting a number in the terminal and then pressing the Enter key. There are a couple of things to note

- the menu may differ depending on which demos have been selected in the project's configuration.
- The first line in terminal is the Boot ROM transmitting its name at a different baud rate (57k)

Entering a number either initiates the execution of the selected option or shows a prompt to a submenu with more options. In this way, users can interact with the application.

It should be stated here that the `peripherals_demo` project is only able to evaluate peripherals that are found on the DK board and do not require additional hardware. In order to test additional hardware components such as those on the sensor board, the `peripherals_demo` project needs to be properly configured.

These additional components are enabled in the `userconfig.h` file, located in `config/default` folder as shown in Code 7. After rebuilding the project, the UART menu, shown above, is extended with the newly defined demo functionality. It is recommended to copy `config/default/userconfig.h` to `config/userconfig.h` and make any changes there. This file overrides the default one during building.

```
/*
 * Below you can enable or disable devices demos for demo_sensors.
 * They require a sensor board.
 *
 * HW_GPADC, AD_GPADC and HW_TIMER2 demos should be disabled to use sensors
demos which use I2C
 * interface for communication with the motherboard.
 */
```


© 2022 Renesas Electronics

```
#define CFG_DEMO_SENSOR_BH1750  (0)
#define CFG_DEMO_SENSOR_BME280  (0)
#define CFG_DEMO_SENSOR_ADXL362 (0)
#define CFG_DEMO_SENSOR_BMM150  (0)
#define CFG_DEMO_SENSOR_BMG160  (0)
```

Code 7: Configuration for additional hardware

 © 2022 Renesas Electronics

7 Power Measurements Demo Application

The Power Measurements Demo Application is a simple connectable advertising demo. It is designed to let the user configure several parameters such as advertising interval and connection parameters using either UART for a Command Line Interface (CLI) or via GPIOs. Several preconfigured settings can be used to examine the effect that they have on the power consumption. This application can be used as a reference for minimizing the power consumption on any given project. This way the user can determine the optimum configuration settings for the device to achieve significant energy savings.

7.1 Basic Services and Features

The Power Measurements demo application supports the configuration of the following parameters:

- Advertising interval.
- Advertising channel map.
- Connection parameters (a device must be connected).
- Recharge period (this is the period of the Sleep Timer).

There are two possible ways of interacting with the application. The choice is made at compile time via `config/power_demo_config.h`

1. Using the predefined CLI commands via UART2. This is selected by setting

```
#define POWER_DEMO_CLI_CONFIGURATION        1
#define POWER_DEMO_GPIO_CONFIGURATION       0
```

2. Using `GPIO` settings and a button (not the `K1` button). The button must be connected in `P1_0` as referred in 7.4 and shown in Figure 31.

```
#define POWER_DEMO_CLI_CONFIGURATION        0
#define POWER_DEMO_GPIO_CONFIGURATION       1
```

This application uses UART2 for logging with `printf()` in both these configurations with the pin mux in `periph_init()` routes UART2 pins to these pins. In the CLI build UART2 is also used for the CLI. There are two ways of accessing UART2 from the host PC. This demo requires `CTS` to be controlled as UART2 is defined to have `auto-flow-control`.

1. UART2 over USB (**ProDK only**)

   For the UART to work on the `COMx` or `/dev/ttyUSBx` port three jumpers are required on J15 as shown in Figure 24. There are the two usual jumpers on Tx (`J15.1-2`) and Rx (`J15.3-4`) plus CTS (`J15.7-8`). CTS must be driven as the UART_BUS macro enables hardware flow control in this project. `J5` must also be removed to stop `P1_6` controlling LED2.

 © 2022 Renesas Electronics

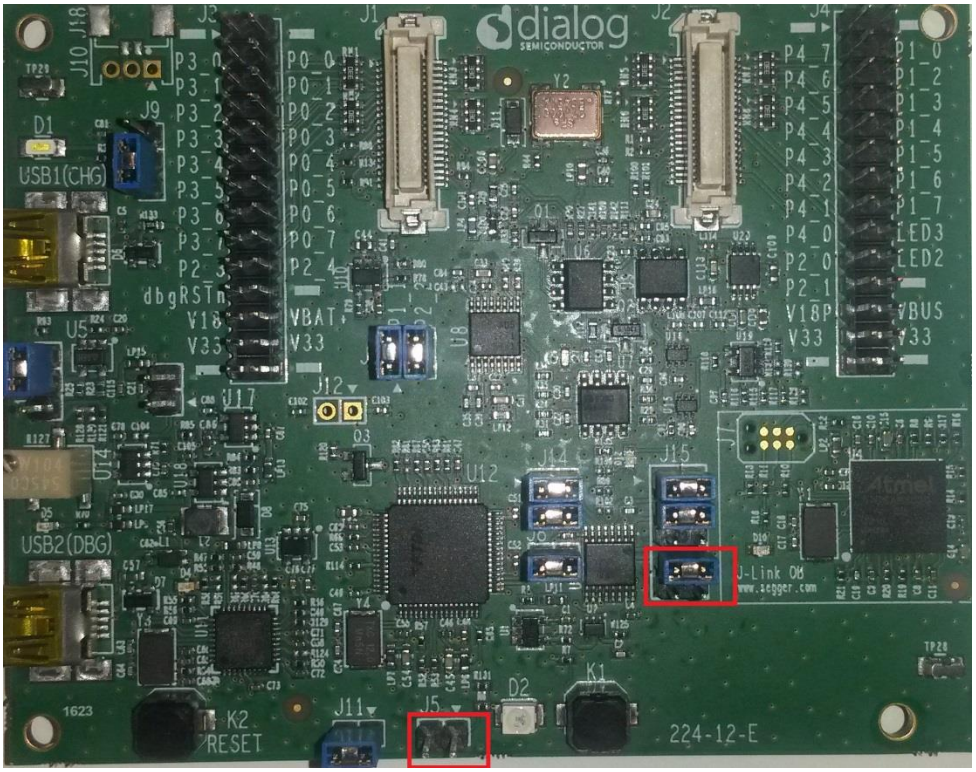


Figure 24: Jumper settings for UART2 and GPIO configuration ProDK Virtual COM port with Flow Control

2. UART2 via the breakout headers (**mandatory on Basic DK or optional on ProDK**)

   In this mode a USB to TTL level UART cable (such as FTDI TTL-232R-RPI) is connected directly to the breakout headers and the on board FTDI USB to UART converter is disconnected. This approach allows UART2 to be used for control in Basic DK.

   On the ProDK it allows demo to be controlled and/or logging to be taken when the board is being powered by battery and no USB cable is connected to `USB2(DBG)`.

   To do this on ProDK remove all jumpers from J15 and connect FTDI TTL-232R-RPI as shown in Figure 25. The orange wire is connected to `P2_3` (Rx), the yellow wire is connected to `P1_3` (Tx), the black wire is connected to GND (`J4.1`) and `P1_6` (CTS) is connected to `J4.28` (GND) to assert CTS.

© 2022 Renesas Electronics

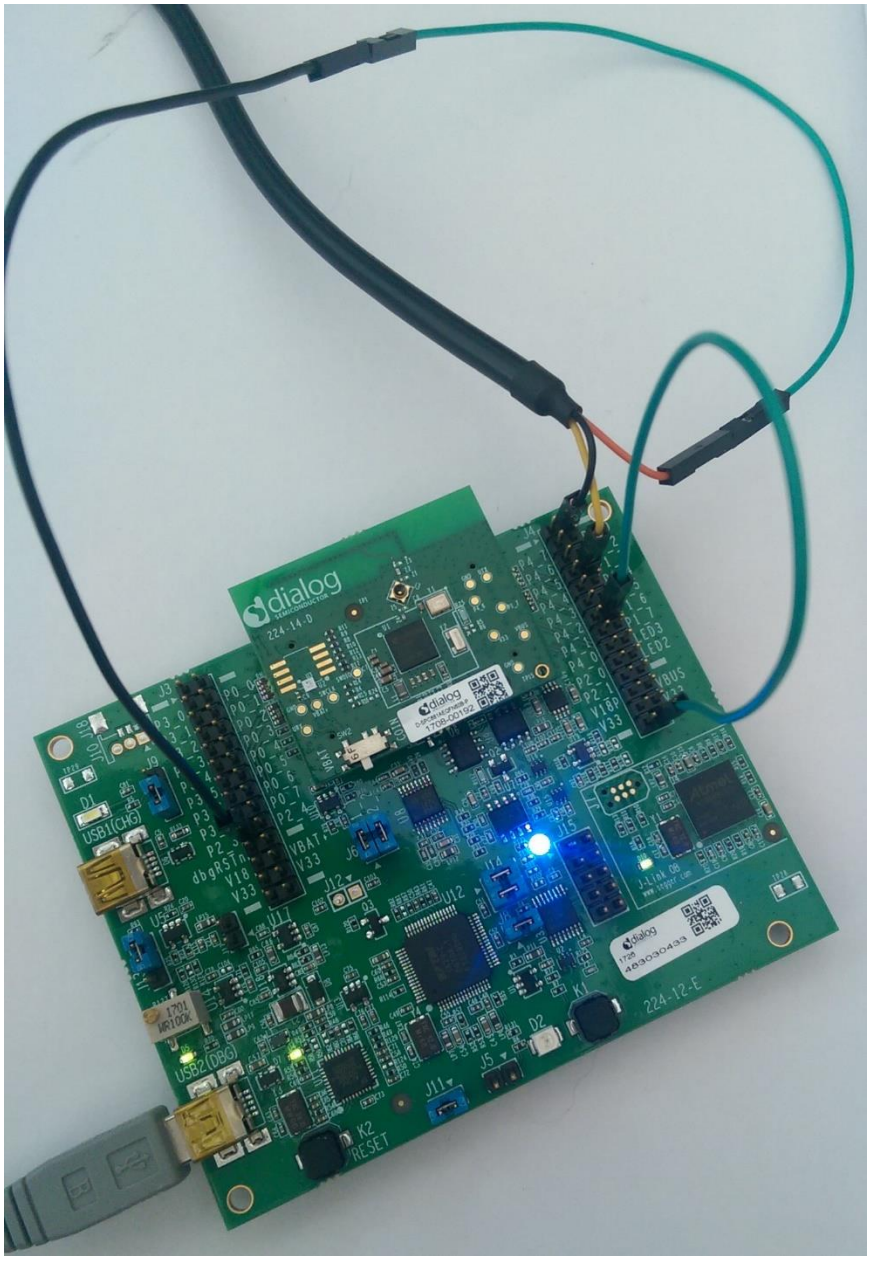


Figure 25: Connect USB-UART cable to ProDK Breakout Headers with Flow Control

To do this on BasicDK remove all jumpers from `J13` and connect FTDI TTL-232R-RPI as shown in Figure 26. The orange wire is connected to `P2_3` (Rx), the yellow wire is connected to `P1_3` (Tx), the black wire is connected to GND (J4.1) and `P1_6` (CTS) is connected to `J4.28` (GND) to assert CTS.

© 2022 Renesas Electronics

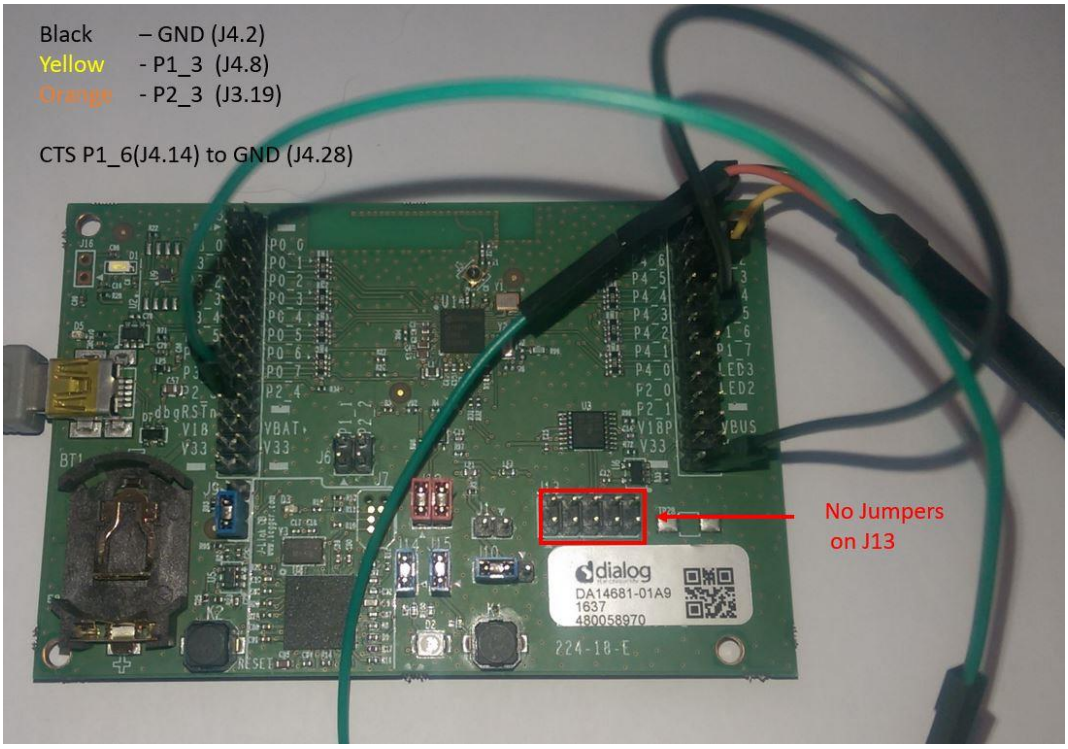


Figure 26: Connect USB-UART cable to BasicDK Breakout Headers with Flow Control

7.2 User Interface

The DK Development Kit is connected to a PC using a USB cable. This cable is used to power the development kit. On ProDK it also and to transfer data between a serial terminal running on the PC and the `power_demo` application running on the DA1468x. On BasicDK this is over a separate USB-UART cable.

For user notification purposes the application uses the serial terminal to print various messages depending on the current configuration and the user's actions. For example, once the application has started, 'Advertising started...' messages are printed out.

7.3 Importing the project

The user must import the project into the Project Explorer of SmartSnippets™ Studio. To import the project located in `<sdk_root_directory>\projects\dk_apps\demos\power_demo` folder use the following steps:

1. Start the SmartSnippets™ Studio by double clicking to the icon located in the Desktop.
2. Go **to File > Import > General > Existing Projects into Workspace** and click **Next**.
3. Find the folder containing the project is located: `<sdk_root_directory>\projects\dk_apps\demos\power_demo` and click **OK**.
4. Tick the `power_demo`.
5. Click **Finish**.

7.3.1 Building the project

The Power Demo Application project, like all other Bluetooth low energy projects, comes with a built-in configuration for QSPI cached execution mode. The project can be built using either the `DA14681-01-Debug_QSPI` or the `DA14681-01-Release_QSPI` configuration. This is done by selecting the project and clicking on the **Build** button on the SmartSnippets™ Studio toolbar as shown in Figure 27 or right click on the project's name and select **Build Project** from the pop-up menu.

© 2022 Renesas Electronics

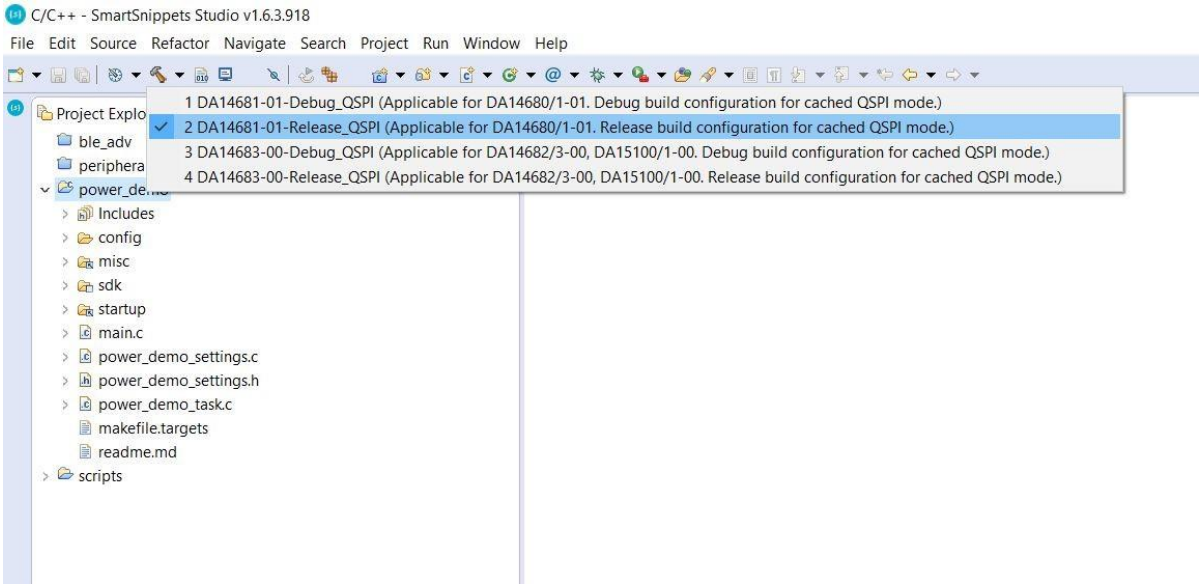


Figure 27: Building Project in Release_QSPI mode

7.3.2 Programming the QSPI Flash

1. Select the project folder, click on the Build Configuration icon as shown in Figure 28 and select the build configuration that you want to program into the QSPI Flash memory.

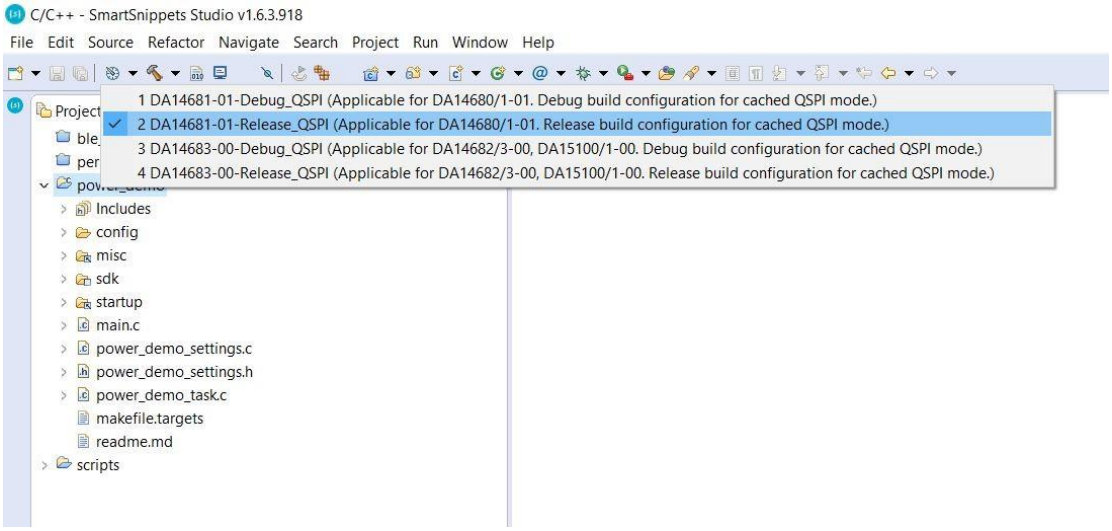


Figure 28: Selecting the build mode to be programmed

2. Under the **External Tool Configurations** menu, click on the `program_qspi_serial_win` option (Or `Run > External Tools > program_qspi_serial_win`) as shown in Figure 29

© 2022 Renesas Electronics

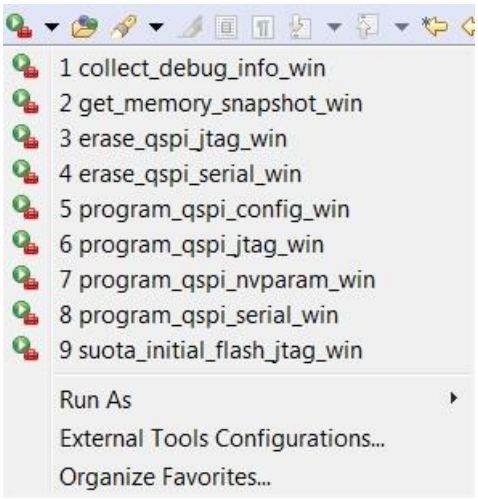


Figure 29: External Tools

Set up TeraTerm as described in section 6.4 for the Peripheral demo.

3. Press K2 RESET button on the DK board.

7.4 Interacting with the Application

After building and loading the project to the DA1468x device, the user is able to interact with it and apply the various settings using serial terminal over UART2 or GPIO configurations. A mobile phone is also needed to apply the connection parameters update settings (the mobile phone must support Bluetooth low energy).

To control the application through UART, macro POWER_DEMO_CLI_CONFIGURATION in the config/power_demo_config.h file must be set to 1.

To control the application using GPIO settings and a button POWER_DEMO_GPIO_CONFIGURATION macro in file config/power_demo_config.h must be set to 1.

The default configuration for the application is shown in Code 8. In order to control the application using the GPIO settings the POWER_DEMO_CLI_CONFIGURATION macro must be set to 0 and the POWER_DEMO_GPIO_CONFIGURATION macro must be set to 1.

