User Manual

DA14585 Serial Port Service Reference Application

UM-B-088

Abstract

This document describes the architecture and the implementation details of the Serial Port Service application running on the SmartBond™ DA14585 development kit.

Contents

Abstract **1**

Contents **2**

Figures **4**

Tables **4**

1 Terms and definitions **6**

2 References **6**

3 Introduction **7**

4 Software features **8**

4.1 Bluetooth 8

4.2 UART 8

4.3 System 8

5 Software architecture **9**

5.1 Software features 9

5.1.1 GAP roles 9

5.1.2 Serial Port Service 9

5.1.3 UART driver for the interrupt driven project 10

5.1.4 Data scheduling and flow control for the interrupt driven project 11

5.1.5 UART driver for the DMA driven project 11

5.1.6 Data scheduling and flow control for the DMA driven project 12

5.2 Source files 13

5.3 Header files 14

6 Code overview and state machines **16**

6.1 Application task state machine 16

6.1.1 Peripheral state machine 16

6.1.2 Central state machine 17

6.2 Application callback functions 18

6.2.1 user_on_connection() 18

6.2.2 user_on_disconnect() 18

6.2.3 user_on_set_dev_config_complete() (Central only) 18

6.2.4 user_on_scanning_completed() (Central only) 18

6.2.5 user_on_adv_report_ind() (Central only) 18

6.2.6 user_on_connect_failed() (Central Only) 18

6.3 Main loop callback functions 19

6.3.1 user_on_init() 19

6.3.2 user_on_db_init_complete() 19

6.3.3 user_on_system_powered() 19

6.3.4 user_before_sleep() 19

6.4 Other application functions 19

6.4.1 user_scan_start() (Central only) 19

6.4.2 user_gapm_cancel() (Central only) 19

6.4.3 user_gattc_exc_mtu_cmd() 19

6.4.4 user_spsc_process_handler() (Central only) 19

6.4.5 user_process_catch() 19

6.4.6 user_gapc_param_update_req_ind_handler() (Central only) 20

6.5 Data management and flow control for interrupt driven project (user_sps_scheduler) 20
6.5.1 user_scheduler_init() 20
6.5.2 user_scheduler_reinit() 20
6.5.3 user_ble_pull() 20
6.5.4 user_ble_push() 20
6.5.5 user_periph_pull() 21
6.5.6 user_periph_push() 21
6.5.7 uart_rx_callback() 21
6.5.8 uart_tx_callback() 21
6.5.9 uart_flow_control_callback() 21
6.5.10 user_override_ble_xon() 21
6.5.11 user_rwip_sleep_check() 21
6.5.12 user_sps_sleep_check() 21
6.5.13 user_sps_sleep_restore() 21
6.6 Cyclic buffer for interrupt driven project (user_buffer) 22
6.6.1 user_buffer_create() 22
6.6.2 user_buffer_write_items() and user_buffer_read_items() 22
6.6.3 user_buffer_read_address() and user_buffer_release_items() 22
6.6.4 user_buffer_write_check() and user_buffer_cfm_write() 22
6.6.5 user_buffer_item_count() 22
6.6.6 user_check_buffer_almost_full() and user_check_buffer_almost_empty() 22
6.7 Data management and flow control for DMA driven project (user_sps_schedule_dma.c) 22
6.7.1 user_dma_sps_sleep_check() 22
6.7.2 user_check_dma_uart_rx_to( ) 22
6.7.3 user_check_set_flow_off( ) 23
6.7.4 bool user_check_set_flow_on( ) 23
6.8 Linked Lists for DMA driven project (user_sps_buffer_dma.c) 23
6.8.1 user_init_queues() 23
6.8.2 user_ble_to_dma_uart() 23
6.8.3 user_dma_uart_to_ble() 23
6.8.4 user_dma_uart_to_ble_confirm() 23
6.9 Application task interface to Serial Port Service 23
6.9.1 user_spss_create_db() / user_spsc_create_db() 23
6.9.2 user_spss_enable() / user_spsc_enable() 23
6.9.3 user_send_ble_data() 23
6.9.4 user_send_ble_flow_ctrl() 24
6.9.5 user_sps_server_enable_cfm_handler() and user_sps_client_enable_cfm_handler() 24
6.9.6 user_sps_server_data_tx_cfm_handler() and user_sps_client_data_tx_cfm_handler() 24
6.9.7 user_sps_server_data_write_ind_handler() and user_sps_client_data_rx_ind_handler() 24
6.9.8 user_sps_server_tx_flow_ctrl_ind_handler() and user_sps_client_tx_flow_ctrl_ind_handler() 24
6.9.9 user_sps_server_error_ind_handler() (Peripheral Only) 24
6.10 Serial Port Service 25
6.11 Sequence diagrams 26

