Dialog SDK 5.0.3 Training Materials – SUOTA (Software Update Over The Air)

2016 February

…personal
…portable
…connected

BLE Dialog Semiconductor SUOTA

SUOTA overview

Step by step procedure to create software update and send over

What would you see as output

BLE Dialog Semiconductor SUOTA

Let's build a demo together …

- **Before we start, we recommend you to …**
  - Take a look at Training material 1 bare bone application
  - Take a look at Training material 2 custom profile application

- **What are you going to learn from this training …**
  - Basic understanding of software update over the air
  - Small assignment to add a characteristic in the custom service database

- **What's next …**
  - Send you queries over Dialog support website
  - Dialog semiconductor BLE Customer support team is always ready to provide you committed support and guidance
  - See **Reference** section of this training slide

SUOTA

Overview

- **Over-the-air programming** (OTA) refers to various methods of distributing new software, configuration settings etc.
- Software-Update-Over-the-air (SUOTA) proprietary service that is implemented by Dialog Semiconductor.
- DA1458x devices are capable of updating software over the air using BLE/Bluetooth smart protocols, if SUOTA proprietary service is activated.
- **Dialog SDK 5.0.3** contains SUOTA compatibility to update software considering both central and peripheral role on DA1458x over the air.

SUOTA

Overview

- SUOTA is instantiated as a GATT Primary Service.

- The service exposes a control point to allow a peer device to initiate software update over the air and define two roles:

  - The “SUOTA Initiator” which transmits the new software image. It is the **GATT client** for the SUOTA service (GAP **Central** Role).

  - The “SUOTA Receiver” which receives the new software image, stores the image into the external **FLASH/EEPROM** device and runs the new image. It is the **GATT server** for SUOTA service (GAP **Peripheral** Role).

SUOTA

Overview

- Dialog SUOTA supports 2 schemes –
  - The secondary bootloader is stored in the external non-volatile memory.
  - The secondary bootloader is burnt into the internal OTP.

	External Non-volatile memory	Internal OTP
Advantages	OTP can stay blank. Useful for development purposes and/or when very low power consumption is not a requirement for the final product.	Fastest boot-up time. Guarantee to boot up anytime.
Disadvantages	In case the external memory is in power down mode and a software reset (e.g.: Watchdog) is triggered, the DA14580 will not boot up properly. The battery has to be removed and replaced.	OTP must be burnt.

SUOTA

Step by step procedure to create software update and send over the air

What would you see as output

SUOTA example

- **prox_reporter.uvprojx** example demonstrates
  - Application running from secondary bootloader stored in SPI flash
  - Software updates over the air
  - How to configure jumpers of DA1458x Dev kit-Pro to run this application
  - **The secondary bootloader is stored in the external non-volatile memory**

- **IDE used KEIL 5**

- **Dialog semiconductor SDK used 5.0.3**

- **Project location: ..\projects\target_apps\ble_examples\prox_reporter\Keil_5**

- **Python 3.5 installation and DA1458x_SUOTA_Multipart_Binary_Generator.zip**

SUOTA

SUOTA files in proximity reporter software project architecture

DA1458x - PRO development kit HW configuration

- sdk_app
  - app_default_handlers.c
  - app.c
  - app_task.c
  - app_security.c
  - app_security_task.c
  - app_bass.c
  - app_bass_task.c
  - app_findme.c
  - app_findme_task.c
  - app_proxr.c
  - app_proxr_task.c
  - app_diss.c
  - app_diss_task.c
  - app_spotar.c
  - app_spotar_task.c
  - app_entry_point.c
  - app_msg_utils.c
  - app_easy_msg_utils.c
  - app_easy_security.c
  - app_easy_timer.c
  - app_customs.c
  - app_customs_task.c
  - app_customs_common.c
- sdk_driver_api
- sdk_app_api
- sdk_arch_api
  - arch_api.h
  - arch_wdg.h
  - arch_console.h
- user_config
  - da1458x_config_advanced.h
  - da1458x_config_basic.h
  - user_callback_config.h
  - user_config.h
  - user_modules_config.h
  - user_periph_setup.h
  - user_profiles_config.h
  - user_config_sw_ver.h
- user_platform
  - user_periph_setup.c
- user_app
  - user_proxr.c

SUOTA

SUOTA

Description of some important files

```
/* Holds DA14580/581/583 basic configuration settings.    */
da1458x_config_basic.h

/* Holds DA14580/581/583 advanced configuration settings. */
da1458x_config_advanced.h

/* Holds user specific information about software version. */
user_config_sw_ver.h

/* Defines which application modules are included or excluded from the user's application. */
user_modules_config.h

            /* The module is excluded. */
            #define EXCLUDE_DLG_SPOTAR          (1)
            /* The module is included. */
            #define EXCLUDE_DLG_SPOTAR          (0)

            /* Note:                                                                  */
            /*      This setting has no effect if the respective module is a BLE Profile        */
            /*      that is not used in the user's application.                       */


/* Callback functions that handle various events or operations. */
user_callback_config.h


/* Holds advertising parameters, connection parameters, etc. */
user_config.h
```