Note 3 Only the desired configuration setting can be enabled each time and not both. Otherwise the application is not built.

```
/**
 * Set power configuration over CLI
 */
#define POWER_DEMO_CLI_CONFIGURATION    (1)

/**
 * Set power configuration through GPIO settings
 */
#define POWER_DEMO_GPIO_CONFIGURATION   (0)

#endif /* POWER_DEMO_CONFIG_H_ */
```

Code 8: Enable UART and/or GPIO configuration

© 2022 Renesas Electronics

7.4.1 Controlling via UART2

The settings to control the application via UART are shown in the Table 3. This assumes that one of the connections described in Section 7.1 has been used to connect to UART2.

Table 3: UART settings

Setting	Value
Baudrate	115200
Data bits	8
Stop bits	1
Parity	01
Flow control	RTS/CTS

7.4.2 Controlling via GPIO

- In this mode the configuration type and index to select are defined by using jumper leads to tie GPIOs **P1_2**, **P1_4**, **P1_5** and **P1_7** either high or low as shown in Table 4 and Table 5.
- The configuration is then set when a "button" is pressed. The "button" press is emulated by using a jumper to pull P1_0 low for a short period. The jumper can be applied between pins J4.2 and J4.4.
- This mode still uses UART2 for logging and so every time a setting is applied by using a jumper on P1_0 a log message appears on a terminal connected to UART2.

Table 4: GPIO configuration type

Configuration type	P1_2	P1_4
Advertising interval	GND	GND
Advertising channel map	VCC	GND
Recharge period	GND	VCC
Connection parameters update	VCC	VCC

Table 5: GPIO configuration index

Configuration index	P1_5	P1_7
0	GND	GND
1	VCC	GND
2	GND	VCC

For example, to set **advertising interval** configuration, user must connect P1_2 to GND and P1_4 to GND. To apply the second configuration index (index 1 from Table 6) P1_5 must be connected to VCC and P1_7 to GND. Finally, to apply these settings P1_0 must be connected to GND as the "button" press.

7.4.3 Set advertising interval

The user can change the advertising interval using either the UART or the GPIOs. Table 6 shows the available settings.

© 2022 Renesas Electronics

Table 6: Advertising interval settings

Configuration index	Interval min [ms]	Interval max [ms]
0	400	600
1	30	60
2	1000	1200

To apply the desirable setting using **serial console** type:

`set_adv_interval <cfg_idx>`

For example to apply the third configuration (index 2 from Table 6) the user must type in Tera Term the following command:

```
> set_adv_interval 2
```

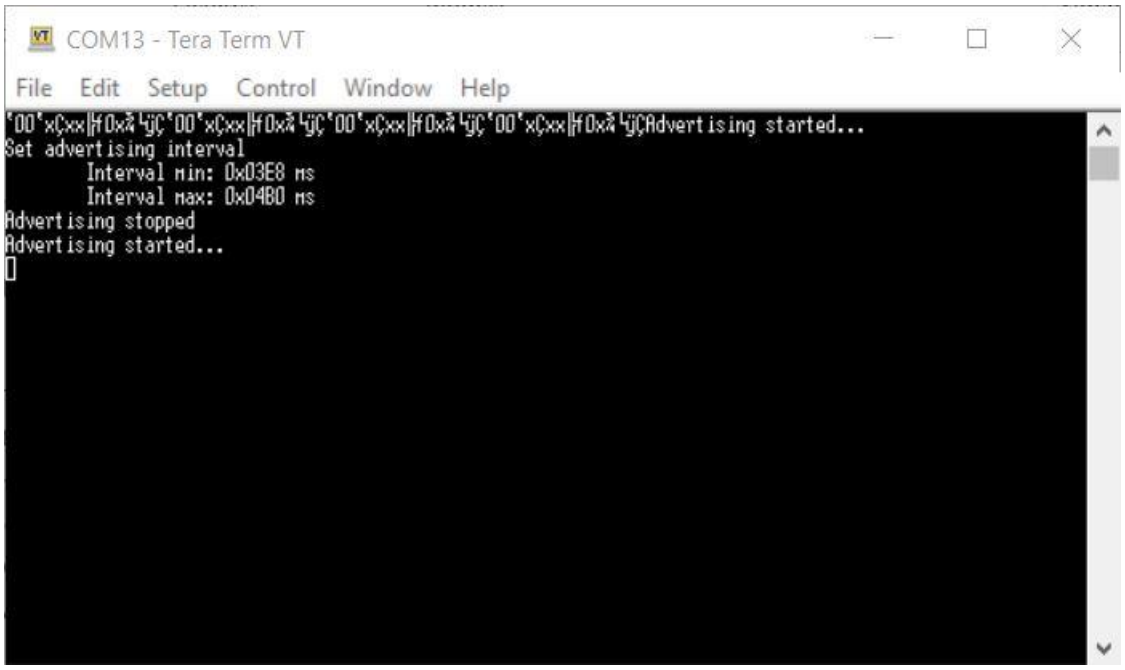


Figure 30: Set advertising interval via CLI

To set the advertising interval using GPIOs user must connect `P1_2` and `P1_4` to GND (Table 4). Then to select the configuration index, Table 6 must be used.

For example, the GPIO connections for the second configuration index (index 1 from Table 6) are shown in Figure 31

© 2022 Renesas Electronics

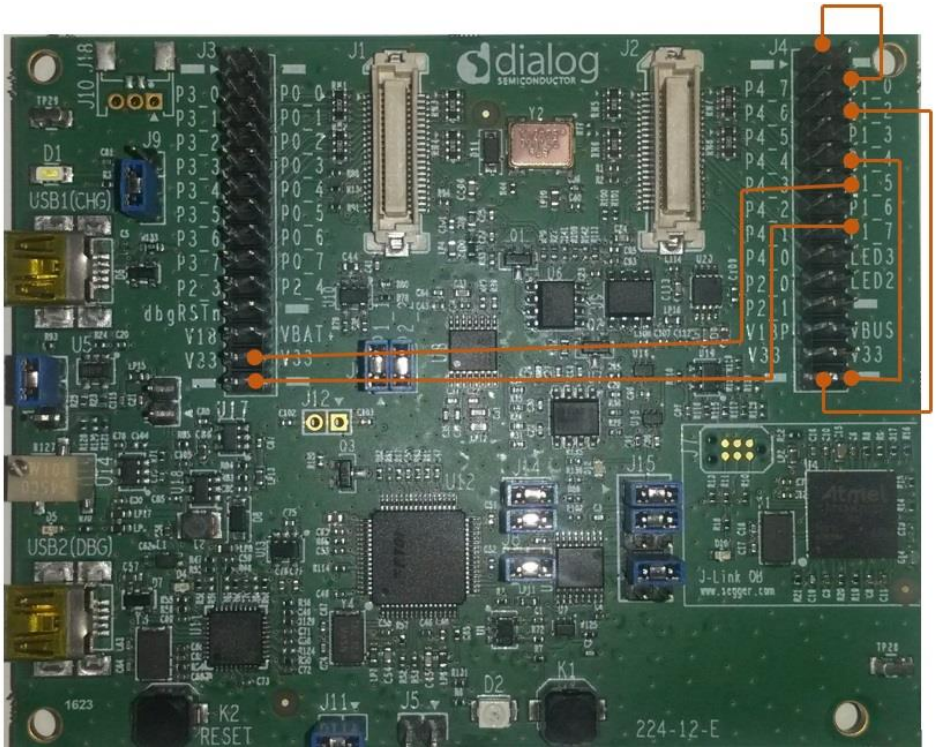


Figure 31: Set Advertising Interval via GPIOs

Note 4 To emulate the button press, `P1_0` should be connected to `GND` for a short time and not permanently, just like a real button press.

On the ProDK the Power Profiler [6] can be used to confirm that there was a change in the advertising interval. Figure 31 shows the output of the Power Profiler, the adverting interval has been set to configuration index 1.

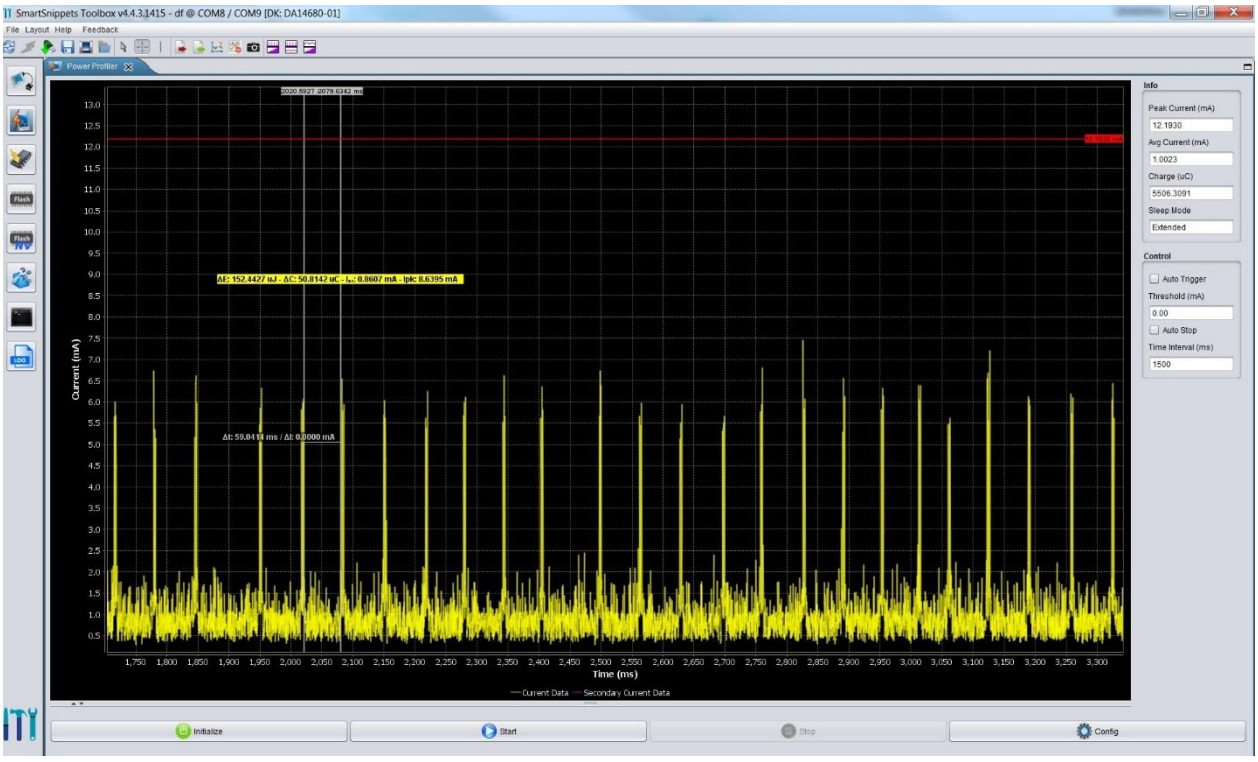


Figure 32: Power Profiler output for the second configuration index

© 2022 Renesas Electronics

7.4.4 Set channel map

The user can set the advertising channel map using either the UART or the GPIOs. Table 7 shows the available settings.

Table 7: Channel map settings

Configuration index	Channel map
0	37 and 38 and 39
1	38 and 39

To apply the desirable setting using the serial console type:

set_adv_channel_map <cfg_idx>

For example to apply the second configuration (index 1 from Table 7) user must type in Tera Term the following command:

```
> set_adv_channel_map 1
```

To set the advertising interval using GPIOs the user must connect P1_2 to VCC and P1_4 to GND (Table 4). Table 7 must be then used to select the configuration index

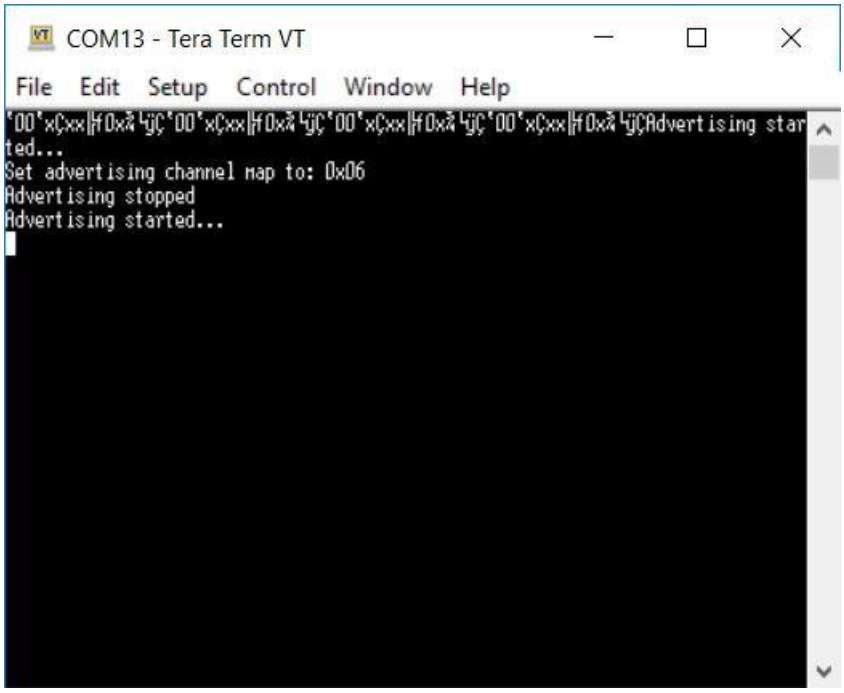


Figure 33: Set advertising channel map via CLI

The Power Profiler can be used on the ProDK to confirm that the advertising channel map actually. Figure 34 shows the output of the Power Profiler before changing the adverting channel map while Figure 35 after channel map has set to configuration index 1.

© 2022 Renesas Electronics

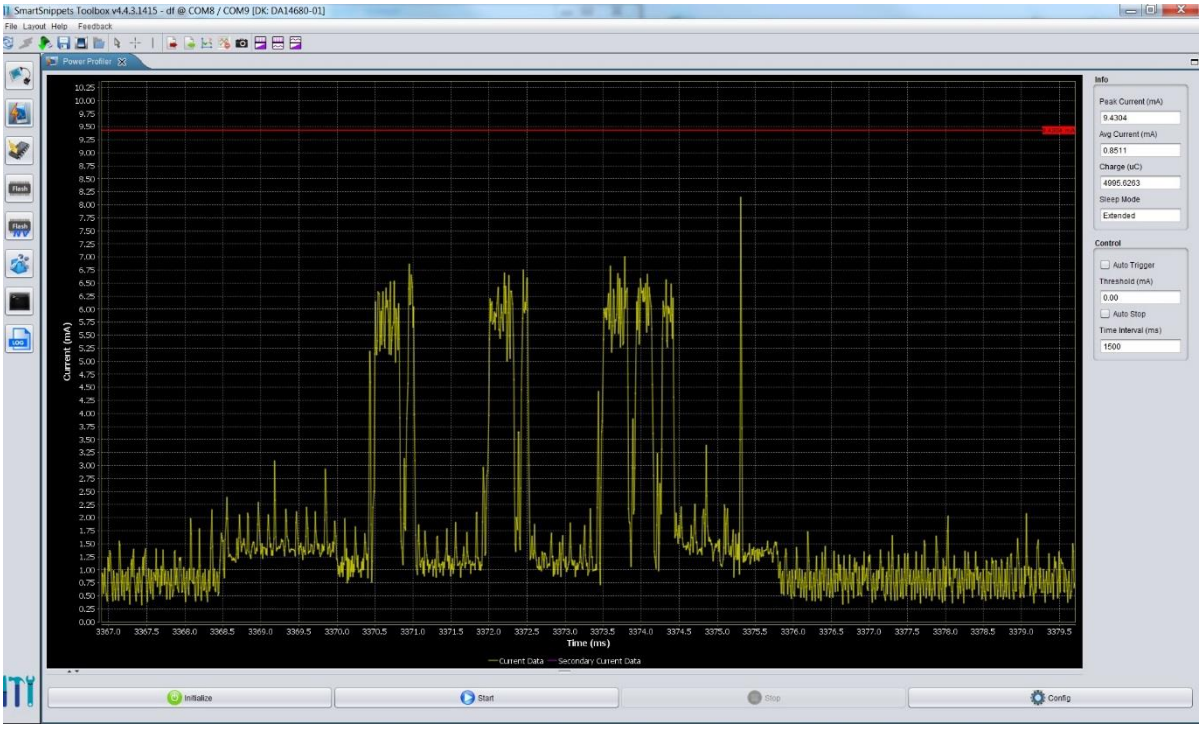


Figure 34: Three advertising channels

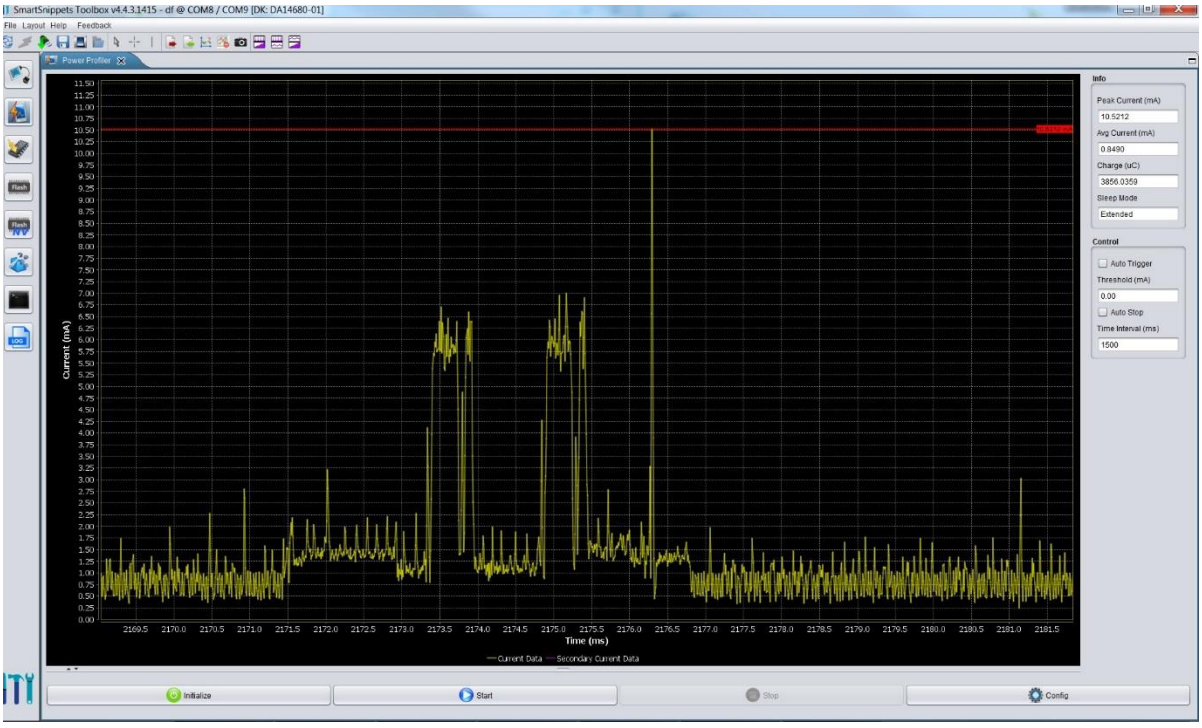


Figure 35: Two advertising channels

7.4.5 Set recharge period

The user can set the recharge period using either the UART or the GPIOs. Table 8 shows the available settings.

© 2022 Renesas Electronics

Table 8: Recharge period settings

Configuration index	Recharge period
0	3000
1	100
2	900

To apply the desirable setting using serial console type:

set_recharge_period <cfg_idx>

For example to apply the second configuration (index 1 from Table 8) user must type in Tera Term the following command:

```
> set_recharge_period 1
```

To set the recharge period using **GPIOs** the user must connect P1_2 to GND and P1_4 to VCC (Table 4). Then to select the configuration index, Table 8 must be used.

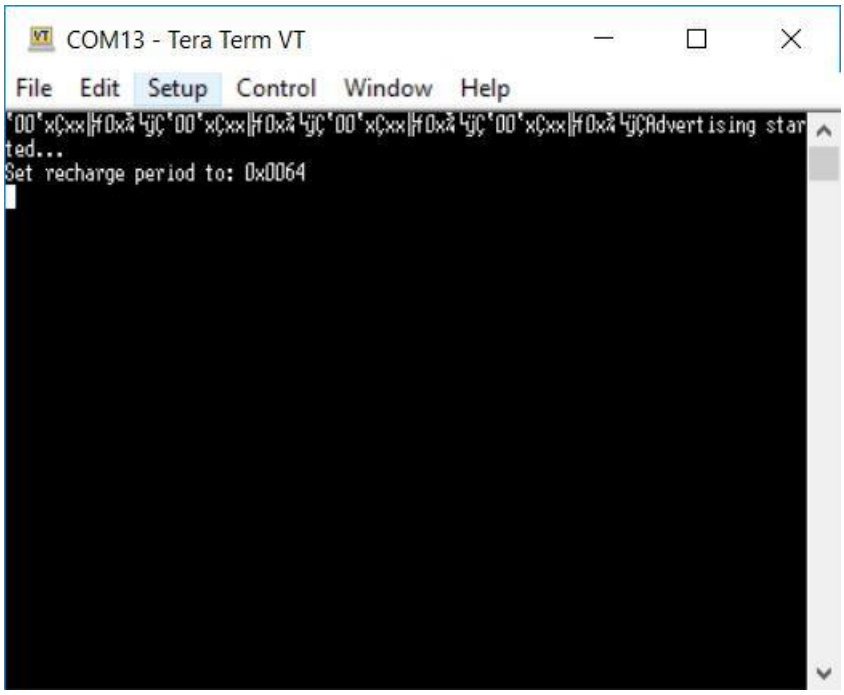


Figure 36: Set recharge period via CLI

The change to the recharge period can be verified by checking the contents of the SLEEP_TIMER_REGISTER sleep Timer which is used to bring-up part of the system periodically to resample the bandgap voltage or to restore the energy of the inductor of the DCDC. So, after changing the recharge period the content of this register is changed. Figure 37 shows the contents of the SLEEP_TIMER_REGISTER after setting the recharge period to 100 ms (configuration index 1). See section 14 in [2] for instructions to configure and view the register mapping for DA14681.