7 DSPS Android and iOS application 30
7.1 Overview 30
7.2 Installation 31
7.2.1 Android application 31
7.2.2 iOS application 31
7.3 Device list 32
7.4 Functionality tabs 33
7.4.1 Console Mode 33
7.4.2 Read/Transfer Mode 33
7.4.3 Data File Streaming Mode 34
7.5 Information and Disclaimer Screens 34
8 Instructions for setting up a demonstration 35
8.1 Hardware setup for Basic DK 35
8.2 Hardware setup for PRO DK 35
8.3 DK to DK connection setup 36
8.4 Android / iOS application to DK connection setup 37
8.5 Run DA14585 application with SmartSnippets 37
9 SPS performance 37
Revision history 40

Figures

Figure 1: DSPS 585 Data and flow control for interrupt driver operation 11
Figure 2: DSPS 585 Data and flow control for DMA driver operation 12
Figure 3: SRS application project overview 13
Figure 4: Peripheral application FSM 16
Figure 4: Central application FSM 17
Figure 5: UART to BLE data transfer sequence diagram. Interrupt driven UART. 26
Figure 6: BLE to UART data transfer sequence diagram. Interrupt driven UART. 27
Figure 7: UART to BLE data transfer sequence diagram. DMA driven UART. 28
Figure 8: BLE to UART data transfer sequence diagram. DMA driven UART. 29
Figure 9: DSPS icon – Android – iOS 30
Figure 10: DSPS on Play Store 31
Figure 11: Installing the DSPS application 31
Figure 12: DSPS on App Store 32
Figure 13: Scan Device Screen 32
Figure 14: Select Operation Mode 33
Figure 15: Application main screens 33
Figure 16: Information & Disclaimer Screens 34
Figure 17: Basic DK pins that should be connected 35
Figure 18: UART with HW flow control on PRO kit 36
Figure 19: Performance results, DLE Devices. DMA driven projects. 39

Tables

Table 1: SPS Project Paths 9
Table 2: Device services/characteristics 9
Table 3: Server Service API messages 10
Table 4: Client Service API messages 10
Table 6: SPS application files 13
Table 7: Header files of the SPS application 14
Table 8: Peripheral application task: FSM states 16
Table 9: State transition table of the peripheral application task FSM 17
Table 10: Central application task: FSM states 17
Table 11: State transition table of the central application task FSM 18
Table 12: Performance results, DLE Devices. Interrupt driven projects. 38

Table 13: Performance results, non DLE Devices. Interrupt driven projects. 38

1 Terms and definitions

BLE	Bluetooth Low Energy
DK	Development Kit
DSPS	Dialog Serial Port Service
FSM	Finite State Machine
GAP	Generic Access Profile
GAPC	Generic Access Profile Controller
GAPM	Generic Access Profile Manager
GATT	Generic ATTribute profile
HWM	High Watermark
LWM	Low Watermark
MTU	Maximum Transmission Unit
SPS	Serial Port Service
UART	Universal Asynchronous Receiver/Transmitter

2 References

[1] UM-B-080, DA14585 Software Developer's Guide, User manual, Dialog Semiconductor.

[2] UM-B-079, DA14585 Software Platform Reference, User manual, Dialog Semiconductor.

[3] RW-BLE-GAP-IS, GAP Interface Specification, Riviera Waves.

[4] UM-B-048, Getting Started with the DA1458x Development Kit – Basic, User manual, Dialog Semiconductor.

[5] UM-B-049, Getting Started with the DA1458x Development Kit – Pro, User Manual, Dialog Semiconductor.

