Abstract

This document describes the architecture and the implementation details of the Serial Port Service application that runs on the DA14531 and DA14585 development kits.
Contents

Abstract ......................................................................................................................... 1

1 Terms and Definitions .............................................................................................. 5

2 References .................................................................................................................. 5

3 Introduction ................................................................................................................ 6

4 Architecture ............................................................................................................... 6
  4.1 Application Task State Machine ............................................................................ 7
    4.1.1 Device State Machine .................................................................................... 7
    4.1.2 Host State Machine ....................................................................................... 8
  4.2 Data Scheduling .................................................................................................... 10
    4.2.1 UART to BLE Transmission .......................................................................... 10
    4.2.2 BLE to UART Transmission .......................................................................... 10
    4.2.3 BLE Flow Control ......................................................................................... 11
  4.3 Serial Port Service (SPS) ..................................................................................... 12
  4.4 Configuration Storage .......................................................................................... 12
    4.4.1 Configuration Image Format ......................................................................... 12
    4.4.2 Configuration Storage Mechanism ............................................................... 14
  4.5 Remote Configuration Service .............................................................................. 15
  4.6 DSPS Configuration .............................................................................................. 17
    4.6.1 Default Values ............................................................................................... 17
  4.7 Update Configuration over SUOTA ....................................................................... 20

5 Secondary Bootloader ............................................................................................... 20

6 How to Build .............................................................................................................. 20
  6.1 To Build a DSPS Device Application ................................................................. 21
    6.1.1 Compilation Switches for DSPS Device Application .................................... 21
  6.2 To Build the DSPS Host Application .................................................................. 22
    6.2.1 Compilation Switches for DSPS Host Device Application .......................... 23

7 Images and Binaries .................................................................................................. 24
  7.1 To Build SUOTA Ready SW Images ................................................................... 24
  7.2 Create and Edit Configuration File ....................................................................... 24
  7.3 Create a Multi-Image for First Time Programming ............................................. 26
  7.4 CLI Flash Programming ....................................................................................... 26
    7.4.1 Hardware setup ........................................................................................... 27
    7.4.2 How to Use cli_flash_programming ......................................................... 27
  7.5 Release Binaries ................................................................................................... 28
    7.5.1 DA14585 Flash Memory Map ..................................................................... 29
    7.5.2 DA14531 Flash Memory Map ..................................................................... 30

8 SmartConsole Android and iOS Application ............................................................. 30
  8.1 Overview ............................................................................................................... 30
  8.2 Installation ............................................................................................................. 31
    8.2.1 Android Application ..................................................................................... 31
    8.2.2 iOS Application ............................................................................................ 32
  8.3 Device List ............................................................................................................. 33
  8.4 Functionality Tabs ................................................................................................. 33
DA14585-531 Serial Port Service Reference Application

8.4.1 Console Mode ................................................................. 35
8.4.2 Read/Transfer Mode ......................................................... 35
8.4.3 Data File Streaming Mode ....................................................... 35
8.5 Information and Disclaimer Screens ........................................... 35

9 SmartConfig ........................................................................... 36
9.1 Installation ............................................................................. 36
  9.1.1 Android Application ......................................................... 36
  9.1.2 iOS application ................................................................. 37
9.2 Device List ............................................................................ 38
9.3 Functionality ......................................................................... 39
9.4 Information and Disclaimer Screens ......................................... 40

10 Hardware Setup ..................................................................... 41
  10.1 DA14585 DK ..................................................................... 41
  10.2 DA14531 DK ..................................................................... 42
  10.3 DA14531 Module ................................................................. 43

11 SPS Performance .................................................................. 44

Appendix A Configuration Parameters .......................................... 46

Appendix B DSPS Configuration Parameters ................................ 48
  B.1 DSPS Device for DA14531 ...................................................... 48
  B.2 DSPS Device for DA14585 ...................................................... 48
  B.3 DSPS Host for DA14531 and DA14585 .................................... 49

Revision History .......................................................................... 51

Figures
Figure 1: DSPS Architecture .......................................................... 7
Figure 2: Device State Machine ......................................................... 8
Figure 3: Host State Machine ............................................................. 9
Figure 4: State Transitions Related to Heap Usage ............................. 11
Figure 5: Configuration Element ....................................................... 14
Figure 6: Configuration Structure ..................................................... 14
Figure 7: Notification Data Format in Discover Elements .................... 15
Figure 8: Write Configuration Data Format In Write Elements .......... 16
Figure 9: Notification Data Format in Write Elements Characteristic ..... 16
Figure 10: Indication Data Format in Read Elements Characteristic ..... 17
Figure 11: Configurator File Menu ................................................... 25
Figure 12: Edit/Delete an Element of the Configuration File ............... 26
Figure 13: DA14531 and DA14531 Module ProDev Kit Motherboard Jumper Configuration .......................... 27
Figure 14: SmartConsole Icon on Android and iOS Device .............. 31
Figure 15: SmartConsole on Play Store ........................................ 32
Figure 16: SmartConsole Application Installation ............................ 32
Figure 17: Scan Device Screen ..................................................... 33
Figure 18: Select Operation Mode ................................................... 34
Figure 19: Application Main Screens ............................................... 34
Figure 20: Information & Disclaimer Screens .................................... 35
Figure 21: SmartConfig Icon – Android – iOS .................................. 36
Figure 22: SmartConfig on Play Store ............................................ 37
Figure 23: SmartConfig Application Installation ............................. 37
Figure 24: SmartConfig in App Store ............................................. 38
Figure 25: Scan Screen ............................................................... 39
Figure 26: Read and Edit Configuration Parameters .......................................................... 40
Figure 27: Information & Disclaimer Screens ................................................................. 41
Figure 28: J1 Jumpers Setup for DA14585 DK .............................................................. 42
Figure 29: DA14531 DK on Mainboard 376-18-B for DSPS ........................................... 43
Figure 30: J1 jumpers setup for the DA14531 TINY™ Module ..................................... 44

Tables

Table 1: Device Application Task States ........................................................................... 7
Table 2: Host Application Task States ............................................................................. 8
Table 3: Device Services/Characteristics ......................................................................... 12
Table 4: Configuration Struct Header ............................................................................. 13
Table 5: Characteristics of the Device Configuration Service .......................................... 15
Table 6: Error Codes of Remote Configuration Service ................................................... 17
Table 7: DA14585 Flash Memory Map ............................................................................ 29
Table 8: DA14585 Product Header ................................................................................ 30
Table 9: DA14531 Flash Memory Map ........................................................................... 30
Table 10: DA14531 Product Header ............................................................................... 30
Table 11: Performance Results for the DA14531 Device ................................................ 44
Table 12: Performance Results for the DA14585 Device ................................................ 45
Table 13: Configuration Parameters .............................................................................. 46
Table 14: DSPS Device Configuration Structure for DA14531 ......................................... 48
Table 15: DSPS Device Configuration Structure for DA14585 ......................................... 48
Table 16: DSPS Host Configuration Structure for DA14531 and DA14585 ..................... 49
1 Terms and Definitions

BLE  Bluetooth Low Energy
DK   Development Kit
SPS  Serial Port Service
DSPS Dialog Serial Port Service
GAP  Generic Access Profile
GPIO General Purpose Input/Output
DLE  Data Length Extension
UART Universal Asynchronous Receiver/Transmitter

2 References

3 Introduction

The Serial Port Service (SPS) emulates serial cable communication. SPS provides a simple substitute for RS-232 connections, which includes the familiar software flow control logic via Bluetooth® Low Energy (BLE). The SPS software distribution includes the application and profile source codes.

The software has been developed for the DA14585 Development Kit (DK) – PRO, DA14531 Development Kit (DK) – Basic or DA14531 TINY™ Module. Software is also developed for Android and iOS tablets and mobile phones, to allow a serial port to be emulated when two DA14585/DA14531 DKs are used, or a DA14585/DA14531 DK and an Android or iOS device. The DA14585 DK can either function in the GAP central role, or in the peripheral role. The Android or iOS device only functions in the GAP central role.

There are two projects provided, a host and a device project that operate in a BLE central and a peripheral role respectively. Both projects use DMA driven UART operation for baud rates up to 921600 that use a hardware flow control only, which is optimized for DLE operation and high speeds.

Both host and device applications are fully configurable with the preprogrammed configuration file in SPI flash. The device application supports a remote configuration service server and can be controlled with a connected smartphone app in runtime. SmartConfig, an Android/iOS mobile phone app can be used to discover, connect and monitor or change the configuration of DSPS devices through a remote configuration BLE service.

The application on the central device automatically starts to scan and connects to the selected device that is in the configuration file or the first peripheral device that supports the serial port service. The Central device also handles situations of connection loss where the flow of data is stopped and automatically tries to re-establish a connection.

DSPS projects are implemented on top of SDK_6 for DA14585/DA14531. To get familiar with the software and hardware of DA14585, read the following documents:

- UM-B-119, DA14585-DA14531 SW Platform Reference [1]
- UM-B-049, DA14585 & DA14586 Getting Started Guide with the PRO-Development Kit [2]
- UM-B-117, DA14531 Getting Started with the Pro Development Kit [3]

4 Architecture

The main software blocks and the high level architecture of DSPS applications are shown in Figure 1.
4.1 Application Task State Machine

4.1.1 Device State Machine

When the code starts to run, a task is created for the main application. The device task application consists of three states as shown in Table 1.

Table 1: Device Application Task States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP_DISABLED</td>
<td>SPS server is disabled, database creation has started</td>
</tr>
<tr>
<td>APP_CONNECTABLE</td>
<td>Database is created, DSPS server is enabled, advertising started</td>
</tr>
<tr>
<td>APP_CONNECTED</td>
<td>Advertising stopped, a central device connected</td>
</tr>
</tbody>
</table>

The first state is **APP_DISABLED**. During this state the SPS server is being enabled, the UART baudrate is set, and the database for the SPS server is created. The database is kept in the server’s profile and its service and characteristics are described in Section 4.3.

After the database is created, the advertising procedure begins. This means that the device advertises the SPS server UUID.

Also two linked lists with zero members are allocated, which will be used for data scheduling. The first list, which will from now on be called **Rx list**, holds buffers received by the UART and that wait to be transmitted by the BLE interface. The second list, which will from now on be called **Tx list**, holds buffers received by the BLE interface and that wait to be transmitted by the UART. Details about data scheduling are described in Section 4.2.

Now the device is in **APP_CONNECTABLE** state and any host that also supports the serial port service can connect to it.

During the connection procedure if the device gets an invalid connection id, the device returns **APP_CONNECTED** state. Otherwise, the device enters **APP_CONNECTED** state and the advertising stops, the created profiles and services are enabled, gets the features of the host device and starts the param update procedure. After completion of the parameter update procedure the data length is set, the default tx size, and finally the MTU exchange procedure starts. Afterwards, independent of the sleep mode that is used (by default the device is in mode sleep on), the following initializations take place:

- Set the default tx size to 244
– Initialize the UART and the DMA channels for the specific connection
– Allocate a DMA Rx buffer and trigger to read from DMA
– Set UART RTS pin to low in order to be ready to receive data

Now the device is ready to receive or send data. Therefore, the device starts asynchronous checks to see either if data are available from BLE to send to the UART Tx queue, or data are available from the UART Rx queue to send over BLE. More details about data scheduling are described in Section 4.2.