© 2022 Renesas Electronics

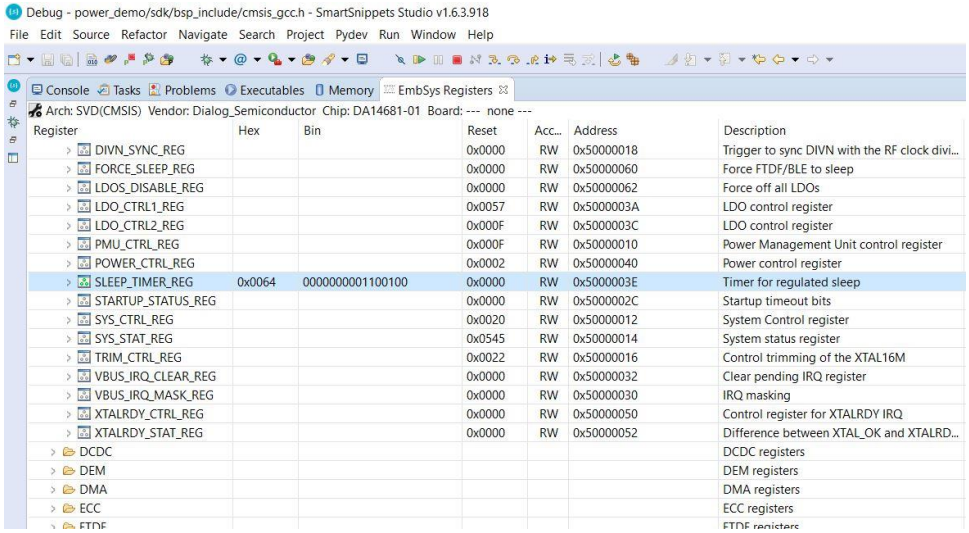


Figure 37: `SLEEP_TIMER_REGISTER` contents

7.4.6 Set connection parameters

Connection parameters can change using either the UART or the GPIOs. The update of the connections parameters only takes place in an active connection. Table 9 shows the available settings.

Table 9: Connection parameters settings

Configuration index	Interval min [ms]	Interval max [ms]	Slave latency	Sup. timeout [ms]
0	400	600	0	1500
1	10	15	0	100
2	1000	1200	0	3000

To apply the desirable setting using serial console type:

`conn_param_update <cfg_idx>`

For example to apply the second configuration (index 1 from Table 9) the user must type in the serial terminal the following command:

```
> conn_param_update 1
```

To update the connection parameters using GPIOs the user must connect `P1_2` and `P1_4` to VCC (Table 4). Then to select the configuration index, Table 9 must be used.

© 2022 Renesas Electronics

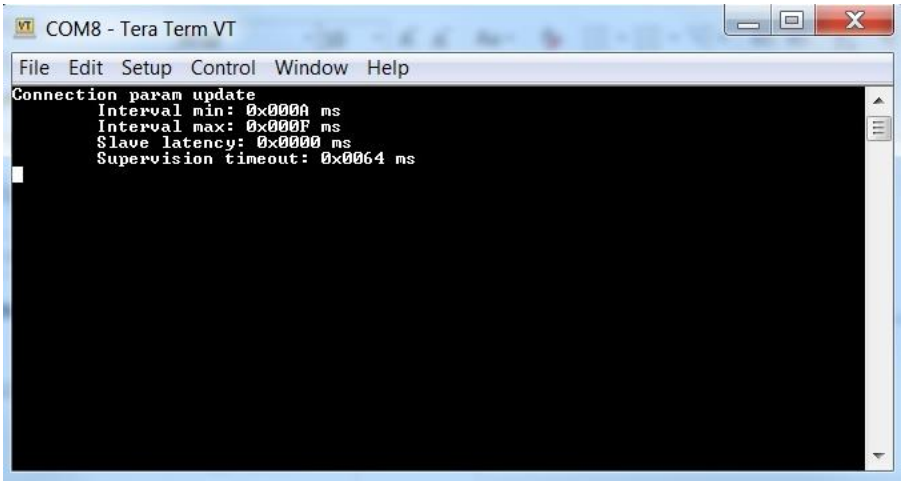


Figure 38: Connection parameters update

Note 5 to measure properly the sleep current with the Power Profiler tool on ProDK or with an external tool the Tera Term UART over USB session must be terminated or remove `J15.7-8` jumper to de-assert CTS as shown in Figure 39. If a UART is required in this configuration use the direct connection to breakout header described in section 7.1.

Doing this prevents any leakage current that may affect the proper measurement of the sleep current.

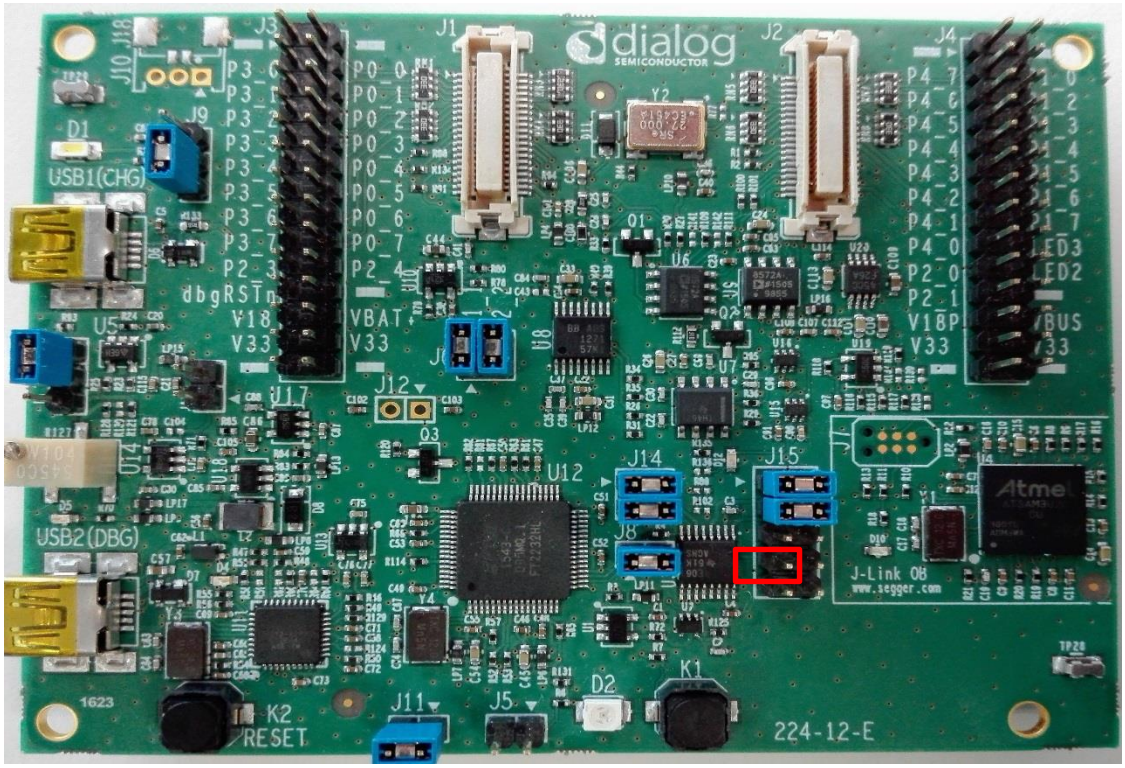


Figure 39: Sleep current measurement

© 2022 Renesas Electronics

8 Create a custom application

8.1 Creating a Bluetooth low energy project

The recommended way to begin a new Bluetooth low energy project is to use one of the existing examples as a basis. The *ble_central* and *ble_peripheral* projects are great starting points for Bluetooth low energy central and peripheral applications respectively, while *ble_multi_link* is appropriate for applications that need to use both roles simultaneously.

The next sections summarize the different aspects to be considered when using one of these existing projects as a template upon which to build a new application.

8.2 Configuring your application

The user can configure any project using a series of MACRO definitions. Some of the options that can be modified are:

- System clocks
- Minimum sleep time
- Charging functionality
- Total heap size
- Chip revision and stepping
- Power up/down peripherals
- Watchdog
- RAM Retention Configuration (refer to section 13.3 of [4]).

The key file for any project is `config/custom_config_qspi.h` which contains all the new configuration options for this project that overrides the default values in the SDK.

8.3 Adding Bluetooth low energy functionality

To extend a Bluetooth low energy project's functionality, the developer should become familiar with the Dialog BLE API. These API header files come with additional Doxygen documentation and are summarized in Table 10. The Doxygen documentation is available in SmartSnippets™ Studio via the API Documentation Open button (at the bottom left) or directly in the SmartSnippets™ DA1468x SDK via `doc/html/index.html`

8.3.1 Including BLE header files

Table 10 : Dialog BLE API header files

File name	Description
`sdk/ble/include/ble_att.h`	Attribute Protocol API: Mostly definitions.
`sdk/ble/include/ble_common.h`	Common API: Functions used for operations not specific to a certain BLE host software component
`sdk/ible/include/ble_gap.h`	GAP API: ● Device parameters configuration: device role, MTU size, device name exposed in the GAP service attribute, etc. ● Air operations: Advertise, scan, connect, respond to connection requests, initiate or respond to connection parameters update, etc. ● Security operations: Initiate and respond to a pairing or bonding procedure, set the security level, unpair, etc.

© 2022 Renesas Electronics

File name	Description
sdk/ble/include/ble_gattc.h	GATT client API: • Discover services, characteristics, etc. of a peer device • Read or write a peer device's attributes • Initiate MTU exchanges • Confirm the reception of indications
sdk/ble/include/ble_gatts.h	GATT server API: • Set up the attribute database • Set attribute values • Notify/indicate characteristic values • Initiate MTU exchanges • Respond to write and read requests
sdk/ble/include/ble_storage.h	BLE persistent storage API.
sdk/ble/include/ble_uuid.h	BLE UUID declarations and helper functions.

8.3.2 Adding BLE services

Table 11 summarizes the API header files of the Bluetooth low energy adopted GATT services already implemented by the SmartSnippets™ DA1468x SDK. These files can be found under <sdk_root_directory>\sdk\ble_services\include. The developer can use these APIs to add these services to another project.

Table 11 : BLE service API header files

File name	Description
ble_service.h	BLE service framework API: • Add service to framework • Handle event using BLE service framework • Elevate permission • Get number of attributes in a service • Add included services
bas.h	Battery Service – BAS
bcs.h	Body Composition Service – BCS
bms.h	Bond Management Service – BMS
cts.h	Current Time Service – CTS
dis.h	Device Information Service – DIS
dlg_debug.h	Dialog Debug Service
dlg_suota.h	Dialog SUOTA Service
hids.h	Human Interface Device Service – HID
hrs.h	Heart Rate Service – HRS
ias.h	Immediate Alert Service – IAS
lls.h	Link Loss Service – LLS
scps.h	Scan Parameters Service – ScPS
sps.h	Serial Port Service – SPS
tps.h	Tx Power Service – TPS

© 2022 Renesas Electronics

File name	Description
uds.h	User Data Service – UDS
wss.h	Weight Scale Service – WSS

8.3.3 Bonding information management

Most aspects of security are handled seamlessly by the BLE Framework. An application that needs to set-up security, for example initiate pairing, do a security request or set-up encryption using previously exchanged keys, needs only to use the appropriate API. Most details of the procedures are handled internally by the BLE Framework and the application is notified only if intervention is needed or when the procedure is completed.

The generation and storage of the security keys and other bonding information is also handled by the BLE Framework. Persistent storage can also be used to store the security keys and bonding data information in the flash. This allows the information to be retrieved by the BLE Framework after a power cycle and so used to reestablish connections with previously bonded devices.

Note 6 For more a detailed description about the Bonding management (API's, Events, MSC's…) please refer to section 7.3 of [4].

8.3.4 Hooks

The BLE Hooks mechanism provides the user application a way to be notified about the exact time of occurrence of specific BLE events.

This mechanism enables the user application to receive notifications of BLE Interrupt Service Routine (ISR) events. These events can be received either directly from inside the BLE ISR, or as task notifications to the application task registered to the BLE manager

To enable this feature, define `dg_configBLE_EVENT_NOTIF_TYPE` to either `BLE_EVENT_NOTIF_USER_ISR` or `BLE_EVENT_NOTIF_USER_TASK`.

When `dg_configBLE_EVENT_NOTIF_TYPE == BLE_EVENT_NOTIF_USER_ISR`, then the following macros can be defined in the application code:

Table 12: Macros for the configuration of the hook functions

Macro name	Description
dg_configBLE_EVENT_NOTIF_HOOK_END_EVENT	The BLE End Event
dg_configBLE_EVENT_NOTIF_HOOK_CSCNT_EVENT	The BLE CSCNT Event
dg_configBLE_EVENT_NOTIF_HOOK_FINE_EVENT	The BLE FINE Event

These macros must be set to the names of functions defined inside the user application and which have the following prototype:

- void func(void); [2]

If a macro is not defined, then the respective notification is suppressed.

Note 7 These functions are called in an ISR context, directly from the BLE ISR. They should therefore be very fast and should NEVER block.

When `dg_configBLE_EVENT_NOTIF_TYPE == BLE_EVENT_NOTIF_USER_TASK`, the user application receives task notifications on the task registered to the BLE manager. Notifications are received using the following bit masks:

[2] The user application does not need to explicitly define the prototype.

© 2022 Renesas Electronics

Table 13: Notification bit masks

Macro name	Description
dg_configBLE_EVENT_NOTIF_MASK_END_EVENT	End Event Mask (Default: bit 24)
dg_configBLE_EVENT_NOTIF_MASK_CSCNT_EVENT	CSCNT Event Mask (Default: bit 25)
dg_configBLE_EVENT_NOTIF_MASK_FINE_EVENT	FINE Event Mask (Default: bit 26)

The bit mask for each of the macros in Table 13 can be redefined as needed.

If one of the macros for callback functions listed in Table 12 (for direct ISR notifications) is defined then the ISR mode takes precedence and the function with the same name is called directly from the ISR instead of sending a task notification for this particular event to the application task.

The macro `dg_configBLE_EVENT_NOTIF_RUNTIME_CONTROL` (`Default: 1`) enables/disables runtime control/masking of notifications.

If `dg_configBLE_EVENT_NOTIF_RUNTIME_CONTROL == 1` then task notifications must be enabled/disabled using the `ble_event_notif[enable|disable]_[end|cscnt|fine]_event()` functions. By default, all notifications are disabled.

If `dg_configBLE_EVENT_NOTIF_RUNTIME_CONTROL == 0`, all notifications are sent unconditionally to the application task.

© 2022 Renesas Electronics

Timing diagrams:

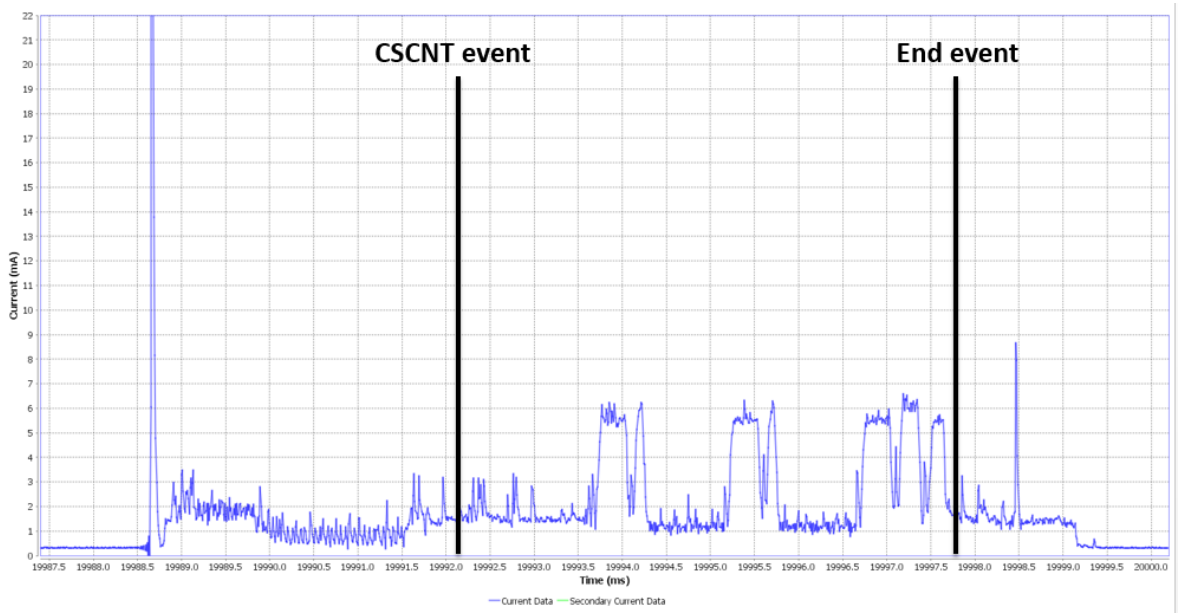


Figure 40: Advertising

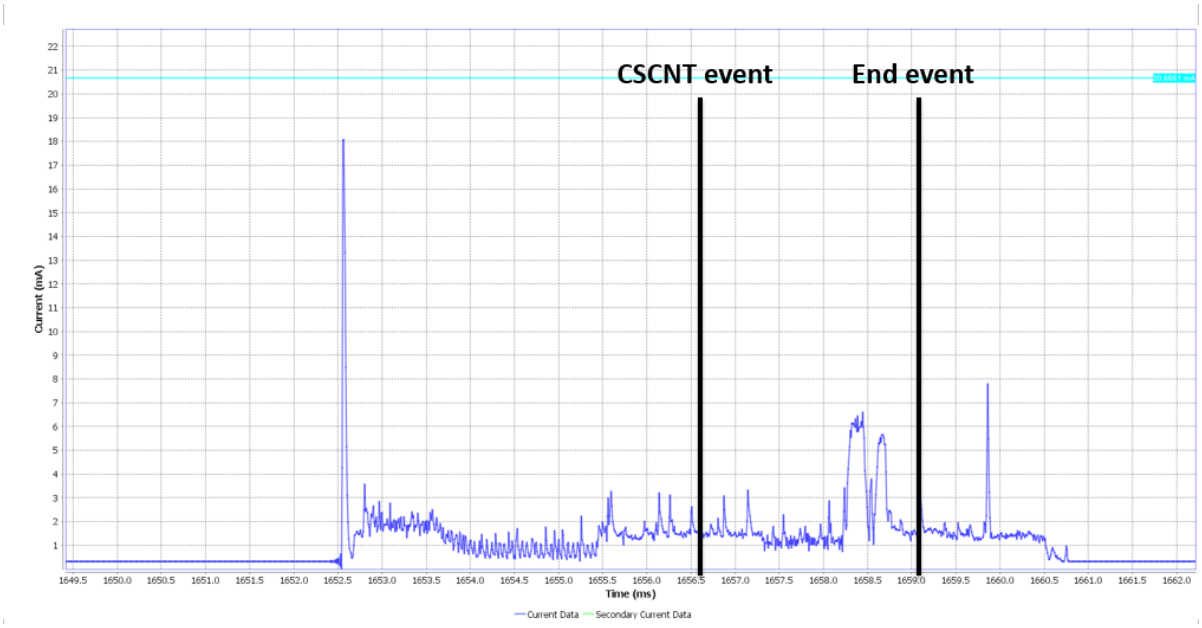


Figure 41: Connected

© 2022 Renesas Electronics

9 Software Upgrade

9.1 Software Upgrade Over The Air (SUOTA)

9.1.1 Introduction

The Bluetooth low energy platform allows the user to update the software of the device wirelessly. This process is called Software Upgrade Over The Air (SUOTA) and is simple enough to be performed by the end user.

When an update procedure is initiated from an Android or iOS device, a new image is first transferred to the Firmware update partition located in the Flash memory and then the device reboots to complete the update. After the reboot is completed, the SUOTA loader transfers the image to the Executable partition and executes it. The new software version should start after the reboot with a small delay.

The SUOTA GATT server runs on the DA1468x device and the GATT client on the Android or iOS device running the SUOTA application.

9.1.2 SUOTA service description

This section gives a brief description of the SUOTA service, responsible for performing software upgrades over BLE. A detailed service characteristic description is given on Table 14.

Table 14: SUOTA service characteristics

Characteristic	SUOTA Version (SUOTA_VERSION definition)	Access	Size	Description
MEM_DEV	since version v1.0	READ WRITE	4	Using this characteristic the client is able to send commands to the SUOTA service. Some of the most commonly used commands are the following: • **SPOTAR_IMG_SPI_FLASH (0x13)**: Prepare for SUOTA. Image is going to be stored to the FLASH memory. • **SPOTAR_REBOOT (0xFD)**: Reboot the device. • **SPOTAR_IMG_END (0xFE)**: Client sent the whole image. SUOTA service is allowed to perform CRC calculations and other sanity tests to verify that the image transfer was successful.