[6] FT_000054, TTL-232R TTL to USB Serial Converter Range of Cables, Datasheet, Future Technology Devices International Ltd.

[7] https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

3 Introduction

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

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

The application on the central device automatically starts scanning and connects to the first discovered peripheral device supporting the serial port service. The Central device also handles situations of connection loss by stopping the flow of data and automatically trying to re-establish a connection. Both central and peripheral devices can operate either in active *mode* or *extended* sleep mode.

There are four projects provided. Two projects (central and peripheral) uses interrupt driven UART operations recommended for UART baud rates lower or equal to 115200 that uses *hardware* or *software* flow control. The other two projects uses DMA driven UART operation for baud rates up to 921600, using hardware flow control only optimized for DLE operation and high speeds.

To get familiar with the DA14585's software and hardware, read any of the following:

- DA14585 Software Development Guide [1],
- DA14585 Platform Reference [1],
- Smart Development Kit – Basic user guide [4]
- Smart Development Kit – PRO user guide [5] documents.

The Android or iOS application, which is described in detail later in this document (see Section 0), scans for BLE peripheral devices that are advertising and displays them in a scan list. The user can select the peripheral device to connect the Android or iOS device.

4 Software features

This section explains the more advanced software features of the Dialog SPS reference application.

4.1 Bluetooth

- GAP Central/Peripheral roles
- GATT-based bidirectional serial link
- Write without Response/Notification methods for data streaming.
- Bluetooth flow control supported.
- Single point-to-point connection
- Automatic reconnection (Note 1) in case of link loss

Note 1 In case of connection loss during data transfer, data loss is possible due to remaining data in the BLE buffers.

4.2 UART

- Hardware and Software (Note 2) flow control are supported for the interrupt driven project
- Hardware flow control only for the DMA driven project
- Binary data transfer supported in hardware flow control mode
- UART baud rates: 9600 to 115200 for the interrupt driven project
- UART baud rates up to 921600 for the DMA driven project

Note 2 Software flow control is partially supported in extended sleep mode. Only incoming serial data can be controlled by DA14585 device. Any incoming XON/XOFF will be ignored during sleep time. On the contrary, software flow control is fully supported in Active mode. Also upon the reception of a flow off signal (0x19) an amount of up to 16 bytes can be transmitted by the DA14585 until transmission stops.

4.3 System

- iOS or Android application.
- Extended sleep mode
- Transfer rates up to 80kbps with 115200 baud rate (Interrupt Driven).
- Transfer rates up to 640 kbps for 921600 baud rate (DMA Driven).
- Memory Size of application image: 18K Code/ 16K Ram for the DMA driven project, 19K Code/12K Ram for the interrupt driven project.

5 Software architecture

5.1 Software features

5.1.1 GAP roles

The DA14585 SPS application supports both GAP central and peripheral roles.

Table 1: SPS Project Paths

Role	Keil UV4 Project Path (Note 3)
Device, Interrupt Driven	\projects\target_apps\dsps\dsps_device\Keil_5\dsps_device.uvprojx
Device, DMA	\projects\target_apps\dsps\dsps_device_dma\Keil_5\dsps_device_dma.uvprojx
Central, Interrupt Driven	\target_apps\dsps\dsps_host\Keil_5\dsps_host.uvprojx
Central, DMA	\target_apps\dsps\dsps_host_dma\Keil_5\dsps_host_dma.uvprojx

The GAP central role application operates as a Serial Port Service client and the GAP peripheral role application as a Serial Port Service server.

Note 3 All projects have been compiled and tested with Keil uVision V5.23.0.0

5.1.2 Serial Port Service

The proprietary Dialog Serial Port Service (DSPS) is used to send and receive data and software flow control signals through a BLE connection. The BLE database is kept in the server's profile and it has two 250-byte characteristics for data transmission and reception and a one byte characteristic for the flow control; 128-bit UUIDs are used for the service and the characteristics.

The Serial Port Service uses a 'Write with no response' method for transmission of data from the GATT client to the GATT server and 'Notify' for the reverse path.

The details of the SP service and its characteristics are outlined in Table 2.