SUOTA

Description of some important files

```
/* Defines which BLE profiles (Bluetooth SIG adopted or custom ones) will be included in user's application.
   each header file denotes the respective BLE profile*/
user_profiles_config.h

            #inlucde "proxr.h"      // Includes Proximity reporter.
            #include "spotar.h"     // Includes SUOTA.


/* Holds hardware related settings relative to the used Development Kit. */
user_periph_setup.h


/* Source code file that handles peripheral (GPIO, UART, SPI, etc.)
   configuration and initialization relative to the Development Kit.*/
user_periph_setup.c


/* Source code file that is implemented as SUOTA reporter application entry point.*/
app_spotar.c


/* Source code file that is implemented as SUOTA receiver application Message Handlers.*/
app_spotar_task.c

/* Source code file that is implemented as Proximity reporter application entry point.*/
app_proxr.c


/* Source code file that is implemented as Proximity reporter application task implementation.*/
app_spotar_task.c
```

Let's do it … preparation for the demo

TODO 1 - Change the default **BD_ADDRESS, this address has to be unique in a BLE network.**

/* @file **da1458x_config_advanced.h** */

```
/* copy and paste in code step 1 change the BLE device address */
#define CFG_NVDS_TAG_BD_ADDRESS                {0x19, 0x00, 0x00, 0x00, 0x00, 0x19}
```

TODO 2 – Check and define **DLG_SPOTAR** module in your application code

/* @file **user_modules_config.h** */

```
/* copy and paste in code step 2 define EXCLUDE_DLG_SPOTAR module in your application code */
#define EXCLUDE_DLG_SPOTAR          (0)          /* included */
```

TODO 3 – Check and include **spotar.h** in your application code to activate custom profile

/* @file **user_profiles_config.h** */

```
#include "diss.h"
/* copy and paste in code step 3 add spotar.h */
#include "spotar.h"
```

Customer services Contents

Let's do it … preparation for the demo

TODO 4 - Information and change your advertising device name

/* @file **user_config.h** */

```
/* default sleep mode. Possible values ARCH_SLEEP_OFF, ARCH_EXT_SLEEP_ON, ARCH_DEEP_SLEEP_ON
   ARCH_EXT_SLEEP_ON, ARCH_DEEP_SLEEP_ON - You cannot debug in these modes
*/
const static sleep_state_t app_default_sleep_mode = ARCH_SLEEP_OFF;
//-------------NON-CONNECTABLE & UNDIRECTED ADVERTISE RELATED COMMON -- //
/// Advertising service data
#define USER_ADVERTISE_DATA     "\x09"\
                                ADV_TYPE_COMPLETE_LIST_16BIT_SERVICE_IDS\
                                ADV_UUID_LINK_LOSS_SERVICE\
                                ADV_UUID_IMMEDIATE_ALERT_SERVICE\
                                ADV_UUID_TX_POWER_SERVICE\
                                ADV_UUID_SPOTAR_SERVICE
/* copy and paste in code step 4 change your advertising device name */
#define USER_DEVICE_NAME     ("SUOTA-1")
```

Customer services Contents

Let's do it … preparation for the demo

TODO 5 - Change the software version

/* @file **ble_580_sw_version.h** */

```
#define DA14580_SW_VERSION "v_5.0.3.0"
#define DA14580_SW_VERSION_DATE "2015-10-14 16:01 "
#define DA14580_SW_VERSION_STATUS "REPOSITORY VERSION"
```

TODO 6 – Build the project and **rename** \out_580**prox_reporter_580.hex** to **fw_1.hex**

Rename **ble_580_sw_version.h** to **fw_1_version.h**

TODO 7 – Create a folder with name **"input"**

TODO 8 – Copy **fw_1_version.h** and **fw_1.hex** files to the folder **input**

Customer services Contents

Let's do it … preparation for the demo

TODO 9 - Information and change your advertising device name

```
/* @file user_config.h */

/* copy and paste in code step 9 change your advertising device name */
#define USER_DEVICE_NAME    ("SUOTA-2")
```

TODO 10 - Change the software version

```
/* @file ble_580_sw_version.h */
#define DA14580_SW_VERSION "v_5.0.3.1"
#define DA14580_SW_VERSION_DATE "2015-10-14 16:11 "
#define DA14580_SW_VERSION_STATUS "REPOSITORY VERSION"
```

TODO 11 – Build the project and **rename** \out_580**prox_reporter_580.hex** to **fw_2.hex**

Rename **ble_580_sw_version.h** to **fw_2_version.h**

TODO 12 – Copy **fw_2_version.h** and **fw_2.hex** files to the folder **input**

TODO 13 – Build the project utilities\secondary_bootloader\secondary_bootloader.uvprojx and **copy** \Out**secondary_bootloader.hex** to the folder named **input**

TODO 14 – Install **python 3.5.1**

Customer services Contents

Let's do it … preparation for the demo

TODO 15 - unzip DA1458x_SUOTA_Multipart_Binary_Generator.zip

TODO 16 - copy and paste your input folder contents (**CAUTION: Do not copy and paste the folder, copy and paste only the content, as there is a zReadme.txt file**; that is useful to make you understand, what are the files being expected in the input folder) inside **DA1458x_SUOTA_Multipart_Binary_Generator** folder

TODO 17 - run command prompt and go to **DA1458x_SUOTA_Multipart_Binary_Generator** folder

TODO 18 - configure "user data configuration section"

Example set IMG_1_ENC to true:

```
# Output image file will be created with default encryption key and init vector value
IMG_1_ENC = True
# below are the default encryption key and init vector value do not change these values
IMG_ENC_KEY_DEF          = "06A9214036B8A15B512E03D534120006"
IMG_ENC_INIT_VEC_DEF     = "3DAFBA429D9EB430B422DA802C9FAC41"
```

TODO 19 - execute "*python project_multipart_binary_v2.py*".

TODO 20 - Check the **output folder** and you will find **fw_multi_part_spi.bin** is created.