© 2022 Renesas Electronics

Characteristic	SUOTA Version (SUOTA_VERSION definition)	Access	Size	Description
GPIO MAP		READ WRITE	4	Used to specify GPIO map of external FLASH device. Currently not applicable.
MEM_INFO		READ	4	Stores the total number of bytes received until now.
PATCH_LEN	since version v1.0	READ WRITE	2	Specifies the number of bytes after which they are received, will send a notification back to the client. This is meant to be used for flow control. The exact value is set by the client during SUOTA. The notification is generated from the "STATUS" characteristic.
PATCH_DATA	since version v1.0	READ WRITE WRITE_NO_RESP	SUOTA v1.0, v1.1, v1.2: 120 bytes SUOTA v1.3 and later: Exact size is specified by PATCH_DATA_CHAR_SIZE	This is the endpoint to which SUOTA image data are sent. The default size for SUOTA versions v1.0, v1.1, v1.2 is fixed at 120 bytes. From versions v1.3 and later the exact size is specified by the "PATCH_DATA_CHAR_SIZE" characteristic, and different values (23 – 509) can be used depending on the throughput requirements.
STATUS	since version v1.0	READ NOTIFY	1	This characteristic is used to notify the client of the status of the SUOTA process. Status notifications are sent to indicate error conditions (for example bad command, or CRC) or to allow flow control during SUOTA process.

© 2022 Renesas Electronics

Characteristic	SUOTA Version (SUOTA_VERSION definition)	Access	Size	Description
L2CAP_PSM	since version v1.2	READ	2	This is an optional characteristic that, if it exists, indicates that the SUOTA service supports both SUOTA over GATT and SUOTA over L2CAP CoC. The value indicates the dynamic L2CAP channel on which the SUOTA service is listening for connections. The absence of this characteristic indicates that only SUOTA over GATT is supported.
VERSION	since version v1.3	READ	1	Indicates the version of the SUOTA service. The value is retrieved from the "SUOTA_VERSION" definition.
MTU	since version v1.3	READ	2	Stores the current value of the MTU, which is going to be either 23 (default), or a bigger value, if MTU exchange took place. This value can be used by the client to retrieve the MTU size (if such API is not available on its side) and write with optimal rate to the "PATCH_DATA" characteristic.
PATCH_DATA_CHAR_SIZE	since version v1.3	READ	2	Specifies the size of the "PATCH_DATA" characteristic.
CCC		READ WRITE	1	Client Characteristic Configuration. Allows the client to enable notifications from the "STATUS" source.

Once the SUOTA service is discovered on the remote device and the client has enabled notifications by writing the CCC characteristic, the SUOTA procedure can be started by issuing the SPOTAR_IMG_SPI_FLASH command. The write command executes successfully only if:

- No more than one device is currently connected to the SUOTA enabled device
- The application hosted in the SUOTA enabled device allows the upgrade to take place
- There is enough memory to allocate the internal working buffers

If any of the above restrictions is violated, then command fails and an error notification is sent back to the client (status SUOTA_SRV_EXIT). After a successful command execution the service is able to receive data either using GATT or L2CAP CoC layer (if the L2CAP_PSM characteristic is available).

© 2022 Renesas Electronics

On SUOTA v1.3 and later, the client can use the value of the characteristic "MTU" to perform ATT write commands to the characteristic PATCH_DATA with optimal size if the client itself has no API to find the optimal packet size. On previous versions the client can either retrieve the MTU value using an OS specific command, or use the default minimum value which is 23 bytes.

On SUOTA v1.3 and later the client can find the size of the PATCH_DATA characteristic by reading the PATCH_DATA_CHAR_SIZE characteristic. On previous versions the size of PATCH_DATA was fixed to 120 bytes.

Following this, the client should specify the value of the patch_len variable by writing the PATCH_LEN characteristic. PATCH_LEN specifies the number of bytes that once received, triggers a notification back to the client. This kind of flow control could be used by the client to avoid flooding the SUOTA enabled device with too much image data. The bigger the value, the better the throughput, since notifications are going to be generated less frequently and therefore the number of missed connection events (where flow has stopped waiting for the notification) is decreased.

For example, if patch_len is set to 500 bytes, notification are going to be sent to the client when byte ranges 1-500, 501-1000, 1001 – 1500 etc. are received. Following the Bluetooth low energy specification, the maximum number of bytes that can be written to the PATCH_DATA characteristic with a single ATT write command is the minimum of MTU – 3 and the size of the PATCH_DATA characteristic.

When the whole image has been sent, the client should issue the SPOTAR_IMG_END command to indicate this to the SUOTA service. The service is going to perform some sanity checks to verify that image transfer took place without errors, and then it is going to generate the appropriate status notification (SUOTA_CMP_OK on success, SUOTA_APP_ERROR or SUOTA_CRC_ERR on error).

Finally, the client could issue an SPOTAR_REBOOT command to force a device reboot. This step is optional, but it is highly recommended.

Note 8 The PATCH_DATA, PATCH_DATA_CHAR_SIZE and PATCH_LEN characteristics are only relevant when SUOTA over GATT is taking place. When L2CAP CoC are used, a connection should be established to the L2CAP_PSM channel via L2CAP CoC and the flow is controlled using L2CAP credits. SUOTA service assigns enough credits to ensure that flow won't stop during the upgrade. Notifications relevant to the PATCH_LEN characteristic are not sent during image transfer, but all other notifications are still valid.

9.1.3 SUOTA Flow

When the software update process is initiated by the SUOTA mobile application, the SUOTA-enabled application downloads the new image and reboots the system. During boot, SUOTA loader verifies the new image, copies it into the executable partition and starts execution as shown in Figure 42.

- **SUOTA enabled application code**

  The execution of a SUOTA-enabled application always starts from the same address. As a result of this during the update, the new application image is stored in a separate location in the Flash memory in the Firmware update partition. After the new image is verified, it must be copied to the Executable Flash partition. Since the application runs from Flash memory, it is impossible to safely overwrite itself with a new image, therefore this part of the update is conducted by the SUOTA loader after the reboot.

- **Bootloader**

  After each reboot, SUOTA loader checks for a valid application image in the Firmware Update partition. If a SUOTA update was performed during the previous run then the new image would have been stored in the Firmware Update partition as part of the update process. The SUOTA loader detects this new image, verifies its checksum and copies it to the Executable partition. When the copy is completed, the SUOTA loader updates the Image header partition with the new image information. Finally, the image data in the Firmware Update partition is marked as invalid so that it is ignored by the loader on a subsequent reboot.

Figure 42 and Figure 43 presents an outline of the overall SUOTA process.

© 2022 Renesas Electronics

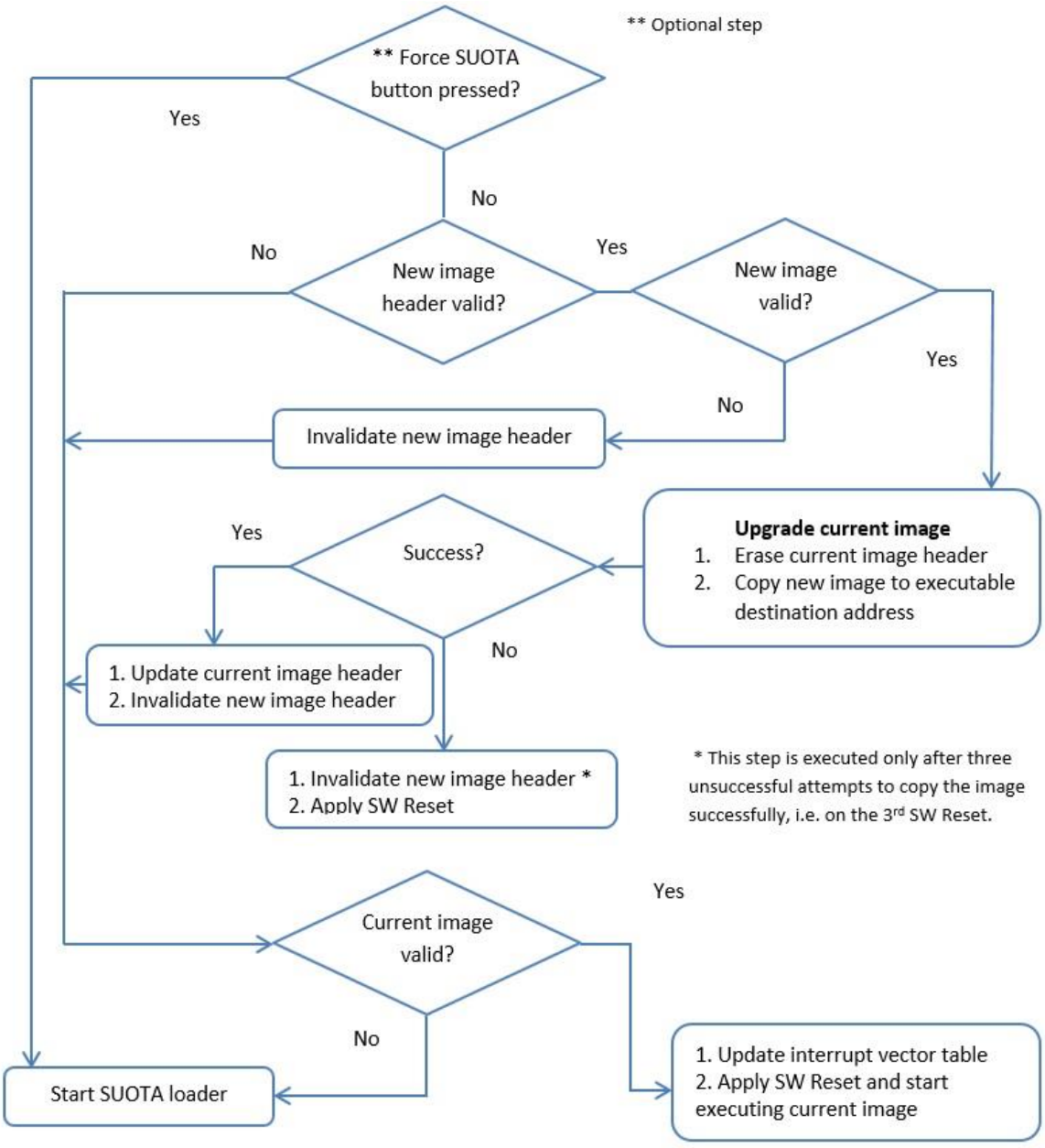


Figure 42: BLE SUOTA loader

© 2022 Renesas Electronics

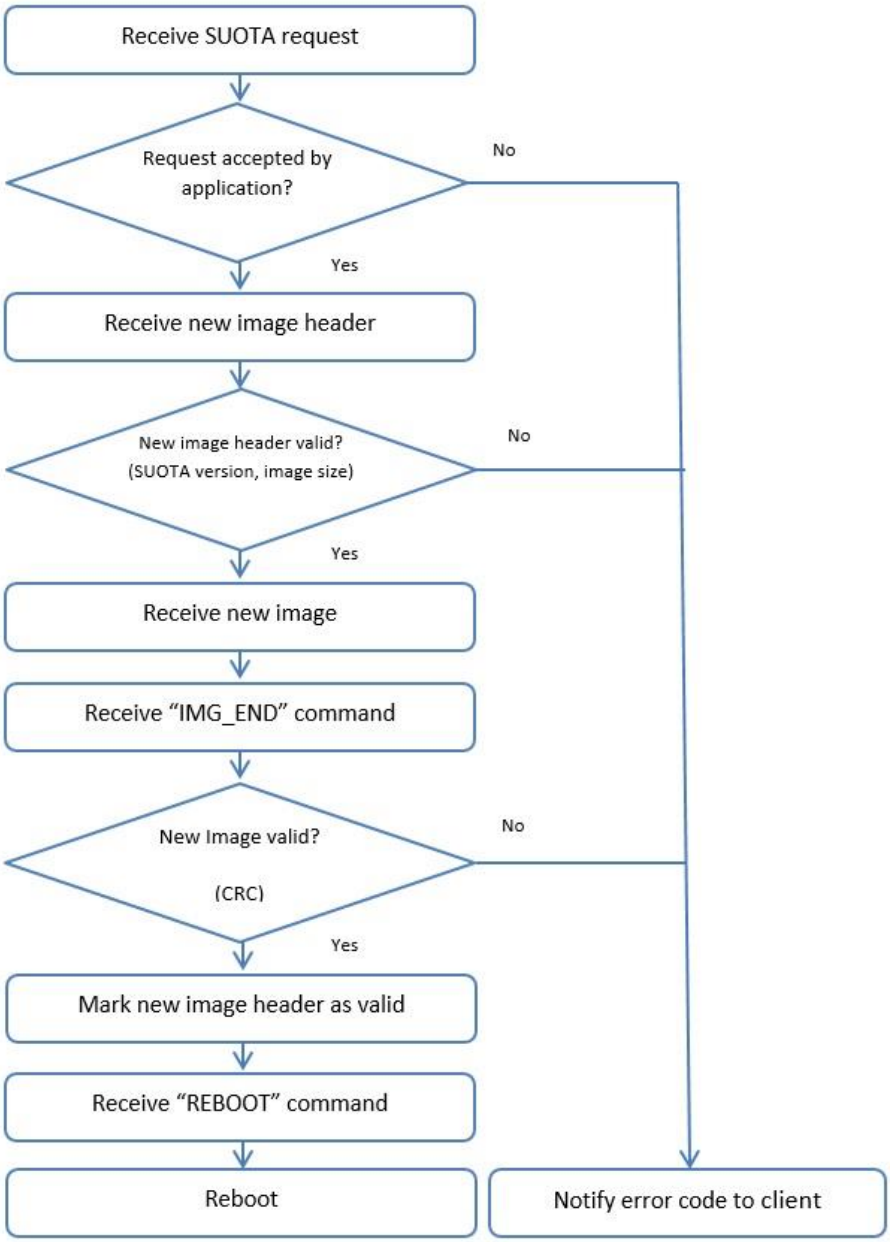


Figure 43: BLE SUOTA Service

© 2022 Renesas Electronics

9.1.4 SUOTA Flash memory layout

The DA1468x platform uses partitions to divide the Flash memory into smaller sections. The NVMS layer provides unified access to those partitions. The applications can use the partition access API to read and write to the Flash memory partitions. The API also allows the modification of the size and position of partitions. The partition layout differs significantly between a SUOTA enabled build and a non-SUOTA enabled build as shown in Figure 44.

To update the software, both the SUOTA enabled application and the SUOTA bootloader need to know the location of the downloaded image in Flash memory. The following partitions are used by an application that supports software update:

- Bootloader partition, contains the bootloader that manages the update process if a new firmware executable has been uploaded.
- Product header partition (a partition with information about a device)
- Image header partition with software version information
- Application executable partition, contains the current application firmware version. In a SUOTA application this is limited to 320kB.
- Firmware update partition, this contains the new updated firmware version that the bootloader will detect on the next reboot. Practically this is limited to 320kB as well by the size of the application executable partition.

The SUOTA Partition layout is color coded in Figure 44. The yellow partitions are the ones that are modified during the update procedure, the blue partitions are accessed during the SUOTA update while green ones that remain intact throughout the update.

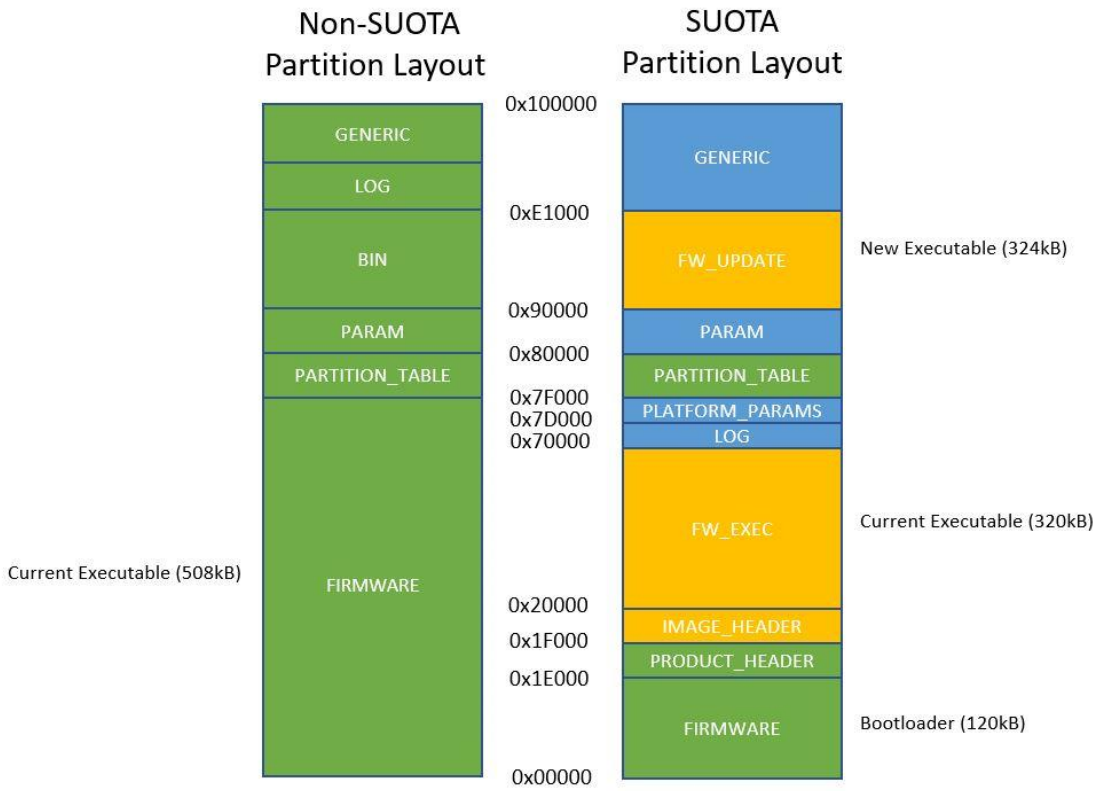


Figure 44: Flash memory partition layout comparison between SUOTA and non-SUOTA build (1Mbyte QSPI Flash)

The SUOTA Flash memory layout is defined in `sdk/bsp/config/1M/suota/partition_table.h` in the `ble_suota_loader` project (the non-SUOTA version is found at `sdk/bsp/config/1M/partition_table.h`.).

 © 2022 Renesas Electronics

Code 9 shows how the partition table is defined in the SUOTA `partition_table.h`. Product and Image header description are given in Table 15 and Table 16 respectively.

```
PARTITION2( 0x000000,0x01E000,NVMS_FIRMWARE_PART       ,0 )
PARTITION2( 0x01E000,0x001000,NVMS_PRODUCT_HEADER_PART ,0 )
PARTITION2( 0x01F000,0x001000,NVMS_IMAGE_HEADER_PART   ,0 )
PARTITION2( 0x020000,0x050000,NVMS_FW_EXEC_PART        ,0 )
PARTITION2( 0x070000,0x00D000,NVMS_LOG_PART            ,0 )
PARTITION2( 0x07D000,0x002000,NVMS_PLATFORM_PARAMS_PART,PARTITION_FLAG_READ_ONLY )
PARTITION2( 0x07F000,0x001000,NVMS_PARTITION_TABLE     ,PARTITION_FLAG_READ_ONLY )
PARTITION2( 0x080000,0x010000,NVMS_PARAM_PART          ,0 )
PARTITION2( 0x090000,0x051000,NVMS_FW_UPDATE_PART      ,0 )
PARTITION2( 0x0E1000,0x01F000,NVMS_GENERIC_PART        ,PARTITION_FLAG_VES )
```

Code 9: Flash memory partition table

Table 15: Product header description

Size	Description
2 bytes	Container Identifier (0x70 0x62)
2 bytes	Flags
4 bytes	Absolute address indicating the location of the current Image
4 bytes	Absolute address indicating the location of the Image Update
8 bytes	Reserved

Table 16: Image header description

Size	Description
2 bytes	FW Image Identifier (0x70 0x61)
2 bytes	Flags
4 bytes	Executable Size
4 bytes	CRC
16 bytes	Version String
4 bytes	Image creation timestamp
4 bytes	Executable location

9.1.5 Performing SUOTA upgrade using a mobile phone

Note 9 The following procedure applies when using Android or iOS devices

The Proximity Reporter application described in section 5 can also be built with SUOTA support. To add SUOTA a different build procedure needs to be followed:

1. Import the following three projects into SmartSnippets™ Studio from these locations.

© 2022 Renesas Electronics

```
scripts:            <sdk_root_directory>\utilities
ble_suota_loader:   <sdk_root_directory>\sdk\bsp\system\loaders
pxp_reporter:       <sdk_root_directory>\projects\dk_apps\demos
```

2. Build the two source code projects in the following configurations:

   - `ble_suota_loader` in `DA14681-01-Release_QSPI` configuration and
   - `pxp_reporter` in `DA14681-01- Release_QSPI_SUOTA` configuration.