Table 2: Device services/characteristics

Service/ Characteristic	UUID	Properties	Size (B)
SPS	0x0783b03e8535b5a07140a304d2495cb7	ATT_CHAR_PROP_RD	-
SPS_SERVER_TX	0x0783b03e8535b5a07140a304d2495cb8	ATT_CHAR_PROP_NTF	250
SPS_SERVER_RX	0x0783b03e8535b5a07140a304d2495cba	ATT_CHAR_PROP_WR_NO_RESP	250
SPS_FLOW_CTRL	0x0783b03e8535b5a07140a304d2495cb9	ATT_CHAR_PROP_WR_NO_RESP| ATT_CHAR_PROP_NTF	1

In the following tables the API messages are outlined, that are used for the communication of the DA14585 application task with the SPS server and client task.

Table 3: Server Service API messages

Message	Direction	Description
SPS_SERVER_ENABLE_REQ	IN	Enables the server service. Sent by the application task at connection establishment.
SPS_SERVER_ENABLE_CFM	OUT	Confirmation of the server service enable message. Sent by the profile to the application at completion of the server service enable task.
SPS_SERVER_DATA_TX_REQ	IN	Sends data to peer device. Sent by application when data is available to transmit.
SPS_SERVER_DATA_TX_CFM	OUT	Confirms that data is put into the BLE TX buffer. This is sent by the profile to inform the application that data is put to send.
SPS_SERVER_DATA_WRITE_IND	OUT	Indication that data has been received. This is sent by the profile to the application including the data that was received.
SPS_SERVER_RX_FLOW_CTRL_REQ	OUT	Sends a flow control BLE byte to the connected client. This is sent by the application when there is a state change or when it is requested.
SPS_SERVER_TX_FLOW_CTRL_IND	IN	Indication of a client request to update the flow control state by sending the device's current BLE TX state.
SPS_SERVER_ERROR_IND	OUT	Indication that an error has occurred.

Table 4: Client Service API messages

Message	Direction	Description
SPS_CLIENT_ENABLE_REQ	IN	Enables the client service. This is sent by the application task when connection is established.
SPS_CLIENT_ENABLE_CFM	OUT	Confirmation of the client service enable message. Sent by the profile to the application at completion of client service enable task.
SPS_CLIENT_DATA_TX_REQ	IN	Sends data to a peer device. This is sent by the application when there is data available to transmit.
SPS_CLIENT_DATA_TX_CFM	OUT	Confirms that data have been put in the BLE TX buffer. Sent by the profile to inform the application that data has been put to send.
SPS_CLIENT_DATA_RX_IND	OUT	Indication that data has been received. Sent by the profile to the application including the data that have been received.
SPS_CLIENT_RX_FLOW_CTRL_REQ	OUT	Sends a flow control BLE byte to the connected client. Sent by the application when there is a state change or when it is requested.
SPS_CLIENT_TX_FLOW_CTRL_IND	IN	Indication of the server request to update the flow control state by the sending device's current BLE TX state.

5.1.3 UART driver for the interrupt driven project

The UART driver controls the UART peripheral module of the DA14585 and manages the serial connection between the DA14585 and the external host device. It uses an interrupt based transmission and reception scheme requiring low CPU resources. It provides both software (XON/XOFF) and hardware (RTS/CTS) flow control schemes, respectively selected by the `CFG_UART_SW_FLOW_CTRL` and `CFG_UART_HW_FLOW_CTRL` definitions in the `user_periph_setup.h` configuration file. Selecting the UART baud rate is done by the setting the value of `BAUDRATE_CONFIG`

(1-5) in the same file. Both baud rate and flow control methods can be set by the configuration wizard tab that is activated when user_periph_setup.h is opened.

5.1.4 Data scheduling and flow control for the interrupt driven project

The data scheduling and part of the flow control mechanism (flow control signals generated from the cyclic buffer) are implemented in the application task layer. On the other hand, incoming flow control signal management is implemented in the BLE SPS profile and UART driver accordingly.