In case a disconnection takes place, the following things will happen:
– Set the tx size to 23
– Advertising starts again and a device task application enters APP_CONNECTABLE state
– If there is data to be send over BLE, this data will not be lost, but stored in the Rx list queue, in order to continue to send them to the host device when reconnected
– Set UART RTS pin to high in order not to receive or send any data from/to the UART interface
– The DMA UART Rx channel is being deactivated

The device application task state machine is shown in Figure 2.

The device application task state machine is shown in Figure 2.

Figure 2: Device State Machine

4.1.2 Host State Machine

When the code starts to run a task is created for the main application. The host task application consists of 3 states as shown in Table 2.

Table 2: Host Application Task States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP_DISABLED</td>
<td>SPS client is disabled.</td>
</tr>
<tr>
<td>APP_CONNECTABLE</td>
<td>Database is created, SPS client is enabled, scanning started.</td>
</tr>
<tr>
<td>APP_CONNECTED</td>
<td>Scanning stopped, connected to a device.</td>
</tr>
</tbody>
</table>
The first state is **APP_DISABLED**. During this state SPS client is being enabled, UART baudrate is set, and the database for the SPS client is created. The database consists of the same characteristics as the server's database, described in Section 4.3.

After database creation finishes the scan procedure begins, where the host scans in order to find a device that supports SPS to connect to it. Now the host enters **APP_CONNECTABLE** state.

Also two linked lists with zero members are allocated, which will be used for data scheduling. One list for Tx and one for Rx. The data scheduling process is described in Section 4.2.

If the host cannot connect to the desired device, the scan procedure stops after some time and restarts again. If the host connects successfully to a device it enters to **APP_CONNECTED** state. Scanning stops and the created profiles and services are enabled.

Afterwards, independent of the sleep mode that is used (by default the device is in sleep on mode), the following initializations take place:

- Set the default tx size to 244
- Initialize the UART and the DMA channels for the specific connection
- Allocate a DMA Rx buffer and trigger to read from DMA
- Set UART RTS pin to low in order to be ready to receive data

Now the device is ready to receive or send data. Therefore, the device starts asynchronous checks to see either if there is data available from BLE to send to the UART Tx queue, or if there is data available from the UART Rx queue to send over BLE. More details about data scheduling are described in Section 4.2.

In case a disconnection takes place, the following things will happen:

- Set the tx size to 23
- Scanning starts again and the device task application enters **APP_CONNECTABLE** state
- If there is data to be send over BLE, this data will not be lost, but stored in the Rx list queue, in order to continue to send them to the device when reconnected
- Set UART RTS pin to high in order not to receive or send any data from/to the UART interface
- The DMA UART Rx channel is being deactivated

The host application task state machine is shown in **Figure 3**.

---

**Figure 3: Host State Machine**
4.2 Data Scheduling

The data scheduling mechanism is implemented in the application task layer. The data flow can be split in two parts:

- Data received by UART and transmitted to the BLE interface (UART->BLE)
- Data received over the BLE interface and transmitted to the UART interface (BLE->UART)

In order to protect the data flow either from UART overflow (UART FIFO is 16 bytes) or heap exhaustion in BLE traffic and possible data loss or system hangs, the use of a Flow control mechanism is necessary. The BLE flow control mechanism is described in detail in Section 4.2.3.

This section describes data scheduling for the host and the device.

At the startup two linked lists with zero members are allocated. The first list, Rx list, holds buffers received by the UART and wait to be transmitted by the BLE interface. The second list, Tx list, holds buffers received by the BLE interface and wait to be transmitted by the UART.

4.2.1 UART to BLE Transmission

In order to minimize data copying and the allocation of heap memory, a buffer optimization scheme is used. The buffers received by DMA UART are put in the Rx list. When data scheduling is done the buffers are detached from the linked list and forwarded from the user space to the profile for transmission.

For the data received over the UART interface, an asynchronous process checks the presence of pending buffers in the Rx list and when there is no data transfer ongoing, it pushes the buffer to the BLE.

When the transfer of buffers over DMA is complete on the UART DMA receive path, an Rx callback puts the new buffer at the end of the Rx list and if the flow of data is not disabled, reinitiates the read of UART via DMA.

There are two different ways to decide if the UART DMA Rx interface is able (flow on state) or not (flow off state) to receive data depending of the chipset that is used.

- DA14531
  The application reads the current heap usage and if the used heap is more than approximately 60%, the device enters flow off state. This means that the RTS signal is high and remains high until heap usage becomes lower than approximately 55%. Then the device enters the flow on state and is able to receive data again on the DMA UART interface.

- DA14585
  For the Rx linked list a buffer utilization level is defined at which the device enters a flow on or a flow off state.

The flow control signaling on the BLE interface is also triggered when the corresponding flow control signal is received from the UART interface. The dma_uart_gpio_callback is triggered when the host CTS signal is toggled. This function sends FLOW_ON or FLOW_OFF over BLE to the peer device in order to block the flow of more data.

4.2.2 BLE to UART Transmission

The buffers that the BLE receives are put in the Tx list. When data scheduling is done the buffers are detached from the linked list and are forwarded from the user space to DMA, for transmission over UART.

An asynchronous process checks the presence of pending buffers in the Tx list and when there is no data transfer ongoing, a transfer to the UART is initiated.

On the UART DMA transmit path a Tx callback is called when the transmission of a buffer over UART is complete. The callback checks if more buffers exist in the Tx list and reinitiates the transfer with a detached buffer from the Tx list.

To decide if the Rx of data from the BLE peer should be blocked (flow off) or be continued (flow on), a buffer utilization level for the Tx linked list is defined. If the contents in the Tx list are more than the...
upper buffer utilization level, a BLE FLOW_OFF packet is sent to peer's BLE in order to block traffic from BLE. If the contents in the Tx list are fewer than the lower buffer utilization level, a BLE FLOW_ON packet is sent to the peer's BLE in order to allow the BLE to continue to send traffic.

4.2.3 BLE Flow Control

BLE traffic is regulated and blocked with the BLE Flow Control mechanism, which is used only for the DA14531 chipset when it receives traffic over BLE. The flow control mechanism depends on the current heap usage. For DA14585 this feature is not supported because it is not needed. Because the available heap size in DA14585 is big enough to handle the traffic. The flow control mechanism is enabled in DA14531's code with the following definitions:

- **CFG_BLE_FLOW_CONTROL**: Enables the flow control mechanism
- **CFG_LOG_HEAP_USAGE**: Gets the current heap usage

The BLE Flow control mechanism consist of three states:

- **DISABLED**: when the heap usage level is lower than 47%
- **REDUCED**: when the heap usage level is between 52% and 63%
- **BLOCKED**: when the heap usage level is higher than 73%

The states are shown in Figure 4.

![Figure 4: State Transitions Related to Heap Usage](image)

When the BLE Flow Control mechanism is in 'DISABLED' state, the BLE subsystem of the DA14531 receives all the packets properly sent from the peer device.

When the BLE Flow Control mechanism is in 'REDUCED' state, the BLE subsystem of the DA14531 receives some of the packets properly sent from the peer device. This triggers the BLE subsystem to NACK other packets that are not received, thus leading the remote peer to do a retransmission. Since the remote peer is properly notified of the NACK, normally it will not trigger a connection timeout on its side. In this state the user can configure how many packets to NACK.

When the BLE Flow Control mechanism is in 'BLOCKED' state, the BLE subsystem of the DA14531 is forced to not receive the packets properly sent from the peer device. This triggers the BLE subsystem to NACK the packets received, thus leading the remote peer to do a retransmission. Since the remote peer is properly notified of the NACK, normally it will not trigger a connection timeout on its side.
But since the DA14531 stack does not receive proper BLE packets, its default behavior would lead to a connection timeout. For this reason, the following custom operation has been added: when an incoming BLE packet is detected, the connection timeout period in the BLE stack is reloaded.

Also, since the BLE subsystem does not receive proper packets, it cannot determine the SN, NESN of the last packet sent by the remote peer, so it cannot prepare and send packets to the remote peer.

4.3 Serial Port Service (SPS)

The Dialog Serial Port Service (DSPS) emulates serial cable communication. DSPS is used to send and receive data, and software flow control signals through a BLE connection. The BLE database has two 250-byte characteristics for serial data transmission and reception, and a one byte characteristic for the flow control. 128-bit UUIDs are used for the service and the characteristics.

<table>
<thead>
<tr>
<th>Service/Characteristic</th>
<th>UUID</th>
<th>Size (bytes)</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPS</td>
<td>0x0783b03e8535b5a07140a304d2495cb7</td>
<td>-</td>
<td>Read</td>
</tr>
<tr>
<td>SPS_SERVER_TX</td>
<td>0x0783b03e8535b5a07140a304d2495cb8</td>
<td>250</td>
<td>Notify</td>
</tr>
<tr>
<td>SPS_SERVER_RX</td>
<td>0x0783b03e8535b5a07140a304d2495cba</td>
<td>250</td>
<td>Write with no response</td>
</tr>
<tr>
<td>SPS_FLOW_CTRL</td>
<td>0x0783b03e8535b5a07140a304d2495cb9</td>
<td>1</td>
<td>Write with no response/ Notify</td>
</tr>
</tbody>
</table>

The Serial Port Service uses a ‘Write with no response’ method for the transmission of data from the GATT client to the GATT server, and a ‘Notify’ method for the reverse path. See Table 3 for more details.

The SPS server or client can request from a peer to stop data transmission over BLE by sending flow control values through the SPS_FLOW_CTRL characteristic. The flow control values are:

FLOW_ON 0x01
FLOW_OFF 0x02

The recipient of the FLOW_OFF value must stop data transmission immediately. Transmission can resume when the value of FLOW_ON is received.

The client sends the flow control value via a GATT Write with no response message and the server sends a notification message.

The SPS client must enable notifications in the CCC attribute of SPS_SERVER_TX and SPS_FLOW_CTRL characteristics to activate the DSPS device’s notifications for data exchange and flow control messages.

4.4 Configuration Storage

The configuration of the DSDS application consists of two parts:

- The Configuration Image
- The Configuration Storage mechanism

The Configuration Image is the image in the flash that keeps the active configuration parameters.

The Configuration Storage mechanism will read from the flash and apply the active configuration parameters or will update them in the flash with the ones desired by the user.

4.4.1 Configuration Image Format

The Configuration Image consists of two parts: header and data.

4.4.1.1 Configuration Header

The first part is the header, which is in the following table:
### Table 4: Configuration Struct Header

<table>
<thead>
<tr>
<th>Byte #</th>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>Signature</td>
<td>0x7054</td>
</tr>
<tr>
<td>2</td>
<td>Valid flag</td>
<td>0xAA</td>
</tr>
<tr>
<td>3</td>
<td>Structure ID</td>
<td>0x00</td>
</tr>
<tr>
<td>4 to 7</td>
<td>CRC</td>
<td>-</td>
</tr>
<tr>
<td>8 to 23</td>
<td>Version</td>
<td>Version</td>
</tr>
<tr>
<td>24, 25</td>
<td>Data size</td>
<td>Depends on application</td>
</tr>
<tr>
<td>26</td>
<td>Number of items</td>
<td>Depends on application</td>
</tr>
<tr>
<td>27</td>
<td>Encryption flag</td>
<td>0</td>
</tr>
<tr>
<td>28 to 63</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

The fields included in the header are:

- **Signature:**
  - A "magic" number that identifies the configuration header.
  - Value: 0x70, 0x54.