Let's do it … preparation for the demo

INTERESTING TASK - create a manual encryption and init vector value key for image 2

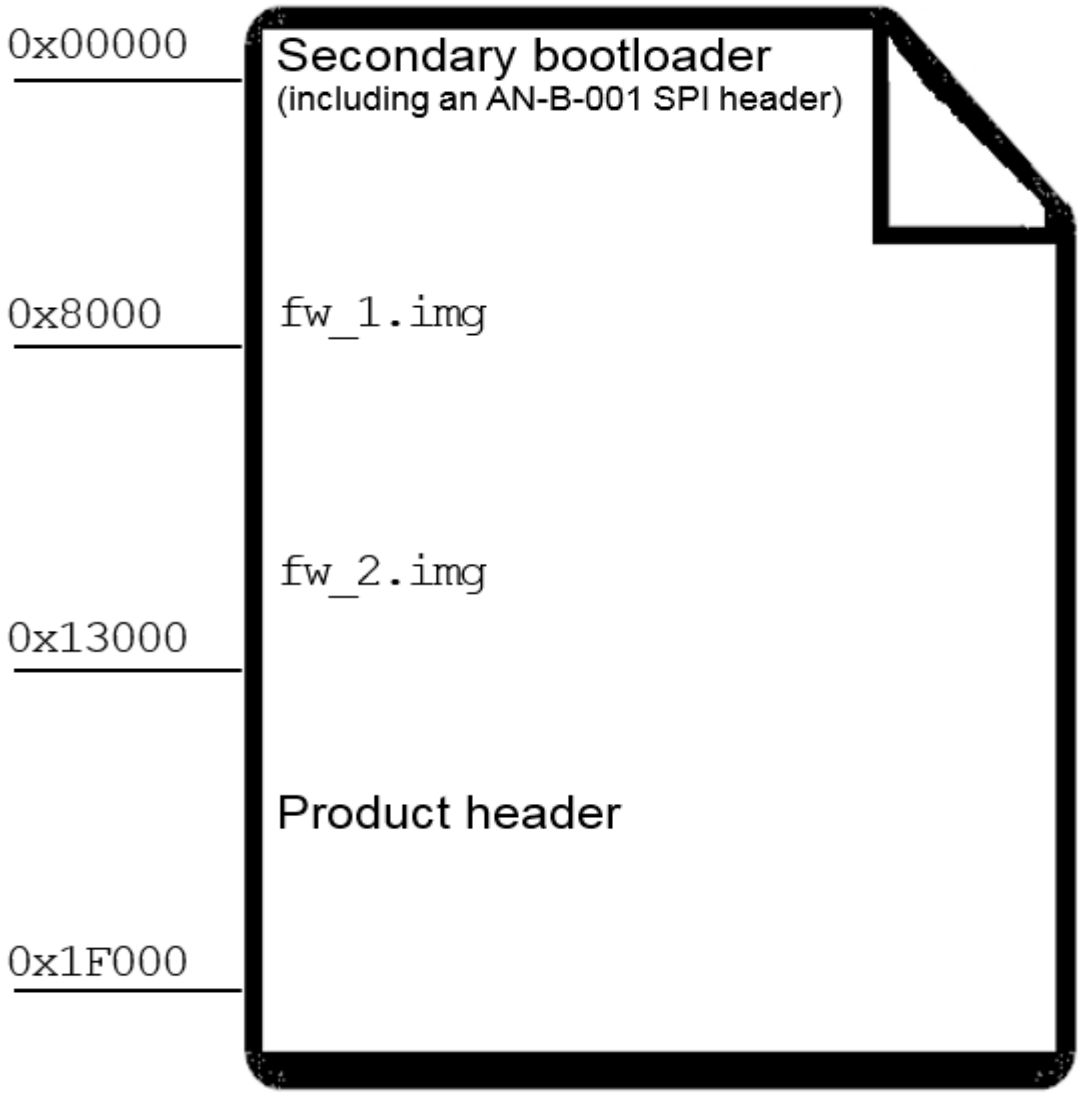
Explanation:

The system configuration of external non-volatile (FLASH) memory is described below:

- SPI/EEPROM flash only (no OTP is used)
- The dual image bootloader is stored at address 0x0
- Image #1 is stored at address 0x8000
- Image #2 is stored at address 0x13000
- The product header is stored at address 0x1F000
- Production settings are stored after the product header

SUOTA

Explanation of memory management



SUOTA

Explanation of memory management

Header

Byte	Field
0	Signature (0x70)
1	Signature (0x50)
2-5	Dummy Bytes
6	Code Size MS Byte
7	Code Size LS Byte
8-	Code Bytes