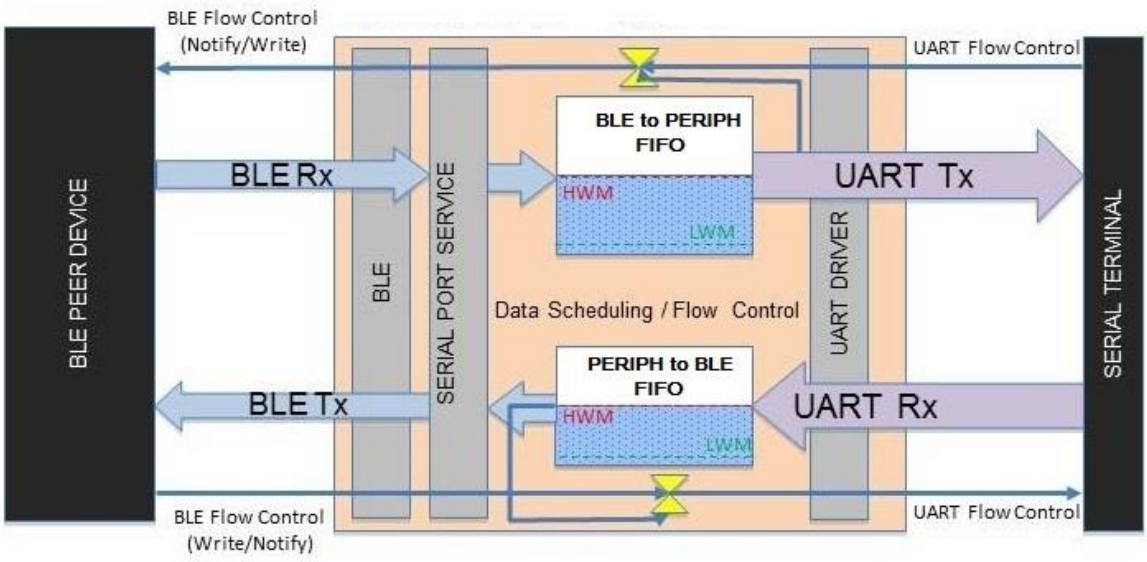

Figure 1: DSPS 585 Data and flow control for interrupt driver operation

Two cyclic FIFOs are allocated at system start-up. The UART driver pushes the data received from the UART interface into the periph_to_ble_buffer, while the Serial Port Service pushes the data received via the BLE interface into the ble_to_periph_buffer. Data transmission scheduling is done using the following methods:

- For the data received over the UART interface, an asynchronous process checks the presence of pending data in the periph_to_ble_buffer. If no data is being transferred, transfer is initiated. It schedules data transmission at the head of the FIFO to the BLE interface, via a data streaming mechanism and the corresponding SPS characteristic. Then a completion event scheme in the application task is used to schedule the next data transmission.
- The scheduling of data transmission to the UART is initiated in the process of pushing the data into the ble_to_periph_buffer, performed by the SPS profile. If the UART interface path is already busy, a check for pending data in the FIFO is done again after the active UART transmit process.

The DA14585 SPS application uses both UART flow control methods: software (XON/XOFF) flow control and hardware (RTS/CTS) flow control. For the BLE interface, a similar flow control mechanism has been developed in SPS using the SPS_FLOW_CTRL characteristic.

High watermark (HWM) and low watermark (LWM) values are defined for each FIFO. This is done to set the buffer utilization level at which a flow off or flow on signal is sent to the corresponding interface. Flow control signaling on the BLE interface is also triggered when the corresponding flow control signal is received from the UART interface.

5.1.5 UART driver for the DMA driven project

The UART driver controls the UART peripheral module of the DA14585 and manages the serial connection between the DA14585 and the external host device. It uses a DMA based transmission and reception scheme requiring minimal CPU resources. It provides hardware (RTS/CTS) flow control scheme only. The selection of the UART baud rate is done by the setting the value of

BAUDRATE_CONFIG (1-8) in the user_periph_setup.h. Both baud can be set by the configuration wizard tab that is activated when user_periph_setup.h is opened.

