Dialog SDK 5 Training Materials – SUOTA (Software Update Over The Air)
2016 February
SUOTA overview

Step by step procedure for SUOTA

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 (OTA)** programming 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** 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).
**Overview**

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

<table>
<thead>
<tr>
<th>Advantages</th>
<th>External Non-volatile memory</th>
<th>Internal OTP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OTP can stay blank. Useful for development purposes and/or when very low power consumption is not a requirement for the final product.</td>
<td>Fastest boot-up time. Guarantee to boot up anytime.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disadvantages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td></td>
<td>OTP must be burnt.</td>
</tr>
</tbody>
</table>
SUOTA

Step by step procedure for SUOTA

What would you see as output
SUOTA example

- **Proximity reporter** 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

- KEIL 5 IDE used, Dialog semiconductor SDK 5 used

- Project location: ..\projects\target_apps\ble_examples\prox_reporter\Keil_5\n
- Python 3.5 installation

- DA1458x_SUOTA_Multipart_Binary_Generator.zip from training 6
SUOTA files in proximity reporter software project architecture

DA1458x - PRO development kit HW configuration
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
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

    #include "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"
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_ID\
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 USERDEVICE_NAME ("SUOTA-1")
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 USERDEVICE_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 - Install python 3.5.1
Let’s do it … preparation for the demo

TODO 14 – unzip DA1458x_SUOTA_Multipart_Binary_Generator.zip

TODO 15 – copy and paste your input folder contents inside DA1458x_SUOTA_Multipart_Binary_Generator folder (please refer to Readme.txt for more information)

The input folder looks like

![Folder contents]

TODO 16 – rename bin\mkimage_580.exe to mkimage.exe;
(User can also copy file SDK5\binaries\host\windows\mkimage\mkimage.exe to the binary folder)

TODO 17 – rename input\secondary_bootloader_580.hex to secondary_bootloader.hex
(User can also build the project SDK5\utilities\secondary_bootloader\secondary_bootloader.uvprojx and copy \Out\secondary_bootloader.hex to the input folder)
TODO 18 - run command prompt and go to `DA1458x_SUOTA_Multipart_Binary_Generator` folder