Same header for the 2 images

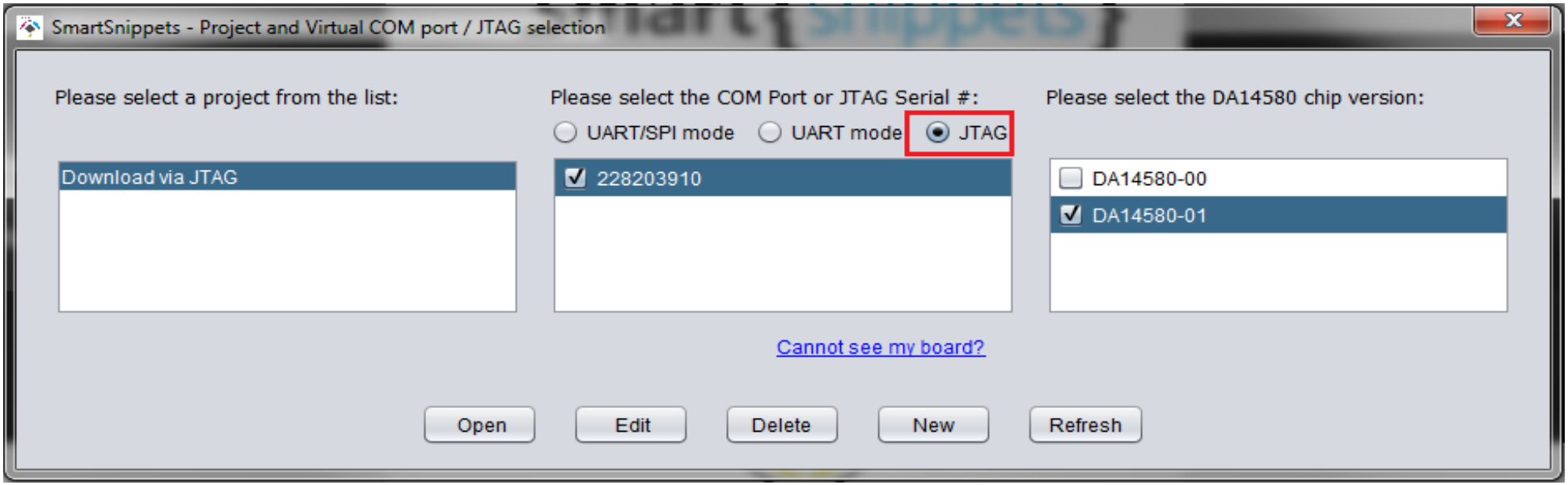
Byte	Field
0,1	Signature (0x70, 0x51)
2	imageid
3	validflag
4-7	Code Size
8-11	CRC
12-27	version
28-31	timestamp
32	Encryption flag
33-63	Reserved

Product header

Byte	Field
0	Signature (0x70)
1	Signature (0x52)
2	Version MS Byte
3	Version LS Byte
4-7	Offset #1
8-11	Offset #2
12-31	Reserved
32-37	BD Address
38	Reserved
39	XTAL 16 Trim Enable
40-43	XTAL 16 Trim Value
44-63	Reserved
64	NVDS

SUOTA erasing SPI FLASH memory

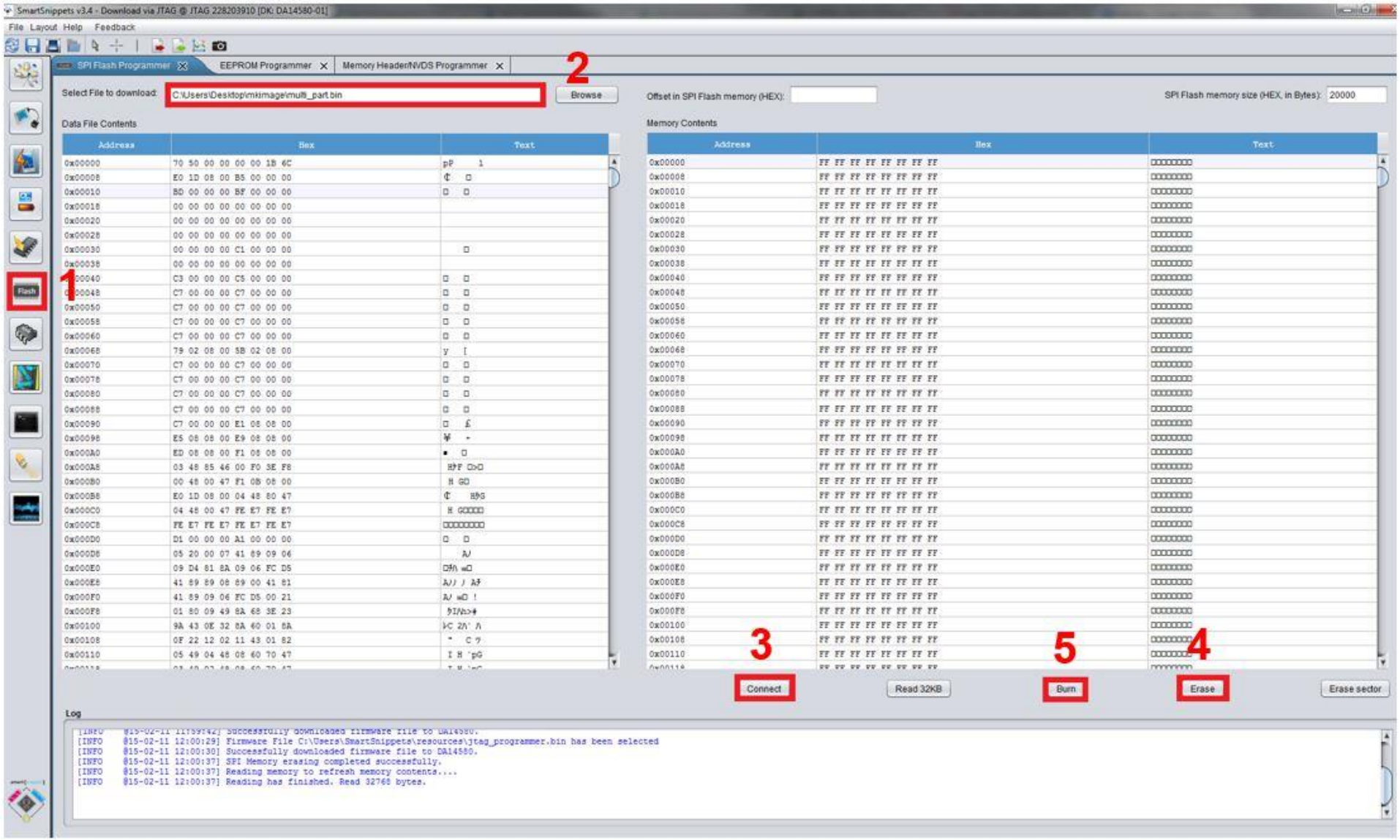
Open Dialog SmartSnippets and make sure you have selected the JTAG connection from the SmartSnippets window as shown below and click Open:



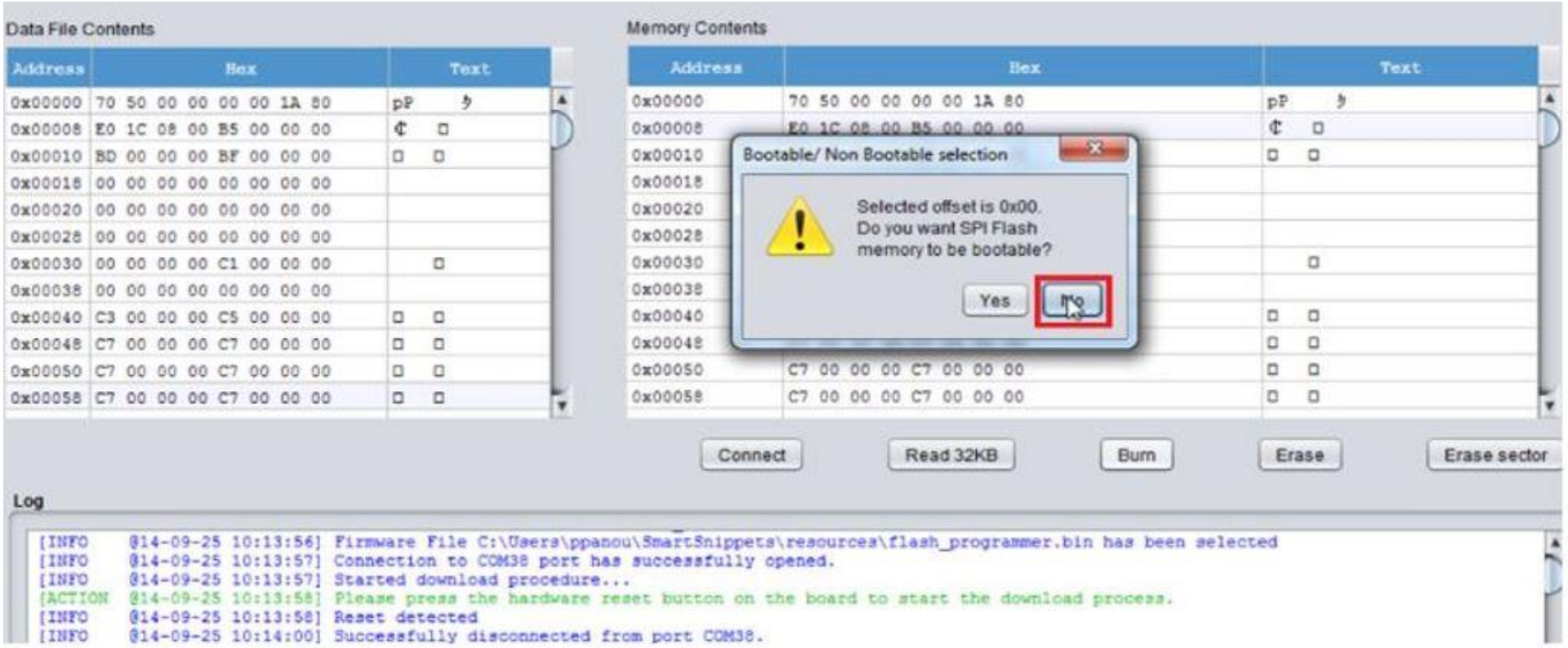
Customer services Contents

- Please follow the steps below:
  - Click on the Flash tab icon on the left side of the SmartSnippets tool
  - Select the **fw_multi_part_spi.bin** file to be downloaded into the external memory
  - Press the ‘Connect’ button
  - Press the ‘ERASE’ button
  - Press the ‘BURN’ button
  - Press the ‘NO’ button