3. Create a SUOTA image. A SUOTA image is a binary file with a proper header that can be sent to a target device from an Android or iOS device. To create the image, build PXP Reporter project, open a command prompt and navigate to `<sdk_root_directory>/projects/dk_apps/demos/pxp_reporter` folder.
4. On Windows run the MKIMAGE script with the following command (Figure 45):

```
>   mkimage.bat DA14681-01-Release_QSPI_SUOTA
```

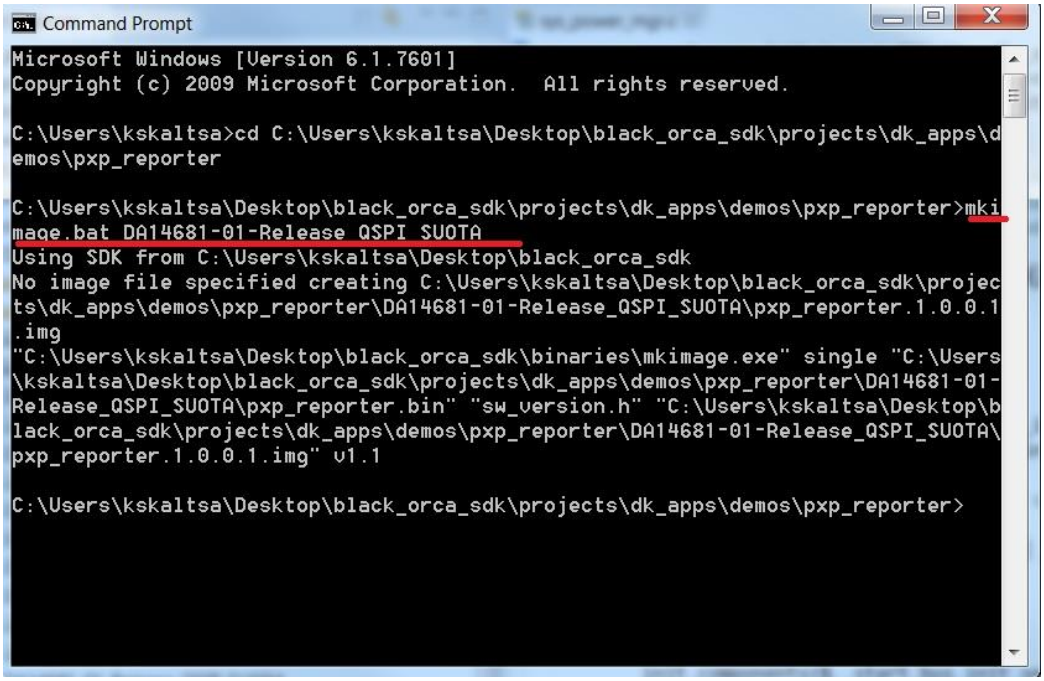


Figure 45: Run mkimage.bat script on Windows

5. On Linux run the mkimage.sh script with the following command (Figure 46)

```
$ ./mkimage.sh DA14681-01-Release_QSPI_SUOTA
```


© 2022 Renesas Electronics

```
iain@iain-Precision-WorkStation-T3500:~/Desktop/work/dialog/DA1468x_DA15xxx_SDK_
1.0.10.1072-clean/projects/dk_apps/demos/pxp_reporter$ ./mkimage.sh DA14681-01-R
elease_QSPI_SUOTA/
Using SDK from /home/iain/Desktop/work/dialog/DA1468x_DA15xxx_SDK_1.0.10.1072-cl
ean
No image file specified creating DA14681-01-Release_QSPI_SUOTA//pxp_reporter.1.0
.0.1.img
iain@iain-Precision-WorkStation-T3500:~/Desktop/work/dialog/DA1468x_DA15xxx_SDK_
1.0.10.1072-clean/projects/dk_apps/demos/pxp_reporter$
```

Figure 46: Run mkimage.sh on Linux

6. A new image named `pxp_reporter.1.0.0.1.img`, containing a version number taken from sw_version.h, is created under `pxp_reporter/DA14681-01-Release_QSPI_SUOTA` folder as shown in Figure 47.

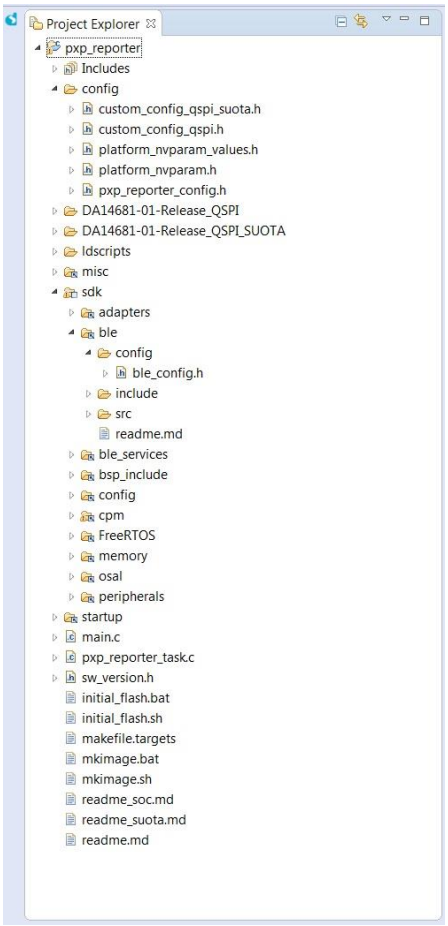


Figure 47: Project directory

7. Download the Dialog SUOTA application from Google PlayStore or Apple App Store.
8. Copy `pxp_reporter.1.0.0.1.img` to an Android phone or tablet or to an iOS device and placed into the SUOTA folder. The folder is automatically created, if it does not exist, on the device by running the "Dialog Suota" application. On Android it is located at the root directory of the "Internal Storage" drive.
9. Erase the Flash memory of DA1468x using the `erase_qspi_jtag_win` script (to ensure the correct partition table is used) and then download `ble_suota_loader` and `pxp_reporter` binaries to DA1468x using `suota_initial_flash_jtag_win` script. This script download both ble_suota_loader and `pxp_reporter` binaries on partitions `FIRMWARE_PART` (bootloader) and `FW_EXEC_PART` respectively.

© 2022 Renesas Electronics

Press the K2 Reset button on the ProDK board. The bootloader (ble_suota_loader) should start executing the pxp_reporter image. Before executing suota_initial_flash_jtag_win ensure pxp_reporter is the selected project at "project explorer"

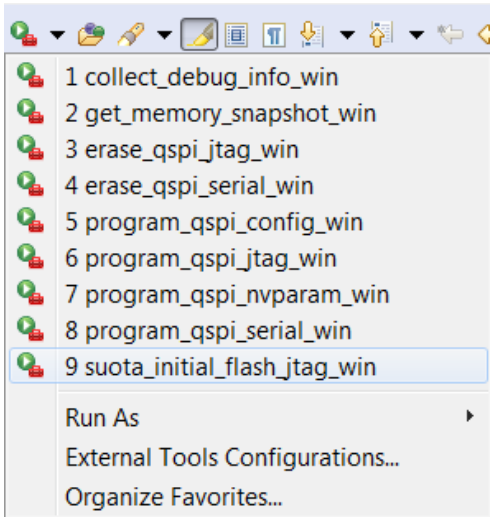


Figure 48: Scripts

10. Launch the Dialog SUOTA application on the Android phone and select the DA1468x device you want to update.

Figure 49: Device selection

11. Select **Update device**.

© 2022 Renesas Electronics

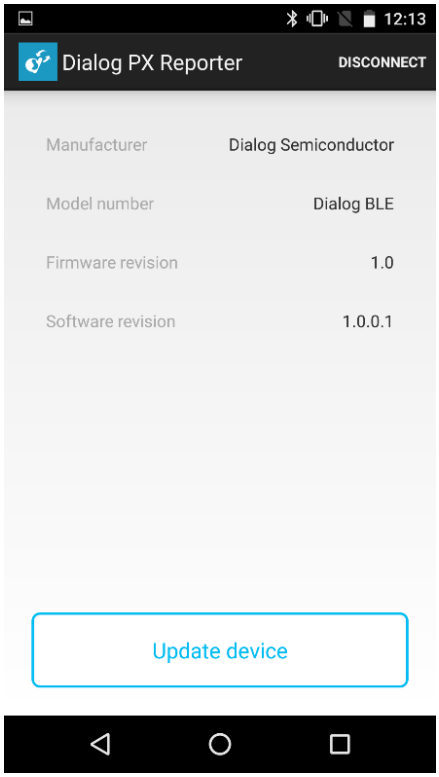


Figure 50: Update device

12. Select the appropriate image file – this is a list of the files in the SUOTA directory.

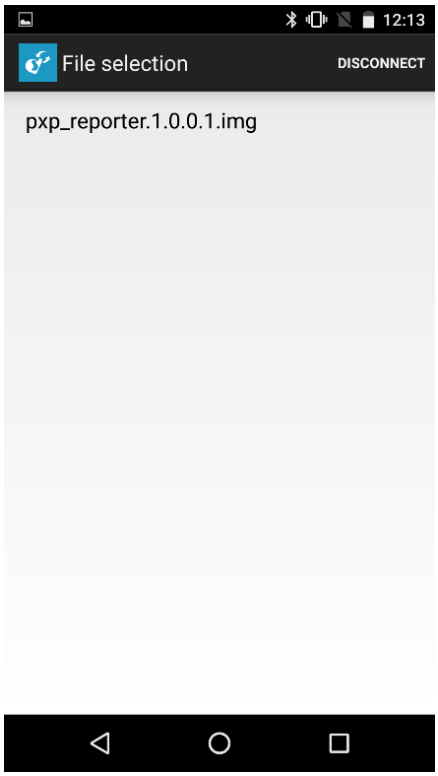
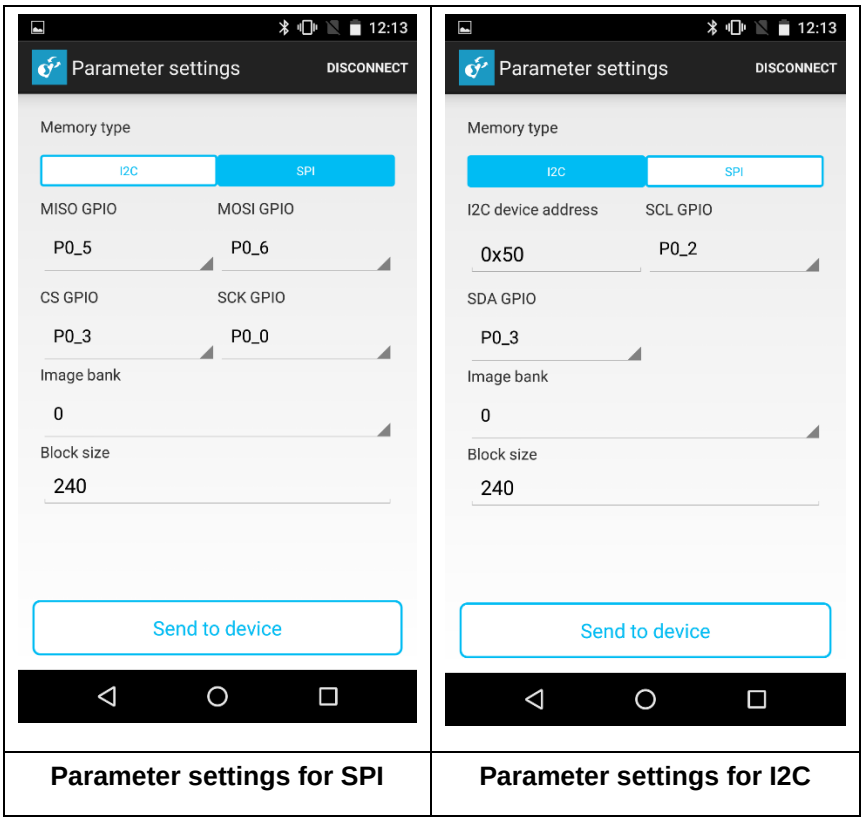


Figure 51: Image file

© 2022 Renesas Electronics

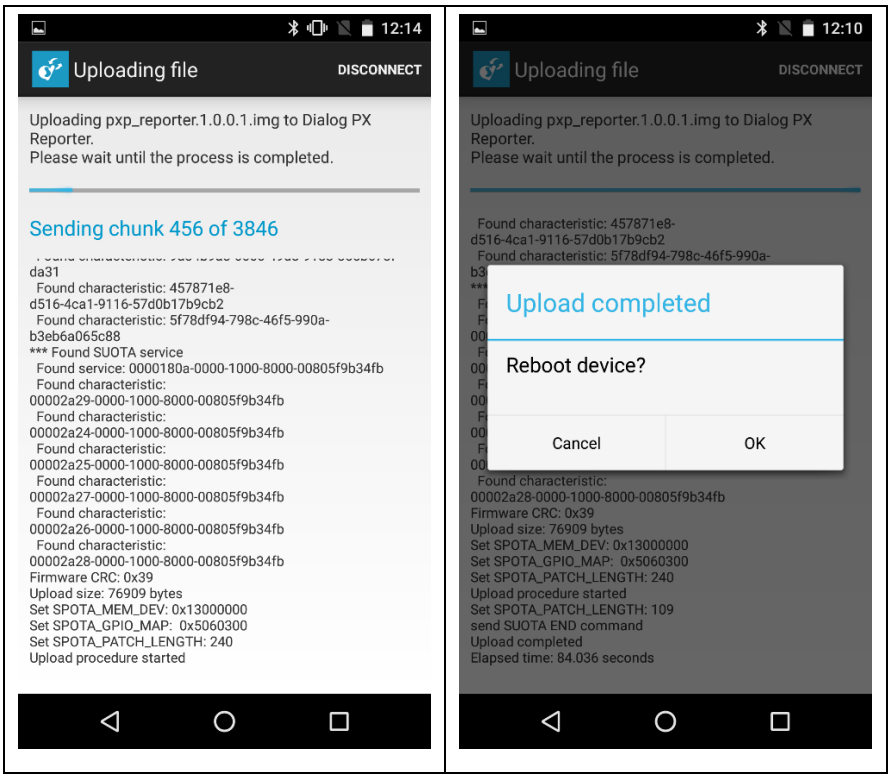
13. This screen is required only by DA1458x devices. For the DA1468x just press **Send to device** as whatever values are here have no effect.



Parameter settings for SPI | **Parameter settings for I2C**

Figure 52: Parameter settings (ignore for DA1468x)

14. Wait until the process is completed. When the image is uploaded, a dialog box pops up and ask to reboot the device. Select **OK**.



© 2022 Renesas Electronics

Uploading the image file	Reboot device

Figure 53: Uploading image and reboot

15. Press **Close** to return to the main menu.

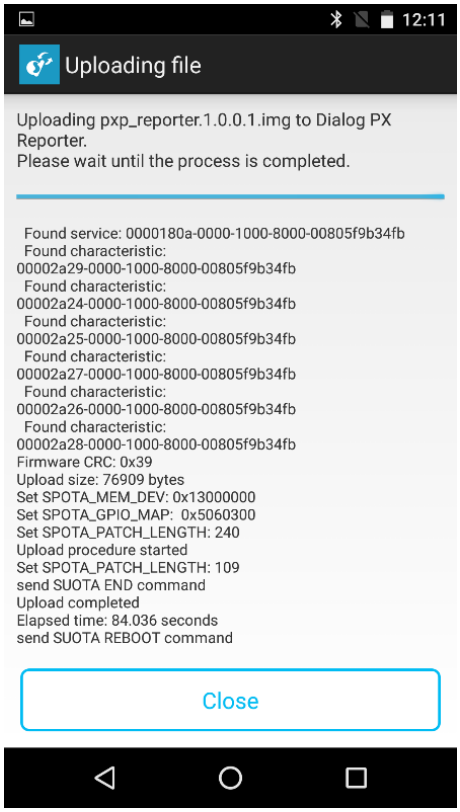


Figure 54: When file upload is finished, press "Close"

9.1.6 Performing SUOTA upgrade using two DA1468x

This section describes the procedure for performing SUOTA using two DA1468x devices.

- one acting as Bluetooth low energy central. It performs as the SUOTA image transmitter running `ble_suota_client`
- one acting as Bluetooth low energy peripheral. It performs as the SUOTA image receiver, initially running `ble_suota_loader` and after first successful SUOTA running `pxp_reporter`.

Using this setup, it is possible to test both SUOTA methods (over GATT and over L2CAP Connection-Oriented Channels) without using any phone. The image to be transferred is stored in the NVMS_BIN_PART partition (Figure 44) in the Flash memory of the BLE central device.

Import the following projects using SmartSnippets™ Studio from the following locations:

```
scripts:            <sdk_root_directory>\utilities

ble_suota_client:   <sdk_root_directory>\projects\dk_apps\features

ble_suota_loader:   <sdk_root_directory>\sdk\bsp\system\loaders

pxp_reporter:       <sdk_root_directory>\projects\dk_apps\demos
```

There are several configuration changes to the SDK required to run this demo.

© 2022 Renesas Electronics

- To communicate successfully between the two devices they need to have different BD addresses. This can be achieved by overriding the address of the device running `ble_suota_loader` by adding this line to its `config/custom_config_qspi.h`:

  ```
  #define defaultBLE_STATIC_ADDRESS        {0x02,0x00,0x80,0xCA,0xEA,0x80}
  ```

- As delivered `ble_suota_loader` has logging disabled, so it is difficult to track if the device is running correctly. To enable logging, change this line in `config/custom_config_qspi.h` to:

  ```
  #define dg_configDEBUG_TRACE                1
  ```

To enable SUOTA over L2CAP Connection-Oriented Channels (COC), both `SUOTA_VERSION` and `SUOTA_PSM` should be defined in the `config/custom_config_qspi.h` in both `ble_suota_loader` and `pxp_reporter` images. This is done by default in SDK.

The overall architecture of the SUOTA demo is shown in Figure 55. The Central device is not enabled for SUOTA, so it is using the normal partition layout (Figure 44).

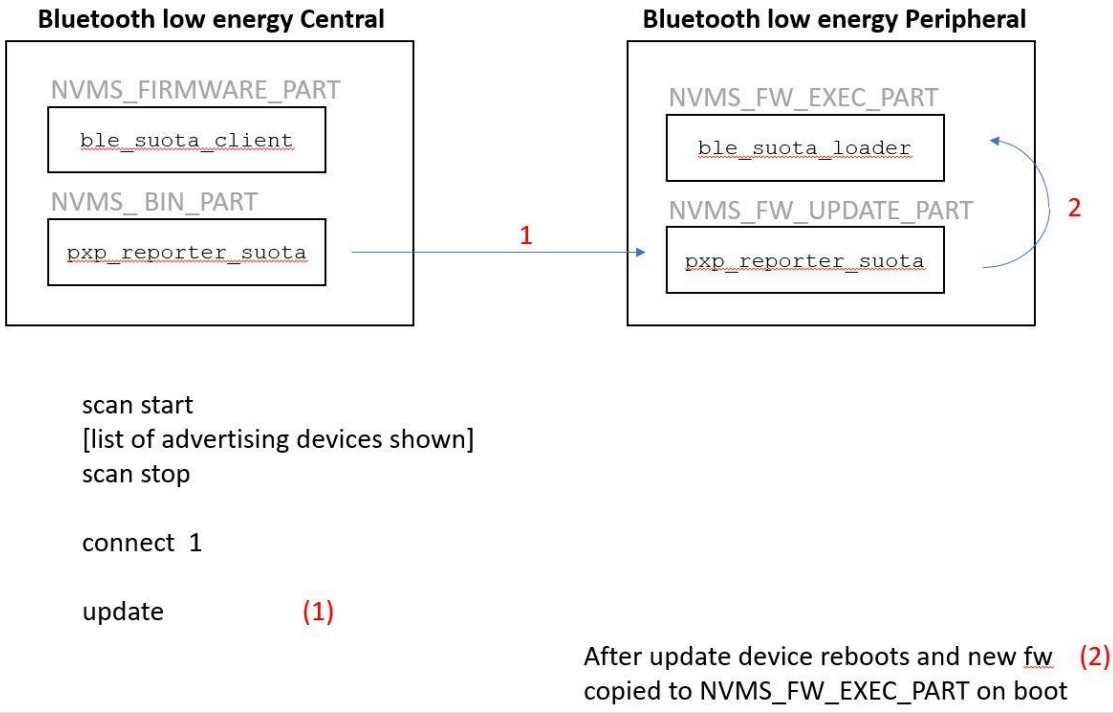


Figure 55: Dual SUOTA architecture

9.1.6.1 Building the Bluetooth low energy Central device

1. Select the build configuration for each of these projects
   - `ble_suota_client` in "`DA14681-01-Release_QSPI`" configuration
   - `pxp_reporter` in "`DA14681-01-Release_QSPI_SUOTA`" configuration.

The aim is to program `ble_suota_client` into the executable partition `NVMS_FW_EXEC_PART` so that it runs and then to put the update image to be transmitted by SUOTA in the `NVMS_FW_UPDATE_PART`.

To prepare the DA1468x acting as Bluetooth low energy central device follow the procedure below:

1. Build the project `ble_suota_client` (See Figure 56).

© 2022 Renesas Electronics

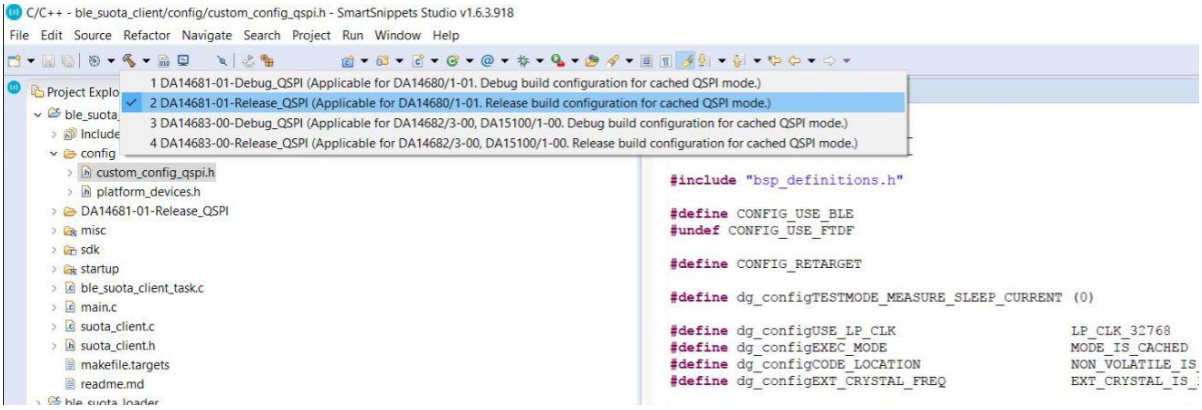


Figure 56: Building `ble_suota_client`

2. Select the External Tool Configurations button, choose the appropriate script file (in Figure 57 the `program_qspi_jtag_win` script is used) to program the QSPI Flash memory and execute it.