- **Valid flag:**
  - Value 0xAA denotes a valid image. The default value in the configuration file should be 0xFF. The DSPS application writes the Valid Flag (0xAA) value in flash when an update is completed, or the mkimage application will do so when a multi-image is built.

- **Structure ID:**
  - A counter for which the application increases the value whenever the configuration struct is updated and stored in an alternative bank.

- **Number of items:**
  - The number of configuration elements in the configuration data.

- **CRC (Not supported - For Future use):**
  - The checksum calculated over the configuration data.

- **Version:**
  - Determines the configuration structure type and version.
  - Can be checked by the application to confirm that the expected data are stored.

- **Data size:**
  - Size of the configuration data (in bytes).

- **Encryption flag (For Future Use):**
  - Indicates whether the configuration data have been encrypted. Value 0x1 means that data is encrypted. Value 0x0 means that data is decrypted. The DSPS application does not use this field at the moment, as encryption is not supported.

In the multi-image, at least one of the Configuration Images must be populated and be valid, in order for the Configuration Storage mechanism to be activated.

#### 4.4.1.2 Configuration Data

The second part is the Configuration Data. Configuration Data is organized in elements. The format of the elements is shown in Figure 5:
The Element ID determines the semantics, type and maximum data size of the configuration data. A full list of the specified Element IDs and the corresponding info is available in Appendix A.

A configuration structure may contain any combination of different configuration fields. An Element ID should be appeared only once in a configuration structure. The actual data size in each element is determined by the element’s length.

The complete format of configuration structure is depicted below:

![Configuration Structure Diagram]

### Figure 5: Configuration Element

4.4.2 Configuration Storage Mechanism

The Configuration Storage Mechanism serves to:

- Read the Configuration Data from the flash
- Apply the Configuration Data to the device
- Store the new Configuration Structure to the flash, when updated by the Remote Configuration Service

In the DSPS application, Configuration Storage support is controlled by the definition CFG_CONFIG_STORAGE. The main functionality is implemented in file user_config_storage.c.

During boot, the Configuration Storage will read the Product Header to get the offset of the two Configuration Images. Then the headers are parsed to read the Valid Flag and the Structure ID flag. The selected Configuration Image is the one that has the Valid Flag 0xAA and the most updated Structure ID. In case that neither Configuration Image is valid, the application will use the hardcoded default values. Once a valid Configuration Image is found, the elements from that Configuration Image are read one by one. If an element is included in the application’s configuration structure, the corresponding Configuration value is updated, otherwise that element is ignored.

The Configuration Storage mechanism is also responsible to save new configuration values to the flash. These new configuration values are updated by the Remote Configuration Service. So, when the Remote Configuration Service updates a configuration parameter, the Configuration Storage is called. In every call, the Configuration Storage will save the whole Configuration Structure in the offset of the Configuration Image that is currently not used. So, at first, it will determine which Configuration Image position to use and then start to create the whole Configuration Image. It will save all the Configuration Data of the DSPS’s Configuration Structure and then start to create the whole Header. First it will set the Valid Flag to Invalid (0xFF) and will keep it like this until all fields are updated. When everything is ready, it will increase the Structure Id by one and set the Valid Flag.
to 0xAA (Valid). In this way, this Configuration Image will become the valid one to be used from now on.

4.5 Remote Configuration Service

The Remote Configuration Service is responsible to inform, update, save and apply the DSPS application's configuration parameters; and does that in real time. Remote Configuration Service support is controlled by the definition CFG_PRF_REMOTE_CONFIG. The main functionality is implemented in the files remote_config.c, remote_config_task.c, user_remote_config.c and user_remote_config_task.c. The Remote Configuration Service is supported only in the Device role, not the Host.

The Remote Configuration Service exposes four characteristics. The characteristics of the service are outlined in Table 5.

Table 5: Characteristics of the Device Configuration Service

<table>
<thead>
<tr>
<th>Characteristic Name</th>
<th>Qualifier</th>
<th>Properties</th>
<th>Max Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration structure version</td>
<td>Mandatory</td>
<td>Read</td>
<td>66</td>
</tr>
<tr>
<td>Discover elements</td>
<td>Mandatory</td>
<td>Write/Notify</td>
<td>130</td>
</tr>
<tr>
<td>Write elements</td>
<td>Mandatory</td>
<td>Write/Notify</td>
<td>133</td>
</tr>
<tr>
<td>Read Element</td>
<td>Mandatory</td>
<td>Write/Notify</td>
<td>132</td>
</tr>
</tbody>
</table>

- **Configuration structure version:**
  - Identifies the type and version of the configuration structure of the application.

- **Discover elements:**
  - The Remote configuration client writes in this characteristic the max number of the element ID that the server must include in the Discover response notification, to trigger the discovery process. The server will respond with one or more Discover response notifications. The max number should not be larger than 64.
  - The server responds to the Discover elements command with one or more Discover responses sent through the same characteristic. The Number of Elements field denotes the number of elements included in the specific message. They should not exceed the max number requested by the remote configuration client.
  - The More field denotes if more messages are following. If it is 1 more discover response messages will follow.
  - If the max number of element IDs creates responses that exceed the MTU size, then it automatically decreases to the maximum value of allowed responses to be transmitted.
  - The element IDs follow in the message.

<table>
<thead>
<tr>
<th>Number of Elements</th>
<th>More</th>
<th>Element 0 ID</th>
<th>Element 1 ID</th>
<th>...</th>
<th>Element n ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>2 bytes</td>
<td>2 bytes</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: Notification Data Format in Discover Elements

- **Write elements:**
  - The Remote configuration client writes the configuration data into this characteristic.
  - The data format is shown in Figure 8:
○ Number of elements: Number of elements to be written in this message.
○ More: Set if more Write configuration messages will follow before the configuration is applied by the device. Cleared if this is the last message and the device must apply the configuration.
○ Payload: Sequence of elements. The number included in a single message depends on the size of the elements. Total size of an element's data must not exceed 128 bytes.
○ The Remote configuration server will reply in a Write configuration process with a notification on the same characteristic. The response will be sent only when the 'More' field in the Write configuration command is clear ('0x0') and will include the following:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Element ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

**Figure 9: Notification Data Format in Write Elements Characteristic**

○ If the write operation completed successfully, the server will send a notification with Error code: 0x00 (No error). Then the Configuration Storage mechanism is called to save the new Configuration Structure to the flash. Afterwards, depending on the parameters that were updated, it may need to take some actions:
  – Connection related parameters: Run Parameters Update.
  – MTU size: Reset chip on the next disconnection, to apply the new MTU.
  – UART Baudrate: Reset chip immediately, to apply the new baudrate.
○ In case the write operation is not completed successfully, the Element ID is the ID of the first element that the error in Error code is detected. If the error code is not related to any Element, then the reserved '0x0000' ID will be used. The list of error codes is displayed in Table 6.
○ The Remote Configuration Server will do some basic data validation, before the updated configuration parameters are saved and applied. This is not done for all parameters. If validation fails, then a response with Error code: "Invalid element data" is generated.
○ If the number of elements is 0, or larger than the elements contained in the DSPS application's Configuration Structure, then a response with Error code: "Invalid number of elements" is generated.
○ If one of the element IDs is not included in the DSPS application's Configuration Structure, then a response with Error code: "Unknown element ID" is generated.
○ If the length of one of the elements is larger than the element's max size, then a response with Error code: "Invalid element length" is generated.
During the whole write operation, if an error happens on any of the elements, then the element's data will not be stored and applied.

Table 6: Error Codes of Remote Configuration Service

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>No error</td>
</tr>
<tr>
<td>0x01</td>
<td>Large element</td>
</tr>
<tr>
<td>0x02</td>
<td>Invalid number of elements</td>
</tr>
<tr>
<td>0x03</td>
<td>Unknown element ID</td>
</tr>
<tr>
<td>0x04</td>
<td>Invalid element length</td>
</tr>
<tr>
<td>0x05</td>
<td>Invalid element data</td>
</tr>
</tbody>
</table>

- **Read Elements:**
  - The Remote configuration client writes a parameter ID into this characteristic, and requests the command server to return the current values of the parameters.
  - The Remote configuration server sends a notification including the configuration data of a parameter, whenever the client requests it by writing the parameter ID to the read command characteristic.
  - The format of the notification data is shown in Figure 10.
  - If the read operation is completed successfully, the server will send a notification with Error code: 0x00 (No error) in the status field.
  - If the read operation is not completed successfully, it will send a notification where the status field will contain one of the Error codes of Table 6. No Data will be sent in this case.
  - If the element ID is not included in the DSPS application's Configuration Structure, then a response with Error code: "Unknown element ID" is generated.
  - If the length of the element is larger than the notification's maximum data allowed (128 bytes), then a response with Error code: "Invalid element length" is generated.
  - If the notification created is larger than the MTU size used, the Remote configuration server breaks the response into smaller fragments, according to the MTU size, and sends each of them as a separate notification response. All the notification responses will use "Large Response" as status, except the last one which will use "No Error".

![Figure 10: Indication Data Format in Read Elements Characteristic](image)

4.6 DSPS Configuration

The DSPS Configuration Structures for both chips DA14531 and DA14558, and both roles Device and Host, are presented in Appendixes B.1, B.2 and B.3. These are the structures handled by the code.

4.6.1 Default Values

The default values of the Configuration Structures, both for the Device role and the Host role are shown in Table 14, Table 15 and Table 16 of Appendix B. These values are hardcoded and will be used in case no valid Configuration Image is present in the flash.
The following section is a guide of how to change these hardcoded values in the code.

### 4.6.1.1 Device and Host Roles Element IDs:

- **ELEM_ID_GAP_MTU**: Contains the MTU value to be used. Is located in the configuration structure `user_gapm_conf` in file `user_sps_device_config.c`, member `channel_map`. The default value is initialized by definition `DEFAULT_MTU`. If changed by the Remote Configuration Service, the device will reset at the next disconnection.

- **ELEM_ID_CONN_INTERVAL_MIN**, **ELEM_ID_CONN_INTERVAL_MAX**: Contain the minimum and maximum Connection Interval. They are located in the configuration structure `user_connection_param_conf` in file `user_sps_device_config.c`, members `intv_min` and `intv_max`. The default values are initialized with hardcoded values. If changed by the Remote Configuration Service, the device will issue a Connection Parameters Request with the updated values.

- **ELEM_ID_CONN_LATENCY**: Contains the Connection Slave Latency. Is located in the configuration structure `user_connection_param_conf` in file `user_sps_device_config.c`, member `latency`. The default value is initialized with a hardcoded value. If changed by the Remote Configuration Service, the device will issue a Connection Parameters Request with the updated value.

- **ELEM_ID_CONN_TIME_OUT**: Contains the Supervision Timeout. Is located in the configuration structure `user_connection_param_conf` in file `user_sps_device_config.c`, member `time_out`. The default value is initialized with a hardcoded value. If changed by the Remote Configuration Service, the device will issue a Connection Parameters Request with the updated value.