Customer services Contents



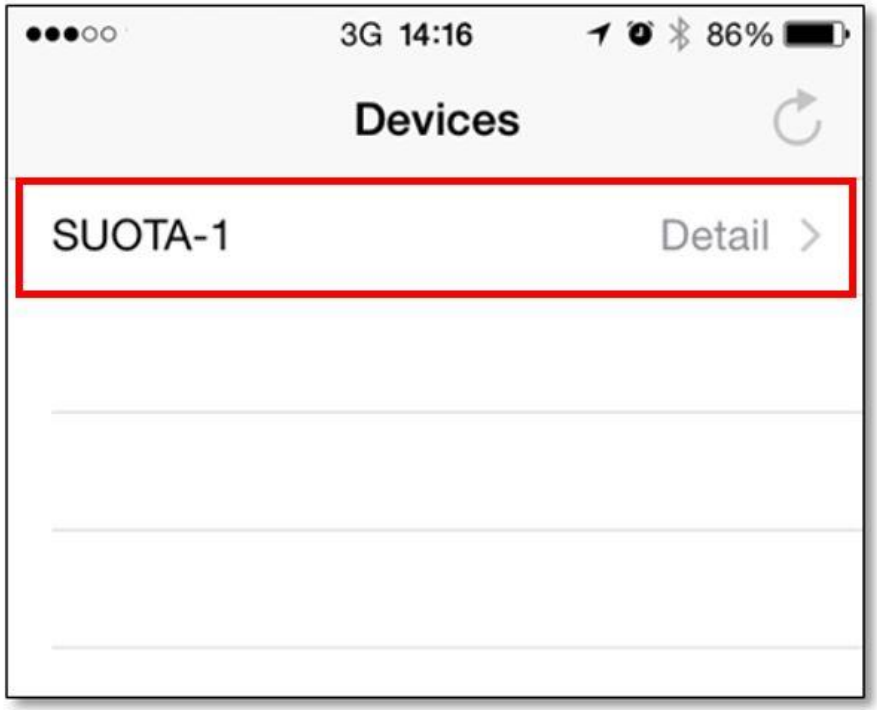
Customer services Contents



What would you see as output

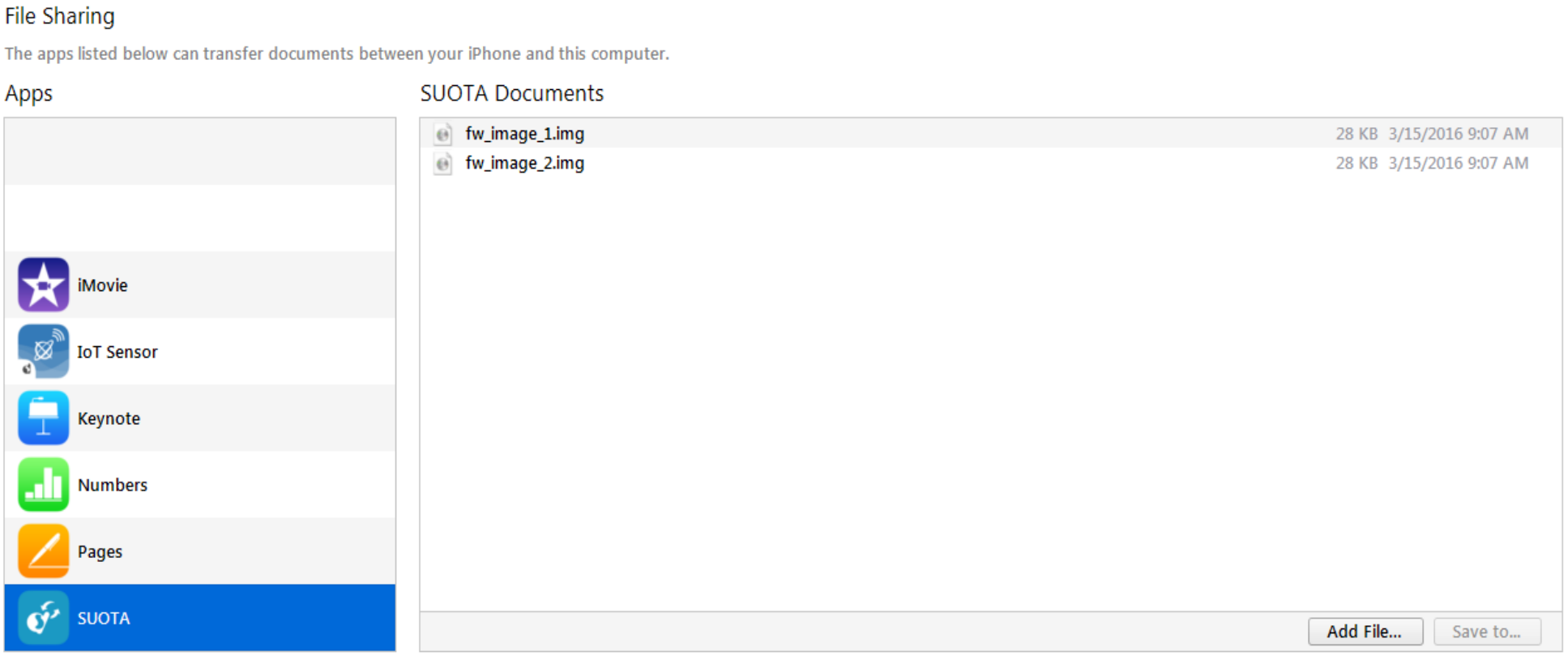
SUOTA

- Reset the DA1458x DevKit – Pro
- Verify DA1458x is advertising with the name SUOTA – 1 in an BLE scanner application iOS/Android device, like LightBLUE or BLE Scanner.



SUOTA

- Running SUOTA from iOS platform



SUOTA

- Running SUOTA from iOS platform

  - Make sure Dialog SUOTA application is downloaded in the iOS device
  - Go to iTune ‘Apps’ section
  - Scroll down to ‘File sharing’ and click on SUOTA app (See the image in the previous slide)
  - Add the ***fw_image_1.img*** and ***fw_image_2.img*** files.

SUOTA

- Start the SUOTA application on the iOS device.
- The DA14580 should advertise at this point and the device name should be detected by the application. If not, click on the clockwise arrow to initiate scanning.
- Click on the SUOTA-1 device to connect and see the DIS info screen. Verify that the “Firmware rev.” field has the same value as the DA14580_SW_VERSION string set during image creation.
- After clicking on the “Update” button, the file selection screen appears. Select fw_image_2.img to update.
- After the file selection, the memory parameters configuration screen is shown. In this screen, the default GPIO settings for SPI FLASH configuration are pre-set. Also, the “Image Bank” is set by default to “Oldest” and the “Block size” to “240”.
- As soon as the “Send to device” button is pressed, the log screen appears with a status bar.