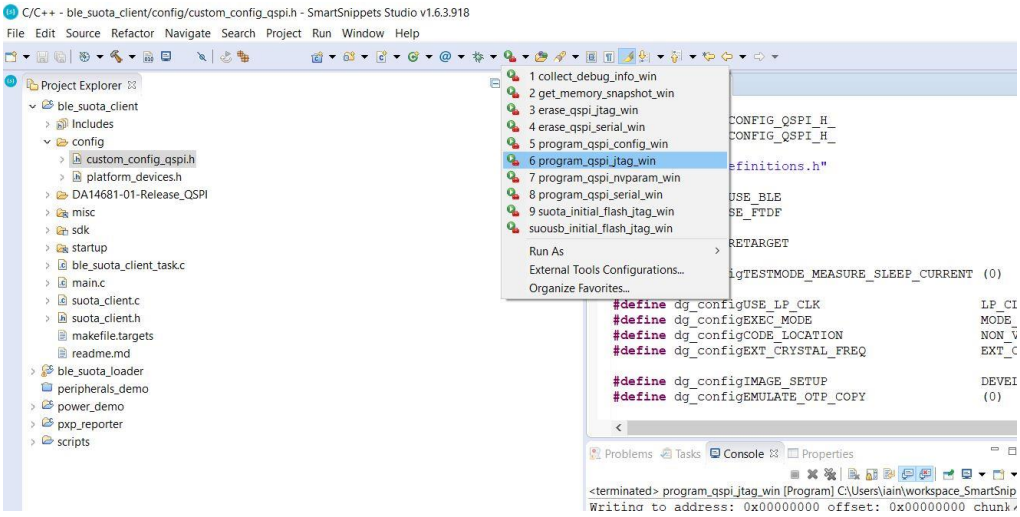


Figure 57: Flash the `ble_suota_client` binary to the QSPI Flash

As soon as the script is executed, a new window pops up asking to choose which of the DKs is the target; select the appropriate device.

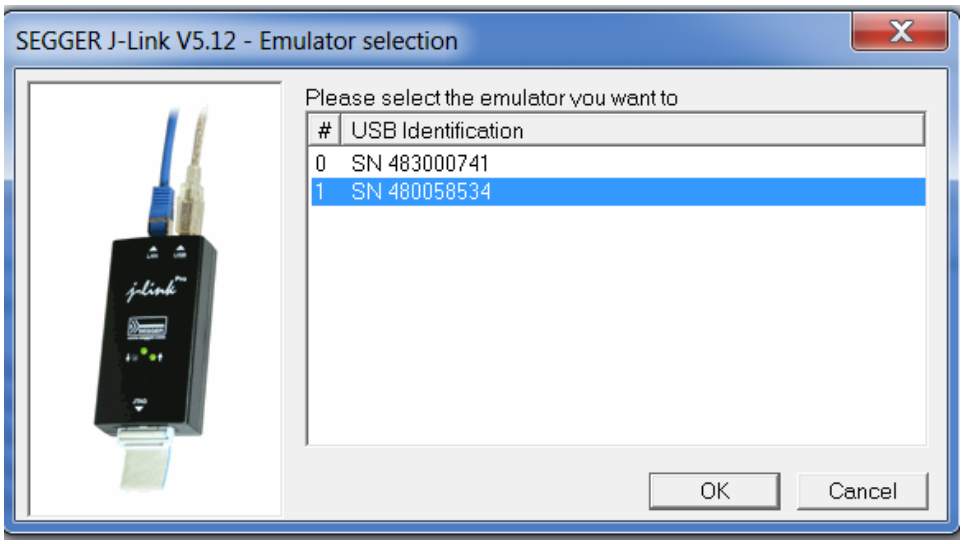


Figure 58: Selecting the target device

© 2022 Renesas Electronics

3. Build the `pxp_reporter` project using the `Release_QSPI_SUOTA` build configuration.

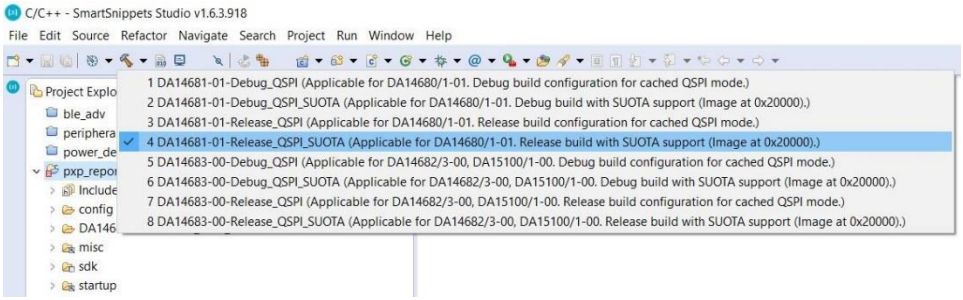


Figure 59: Building pxp_reporter for SUOTA

4. Create a SUOTA image. To create the image, open a command prompt, navigate to `<sdk_root_directory>/projects/dk_apps/demos/pxp_reporter` folder and run the following command (Figure 60) on Windows:

```
>   mkimage.bat DA14681-01-Release_QSPI_SUOTA
```

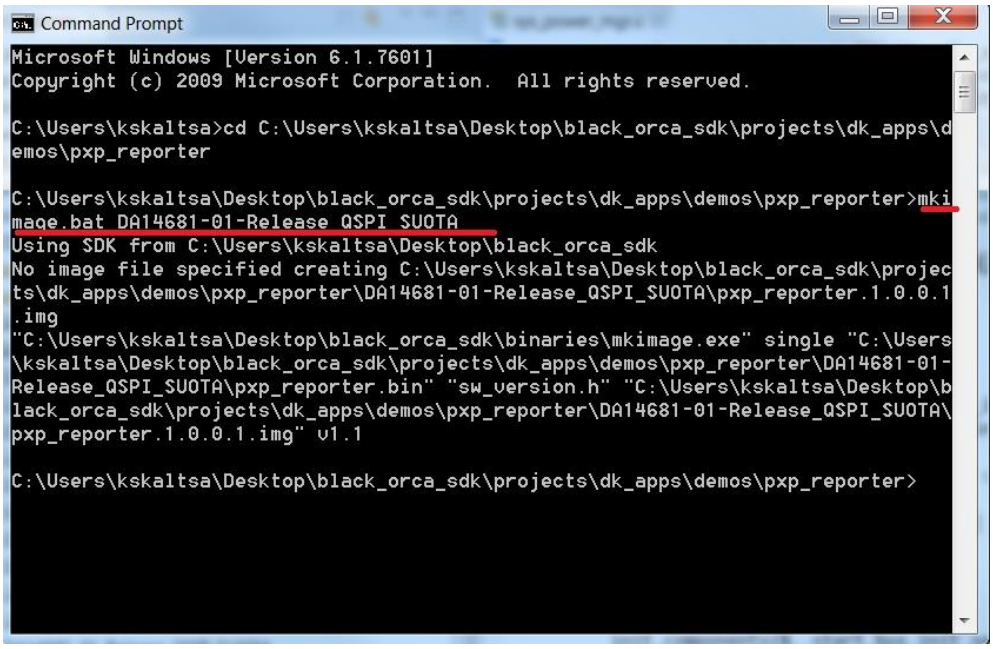


Figure 60: Creating image

For Linux use the command

```
$ ./mkimage.sh DA14681-01-Release_QSPI_SUOTA
```

5. Copy the image from `<sdk_root_directory>/projects/dk_apps/demos/pxp_reporter/DA14681-01-Release_QSPI_SUOTA\` and paste it to `<sdk_root_directory>/binaries`
6. Use `cli_programmer` to download the binary pxp_reporter.1.0.0.1.img to the `NVMS_FW_UPDATE_PART` partition. Open a command prompt, navigate to the `<sdk_root_directory>\binaries` folder and run one of the following commands

© 2022 Renesas Electronics

- On a ProDK make sure that CTS is not connected on the UART by ensuring there is no jumper on J15.7-8 as shown in Figure 61.
- On a BasicDK use the no flow control configuration described in Figure 15.

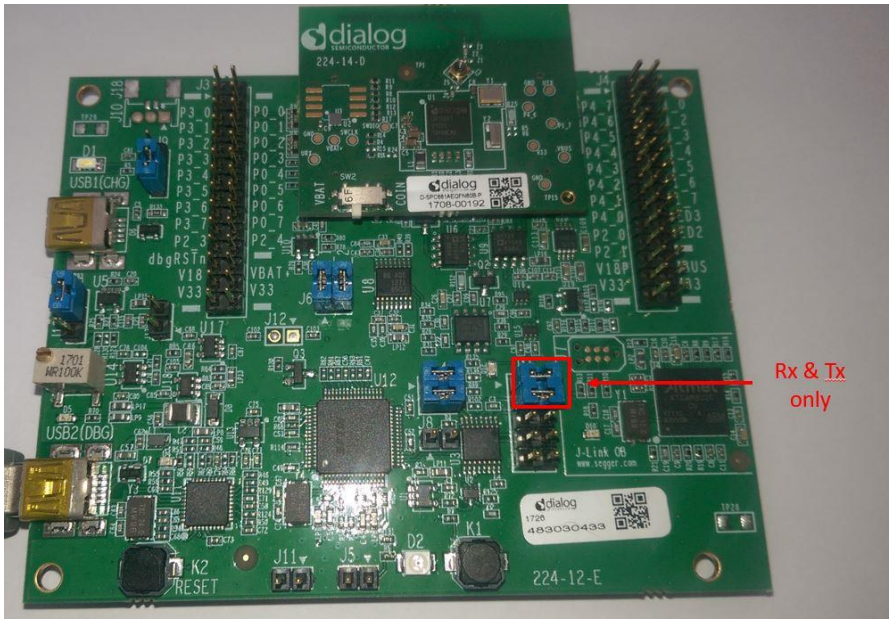


Figure 61: Jumpers only on Rx and Tx for ProDK no Flow Control

To download `pxp_reporter.1.0.0.1.img` using UART as shown in Figure 62. During this procedure the user is asked to press the `K2` Reset button.

```
> cli_programmer.exe COM13 write_qspi 0x00090000 pxp_reporter.1.0.0.1.img
```

To download `pxp_reporter.1.0.0.1.img` using SWD

```
> cli_programmer.exe gdbserver write_qspi 0x00090000 pxp_reporter.1.0.0.1.img
```

```
Command Prompt
C:\Users\iain\workspace_SmartSnippets_Studio-doc\binaries>cli_programmer COM13 write_qspi 0x00090000 pxp_reporter.1.0.0.1.img
cli_programmer 1.24
Copyright (c) 2015-2017 Dialog Semiconductor

Using serial port COM13 at baud rate 57600.
bootloader file not specified, using internal uartboot.bin

Connecting to device...
Setting serial port baud rate to 57600.
Press RESET.
Uploading boot loader/application executable...
Executable uploaded.
Setting serial port baud rate to 57600.
Writing to address: 0x00090000 offset: 0x00000000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00002000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00004000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00006000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00008000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x0000a000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x0000c000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x0000e000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00010000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00012000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00014000 chunk size: 0x00002000
Writing to address: 0x00090000 offset: 0x00016000 chunk size: 0x00000134
done.

C:\Users\iain\workspace_SmartSnippets_Studio-doc\binaries>
```

Figure 62: Uploading image to the Client

© 2022 Renesas Electronics

On a Linux host the equivalent commands are for a Serial download (Jumper J15.7-8 on Pro DK must not be installed for this to work)

```
$ ./cli_programmer.sh COM13 write_qspi 0x00090000 pxp_reporter.1.0.0.1.img
```

And for SWD

```
$ ./cli_programmer.sh gdbserver write_qspi 0x00090000 pxp_reporter.1.0.0.1.img
```

9.1.6.2 Building the Bluetooth low energy peripheral device

1. Build the project `ble_suota_loader` in "`DA14681-01-Release_QSPI`" configuration.

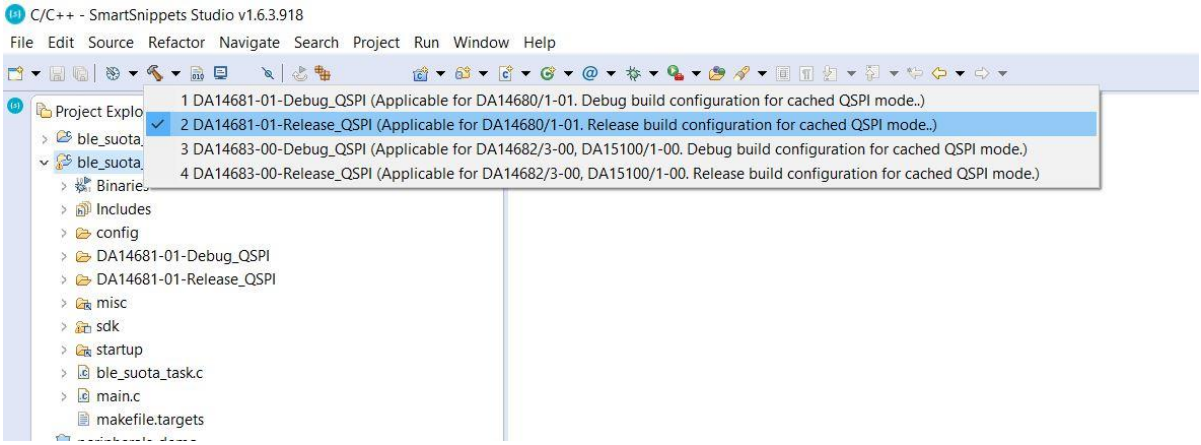


Figure 63 : Building ble_suota_loader project

2. Select the External Tool Configurations menu, choose the appropriate script file (in this example, the `program_qspi_jtag_win` script is used) to program the QSPI Flash memory, and execute/run the script.

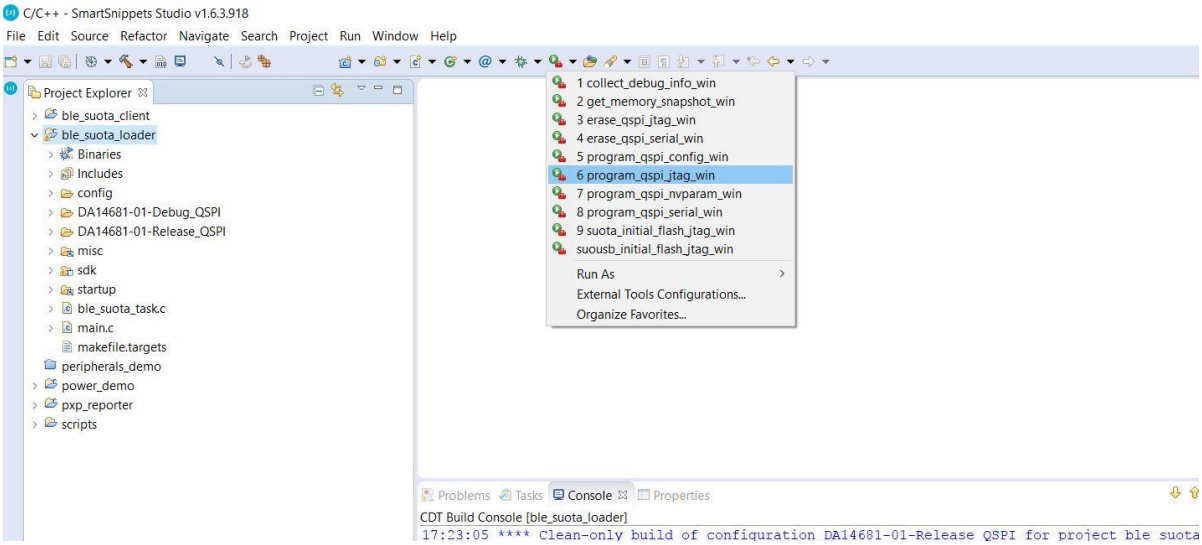
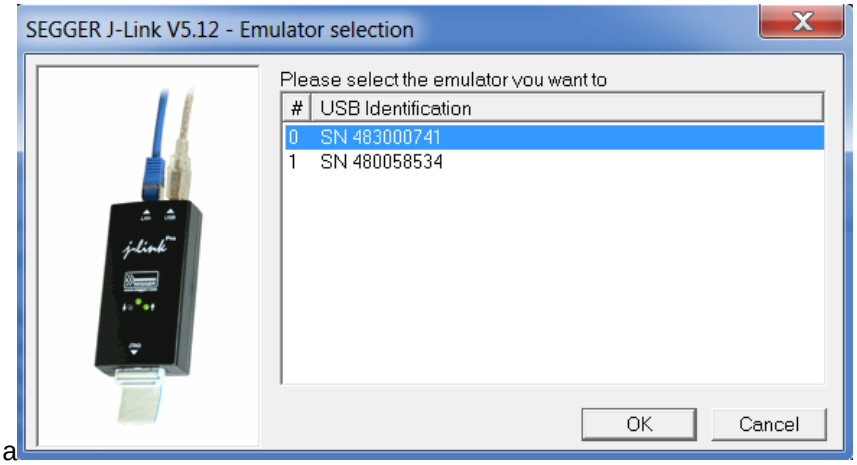


Figure 64: Flashing ble_suota_loader to QSPI Flash

As soon as the script is executed, a new window pops up asking to choose which device is the target; select the appropriate device.

© 2022 Renesas Electronics



a

Figure 65: Selecting the target device

3. Press the K2 Reset button.

9.1.6.3 Running the software upgrade procedure

When the previous procedure has finished, the two DA1468x devices are ready to communicate. To load the image, the following steps should be followed.

A Pro DK running the Bluetooth low energy central device needs to have a jumper put on J15.7-8 to connect the CTS line (Figure 66) for `ble_suota_client` to run correctly. This jumper must be removed if UART is used to reprogram a different update image with `cli_programmer`.

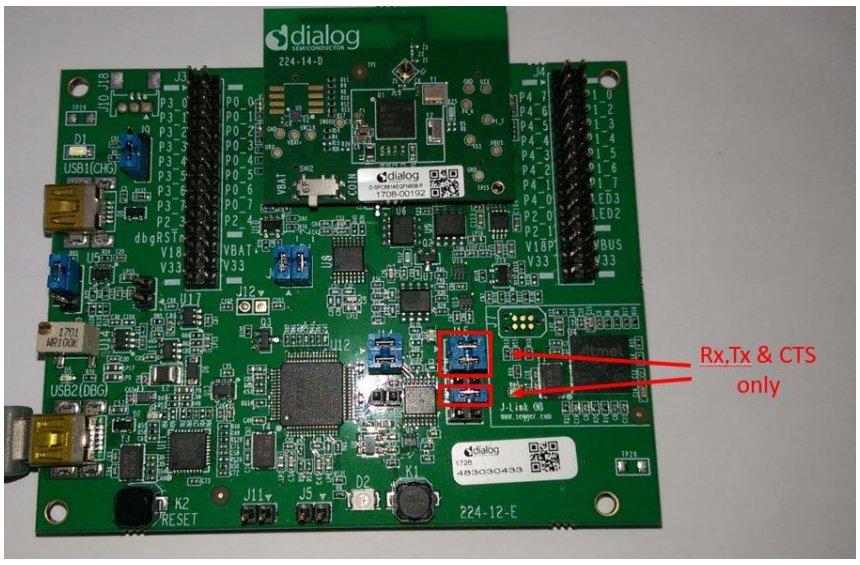


Figure 66: Jumpers on Rx, Tx and CTS

A Basic DK running the Bluetooth low energy central device needs to CTS line driven as shown in Figure 26) for `ble_suota_client` to run correctly. The jumper wire connecting CTS to GND must be removed if UART is used to reprogram a different update image with `cli_programmer`.

Open two terminals, one for each device. A serial terminal is needed to control each device. In the example below, "TeraTerm" is used for this purpose

Configure serial terminal for the Bluetooth low energy central device:

- Select the serial port number for the client.