- **ELEM_ID_USER_ADV_DATA**: Contains the Data of the Advertising frame. Is located in the configuration array `user_advertise_data` in file `user_sps_device_config.c`. The default value is initialized with definition `USER_ADVERTISE_DATA`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_USER_ADV_DATA_LEN**: Contains the length of the Advertising Data. Is located in the configuration variable `user_advertise_data_len` in file `user_sps_device_config.c`. The default value is initialized with definition `USER_ADVERTISE_DATA_LEN`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_PERIPH_UART_BAUDRATE**: Contains the UART's baudrate. Is located in the configuration variable `uart_baudrate_param` in file `user_sps_device_config.c`. The default value is initialized with definition `BAUDRATE_CONFIG`. If changed by the Remote Configuration Service, the device will reset immediately.

### 4.6.1.2 Device Role Only Element IDs:

- **ELEM_ID_ADV_INTERVAL_MIN**, **ELEM_ID_ADV_INTERVAL_MAX**: Contain the minimum and maximum Interval for Advertising. They are located in the configuration structure `user_adv_conf` in file `user_sps_device_config.c`, members `intv_min` and `intv_max`. The default values is initialized by definition `DEFAULT_ADVERTISE_INTERVAL`. If changed by the Remote Configuration Service, the new values are adopted immediately.

- **ELEM_ID_ADV_CHANNEL_MAP**: Contains the Advertising Channel Map. Is located in the configuration structure `user_adv_conf` in file `user_sps_device_config.c`, member `channel_map`. The default value is initialized by enumeration `adv_channel_map`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_USER_ADV_SCAN_RESP_DATA**: Contains the Data of the Advertising Scan Response. Is located in the configuration array `user_adv_scan_resp_data` in file `user_sps_device_config.c`. The default value is initialized with definition `USER_ADVERTISE_SCAN_RESPONSE_DATA`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_USER_ADV_SCAN_RESP_DATA_LEN**: Contains the length of the Data of the Advertising Scan Response. Is located in the configuration variable `user_adv_scan_resp_data_len` in file `user_sps_device_config.c`. The default value is
initialized with definition USER_ADVERTISE_SCAN_RESPONSE_DATA_LEN. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_USER_DEVICE_NAME**: Contains the Name of the Device. Is located in the configuration structure `device_info` in file `app.c`, member `dev_name.name`. The default value is initialized with definition `USER_DEVICE_NAME`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_USER_DEVICE_NAME_LEN**: Contains the length of Device Name. Is located in the configuration structure `device_info` in file `app.c`, member `dev_name.length`. The default value is initialized with definition `USER_DEVICE_NAME_LEN`. If changed by the Remote Configuration Service, the new value is adopted immediately.

- **ELEM_ID_GPIO_UART1_XXX (XXX = RX, TX, CTS, RTS)**: Contain the GPIO port and GPIO pin used for the UART pins. They are located in configuration structures `gpio_uart1_XXX`. The default values are initialized with definitions `GPIO_UART1_XXX_PIN` and `GPIO_UART1_XXX_PORT`. If changed by the Remote Configuration Service, the device will reset immediately.

- **ELEM_ID_GPIO_ACTIVE_STATUS_PIN**: Contains the GPIO port and GPIO pin to be used as an `ACTIVE_STATUS` pin. Is located in the configuration structure `gpio_reset_status`. The default value is initialized with definitions `GPIO_ACTIVE_STATUS_PIN` and `GPIO_ACTIVE_STATUS_PORT`. If changed by the Remote Configuration Service, the device will reset immediately. If the values are set to `GPIO_PORT_INVALID (0xFF)` and `GPIO_PIN_INVALID (0xFF)`, the Active Status functionality is disabled.

- **ELEM_ID_GPIO_PORT_PIN**: Contains the GPIO port and GPIO pin to be used as the Power On Reset pin. Is located in the configuration structure `gpio_por_pin`. The default value is initialized with definitions `GPIO_PORT_PIN` and `GPIO_PORT`. If changed by the Remote Configuration Service, the device will reset immediately. If the values are set to `GPIO_PORT_INVALID (0xFF)` and `GPIO_PIN_INVALID (0xFF)`, the Power On Reset functionality is disabled.

- **ELEM_ID_GPIO_PORT_PIN_POLARITY**: Contains the Polarity of the GPIO used as Power On Reset pin. Is located in the configuration variable `gpio_por_pin_polarity` in file `user_sps_device_config.c`. The default value is initialized with enumeration `GPIO_PORT_PIN_POLARITY`. If changed by the Remote Configuration Service, the device will reset immediately.

- **ELEM_ID_GPIO_PORT_PIN_TIMEOUT**: Contains the Timeout of Power On Reset functionality. Is located in the configuration variable `gpio_por_pin_timeout` in file `user_sps_device_config.c`. The default value is initialized with hardcoded value. If changed by the Remote Configuration Service, the device will reset immediately. If the value set to 0, the functionality is disabled.

### 4.6.1.3 Host Role Only Element IDs:

- **ELEM_ID_CONN_CE_LEN_MIN, ELEM_ID_CONN_CE_LEN_MAX**: Contain the minimum and maximum Connection Event Duration. They are located in the configuration structure `user_connection_param_conf` in file `user_sps_host_config.c`, members `ce_len_min` and `ce_len_max`. The default values are initialized with hardcoded values.

- **ELEM_ID_CENTRAL_CODE**: Contains the Operation of the device. Is located in the configuration structure `user_central_conf` in file `user_sps_host_config.c`, member `code`. The default value is initialized with enumeration `gapm_operation`.

- **ELEM_ID_CENTRAL_SCAN_INTERVAL**: Contains the Scan Interval. Is located in the configuration structure `user_central_conf` in file `user_sps_host_config.c`, member `scan_interval`. The default value is initialized with a hardcoded value.

- **ELEM_ID_CENTRAL_SCAN_WINDOW**: Contains the Scan Window size. Is located in the configuration structure `user_central_conf` in file `user_sps_host_config.c`, member `scan_window`. The default value is initialized with a hardcoded value.

- **ELEM_ID_CENTRAL_PEER_ADDR_0**: Contains the Address of the first Client device to look for. Is located in the configuration structure `user_central_conf` in file
4.7 Update Configuration over SUOTA

The SUOTA mechanism is used to update the application image in the device's flash. After a successful update, this image becomes the active one. See reference document [1].

The SUOTA mechanism is upgraded to support configuration images as well. This way, the user can update the whole configuration structure without the need to change all parameters one by one. In the SUOTA mobile application the user selects the desired configuration image and updates the device. In the device, the SUOTA service, with the use of the image's header, identifies that the new image is a configuration image and stores the new image to the suitable offset in the flash, overwrites in this way the configuration image that is not currently active. If the new image is successfully stored, the SUOTA application makes this the active one.

5 Secondary Bootloader

The secondary bootloader is a separate application which is located under:

utilities\secondary_bootloader\secondary_bootloader.uvprojx

In order to build the secondary bootloader, choose the desired target DA14585 for the DA14585 chipset or DA14531 for the DA14531 chipset. After a successful build a secondary_bootloader_585.hex or a secondary_bootloader_531.hex are located under:

utilities\secondary_bootloader\out_DA14585\Objects\ or

utilities\secondary_bootloader\out_DA14531\Objects\

At power up or reset of the DA14585/DA14531, the primary boot code (ROM code) loads the secondary bootloader (from OTP memory or flash) and the secondary bootloader proceeds and copies the FLASH image to the system RAM and programs the execution.

The secondary bootloader allows the DA14585/DA14531 to boot from an external SPI Flash memory or the UART interface. Support for 1-wire and 2-wires UART exists for DA14531. For more information about the 1-wire and 2-wires UART configuration for DA14531, see reference document [5].

The secondary bootloader supports a dual image boot scheme, which is used in Software Update Over the Air (SUOTA) applications to update product firmware in the field. The dual image bootloader specifies how two application images can be stored in an external SPI Flash memory and defines how the current image is determined. The secondary bootloader application is configured to operate in dual image mode if the option SUPPORT_AN_B_001 is undefined in the header file bootloader.h. Application image decryption support is enabled if the option AES_ENCRYPTED_IMAGE_SUPPORTED is defined to 1 in the same header file.

Please see the flash memory map of DA14585 (Section 7.5.1) and DA14531 (Section 7.5.2) in order to see where the secondary bootloader is located in the flash memory.

6 How to Build

The Serial port service consists of two parts:

- Server (peripheral device or just device)
- Client (central or host device)
This section describes the way how to build the corresponding projects for a device or a host with different compilation options during the build and the way to run them when the debugger is used. Aside from the debugger, the program can run with SmartSnippets or be burned into the flash. These procedures will be described in other sections.

6.1 To Build a DSPS Device Application

1. Extract the DSPS application zip file.
2. Use Keil uVision to open
   
   projects\target_apps\dsps\dsps_device\Keil_5\dsps_device.uvprjx.

3. Select the desired target:
   Choose **dsps_531** for the DA14531 chipset.
   or
   Choose **dsps_585** for the DA14585 chipset.

4. To use the default compilation options, just press **F7** to build the project.
   **NOTE:** To change any of the compilation switches, please follow the instructions described in section 6.1.1.

5. After a successfully build, a **dsps_device_531.hex** or **dsps_device_585.hex** file is produced in directory:
   
   \target_apps\dsps\dsps_device\Keil_5\out_531\  or
   \target_apps\dsps\dsps_device\Keil_5\out_585\ 
   for either DA14531 or DA14585 respectively.

6. Configure the debugger:
   a. Press ‘Alt + F7’.
   b. In the **Debug** tab, click the **Settings** button.
   c. Select the appropriate **SN** and then in **Port** field ‘**SW**’.
   d. Click **Apply** in order to save.
   e. Click **OK** to exit.

7. Start debugging session by pressing ‘**Ctrl + F5**’, run the program by pressing ‘**F5**’. Now the device is in Connectable state and any Central device is able to detect it and connect to it.

8. Stop the debugger session by pressing again ‘**Ctrl + F5**’.

6.1.1 Compilation Switches for DSPS Device Application

Before the DSPS device application is compiled, the user can modify the following configuration options:

- In **user_profiles_config.h**, there are four profiles defined by default:
  - **CFG_PRF_SPSS**: Profile of SPS server, needs to be defined.
  - **CFG_PRF_REMOTE_CONFIG**: Profile for remote configuration. If undefined, the functionality of Remote Configuration is disabled.
  - **CFG_PRF_SUOTAR**: Profile for Software update over the air. If defined, it can be also used to update the device’s Configuration Image.
  - **CFG_PRF_DISS**: Profile for Device Information Service. If defined, Device Information Service is supported.

- In **da1458x_config_advanced.h** there are the following defines:
  - **CFG_NVDS_TAG_BD_ADDRESS**: Changes the BD address of the DSPS peripheral device. This BD address is used only if no other BD address is defined in OTP.
  - **CFG_LOG_HEAP_USAGE** and **CFG_BLE_FLOW_CONTROL**, both definitions are used for the DA14531 chipset and have to be undefined for the DA14585 chipset. As detailed in Section, the BLE FLOW Control for DA14531 is based on the current heap usage. Undefining these definitions, there is the risk that during data exchange the available heap...
will exhaust and it is possible that the system hangs. It is not recommended to undefine these definitions.