SUOTA

- When the image is uploaded successfully, reboot the device in order to start advertising as SUOTA-2

- The DA14580 should advertise at this point and the SUOTA-2 device should be detected by the application. Click on the device to connect and verify the “Firmware rev.” value.

Follow the same procedure, for android devices. If you are playing around android.

IMPORTANT NOTE: AVOID THE SAME IMAGE ERROR

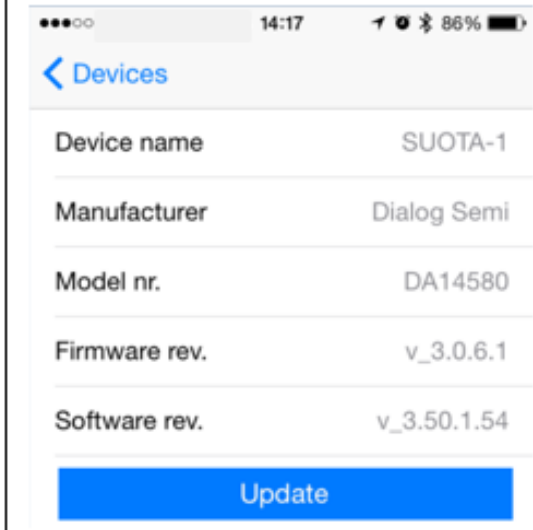
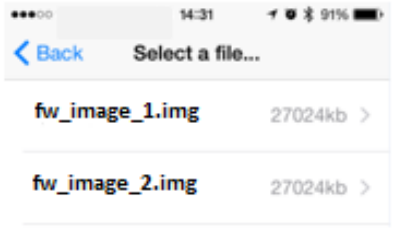
When the user tries to update an image that has the same software version **and** the same timestamp as the new image, a “Same Image Error” message is displayed on the iOS screen.
To avoid this error during a demo do one of the following:

- If two images are used, as in this example, then always update both memory banks with the same image. For example, in this demo description, the SUOTA_1.img was used for both image banks when creating the multi_part.bin (step 11). When the SUOTA app was used to upload SUOTA_02.img, only one of the memory banks has been updated. The other one still holds SUOTA_1.img. To make sure that the remaining SUOTA_1.img is updated with SUOTA _2.img, upload SUOTA _2.img again. If you want to switch back to SUOTA _1.img, then upload SUOTA _1.img twice to replace both image banks. By uploading the same image twice (replacing the old images in both memory banks), the “Same Image error” is eliminated.

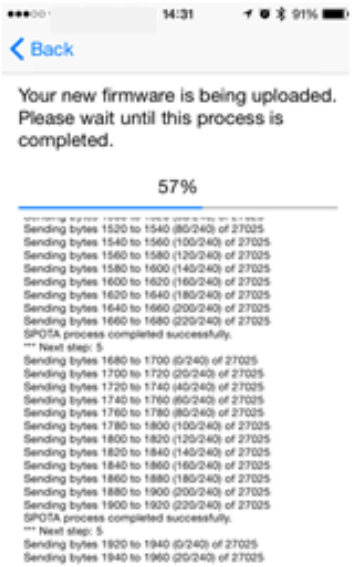
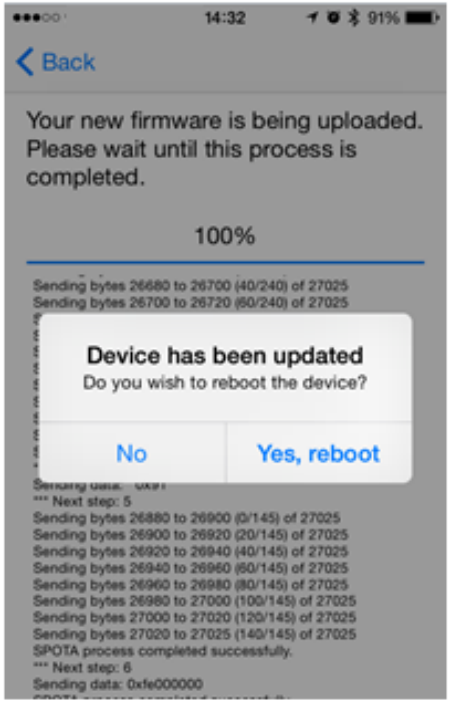
- Create and use three images and sequentially upload one after the other. By doing this it is guaranteed that “Same Image Error” will not happen.

- Note that in normal use the “Same Image Error” rarely happens. Customer will normally create a new image to update an old one. However, in the case of a demo, the same files are used to switch from one image to another and back, so it is possible that a “Same Image Error” might occur if the two memory banks implementation is not well understood.