© 2022 Renesas Electronics

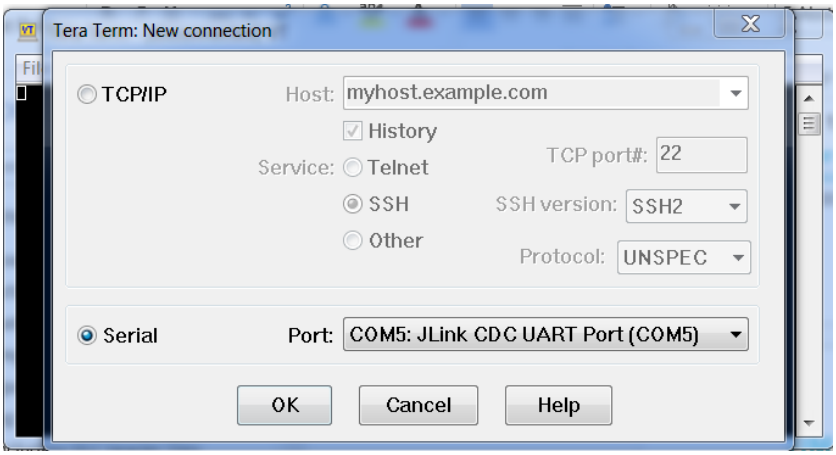


Figure 67: Specifying the serial port number

- From the Menu bar choose `Setup > Serial port ...` and configure the serial port as **115200-8-n-1** as shown in Figure 68.

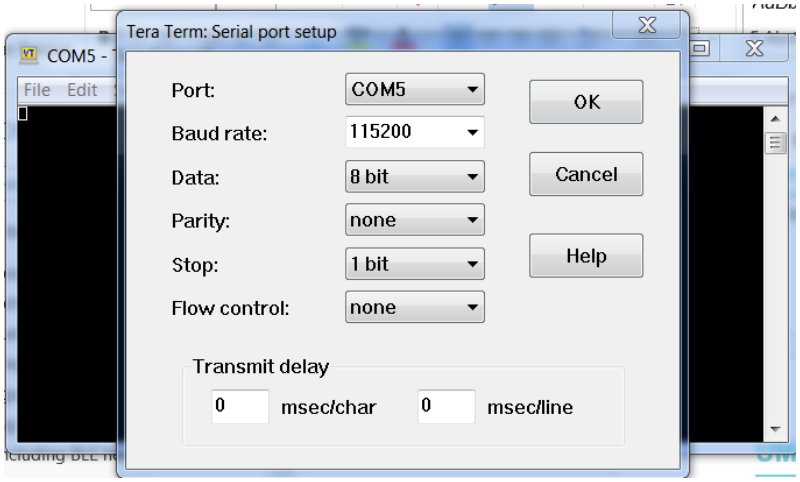


Figure 68: Configuring the serial port

- Press the K2 RESET button.

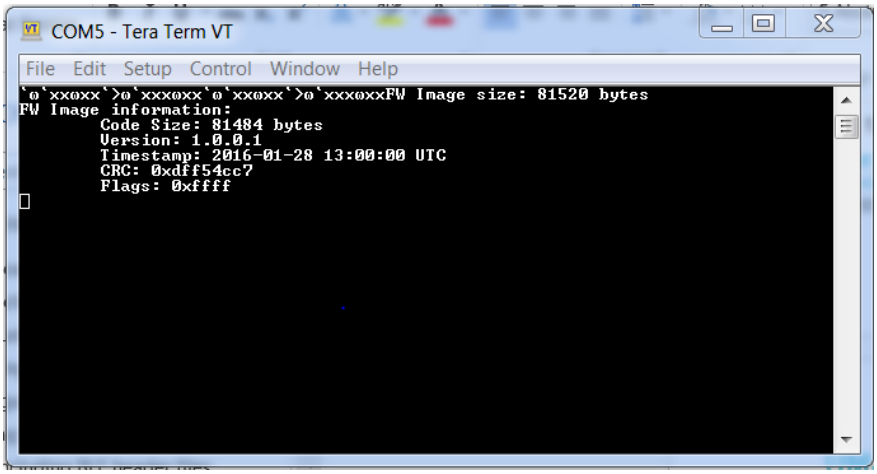


Figure 69: Information regarding the image stored in the NVMS_BIN_PART partition are displayed during boot

© 2022 Renesas Electronics

The Bluetooth low energy peripheral device does not have flow control on its UART so on a ProDK on the virtualCOM port configure as shown in Figure 61Figure 24 and on a BasicDK with an FTDI cable connect as in Figure 15.

- Select the appropriate serial port number for the loader.

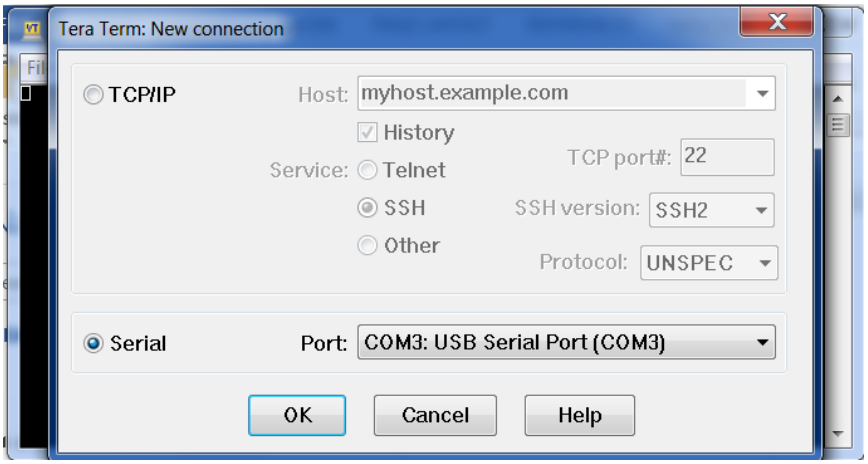


Figure 70: Specifying the serial port number

- From the Menu bar choose `Setup > Serial port ...` and configure the serial port as **115200-8-n-1** as shown in Figure 71.

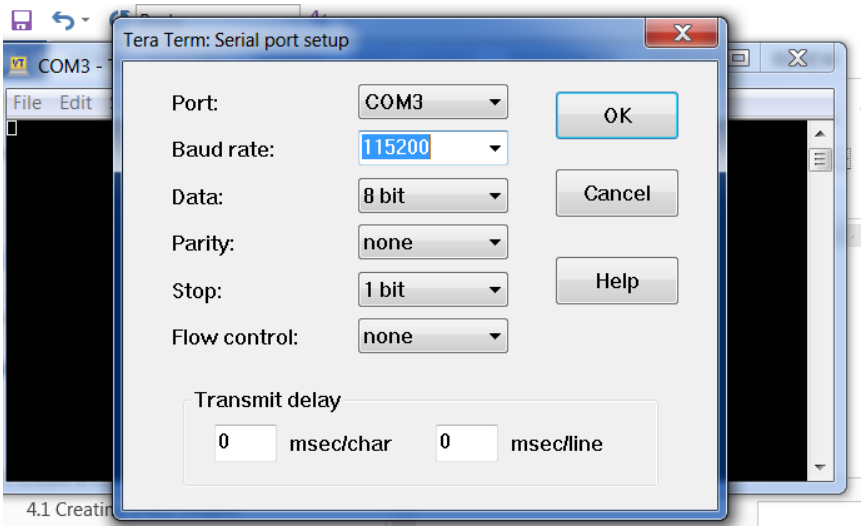


Figure 71: Configuring the serial port

- Press the `K2` Reset button.

© 2022 Renesas Electronics

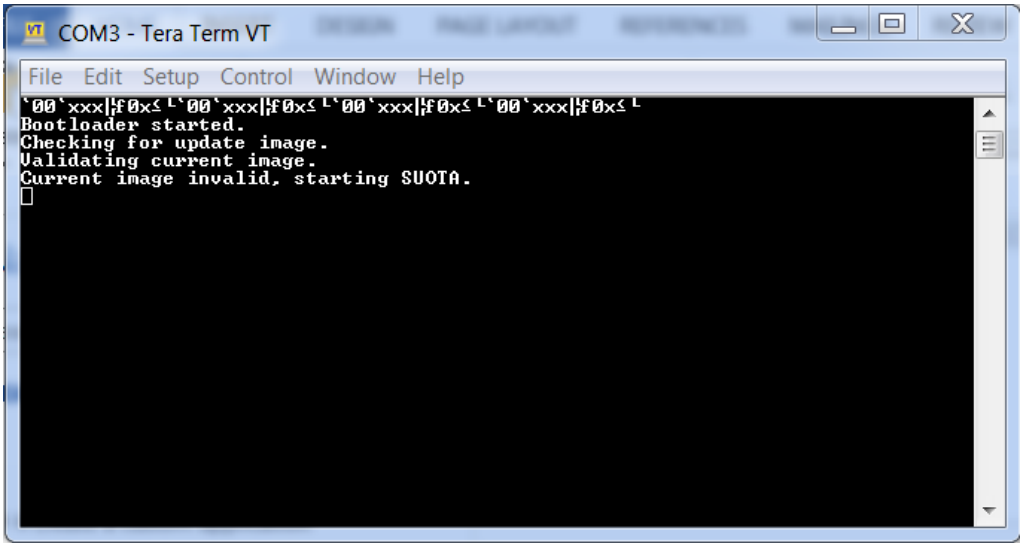


Figure 72: ble_suota_loader information is displayed during boot

- In the serial terminal of the Bluetooth low energy central device, write the following command:

```
> scan start
```

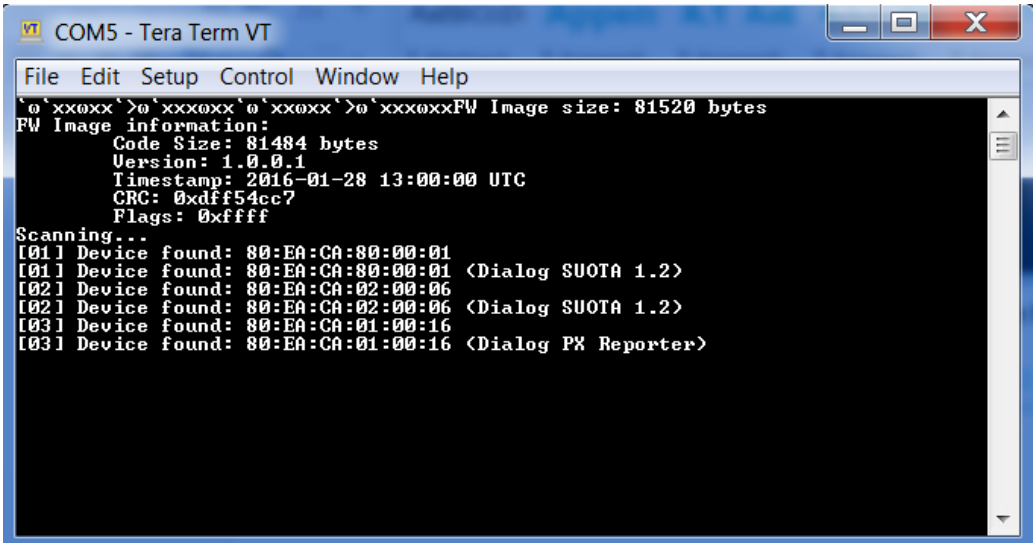


Figure 73: Scanning for available devices

The Bluetooth low energy central device starts scanning for available devices immediately (Figure 73). In this in example, form the devices listed Figure 73, the Bluetooth low energy peripheral device, is the device with sequence number `[01]` and an advertising name of `<Dialog SUOTA 1.2>`.

- As soon as the Bluetooth low energy peripheral device is found, the scanning operation can be stopped with the following command:

```
> scan stop
```


© 2022 Renesas Electronics

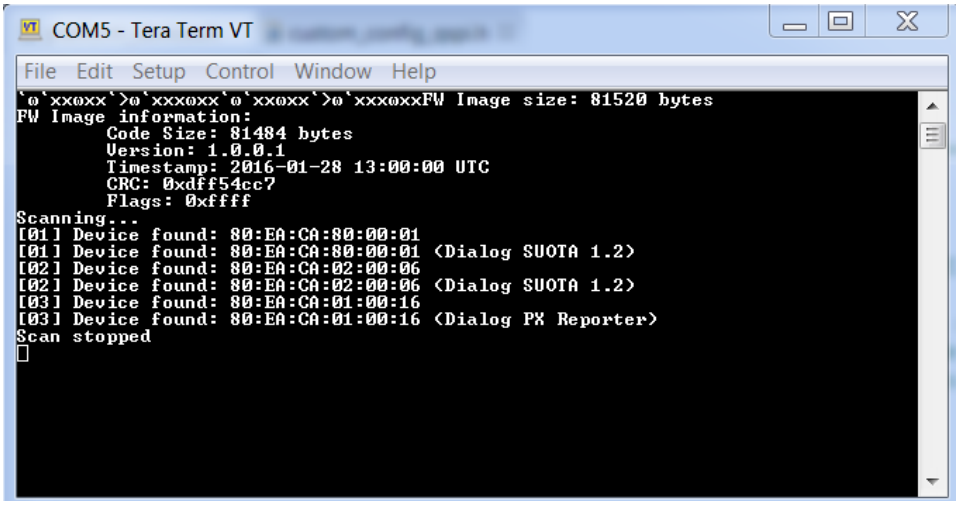


Figure 74: Stop scanning procedure

- A connection to the Bluetooth low energy peripheral device can be initiated with the command:

```
> connect 1
```

The first argument of the "connect" command refers to the device index on the scan result list. Once a connection is established, the application automatically queries the remote device for available services and device information. The characteristic values of the Device Information Service (DIS) are read. The following output is printed on the terminal:

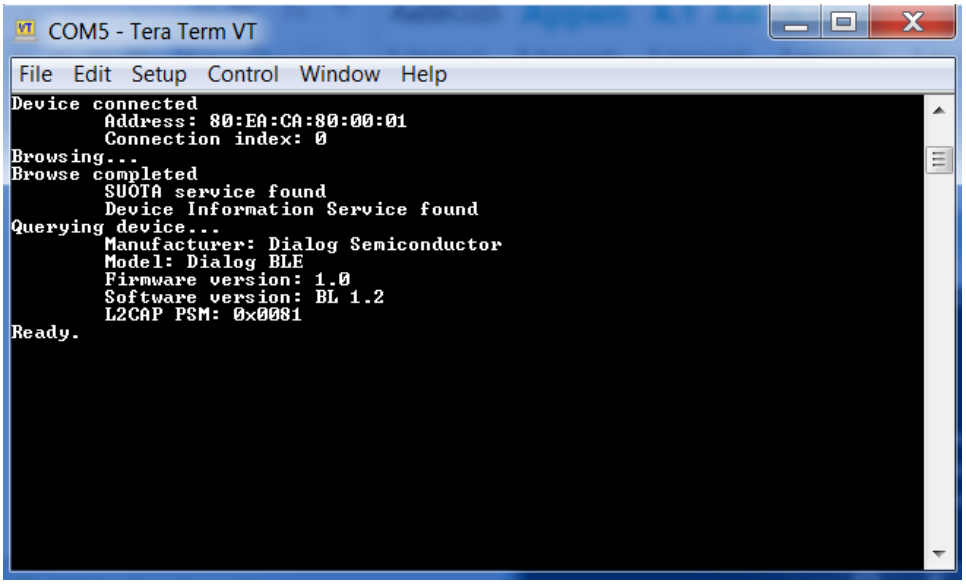


Figure 75: Connecting to loader device

The presence of "L2CAP PSM" indicates that the remote device supports SUOTA and over-L2CAP COC.

- To update a device supporting L2CAP COC over L2CAP, issue the **update** command. To update the same device over GATT, issue the **update gatt** command. If the remote device does not support L2CAP COC ("L2CAP PSM" is not displayed), both **update** and **update gatt** commands begin SUOTA over GATT.

```
> update
```


© 2022 Renesas Electronics

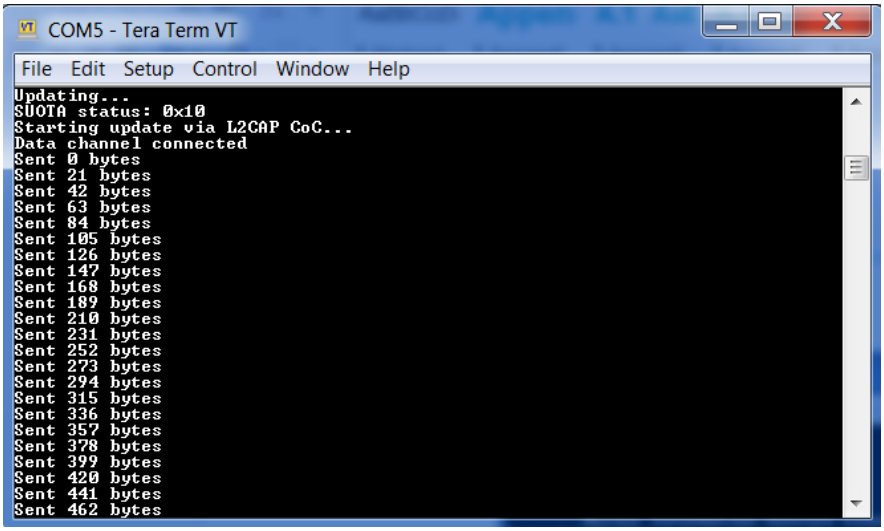


Figure 76: Updating with new image the loader device

After the image transfer has been completed, the remote device disconnects and reboot as shown in Figure 77.

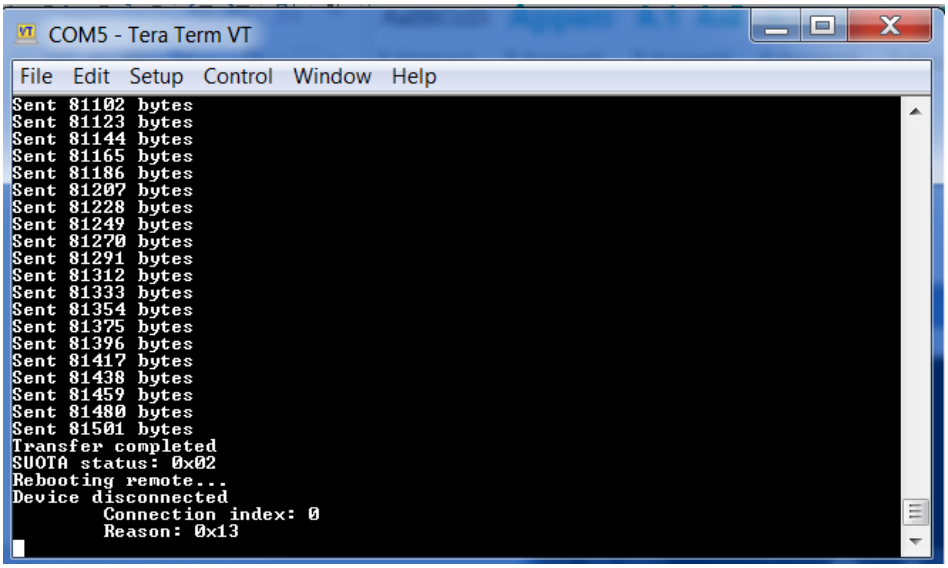


Figure 77: Transfer complete

When the Bluetooth low energy peripheral device boots, `ble_suota_loader` transfers the new image from the `NVMS_FW_UPDATE_PART` to the `NVMS_FW_EXEC_PART` partition, and execution of the new image begins.

© 2022 Renesas Electronics

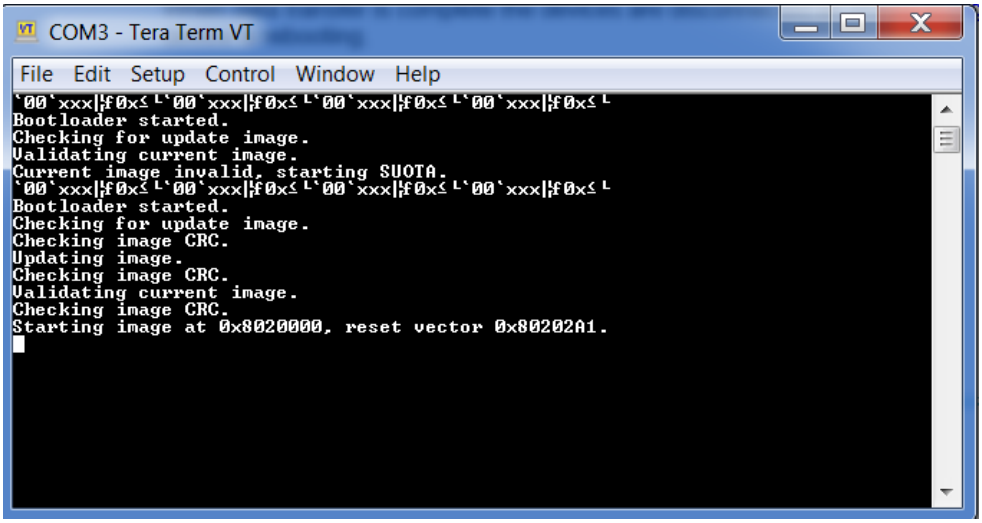


Figure 78: Rebooting and loading image

The software upgrade has finished. Now the Bluetooth low energy peripheral device must start advertising as `<Dialog PX Reporter>`. To verify that, scan again from Bluetooth low energy central device's terminal.

```
> scan start
```

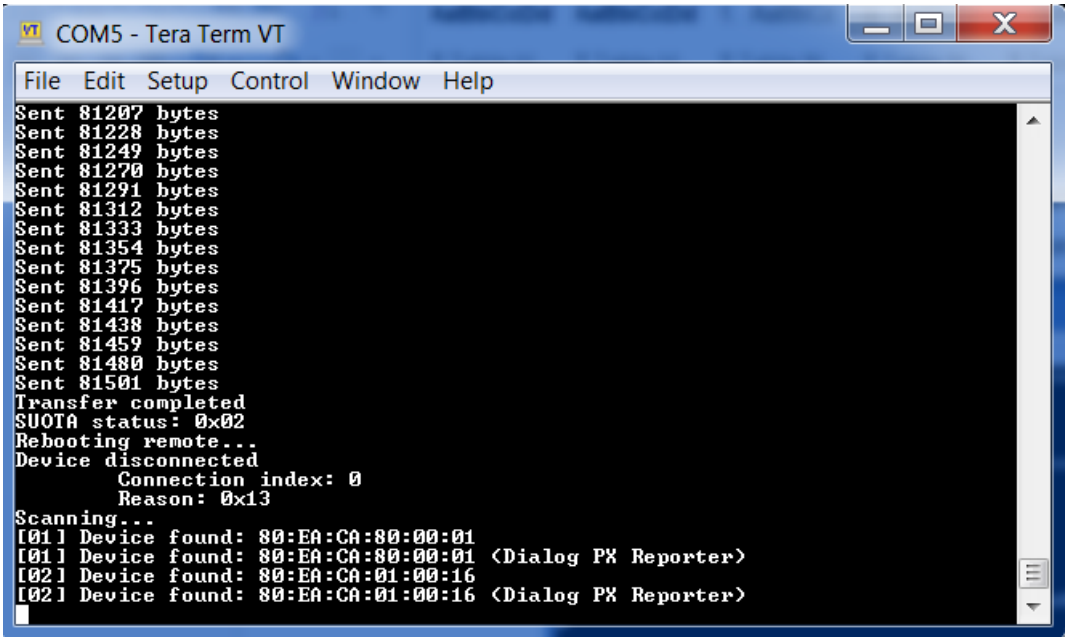


Figure 79: Verifying that loader is running PX Reporter

9.1.7 SUOTA in Production and Field deployment

When the device is deployed in the field it contains both the bootloader (`ble_suota_loader`) which checks for the presence of a new firmware image in the update partition as well as the existing application in the execute partition (`pxp_reporter` in this case).