- **CFG_BLE_METRICS**: If defined, the user gets statistics about the BLE traffic.
- **MSG_HEAP_SZ**: Change the heap size that is available for messages. Needs to be defined. The maximum sizes are used for the DA14531 and the DA14585 by default. It is not recommended to make changes to this parameter, as it may have serious impact on the overall DSPS functionality.
- **CFG_CONFIG_STORAGE**: Profile for Configuration storage. If defined, parameters from a valid configuration image in the flash will be applied and, also, the configuration parameters can be updated, if Remote Configuration Service is enabled.
- **CFG_EXTERNAL_WAKEUP**: Enables the functionality of wakeup from the external processor that is running the host application. This is done with the use of a gpio_ext_wakeup GPIO. The selected GPIO can be included in the Configuration Image of the flash and can be configured by the Remote Configuration Service.
- **CFG_WAKEUP_EXT_PROCESSOR**: Enables the functionality of waking up the external processor that runs the host application. This is done with the use of a gpio_wakeup_ext_host GPIO. The selected GPIO can be included in the Configuration Image of the flash and can be configured by the Remote Configuration Service.

- In `da1458x_config_basic.h` there are the following defines:
  - **CFG_DEVELOPMENT_DEBUG**: If enabled, there is more debug information available and in combination with definition of CFG_PRINTF you can have print messages on UART2 console.
  - **CFG_PRINTF**: Enables logs that will be printed in the UART2 console regarding connect, disconnect and metrics.
  - **USE_DLE**: If it is undefined the device sends and receives packets without DLE support. By default it is enabled, for higher throughput performance.
  - **CFG_MAX_CONNECTIONS**: The maximum concurrent connections supported by the application. Needs to be defined to 1, to optimize the memory utilization.

**Note**: In order to enable some of the above-mentioned definitions for the DA14531 you need to disable first one of the profiles defined in `user_profiles_config.h`, due to the limited memory area. For the DA14585 you can just enable them without any problem.

### 6.2 To Build the DSPS Host Application

1. Extract the DSPS application zip file.
2. Use Keil uVision to open `projects\target_apps\dsps\dsps_host\Keil_5\dsps_host.uvprojx`.
3. Select the desired target:
   - Choose dsps_531 for the DA14531 chipset.
   - Or
   - Choose dsps_585 for the DA14585 chipset.
4. Before the host app is built, you can configure the desired peer to which the host wants to connect to in the following way: In file `user_sps_host_config.c` go to the `user_central_config` struct and fill in the member `peer_addr_0` the BD address of the desired DSPS device. (i.e. `peer_addr_0 = {0xCD, 0x00, 0x00, 0xCA, 0xEA, 0x80}`). The default BD address in the DSPS device project is set in `da1458x_config_advanced.h` with the definition `CFG_NVDS_TAG_BD_ADDRESS`.
5. To use the default compilation options just press F7 to build the project. To change any compilation switches, please follow the instructions described in section 6.2.1.
6. After a successful build, file `dsps_host_531.hex` or `dsps_host_585.hex` is created in folder:
   - `\target_apps\dsps\dsps_host\Keil_5\out_531\`
   - or
   - `\target_apps\dsps\dsps_host\Keil_5\out_585\`

for DA14531 and DA14585 respectively.
7. Configure the debugger:
   a. Press ‘Alt + F7’.
   b. In the **Debug** tab, click the **Settings** button.
   c. Select the appropriate **SN** and then in the **Port** field select ‘SW’.
   d. Click **Apply** to save.
   e. Click **OK** to exit.

8. Start the debugging session:
   a. Press ‘Ctrl + F5’.
   b. Press ‘F5’ to run the program.

Now the host starts to scan and when the desired peer is in Connectable state it will connect to it automatically. Without setting a desired peer in Step 5, the host will connect to any available DSPS device that is in Connectable state.

9. Press again ‘Ctrl + F5’ to stop the debugger session.

### 6.2.1 Compilation Switches for DSPS Host Device Application

Before the DSPS host application is compiled, the user has the ability to modify the following configuration options:

- **user_profiles_config.h**, there is one profile defined by default:
  - **CFG_PRF_SPSC**: Profile of SPS client, needs to be defined.

- **da1458x_config_advanced.h** there are the following defines:
  - **CFG_NVDS_TAG_BD_ADDRESS**: Changes the BD address of the DSPS central device. This BD address is used only if no other BD address is defined in OTP.
  - **CFG_LOG_HEAP_USAGE** and **CFG_BLE_FLOW_CONTROL** are definitions both used for the DA14531 chipset and have to be undefined for the DA14585 chipset. This is described in section 4.2.3. The BLE FLOW Control for the DA14531 is based on the current heap usage. If you undefine these definitions, there is the risk that during data exchange the available heap will exhaust and the possibility that the system will hang. It is not recommended to undefine these definitions.
  - **CFG_BLE_METRICS**: If defined, the user gets statistics about the BLE traffic.
  - **MSG_HEAP_SZ**: Changes the heap size that is available for messages, needs to be defined. The maximum sizes are used by default for the DA14531 and the DA14585. It is not recommended to make changes to this parameter, as it may have serious impact on the overall DSPS functionality.
  - **CFG_CONFIG_STORAGE**: Profile for Configuration storage. If defined, the parameters from a valid configuration image in the flash will be applied.
  - **CFG_EXTERNAL_WAKEUP**: Enables the functionality of wakeup from the external processor that is running the host application. This is done with the usage of a gpio_ext_wakeup GPIO. The selected GPIO can be included in the Configuration Image of the flash and configured with this define.
  - **CFG_WAKEUP_EXT_PROCESSOR**: Enables the functionality to wake up the external processor that is running the host application. This is done with the use of a gpio_wakeup_ext_host GPIO. The selected GPIO can be included in the Configuration Image of the flash and configured with this define.

- **da1458x_config_basic.h** there are the following defines:
  - **CFG_DEVELOPMENT_DEBUG**: If enabled, there is more debug information available and in combination with the definition of **CFG_PRINTF** you can get print messages on the UART2 console.
  - **CFG_PRINTF**: Enables logs about connect, disconnect and metrics information that will be printed in the UART2 console.
  - **USE_DLE**: If undefined, the device sends and receives packets without DLE support. By default this define is enabled, for higher throughput performance.
7 Images and Binaries

This section describes how to create an image (single, multi or configuration) and the final binary file that is written in SPI flash, from the .hex file derived after building the device (Section 6.1) or the host applications (Section 6.2).

Application mkimage.exe is a CLI python application to build a single image that is SUOTA compatible, add a configuration struct to the image and to create a final multi image that will be written to SPI flash.

Configurator is a graphic interface that, among others, gives the user the ability to create configuration struct images. This is described in the following section.

But before starting the procedure of creating any image, the generated .hex files must be converted to binary files with the hex2bin.exe application. For example, hex2bin.exe dsps_device_531.hex.

7.1 To Build SUOTA Ready SW Images

Syntax: mkimage.exe single in_file version_file out_file

Prepend the header to the raw binary image 'in_file' (e.g. .bin file). The header contains version, timestamp and housekeeping information, which is partially extracted from header file 'version_file'.

Arguments:

- 'in_file' is the raw binary to be inserted into the image
- 'version_file' is normally stored in a header file and is available in dsps_585\projects\target_apps\dsps\sw_version. Depending on the DSPS role, 'version_file' can be either dsps_device_sw_version.h for device or dsps_host_sw_version.h for host. The mkimage application looks in these files for definitions such as:
  - #define SW_VERSION        "version"
  - #define SW_VERSION_DATE  "date"
- 'out_file' is the resulting image (e.g. .img file)

7.2 Create and Edit Configuration File

Configurator is a graphical user interface application written in python, to create configuration structure files in binary format. The user is able to:

- Create a new configuration file
- Open an existing configuration file and read the included elements and their values
- Modify the configuration file by adding/removing elements or change the value of included elements
- Save changes in the same file or in a new file
At startup, the application will open and read XML file config-spec.xml, which is in the same file location as the application. The xml file is parsed with which the list of Groups, Element IDs and types are inserted into the application. The XML file has all the specified groups and element IDs with their corresponding fields:

- **names**: the name of the element
- **types**: the type of the element as defined in the DSPS device or the host application
- **units**: the measurement unit of each element
- **max_sizes**: the maximum size of each element
- **min**: the minimum value
- **max**: the maximum value

This is an example of an element and its corresponding fields:

```xml
<group id="0x01" name="ADVERTISE">
  <element id="0x00">
    <name>Advertise Interval Minimum</name>
    <type>uint16_t</type>
    <units>BLE slots</units>
    <max_size>2</max_size>
    <min>32</min>
    <max>16384</max>
  </element>
</group>
```

The number of Groups and Elements is transparent to the Configurator application. New Groups or Elements can be added to the XML file if the format described above is followed. In this case, the new item should be added in the DSPS Configuration Struct (Section 4.6) and code changes should be made to handle the new item in the XML file.

In the left-hand side frame of the application (see Figure 12), the user can select the new Group ID and the Element ID that is to be included in the configuration file. Click the **Add** button to add the element in the Configuration Structure shown in the right-hand side frame of the application. The configurator checks and validates the applied values based on the types and the max sizes of the element’s value. The user can save the changes either every time an element is added, or after completing all changes and then select the **Save** or **Save as** option that is under the **File** menu. When the structure is saved in the file (binary format), all the header’s fields aside from version and encryption flag (provided by the user) will be calculated and the header element will precede the configuration elements.
7.3 Create a Multi-Image for First Time Programming

Syntax: `mkimage.exe multi in_img off1 off2 in_cfg off3 off4 off5 out_file [bloader]`

To create a multi-part image 'out_file', the application image and the configuration image are packed together into one image that can be written to SPI flash.

Arguments:
- 'in_img' is the output from a single procedure, described in Section 7.1
- 'off1' is the offset for the application image
- 'off2' is the offset for the alternative application image
- 'in_cfg' is the file with the configuration image to be inserted into the output image, which is generated by the Configurator application (Section 7.2)
- 'off3' is the offset for the configuration image
- 'off4' is the offset for the alternative configuration image (filled with the same in_cfg)
- 'off5' is offset for the product header
- 'bloader' (.bin file) at offset 0, if the secondary bootloader image is provided. The bloader bin file must always be in offset 0x0

The offset values are available in either Section 7.5.1 or Section 7.5.2 depending to selected target in the build procedure.

7.4 CLI Flash Programming

Application cli_flash_programming is a command line tool, written in python language, which can be used to access and program the SPI Flash for DA14531 devices. The tool communicates with the SPI flash either over a virtual COM port or over a JTAG interface. After the communication interface is selected, the user is able to:
- Erase the entire or part of the SPI flash
- Read all or a specific number of bytes from the SPI flash
- Write a binary or a specific number of bytes from a binary file to the SPI flash
7.4.1 Hardware setup

For DA14531 devices the only acceptable UART configuration is through single-wire UART that is connected to pin P0_5 and operates at a 115200 baudrate. In order to access the SPI Flash of the DA14531 Module, the jumpers of DK’s SPI Flash should be removed. The figure below shows the hardware setup that is needed, in both DA14531 and DA14531 Module devices, in order to access the SPI flash either with the use of UART or the JTAG interface.

![Hardware setup diagram]

Figure 13: DA14531 and DA14531 Module ProDev Kit Motherboard Jumper Configuration

7.4.2 How to Use cli_flash_programming

The user must specify the communication interface and the operation (command) to be executed at the SPI flash. All arguments and commands are described in detail below. To write an application image to the SPI flash, the whole the SPI flash memory must be erased first.