SUOTA

Step 1	Step 2	Step 3	Step 4
			

SUOTA

Step 5	Step 6	Step 7	Step 8
			

Before we end …

Multi-part binary in OTP memory

Let’s do it … preparation for the OTP demo

Explanation:

The system configuration of internal OTP memory is described below:

- SPI/EEPROM flash and OTP are used
- The dual image bootloader is stored in the OTP
- Image #1 is stored at address 0x8000
- Image #2 is stored at address 0x13000
- The product header is stored at address 0x1F000
- Production settings are stored after the product header or in the OTP

SUOTA OTP memory explained

Explanation of memory management

Same header for the 2 images

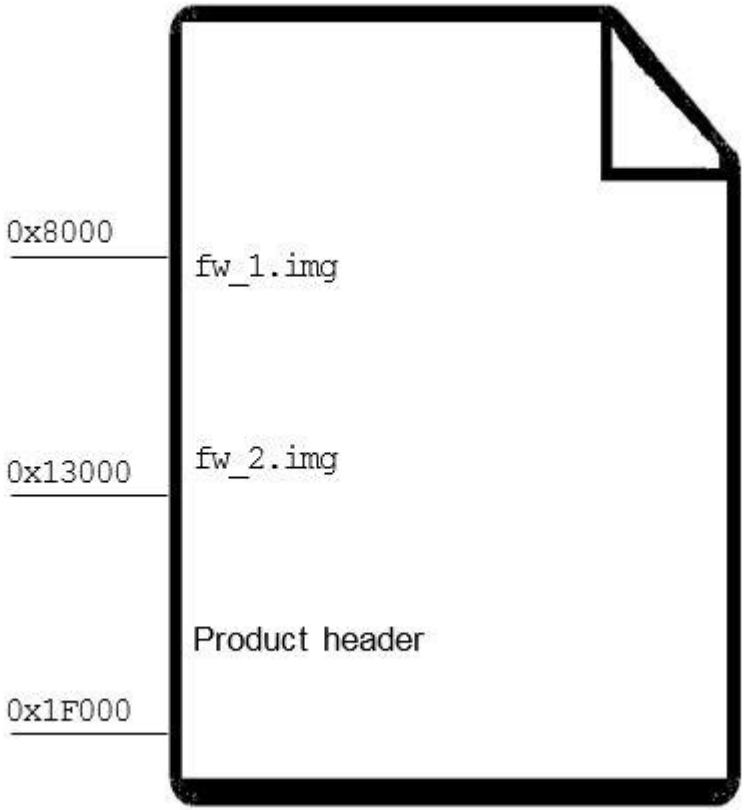
Byte	Field
0,1	Signature (0x70, 0x51)
2	imageid
3	validflag
4-7	Code Size
8-11	CRC
12-27	version
28-31	timestamp
32	Encryption flag
33-63	Reserved

Product header

Byte	Field
0	Signature (0x70)
1	Signature (0x52)
2	Version MS Byte
3	Version LS Byte
4-7	Offset #1
8-11	Offset #2
12-31	Reserved
32-37	BD Address
38	Reserved
39	XTAL 16 Trim Enable
40-43	XTAL 16 Trim Value
44-63	Reserved
64	NVDS

Multi-part binary in OTP memory

OTP memory management



Multi-part binary in OTP memory

How to activate in the python script

Find '**BOOT_2ND_LOADER_IN_OTP = False**' and make it true in the script

Why to use python script

- The process of creating multi-part binary along with 2 firmware images in not a straight forward process.
- You need to execute different Dialog utility software stored in different places, in the SDK 5.0.3.
- To keep it simple, a python script is created where you can set all you input in the # **USER DATA CONFIGURATION SECTION** #
- Place all you necessary files in the '**input folder**' and you will generate a very nice output folder containing files necessary to run your SOUTA application from OTP or from Secondary boot loader stored in FLASH memory.

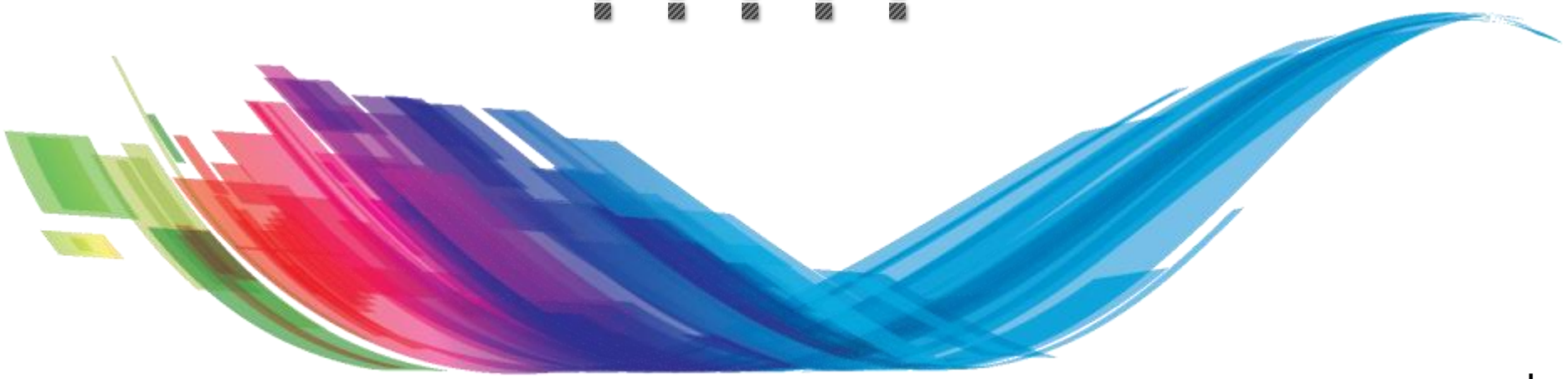
BLE Contents

Reference

- https://www.wikiwand.com/en/Over-the-air_programming
- http://support.dialog-semiconductor.com/connectivity
- **Register with us for extensive support**
  - http://support.dialog-semiconductor.com/user/register
  - Dialog semiconductor application note '**AN-B-010** DA14580 using SUOTA'

The Power To Be…

?????



…personal
…portable
…connected