TODO 19 - 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 = "06A9214036B8A15B512E03D53412006"
IMG_ENC_INIT_VEC_DEF = "3DAFBA429D9EB430B422DA802C9FAC41"
```

TODO 20 - execute "python project_multpart_binary_v2.py".

TODO 21 - Check the output folder and you will find `fw_multi_part_spi.bin` is created.
**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
Primary bootloader
(including an AN-B-001 SPI header)

fw_1.img

fw_2.img

Product header

Secondary bootloader
### Explanation of memory management

#### Header

<table>
<thead>
<tr>
<th>Byte</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signature (0x70)</td>
</tr>
<tr>
<td>1</td>
<td>Signature (0x50)</td>
</tr>
<tr>
<td>2-5</td>
<td>Dummy Bytes</td>
</tr>
<tr>
<td>6</td>
<td>Code Size MS Byte</td>
</tr>
<tr>
<td>7</td>
<td>Code Size LS Byte</td>
</tr>
<tr>
<td>8-</td>
<td>Code Bytes</td>
</tr>
</tbody>
</table>

#### Product header

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

#### Same header for the 2 images

<table>
<thead>
<tr>
<th>Byte</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>Signature (0x70, 0x51)</td>
</tr>
<tr>
<td>2</td>
<td>imageid</td>
</tr>
<tr>
<td>3</td>
<td>validflag</td>
</tr>
<tr>
<td>4-7</td>
<td>Code Size</td>
</tr>
<tr>
<td>8-11</td>
<td>CRC</td>
</tr>
<tr>
<td>12-27</td>
<td>version</td>
</tr>
<tr>
<td>28-31</td>
<td>timestamp</td>
</tr>
<tr>
<td>32</td>
<td>Encryption flag</td>
</tr>
<tr>
<td>33-63</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
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

![Image of a software interface](image)

1. **Select File**
2. **Browse**
3. **Address**
4. **Write**
5. **Erase**
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.
Running SUOTA from iOS platform

File Sharing
The apps listed below can transfer documents between your iPhone and this computer.

<table>
<thead>
<tr>
<th>Apps</th>
<th>SUOTA Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>iMovie</td>
<td></td>
</tr>
<tr>
<td>IoT Sensor</td>
<td></td>
</tr>
<tr>
<td>Keynote</td>
<td></td>
</tr>
<tr>
<td>Numbers</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td></td>
</tr>
<tr>
<td>SUOTA</td>
<td></td>
</tr>
</tbody>
</table>

fw_image_1.png
fw_image_2.png
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.
- 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.
<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Suota Logo" /></td>
<td><img src="image2" alt="Devices Screen" /></td>
<td><img src="image3" alt="Device Details" /></td>
<td><img src="image4" alt="File Selection" /></td>
</tr>
</tbody>
</table>

**SUOTA**

Device name: SUOTA-1
Manufacturer: Dialog Semi
Model nr.: DA14580
Firmware rev.: v_3.0.6.1
Software rev.: v_3.50.1.54

Update

- fw_image_1.img 27024kb
- fw_image_2.img 27024kb
### Step 5

**Select a file...**
- Selected file: fw_2.img
- Select memory type:
  - I2C
  - SPI
  - MISO GPIO
  - MOSI GPIO
  - P0.5
  - P0.6
  - CS GPIO
  - SCK GPIO
  - P0.3
  - P0.0
- Image bank:
  - Oldest
  - Bank 1
  - Bank 2
- Block size: 240

**Send to device**

### Step 6

**Back**

Your new firmware is being uploaded. Please wait until this process is completed.

- 57%

### Step 7

**Back**

Your new firmware is being uploaded. Please wait until this process is completed.

- 100%

**Device has been updated**

Do you wish to reboot the device?

- No
- Yes, reboot

### Step 8

**Device**

- **Device name:** SUOTA-2
- **Manufacturer:** Dialog Semi
- **Model nr.:** DA14580
- **Firmware rev.:** v.3.0.6.2
- **Software rev.:** v.3.50.1.54

**Update**
SUOTA

Before we end ...
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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>Signature (0x70, 0x51)</td>
</tr>
<tr>
<td>2</td>
<td>imageid</td>
</tr>
<tr>
<td>3</td>
<td>validflag</td>
</tr>
<tr>
<td>4-7</td>
<td>Code Size</td>
</tr>
<tr>
<td>8-11</td>
<td>CRC</td>
</tr>
<tr>
<td>12-27</td>
<td>version</td>
</tr>
<tr>
<td>28-31</td>
<td>timestamp</td>
</tr>
<tr>
<td>32</td>
<td>Encryption flag</td>
</tr>
<tr>
<td>33-63</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Product header

<table>
<thead>
<tr>
<th>Byte</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signature (0x70)</td>
</tr>
<tr>
<td>1</td>
<td>Signature (0x52)</td>
</tr>
<tr>
<td>2</td>
<td>Version MS Byte</td>
</tr>
<tr>
<td>3</td>
<td>Version LS Byte</td>
</tr>
<tr>
<td>4-7</td>
<td>Offset #1</td>
</tr>
<tr>
<td>8-11</td>
<td>Offset #2</td>
</tr>
<tr>
<td>12-31</td>
<td>Reserved</td>
</tr>
<tr>
<td>32-37</td>
<td>BD Address</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>XTAL 16 Trim Enable</td>
</tr>
<tr>
<td>40-43</td>
<td>XTAL 16 Trim Value</td>
</tr>
<tr>
<td>44-63</td>
<td>Reserved</td>
</tr>
<tr>
<td>64</td>
<td>NVDS</td>
</tr>
</tbody>
</table>
Multi-part binary in OTP memory

OTP memory management

0x8000  fw_1.img
0x13000  fw_2.img
0x1f000  Product header
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 was not a straightforward process.

- You need to execute different Dialog utility software stored in different places, in the SDK 5.0.x to generate these images.

- To keep it simple, a python script is created where you can set all your input in the # USER DATA CONFIGURATION SECTION # of project_multipart_binary_v2.py

- Place all your necessary files in the ‘input folder’ and the script will generate an ‘output folder’ containing the files necessary to run your SOUTA application from OTP or from Secondary boot loader stored in FLASH memory.
BLE Contents

Reference

- http://support.dialog-semiconductor.com/connectivity

**Register with us for extensive support**

- [http://support.dialog-semiconductor.com/user/register](http://support.dialog-semiconductor.com/user/register)
- Dialog semiconductor application note ‘**AN-B-010 DA14580 using SUOTA**’
- Dialog semiconductor user manual ‘**UM-B-012 DA14580/581/583 Creation of a secondary bootloader**’
The Power To Be...