Arguments:

- uart: use the UART interface to communicate with and program the SPI flash.
  - port <PORT>: Specify the serial port to which the device is connected. If the --port argument is either empty or not found during the scan of all available Virtual COM ports, an autodetection graphical interface will display with a list of the available Virtual COM ports.
  - firmware <PATH_FIRMWARE_FILE>: Specify the file to be used as flash_programmer_531.bin instead of the default, which is located in the same folder as cli_flash_programming.exe.
  - spi_cfg <CS_PORT,CS_PIN,CLK_PORT,CLK_PIN,DO_PORT,DO_PIN,DI_PORT,DI_PIN>: Define the GPIO pin assignment for the SPI flash. The configuration is a comma separated list of four pairs of numbers. Each pair describes the port and pin for CS, CLK, DO and DI. If the --spi_cfg argument is not selected, then the default SPI Flash pins are used: --spi_cfg 0,1,0,4,0,0,0,3.
  - port_selector <PORT_SELECTOR>: Define the GPIO pin assignment for UART. The cli_flash_programming uses by default selector 5 through pin P0_5 at baudrate speed 115200 for DA14531. This is the only option for DA14531.
● jlink: use the JTAG interface over JLink to communicate with and to program the SPI flash.
  ○ serialno <SERIAL NUMBER>: Specify the serial number of the board to be connected.

● verbose: increases the logging verbosity.
  ○ If this option is not added, then the default logging level is set to Warning
  ○ v: increases the logging level to Info
  ○ vv: sets the logging level to its maximum value for Debugging reasons.

● --help: prints a complete help message for all the options in the current parser and then exits.
  ○ cli_flash_programming.exe -h: prints the help messages for all the top level arguments and options.
  ○ cli_flash_programming.exe erase -h: prints the related help messages of the required parameters of the erase command. The above command can be used respectively for read and write commands.

● device <DEVICE NAME>: Specify the device name. In our case DA14531 is acceptable.

● bootable: Automatically add an AN-B-001 header at offset 0 of the SPI flash memory. The secondary bootloader will copy only the number of bytes defined in the SPI Flash header.

Commands:

● erase: erase the whole or a sector of the SPI flash.
  ○ start: the start address to be erased. Default is 0.
  ○ size: the size of the area to be erased. Default is -1. If the start address is not aligned to a page, then the corresponding sector will be erased.

● read: read the entire or a specified size from the SPI flash.
  ○ start: the start address to be read. Default is 0.
  ○ size: the size of the area to be read. Default is 1.
  ○ bin: optional output file where to store read data.

● write: all contents of a binary file or specified size to the SPI flash.
  ○ start: the start address to be written.
  ○ size: the size of the area to be written. If the size is equal to 0, write the entire file to the SPI flash.
  ○ bin: the input file that will be written to the SPI flash.

7.5 Release Binaries

In the release folder, there is a bin/ folder, which contains all the related binary and image files. The content of the folder is:

● The binary files after building the DSPS project for both chips (DA14531 and DA14585) and both roles (Device and Host):
  ○ dsps_device_531.bin
  ○ dsps_device_585.bin
  ○ dsps_host_531.bin
  ○ dsps_host_585.bin

● The single images (see 7.1) of the DSPS application for both chips (DA14531 and DA14585) and both roles (Device and Host):
  ○ dsps_device_531.img
  ○ dsps_device_585.img
  ○ dsps_host_531.img
  ○ dsps_host_585.img
● The multi images (see 7.3) of the DSPS application for both chips (DA14531 and DA14585) and both roles (Device and Host):
  ○ dsps_device_531_multi.bin
  ○ dsps_device_585_multi.bin
  ○ dsps_host_531_multi.bin
  ○ dsps_host_585_multi.bin

● The configuration structure binary files for both chips (DA14531 and DA14585) and both roles (Device and Host). In the Host role the configuration structure is the same for both chips:
  ○ spss_device_config_struct_531.img
  ○ spss_device_config_struct_585.img
  ○ spss_host_config_struct_531_585.img

● The secondary bootloader binary files for both chips (DA14531 and DA14585), which is used to create the multi images (see 7.3):
  ○ secondary_bootloader_531.bin
  ○ secondary_bootloader_585.bin

● The application executables used to create the images (i) and the configuration structures (ii):
  ○ mkimage.exe
  ○ configurator.exe

● The XML file, used by the Configurator and the SmartConfig application, which contains all the supported configuration elements, their properties and the necessary enumerations and data types:
  ○ config-spec.xml

● The cli_flash_programming executable (see 7.4), which programs the SPI flash over either UART or JTAG:
  ○ i. cli_flash_programming.exe

● The binary of the flash_programmer application used to upload and read back the application code that runs on platform DA14531:
  ○ i. flash_programmer_531.bin

The multi-image follows the memory map of the flash of each chip. The secondary bootloader must be stored in the beginning of the image, in offset 0x0. The Product Header is produced during the multi-image creation and contains the offsets of the Application Images and the Configuration Structure images. The offset of the Product Header is defined during the build procedure of the multi-image and is hardcoded in the code of the DSPS host/device and secondary bootloader projects. So, if someone wants to change the Product Header offset, he must also change it in the code.

The Flash memory maps and the Product Headers of the multi images of both chips are shown in the following sectors.

### 7.5.1 DA14585 Flash Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>Secondary Bootloader</td>
</tr>
<tr>
<td>0x06000</td>
<td>DSPS Image 1</td>
</tr>
<tr>
<td>0x1A000</td>
<td>DSPS Image 2</td>
</tr>
<tr>
<td>0x2E000</td>
<td>DSPS Configuration Image 1</td>
</tr>
<tr>
<td>0x2F000</td>
<td>DSPS Configuration Image 2</td>
</tr>
<tr>
<td>0x39000</td>
<td>Product Header</td>
</tr>
</tbody>
</table>
Table 8: DA14585 Product Header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size (in bytes)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signature</td>
<td>2</td>
<td>0x7052</td>
</tr>
<tr>
<td>2</td>
<td>Version</td>
<td>2</td>
<td>&lt;version&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Image bank1 offset</td>
<td>4</td>
<td>0x06000</td>
</tr>
<tr>
<td>8</td>
<td>Image bank1 offset</td>
<td>4</td>
<td>0x1A000</td>
</tr>
<tr>
<td>12</td>
<td>Configuration bank1 offset</td>
<td>4</td>
<td>0x2E000</td>
</tr>
<tr>
<td>16</td>
<td>Configuration bank1 offset</td>
<td>4</td>
<td>0x2F000</td>
</tr>
</tbody>
</table>

7.5.2 DA14531 Flash Memory Map

Table 9: DA14531 Flash Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>Secondary Bootloader</td>
</tr>
<tr>
<td>0x06000</td>
<td>DSPS Image 1</td>
</tr>
<tr>
<td>0x10000</td>
<td>DSPS Image 2</td>
</tr>
<tr>
<td>0x1A000</td>
<td>DSPS Configuration Image 1</td>
</tr>
<tr>
<td>0x1B000</td>
<td>DSPS Configuration Image 2</td>
</tr>
<tr>
<td>0x1F000</td>
<td>Product Header</td>
</tr>
</tbody>
</table>

Table 10: DA14531 Product Header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size (in bytes)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Signature</td>
<td>2</td>
<td>0x7052</td>
</tr>
<tr>
<td>2</td>
<td>Version</td>
<td>2</td>
<td>&lt;version&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Image bank1 offset</td>
<td>4</td>
<td>0x06000</td>
</tr>
<tr>
<td>8</td>
<td>Image bank1 offset</td>
<td>4</td>
<td>0x10000</td>
</tr>
<tr>
<td>12</td>
<td>Configuration bank1 offset</td>
<td>4</td>
<td>0x1B000</td>
</tr>
<tr>
<td>16</td>
<td>Configuration bank1 offset</td>
<td>4</td>
<td>0x1F000</td>
</tr>
</tbody>
</table>

8 SmartConsole Android and iOS Application

8.1 Overview

The Dialog Serial Port Service (DSPS) reference application comes with an SPS demo application for Android and iOS systems. The SmartConsole application can discover, connect and exchange data with SPS-enabled devices within the Bluetooth RF range of the mobile device.

Features

- Device discovery
- Connection to a discovered device
- Receive data (ASCII or HEX) from a peer device with flow control
- Transmit data to a peer device, either once or repeatedly
- Transmit file(s) to a peer device with flow control
Figure 14: SmartConsole Icon on Android and iOS Device

8.2 Installation

8.2.1 Android Application

The Android application is available in the Google Play Store and is easily installed from there just like any other Android application. To find the application, search for ‘Dialog SmartConsole’ (Figure 15). Then open the application's description and tap the Install button. During the installation the permission to access the device resources is requested as shown in Figure 16. Please grant this permission. After the installation is complete, the application can be opened either with the Open button of the ‘Play store’ or from the Android application drawer.
8.2.2 iOS Application

The iOS application is available in the Apple App Store and is easily installed from there like any other iOS application. To find the application, search for ‘SmartConsole’ or ‘Dialog SmartConsole’ in
the App Store. Next, the user has to open the application's description and then tap the cloud button to download and install the software. After the installation is complete, the application can be opened either with the Open button in the 'App store' or from the iOS desktop.

8.3 Device List
The user must tap the SmartConsole icon to start the application. The scan screen is shown (Figure 17) and the user has to tap the SCAN button. While scanning, all devices found are displayed (device name and Bluetooth address). The complete list of all discovered devices are listed on the screen when the search has finished. To refresh the list of devices found, the user can tap the SCAN button again. The user must tap on the name of the desired device to connect to that device.

![Figure 17: Scan Device Screen](image)

8.4 Functionality Tabs
When connected to a device, a side menu environment is shown to the user when the List Menu button is tapped, which provides the functionality of the application. After connection, the user may navigate to three operating mode screens, an Information screen and a Disclaimer screen that will be explained in the following paragraphs.
Figure 18: Select Operation Mode

Figure 19: Application Main Screens
8.4.1 Console Mode

The console mode is a serial port console emulation mode. Any character that is entered on the keyboard is immediately sent to the peer device and any character received is displayed in the reception display box.

8.4.2 Read/Transfer Mode

The Read/Transfer mode allows the user to send data to and receive data from the peer device. Initially, the reception (RX) display box is shown, where received data can be displayed in ASCII or HEX format. Also, the user can clear the received data, enable or disable the incoming flow by tapping the START or STOP button respectively.

For the Transfer (TX) part, a text box is shown where data can be entered via the keyboard or pasted by keeping the field pressed and select ‘paste’ when the context menu appears. Tap the SEND button to send the displayed data once or repeatedly (depending on the state of the ‘cyclic sending’ toggle switch) with the chosen interval.

8.4.3 Data File Streaming Mode

The user also has the ability to send a file instead of individual strings of characters. This function can be accessed in the ‘Data File Streaming’ screen, where the user can browse the device’s file system to find the file to be transmitted.

8.5 Information and Disclaimer Screens

The ‘Information’ screen displays contact information and the version of the application. See Figure 20. The Disclaimer screen contains the disclaimer notes.

Figure 20: Information & Disclaimer Screens
9  SmartConfig

The SmartConfig Android application can discover and connect to DSPS devices. It serves to discover, read and modify the configuration of the device.