The first part of this code resides in the bootloader, where it checks if a new firmware version is available on the update partition on every single reboot. The second part is inside the SUOTA-enabled application. This part decides how and when the new image can be downloaded. The user application must broadcast the presence of the SUOTA service (using the `DIALOG SUOTA UUID`) in its advertising data. The Android or IOS SUOTA application can then identify a SUOTA capable device and initiate a software update.

© 2022 Renesas Electronics

9.1.8 Recommendations

Only one SUOTA-capable Bluetooth low energy peripheral device should be connected to the Bluetooth low energy central device (e.g. an Android device running the Dialog SUOTA application) that performs a software update at any given time. If more than one device is connected to the accessory, then the SUOTA process should never be initiated. The application should decide how to handle this case.

9.2 Software Upgrade Over USB (SUOUSB)

9.2.1 Introduction

The DA1468x platform also allows the user to update the software over USB CDC. This process is called Software Upgrade Over USB (SUOUSB), and is simple enough to be performed by the end user. **It can only be done on a ProDK.**

The SUOUSB process starts with putting the DA1468x in SUOUSB mode, then connecting USB1 to the host PC (Windows or Linux) which then initiates the transfer of the new image to the Firmware Update partition in the Flash memory of the DA1468x. After the transfer is completed the device reboots to complete the update with the SUOUSB loader transferring the image to the Executable partition and executing it. The new software version should start after the reboot with a small delay.

9.2.2 QSPI based SUOUSB

Import the following three projects into SmartSnippets Studio from these locations.

```
scripts:          <sdk_root_directory>\utilities
suousb_loader:    <sdk_root_directory>\projects\dk_apps\features\suousb_loader
pxp_reporter:     <sdk_root_directory>\projects\dk_apps\demos
```

9.2.2.1 Prepare bootloader

Build the `suousb_loader` using the `DA14681-01-Release_QSPI` configuration.

9.2.2.2 Prepare main image

To make the flash partition table of `pxp_reporter` the same with the `suousb_loader` project, below defines must be included:

```
#define dg_configIMAGE_FLASH_OFFSET             (0x20000)
#define USE_PARTITION_TABLE_1MB_WITH_SUOTA
```

Note 10 The build configurations for SUOTA `DA14681-01-Release_QSPI_SUOTA` and `DA14681-01-Debug_QSPI_SUOTA` have the defines already included. These configurations can be used without any changes.

1. Build the project with a configuration like `DA14681-01-Release_QSPI` or `DA14681-01-Debug_QSPI` with these defines changed or `DA14681-01-Release_QSPI_SUOTA` or `DA14681-01-Debug_QSPI_SUOTA` which has them already defined.
2. Erase flash entirely using `erase_qspi_jtag_win` (or `_linux`).
3. Download the bootloader and the main image to flash using the script `suousb_initial_flash_jtag_win` (or `_linux`).
4. The `suousb_loader` and `pxp_reporter` projects must be built in this order so that when the mkimage tool is run in the next stage the `pxp_reporter` project was the last active project.

9.2.2.3 Prepare SUOUSB image for test

SUOUSB image is a binary file with a proper header that can be sent to a target device from Windows and Linux. To create an image, open command prompt in a project folder like `pxp_reporter` located in `<sdk_root_directory>/projects/dk_apps/features/suousb_loader` and run script to create the image file.

 © 2022 Renesas Electronics

- To build an image in Windows run:

  ```
  > mkimage.bat DA14681-01-Release_QSPI_SUOTA
  ```

- To build an image in Linux run:

  ```
  > ./mkimage.sh DA14681-01-Release_QSPI_SUOTA
  ```

It prepares an image file with the following naming format `pxp_reporter.1.0.0.1.img`. The version number in the image file is taken from file `sw_version.h`.

9.2.2.4 Running the SUOUSB process

The first step is to connect a USB cable from the host PC to USB1 connector on ProDK – note this is the charger USB port and not the debug one.

Put the DA1468x into USB-CDC mode by resetting the device while holding down button `K1`.

Press `K1` button. While it is pressed, press and release `K2 RESET` button. Then release `K1` button.

When the target enters the download mode, the messages below are shown on the serial console of host through UART if USB2 is also connected for debug.

- Bootloader started.
- Checking status of K1 Button.
- K1 Button is pressed, starting SUOUSB service without booting application.

Now that SUOUSB mode has started it enumerates a USB-CDC port that appears as another `COMx` port in Windows and `/dev/ttyACM0` in Linux.

9.2.2.5 Transfer from a Windows host

- Open a command prompt window at `<sdk_root_folder>\utilities\suousb_host\` in Windows.
- Build using the following command:

  ```
  > C:\DiaSemi\SmartSnippetsStudio\Tools\mingw64_targeting32\bin\gcc.exe -o
  host_usb_updater.exe host_usb_updater.c
  ```

Note 11 Another gcc.exe can be used. (e.g, cygwin or mingw.)

- Run using the following command:

  ```
  > host_usb_updater.exe 24
  ..\..\..\..\..\projects\dk_apps\demos\pxp_reporter\DA14681-01-
  Release_QSPI\pxp_reporter.1.0.0.1.img -verbose
  ```

  - The number `24` is the com port number of USB-CDC device. User can see the com port number on Windows device manager.
  - Debug message can be enabled by `-verbose` option.

9.2.2.6 Transfer from a Linux host

- Open a command terminal in `<sdk_root_directory>\utilities\suousb_host\` on Linux machine.
- Build using the following command

  ```
  > gcc -o host_usb_updater host_usb_updater.c
  ```

- Run using the following command

  ```
  > sudo ./host_usb_updater /dev/ttyACM0 ./pxp_reporter.1.0.0.1.img
  ```

  - The `/dev/ttyACM0` is the usb-cdc driver of Linux. It can be changed according to test machine.

- Sometimes a modemmanager in Linux system like Ubuntu might interrupt the usb-cdc communication. So, it should be disabled using one of the methods below.

© 2022 Renesas Electronics

- Remove the modemmanger :
- `> sudo apt-get remove modemmanager`
- Disable the modemmanger in case of usb-cdc communication by adding the rule below to `/etc/udev/rules.d/10-local.rules`

```
ATTRS{idVendor}=="2dcf", ATTRS{idProduct}=="6001",
ENV{ID_MM_DEVICE_IGNORE}="1"
```

9.2.3 RAM based SUOUSB

Import the following three projects into SmartSnippets Studio from these locations.

```
scripts:          <sdk_root_directory>\utilities
suousb_loader:    <sdk_root_directory>\projects\dk_apps\features\suousb_loader
pxp_reporter:     <sdk_root_directory>\projects\dk_apps\demos
```

9.2.3.1 Prepare bootloader

Build the `suousb_loader` using the `DA14681-01-Release_RAM` configuration.

9.2.3.2 Prepare main image

Prepare the main image to be downloaded and programmed to QSPI with the following steps:

1. To make the flash partition table of `pxp_reporter` the same with the `suousb_loader` project, below defines must be included:

```
#define dg_configIMAGE_FLASH_OFFSET             (0x20000)
#define USE_PARTITION_TABLE_1MB_WITH_SUOTA
```

Note 12 The build configurations for SUOTA `DA14681-01-Release_QSPI_SUOTA` and `DA14681-01-Debug_QSPI_SUOTA` have the defines already included. These configurations can be used without any changes.

2. Build the project with a configuration like `DA14681-01-Release_QSPI` or `DA14681-01-Debug_QSPI` with these defines changed or `DA14681-01-Release_QSPI_SUOTA` or `DA14681-01-Debug_QSPI_SUOTA` which has them already defined.
3. Erase flash entirely using `erase_qspi_jtag_win` (or `_linux`).
4. Download the bootloader and the main image to flash using the script `suousb_initial_flash_jtag_win` (or `_linux`).
5. The `suousb_loader` and `pxp_reporter` projects must be built in this order so that when the mkimage tool is run in the next stage the `pxp_reporter` project was the last active project.

9.2.3.3 Prepare SUOUSB image for test

SUOUSB image is a binary file with a proper header that can be sent to a target device from Windows and Linux. To create an image, open command prompt in a project folder like `pxp_reporter` located in `<sdk_root_directory>/projects/dk_apps/features/suousb_loader` and run script to create the image file.

- To build an image in Windows run:

```
> mkimage.bat <build_configuration>
```

Where `build_configuration` may be `DA14681-01-Release_QSPI`, `DA14681-01-Debug_QSPI`, etc.

- To build an image in Linux run:

```
> ./mkimage.sh <build_configuration>
```

Where `build_configuration` may be `DA14681-01-Release_QSPI`, `DA14681-01-Debug_QSPI`, etc.

© 2022 Renesas Electronics

It prepares an image file like `pxp_reporter.1.0.0.1.img`. The file name contains a version number taken from file `sw_version.h`.

9.2.3.4 Running the SUOUSB process

The first step is to connect a USB cable from the host PC to USB1 connector on ProDK – note this is the charger USB port and not the debug one.

To enter the download mode user should follow the next steps:

1. Load `suousb_loader` into RAM using RAM script.
2. `suousb_loader` should be paused at break point of the main function.
3. Press `K1` button of board and select **resume** debug mode in SmartSnippets™ Studio.
4. Now that SUOUSB mode has started it enumerates a USB-CDC port that appears as another `COMx` port in Windows and `/dev/ttyACM0` in Linux.

Everything else is the same as in the QSPI based SUOUSB and the windows and linux host download instructions are the same as in the QSPI based SUOUSB in sections 9.2.2.5 and 9.2.2.6.

9.2.4 Use both SUOUSB and SUOTA

SUOUSB and SUOTA can be enabled together in the image. The SUOUSB is applied to the bootloader (`suousb_loader`) while SUOTA is applied to the main image e.g. `` `pxp_reporter` ``. For the preparation of the bootloader and the main image please follow the steps described in the corresponding sections of the SUOSB and SUOTA.

© 2022 Renesas Electronics

10 Enabling features on the Proximity Reporter application

The following paragraphs explain how to enable SmartSnippets™ DA1468x SDK features on the Proximity Reporter application (the same can be applied on any project). Modifications are needed depending the specific implementation of each project.

10.1 Enabling the Charger

A very useful feature that Proximity Reporter provides is the ability to enable the charger. The configuration of the Charger can be divided in three parts:

- Configuration of the USB.
- Charging algorithm configuration.
- Charging parameters.

For more information about the charger please refer to the Platform Reference Manual [4].

All these configuration options are defined in `config/custom_config_qspi_suota.h` header file.

```
#define dg_configBATTERY_TYPE                    (BATTERY_TYPE_CUSTOM)
#define dg_configBATTERY_CHARGE_VOLTAGE          0xD     // 4.35V
#define dg_configBATTERY_TYPE_CUSTOM_ADC_VOLTAGE (3563)
//#define dg_configBATTERY_LOW_LEVEL             (2457)  // 3V
#define dg_configPRECHARGING_THRESHOLD           (2462)  // 3.006V
#define dg_configCHARGING_THRESHOLD              (2498)  // 3.05V
#define dg_configBATTERY_CHARGE_CURRENT          2       // 30mA
#define dg_configBATTERY_PRECHARGE_CURRENT       20      // 2.1mA
#define dg_configBATTERY_CHARGE_NTC              1       // disabled
#define dg_configPRECHARGING_TIMEOUT             (30 * 60 * 100)  // N x 10msec
#define dg_configUSE_USB                         1
#define dg_configUSE_USB_CHARGER                 1
#define dg_configALLOW_CHARGING_NOT_ENUM         1
#define dg_configUSE_NOT_ENUM_CHARGING_TIMEOUT   0
```

Code 10: Charger configuration

10.2 Configuration for SUOTA

The following sections describe the Proximity Reporter demo from the SUOTA point of view. The steps need to enable/configure SUOTA in an application are the following:

1. Create or modify a header file containing information about the version used for creating images.
2. Add code to include the SUOTA in an application.
3. Configure the application start address.

10.2.1 Version header file

Code 11 shows the content of `sw_version.h`, which is the header file that should be modified whenever a new version is produced. This header file is important because it is required by the `mkimage` tool in order to create the new image file.

 © 2022 Renesas Electronics

```
#define BLACKORCA_SW_VERSION "1.0.0.1"
#define BLACKORCA_SW_VERSION_DATE "2016-01-28 15:00"
#define BLACKORCA_SW_VERSION_STATUS "REPOSITORY VERSION"
```

Code 11: sw_version.h

10.2.2 Code analysis

All SUOTA-related lines of code are inside `#if dg_configSUOTA_SUPPORT #endif` blocks. The preprocessor macro for enabling the SUOTA is already in a configuration header file located in `<sdk_root_directory>/projects/dk_apps/demos/pxp_reporter/config/custom_config_qspi_suota.h`:

```
#define dg_configSUOTA_SUPPORT              (1)
```

Code 12: Macro to enable SUOTA

The desired SUOTA version is defined using the `SUOTA_VERSION` definition. SUOTA version v1.1 (`SUOTA_VESRION_1_1`) allows the application to be upgraded only over GATT, whereas versions v1.2, v1.3 (`SUOTA_VERSION_1_2, SUOTA_VERSION_1_3`) allow the application to be upgraded both over GATT and L2CAP COC. To enable SUOTA over L2CAP COC, `SUOTA_PSM` should also be defined, in addition to `SUOTA_VERSION`. This definition specifies the PSM that must be used to establish the L2CAP COC connection.

```
/*
 * SUOTA loader configuration:
 * - To enable SUOTA over GATT only, set SUOTA_VERSION to any version >=
SUOTA_VERSION_1_1
 *      and leave SUOTA_PSM undefined.
 * - To enable SUOTA over GATT and L2CAP CoC, set SUOTA_VERSION to any version
>= SUOTA_VERSION_1_2
 *      and also define SUOTA_PSM to match the desired PSM. In this case the
central device
 *      can use either of both according to its preference.
 */
#define SUOTA_VERSION   SUOTA_VERSION_1_3
#define SUOTA_PSM       0x81
```

Code 13: Defining SUOTA version and L2CAP COC PSM

In addition, when support for L2CAP COC is enabled (`SUOTA_PSM` is defined), L2CAP related events should be passed from the application to the SUOTA service as shown in Code 14.

```
#if dg_configSUOTA_SUPPORT && defined (SUOTA_PSM)
                                case BLE_EVT_L2CAP_CONNECTED:
                                case BLE_EVT_L2CAP_DISCONNECTED:
                                case BLE_EVT_L2CAP_DATA_IND:
                                        suota_l2cap_event(suota, hdr);
                                        break;
#endif
```

Code 14: Passing L2CAP events to the SUOTA service

For SUOTA to be enabled in the Proximity Reporter application, the `pxp_reporter_task.c` should be modified accordingly. The code snippet presented in Code 15 should be included in the `pxp_reporter_task()` function to declare the variable that handles the service:

© 2022 Renesas Electronics

```
ble_service_t *suota;
```

Code 15: Declare SUOTA variable

Additionally, the SUOTA service should be initialized and added to the BLE framework, inside the same function. This service is the starting point where all services are created. In addition, it is important to add the Device Information Service (DIS) as Android and iOS applications relies on DIS, as shown in Code 16.

```
/* Register SUOTA
*
* SUOTA instance should be registered in ble_service framework in order for
events
* inside service to be processed properly.
*/
suota = suota_init(&suota_cb);
OS_ASSERT(suota != NULL);
/*
* Register DIS
*
* DIS doesn't contain any dynamic data thus it doesn't need to be registered in
* ble_service framework (but it's not an error to do so).
*/
dis_init(NULL, &dis_info);
```

Code 16: Register SUOTA and DIS

Code 17 shows the required data for the DIS standard BLE service. This project uses the same header file version for both building the image and for providing data for the DIS.

```
/* Device Information Service data
 *
 * Manufacturer Name String is mandatory for devices supporting HRP.
 */
static const dis_device_info_t dis_info = {
        .manufacturer = "Dialog Semiconductor",
        .model_number = "Dialog BLE",
        .serial_number = "123456",
        .hw_revision = "REV.D",
        .fw_revision = "1.0",
        .sw_revision = BLACKORCA_SW_VERSION,
};
```

Code 17: DIS data

The header files shown in Code 18 must be included to build these changes.

```
#include "dis.h"
#include "dlg_suota.h"
#include "sw_version.h"
```

Code 18: Header files

© 2022 Renesas Electronics

In order to make the SUOTA process operational, the SUOTA service has to be included in the advertising data.

```
/*
 * PXP advertising and scan response data
 * While not required, PXP specification states that PX reporter device using
 * peripheral role can advertise support for LLS. Device name is set in scan
 * response to make it easily recognizable.
 */
static const uint8_t adv_data[] = {


#if dg_configSUOTA_SUPPORT
        0x07, GAP_DATA_TYPE_UUID16_LIST_INC,
        0x03, 0x18, // = 0x1803 (LLS UUID)
        0x02, 0x18, // = 0x1802 (IAS UUID)
        0xF5, 0xFE, // = 0xFEF5 (DIALOG SUOTA UUID)
#if dg_configSUOTA_SUPPORT
     0x07, GAP_DATA_TYPE_UUID16_LIST_INC,
     0x03, 0x18, // = 0x1803 (LLS UUID)
     0x02, 0x18, // = 0x1802 (IAS UUID)
     0xF5, 0xFE, // = 0xFEF5 (DIALOG SUOTA UUID)
```

Code 19: Advertising and scan response data

10.2.3 Application start address

SUOTA-enabled applications should be compiled for execution from address `0x20000`.

The following lines are already there but it is likely that `CODE_BASE_ADDRESS` is always set to `0x8000000`. This does not work when a bootloader is present. The modification required to `CODE_BASE_ADDRESS` is highlighted with red font in Code 20.

```
#  if (dg_configEXEC_MODE == MODE_IS_MIRRORED)
        #warning "QSPI mirrored execution mode is not supported!"
        #undef CODE_SIZE
        #define CODE_SIZE               0
#  else
        #define CODE_BASE_ADDRESS       0x8000000 + dg_configIMAGE_FLASH_OFFSET
        #define RAM_BASE_ADDRESS        0x7FC0000
#  endif
```

Code 20: Code Base Address

These changes are not sufficient in themselves and the application still builds from address 0 unless `dg_configIMAGE_FLASH_OFFSET` is defined.

In case of the Proximity Reporter application, this macro is defined in the `config/custom_config_qspi_suota.h` file. Code 21 defines the correct starting address.

© 2022 Renesas Electronics

```
#define dg_configIMAGE_FLASH_OFFSET             (0x20000)
```

Code 21: Set starting address

The Proximity Reporter includes two configurations: one without the SUOTA functionality, which allows building an application for address `0x0`, and another one with the SUOTA functionality, which builds for address `0x20000`.

© 2022 Renesas Electronics

Revision history

Revision	Date	Description
1.0	19-Nov-2015	First released version
2.0	22-Apr-2016	Update for SmartSnippets DA1468x SDK Release 1.0.4 .812
2.1	17-Jun-2016	Update for SmartSnippets DA1468x SDK Engineering Release 1.0.5.885
3.0	26-Jul-2016	Update for SmartSnippets DA1468x SDK Release 1.0.6.968
3.1	26-Jul-2016	Chapter 10 deleted because of improvements in scripts.
4.0	07-Dec-2016	Update for SmartSnippets DA1468x SDK Release 1.0.8
5.0	21-Jul-2017	Update for SmartSnippets DA1468x SDK Release 1.0.10
5.03	15-Nov-2017	Update for SmartSnippets DA1468x SDK Release 1.0.10
6.0	14-Dec-2017	Update for SmartSnippets DA1468x SDK Release 1.0.12
6.1	24-Feb-2022	Updated logo, disclaimer, copyright.

© 2022 Renesas Electronics

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

© 2022 Renesas Electronics