![Figure 21: SmartConfig Icon – Android – iOS](image)

9.1  Installation

9.1.1  Android Application

The SmartConfig application for Android is available in the Google ‘Play Store‘ and is easily installed from there just like any other Android application. The user can search for ‘Dialog SmartConfig’ (Figure 22) to find the application. Next, open the application's description and tap the Install button. During the installation the permission to access the device resources and location is requested as shown in Figure 23. Please grant this permission. After installation is complete, the application can be opened either with the Open button in the ‘Play store‘ or from the Android application drawer.
9.1.2 iOS application

The iOS application is available in the Apple ‘App Store’ and is easily installed from there just like any other iOS application. Search for ‘SmartConsole’ or ‘Dialog SmartConsole’ to find the application.
Next, open the application’s description and tap the cloud button (Figure 24) to download and install the application. After installation is complete, the application can be opened either with the Open button in the ‘App store’ or from the iOS desktop.

![SmartConfig in App Store](image)

**Figure 24: SmartConfig in App Store**

### 9.2 Device List

When the application is started the scan screen is displayed and the DSPS devices that are within range appear. The device name, BD Address, RSSI and the advertised services are displayed in the list of discovered devices. The user can connect to any of them by tapping on that item in the list.

To do a rescan, simply tap the SCAN button on the top of the screen.
9.3 Functionality

When the SmartConfig application connects to a DSPS device, the discovery of the configuration parameters (elements) is automatically started, followed by successive reads of the current configuration values of the parameters. The elements and the corresponding values are displayed on the screen as shown in Figure 26. To change the value of any parameter, simply tap on the parameter. Next, a list of all valid options or an edit textbox to enter a new value is displayed. The new value will be applied locally but all changed values will be sent to the remote control service and applied in the DSPS device when the APPLY button is tapped. The user can undo the changes before they are applied to the DSPS device or refresh the values with the device's values by tapping any of the corresponding buttons on the top of the parameters list.

After a successful update of the parameters, the Apply and Undo buttons are disabled. In case an Apply fails, a "Write failed" message is shown on screen and the parameter change is NOT stored in the device's flash.
The SmartConfig application uses a built-in XML file that describes all the supported configuration elements, their parameters (Group ID, Elemt ID, Name, Type, Units, Maximum size, Minimum and maximum values) and the definitions of enumerations and custom types used. If the application finds an XML file with the name config-spec.xml, it uses this to replace the built-in XML file. Since the Configurator application also uses the same XML file (see 7.2), this is a way to keep both applications synchronized. The Android SmartConfig application looks in the folder "Dialog Semiconductor/SmartConfig", while the iOS one looks into the application’s Documents. Moreover, if SmartConfig finds another XML file named user-config-spec.xml, it merges this with the XML file that it finally uses (either the built-in or the config-spec.xml). This way, the user can add extra elements or different element properties, without having to alter the common config-spec.xml file.

9.4 Information and Disclaimer Screens

The 'Information' screen displays contact information and the version of the application. See Figure 27. The Disclaimer screen contains the disclaimer notes.
10 Hardware Setup

As mentioned before, DSPS emulates serial cable communication. Data is transferred over UART to BLE and over BLE to UART. For DSPS to work properly, the signals UCTS, URTS,URX and UTX of mainboard 376-18-B should connect in a proper way for each DK and module. This section describes the correct way for each chipset to be able to transfer or receive data with the use of DSPS.

The configuration of the UART pins described below are located in file user_periph_setup.h.

10.1 DA14585 DK

For DSPS functionality there are three jumpers needed on the J1 connector of the mainboard 376-18-B at the following pins:

- J1:11-J1:12
- J1:13-J1:14
- J1:17-J1:18

And one jumper cable is needed from J1:15 to J2:7 (P02).

Remove the jumper from J1:19-J1:20 in case it exists.

Figure 28 shows the proper jumpers setup of the J1 header.
Figure 28: J1 Jumpers Setup for DA14585 DK

10.2 DA14531 DK

For the DSPS functionality to work with the DA14531 DK, four cables are needed that go from the J1 connector to the J2 connector at the following pins:

- **UCTS**: J1:11 - J2:29 (pink cable in Figure 29)
- **URTS**: J1:13 - J2:28 (yellow cable in Figure 29)
- **URX**: J1:15 - J2:26 (red cable in Figure 29)
- **UTX**: J1:17 - J2:27 (blue cable in Figure 29)

Figure 29 shows where the cables should be for a proper functional DSPS.
10.3 DA14531 Module

For the DSPS functionality, four jumpers are needed on the J1 connector of the mainboard 376-18-B at the following pins:

- J1:11-J1:12
- J1:13-J1:14
- J1:15-J1:16
- J1:17-J1:18

Remove the jumper from J1:19-J1:20 in case it exists.

Figure 30 shows the proper jumpers setup of the J1 header.
11 SPS Performance

To measure SPS performance, throughput tests have been done for the DA14531 and the DA14585 chipsets. The UART baudrate was set to 921600 bits/s in all tests. The connection interval is 25 msec and MTU negotiated to 247, but for iPhone tests MTU=185 bytes).

For the test the following hosts were used: DA14531, DA14585, iPhone XR (iOS 13.1.2) and Samsung S9 (Android 8.0.0-payload size=251 bytes).

The SPS performance analysis gives the following results (Table 11 and Table 12) regarding the service's throughput.

Table 11: Performance Results for the DA14531 Device

<table>
<thead>
<tr>
<th>Device</th>
<th>Host</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DA14531</td>
<td>DA14531</td>
<td>iPhone 7 Plus</td>
<td>Samsung S9</td>
</tr>
<tr>
<td>Tx to host</td>
<td>493.27 kbps</td>
<td>185.44 kbps</td>
<td>470.46 kbps</td>
</tr>
<tr>
<td></td>
<td>61.65 kBytes/s</td>
<td>23.18 kBytes/s</td>
<td>58.80 kBytes/s</td>
</tr>
<tr>
<td>Rx from host</td>
<td>551.84 kbps</td>
<td>94.3 kbps</td>
<td>152.35 kbps</td>
</tr>
<tr>
<td></td>
<td>68.98 kBytes/s</td>
<td>11.8 kBytes/s</td>
<td>19.04 kBytes/s</td>
</tr>
</tbody>
</table>
Table 12: Performance Results for the DA14585 Device

<table>
<thead>
<tr>
<th>Device</th>
<th>Hosts</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DA14585</td>
<td>DA14585</td>
<td>iPhone Plus</td>
<td>Samsung S9</td>
</tr>
<tr>
<td>Tx to host</td>
<td>516.28 kbps</td>
<td>187.6 kbps</td>
<td>513.11 kbps</td>
</tr>
<tr>
<td></td>
<td>64.53 kBytes/s</td>
<td>23.45 kBytes/s</td>
<td>64.13 kBytes/s</td>
</tr>
<tr>
<td>Rx from host</td>
<td>567.55 kbps</td>
<td>94.3 kbps</td>
<td>152.14 kbps</td>
</tr>
<tr>
<td></td>
<td>70.94 kBytes/s</td>
<td>11.8 kBytes/s</td>
<td>19.01 kBytes/s</td>
</tr>
</tbody>
</table>
Appendix A Configuration Parameters

All the configuration parameters that are supported until now are shown in the next table:

**Table 13: Configuration Parameters**

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_INTERVAL_MIN</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_INTERVAL_MAX</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_CHANNEL_MAP</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_MODE</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_FILTER_POLICY</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_PEER_ADDR</td>
</tr>
<tr>
<td>0x0100</td>
<td>ELEM_ID_ADV_PEER_ADDR_TYPE</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_ROLE</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_MTU</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_ADDR_TYPE</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_ADDRESS</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_REGEN_ADDR_DUR</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_DEVICE_IRK</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_ATTR_CONFIG</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_SERVICE_START_HANDLE</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_SERVICE_START_HANDLE</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_MAX_MPS</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_MAX_TX_OCTETS</td>
</tr>
<tr>
<td>0x0200</td>
<td>ELEM_ID_GAP_MAX_TX_TIME</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_INTERVAL_MIN</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_INTERVAL_MAX</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_LATENCY</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_TIME_OUT</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_CE_LEN_MIN</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_CONN_CE_LEN_MAX</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_ADV_DATA</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_ADV_DATA_LEN</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_SCAN_RESP_DATA</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_SCAN_RESP_DATA_LEN</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_DEVICE_NAME</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_USER_DEVICE_NAME_LEN</td>
</tr>
<tr>
<td>0x0500</td>
<td>ELEM_ID_PERIPH_UART_BAUDRATE</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_CODE</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_ADDR_SRC</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_SCAN_INTERVAL</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_SCAN_WINDOW</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_PEER_ADDR_0</td>
</tr>
<tr>
<td>0x0600</td>
<td>ELEM_ID_CENTRAL_PEER_ADDR_0_TYPE</td>
</tr>
<tr>
<td>0x0700</td>
<td>ELEM_ID_GPIO_UART1_RX</td>
</tr>
<tr>
<td>0x0701</td>
<td>ELEM_ID_GPIO_UART1_TX</td>
</tr>
<tr>
<td>0x0702</td>
<td>ELEM_ID_GPIO_UART1_CTS</td>
</tr>
<tr>
<td>0x0703</td>
<td>ELEM_ID_GPIO_UART1_RTS</td>
</tr>
<tr>
<td>0x0704</td>
<td>ELEM_ID_GPIO_EXT_WAKEUP</td>
</tr>
<tr>
<td>0x0705</td>
<td>ELEM_ID_GPIO_EXT_WAKEUP_POLARITY</td>
</tr>
<tr>
<td>0x0706</td>
<td>ELEM_ID_GPIO_EXT_WAKEUP_DEBOUNCE</td>
</tr>
<tr>
<td>0x0707</td>
<td>ELEM_ID_GPIO_ACTIVE_STATUS_PIN</td>
</tr>
<tr>
<td>0x0708</td>
<td>ELEM_ID_GPIO_WAKEUP_EXT_HOST</td>
</tr>
<tr>
<td>0x0709</td>
<td>ELEM_ID_GPIO_POR_PIN</td>
</tr>
<tr>
<td>0x070A</td>
<td>ELEM_ID_GPIO_POR_PIN_POLARITY</td>
</tr>
<tr>
<td>0x070B</td>
<td>ELEM_ID_GPIO_POR_PIN_TIMEOUT</td>
</tr>
<tr>
<td>0x070C</td>
<td>ELEM_ID_GPIO_UART2_RX</td>
</tr>
<tr>
<td>0x070D</td>
<td>ELEM_ID_GPIO_UART2_TX</td>
</tr>
</tbody>
</table>
Appendix B DSPS Configuration Parameters

The DSPS application supports both DA14531 and DA14585 chips, and both Device and Host roles. In the Host role, both chips use the same configuration structure. So, the DSPS application supports the following three configuration structures.

B.1 DSPS Device for DA14531

Table 14: DSPS Device Configuration Structure for DA14531

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Name</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0100</td>
<td>ELEM_ID_PARAM_ADV_INTERVAL_MIN</td>
<td>160 ble slots (100 ms)</td>
</tr>
<tr>
<td>0x0101</td>
<td>ELEM_ID_PARAM_ADV_INTERVAL_MAX</td>
<td>160 ble slots (100 ms)</td>
</tr>
<tr>
<td>0x0102</td>
<td>ELEM_ID_PARAM_ADV_CHANNEL_MAP</td>
<td>7 (ADV_ALL_CHNLS_EN)</td>
</tr>
<tr>
<td>0x0201</td>
<td>ELEM_ID_PARAM_GAP_MTU</td>
<td>247 bytes</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_PARAM_CONN_INTERVAL_MIN</td>
<td>20 ble doubleslots (25 ms)</td>
</tr>
<tr>
<td>0x0301</td>
<td>ELEM_ID_PARAM_CONN_INTERVAL_MAX</td>
<td>20 ble doubleslots (25 ms)</td>
</tr>
<tr>
<td>0x0302</td>
<td>ELEM_ID_PARAM_CONN_LATENCY</td>
<td>0 ms</td>
</tr>
<tr>
<td>0x0303</td>
<td>ELEM_ID_PARAM_CONN_TIME_OUT</td>
<td>1000 timerunits (10000 ms)</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_PARAM_USER_ADV_DATA</td>
<td>x11</td>
</tr>
<tr>
<td>0x0401</td>
<td>ELEM_ID_PARAM_USER_ADV_DATA_LEN</td>
<td>22 bytes</td>
</tr>
<tr>
<td>0x0402</td>
<td>ELEM_ID_PARAM_SCAN_RESP_DATA</td>
<td>Null</td>
</tr>
<tr>
<td>0x0403</td>
<td>ELEM_ID_PARAM_SCAN_RESP_DATA_LEN</td>
<td>0 bytes</td>
</tr>
<tr>
<td>0x0404</td>
<td>ELEM_ID_PARAM_USER_DEVICE_NAME</td>
<td>SPS-531</td>
</tr>
<tr>
<td>0x0405</td>
<td>ELEM_ID_PARAM_DEVIC_NAME_LEN</td>
<td>7</td>
</tr>
<tr>
<td>0x0500</td>
<td>ELEM_ID_PARAM_PERIPH_UART_BAUDRATE</td>
<td>8 (921600 Kbps)</td>
</tr>
<tr>
<td>0x0700</td>
<td>ELEM_ID_GPIO_UART1_RX</td>
<td>GPIO 5, PORT 0</td>
</tr>
<tr>
<td>0x0701</td>
<td>ELEM_ID_GPIO_UART1_TX</td>
<td>GPIO 6, PORT 0</td>
</tr>
<tr>
<td>0x0702</td>
<td>ELEM_ID_GPIO_UART1_CTS</td>
<td>GPIO 8, PORT 0</td>
</tr>
<tr>
<td>0x0703</td>
<td>ELEM_ID_GPIO_UART1_RTS</td>
<td>GPIO 7, PORT 0</td>
</tr>
<tr>
<td>0x0707</td>
<td>ELEM_ID_GPIO_ACTIVE_STATUS_PIN</td>
<td>GPIO 11, PORT 0</td>
</tr>
<tr>
<td>0x0709</td>
<td>ELEM_ID_GPIO_PER_PIN</td>
<td>GPIO 9, PORT 0</td>
</tr>
<tr>
<td>0x070A</td>
<td>ELEM_ID_GPIO_PER_PIN_POLARITY</td>
<td>1 (High)</td>
</tr>
<tr>
<td>0x070B</td>
<td>ELEM_ID_GPIO_PER_PIN_TIMEOUT</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

B.2 DSPS Device for DA14585

Table 15: DSPS Device Configuration Structure for DA14585

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Name</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0100</td>
<td>ELEM_ID_PARAM_ADV_INTERVAL_MIN</td>
<td>160 ble slots (100 ms)</td>
</tr>
<tr>
<td>0x0101</td>
<td>ELEM_ID_PARAM_ADV_INTERVAL_MAX</td>
<td>160 ble slots (100 ms)</td>
</tr>
<tr>
<td>0x0102</td>
<td>ELEM_ID_PARAM_ADV_CHANNEL_MAP</td>
<td>7 (ADV_ALL_CHNLS_EN)</td>
</tr>
</tbody>
</table>
Table 16: DSPS Host Configuration Structure for DA14531 and DA14585

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Name</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0201</td>
<td>ELEM_ID_PARAM_GAP_MTU</td>
<td>247 bytes</td>
</tr>
<tr>
<td>0x0300</td>
<td>ELEM_ID_PARAM_CONN_INTERVAL_MIN</td>
<td>20 ble doubleslots (25 ms)</td>
</tr>
<tr>
<td>0x0301</td>
<td>ELEM_ID_PARAM_CONN_INTERVAL_MAX</td>
<td>20 ble doubleslots (25 ms)</td>
</tr>
<tr>
<td>0x0302</td>
<td>ELEM_ID_PARAM_CONN_LATENCY</td>
<td>0 ms</td>
</tr>
<tr>
<td>0x0303</td>
<td>ELEM_ID_PARAM_CONN_TIME_OUT</td>
<td>1000 timerunits (10000 ms)</td>
</tr>
<tr>
<td>0x0400</td>
<td>ELEM_ID_PARAM_USER_ADV_DATA</td>
<td>\x11\x07\x87\x5c\x49\xd2\x04\x03\x71\xa0\xb5\x35\x85\x3e\xb0\x83\x07\x03\x03\x5c\x49</td>
</tr>
<tr>
<td>0x0401</td>
<td>ELEM_ID_PARAM_USER_ADV_DATA_LEN</td>
<td>22 bytes</td>
</tr>
<tr>
<td>0x0402</td>
<td>ELEM_ID_PARAM_ADV_SCAN_RESP_DATA</td>
<td>Null</td>
</tr>
<tr>
<td>0x0403</td>
<td>ELEM_ID_PARAM_ADV_SCAN_RESP_DATA_LEN</td>
<td>0 bytes</td>
</tr>
<tr>
<td>0x0404</td>
<td>ELEM_ID_PARAM_USER_DEVICE_NAME</td>
<td>SPS-585</td>
</tr>
<tr>
<td>0x0405</td>
<td>ELEM_ID_PARAM_DEVICENAME_LEN</td>
<td>7</td>
</tr>
<tr>
<td>0x0500</td>
<td>ELEM_ID_PARAM_PERIPH_UART_BAUDRATE</td>
<td>8 (921600 Kbps)</td>
</tr>
<tr>
<td>0x0700</td>
<td>ELEM_ID_GPIO_UART1_TX</td>
<td>GPIO 4, PORT 0</td>
</tr>
<tr>
<td>0x0701</td>
<td>ELEM_ID_GPIO_UART1_RX</td>
<td>GPIO 5, PORT 0</td>
</tr>
<tr>
<td>0x0702</td>
<td>ELEM_ID_GPIO_UART1_CTS</td>
<td>GPIO 7, PORT 0</td>
</tr>
<tr>
<td>0x0703</td>
<td>ELEM_ID_GPIO_UART1_RTS</td>
<td>GPIO 6, PORT 0</td>
</tr>
<tr>
<td>0x0704</td>
<td>ELEM_ID_GPIO_ACTIVE_STATUS_PIN</td>
<td>GPIO 5, PORT 0</td>
</tr>
<tr>
<td>0x0705</td>
<td>ELEM_ID_GPIO_POR_PIN</td>
<td>GPIO 9, PORT 0</td>
</tr>
<tr>
<td>0x0706</td>
<td>ELEM_ID_GPIO_POR_PIN_POLARITY</td>
<td>1 (High)</td>
</tr>
<tr>
<td>0x0707</td>
<td>ELEM_ID_GPIO_POR_PIN_TIMEOUT</td>
<td>0 ms</td>
</tr>
</tbody>
</table>

B.3 DSPS Host for DA14531 and DA14585
| 0x0605 | ELEM_ID_CENTRAL_PEER_ADDR_0_TYPE | 0 (public) |
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>23-Jun-2020</td>
<td>Update for 6.150.4.50 Official Release of DSPS reference design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ cli_flash_programmer added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ SmartConsole for iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ Minor changes</td>
</tr>
<tr>
<td>1.1</td>
<td>03-Apr-2020</td>
<td>Update for 6.150.3.45 Engineering Release of DSPS reference design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ Revise document structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ Support of DA14531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ Addition of BLE flow control mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td>○ Introduce Configuration Structure, Remote Configuration Service and corresponding tools and apps</td>
</tr>
<tr>
<td>1.0</td>
<td>24-Nov-2017</td>
<td>Initial version.</td>
</tr>
</tbody>
</table>
Status Definitions

<table>
<thead>
<tr>
<th>Status</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAFT</td>
<td>The content of this document is under review and subject to formal approval, which may result in modifications or additions.</td>
</tr>
<tr>
<td>APPROVED or unmarked</td>
<td>The content of this document has been approved for publication.</td>
</tr>
</tbody>
</table>

Disclaimer

Unless otherwise agreed in writing, the Dialog Semiconductor products (and any associated software) referred to in this document are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of a Dialog Semiconductor product (or associated software) can reasonably be expected to result in personal injury, death or severe property or environmental damage. Dialog Semiconductor and its suppliers accept no liability for inclusion and/or use of Dialog Semiconductor products (and any associated software) in such equipment or applications and therefore such inclusion and/or use is at the customer’s own risk.

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, express or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications. Notwithstanding the foregoing, for any automotive grade version of the device, Dialog Semiconductor reserves the right to change the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications, in accordance with its standard automotive change notification process.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document is subject to Dialog Semiconductor’s Standard Terms and Conditions of Sale, available on the company website (www.dialog-semiconductor.com) unless otherwise stated.

Dialog, Dialog Semiconductor and the Dialog logo are trademarks of Dialog Semiconductor Plc or its subsidiaries. All other product or service names and marks are the property of their respective owners.

© 2020 Dialog Semiconductor. All rights reserved.

RoHS Compliance

Dialog Semiconductor’s suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

<table>
<thead>
<tr>
<th>Country</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>United Kingdom (Headquarters)</td>
<td>Dialog Semiconductor (UK) LTD  Phone: +44 1793 757700</td>
</tr>
<tr>
<td>Germany</td>
<td>Dialog Semiconductor GmbH  Phone: +49 7021 805-0</td>
</tr>
<tr>
<td>The Netherlands</td>
<td>Dialog Semiconductor B.V.  Phone: +31 73 640 8822</td>
</tr>
<tr>
<td>North America</td>
<td>Dialog Semiconductor Inc.  Phone: +1 408 845 8500</td>
</tr>
<tr>
<td>Japan</td>
<td>Dialog Semiconductor K. K.  Phone: +81 3 5769 5100</td>
</tr>
<tr>
<td>Taiwan</td>
<td>Dialog Semiconductor Taiwan  Phone: +886 281 786 222</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>Dialog Semiconductor Hong Kong  Phone: +852 2607 4271</td>
</tr>
<tr>
<td>Korea</td>
<td>Dialog Semiconductor Korea  Phone: +82 2 3469 8200</td>
</tr>
<tr>
<td>China (Shenzhen)</td>
<td>Dialog Semiconductor China  Phone: +86 755 2981 3669</td>
</tr>
<tr>
<td>China (Shanghai)</td>
<td>Dialog Semiconductor China  Phone: +86 21 5424 9058</td>
</tr>
</tbody>
</table>

Email: enquiry@diasemi.com  Web site: www.dialog-semiconductor.com