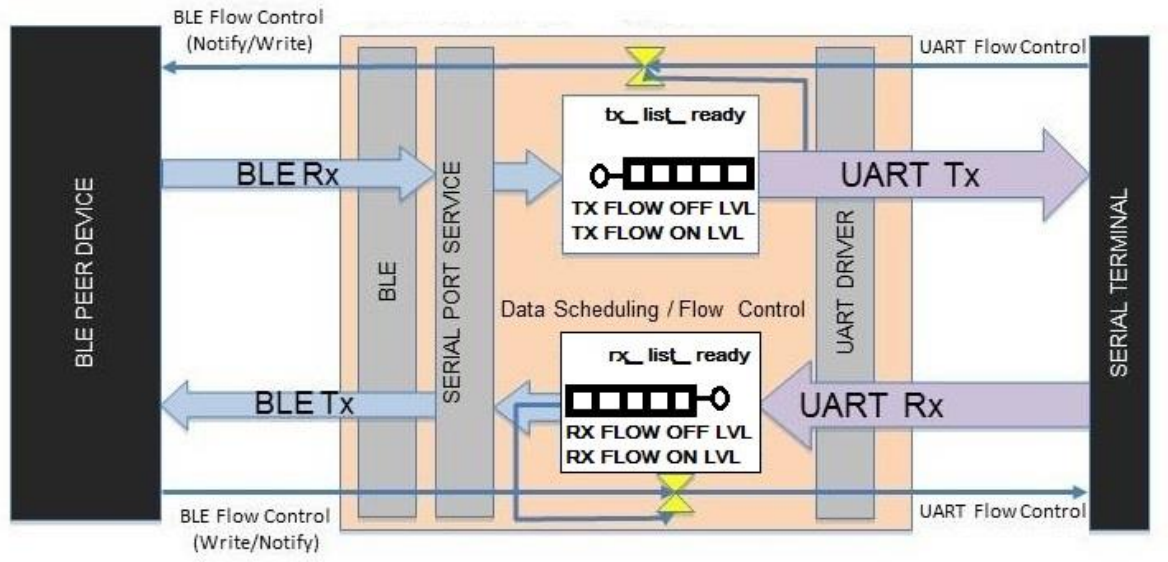

Figure 2: DSPS 585 Data and flow control for DMA driver operation

5.1.6 Data scheduling and flow control for the DMA driven project

The data scheduling and part of the flow control mechanism (flow control signals generated from the cyclic buffer) are implemented in the application task layer. Incoming flow control signal management is implemented in the BLE SPS profile and UART driver accordingly.
Two linked lists with zero members are allocated on startup. The dma_uart.rx_list_ready holds buffers received by the UART and waiting to be transmitted by the BLE interface.
The dma_uart.tx_list_ready holds buffers received via the BLE interface waiting to be transmitted by the UART. The scheduling of data transmission is performed using the following methods:

In order to minimize data copying and the allocation of heap memory a buffer optimization scheme has been used. The buffers used in DMA UART receive are type of gattc_write_cmd (central) or gattc_send_evt_cmd (peripheral) and are placed in dma_uart.rx_list_ready. When scheduling is performed they are detached from the linked list and they are forwarded from the user space to profile for transmission. For the opposite direction, the buffers received by the BLE gattc_event_ind (central) or gattc_write_req_ind (peripheral) are placed in dma_uart.tx_list_ready. When scheduling is performed they are detached from the linked list and they are forwarded from the user space to DMA for transmission over UART.

For the data received over the UART interface, an asynchronous process (user_uart_to_ble) checks the presence of pending buffers in the dma_uart.rx_list_ready and when there is no data transfer ongoing it initiates a transfer by sending SPS_CLIENT_DATA_TX_REQ (central) or SPS_SERVER_DATA_TX_REQ (peripheral). For the opposite direction, an asynchronous process (dma_uart_tx_async) checks the presence of pending buffers (dma_uart.tx_list_ready) and when there is no data transfer ongoing it initiates a transfer by calling dma_uart_write.

When the transfer of buffers over DMA is complete two callbacks are used. On the UART DMA receive path the dma_uart_rx_callback places the new buffer at the end of dma_uart.rx_list_ready and if the flow of data is not disabled reinitiates the read of UART via DMA. On the UART DMA transmit path the dma_uart_tx_callback is called when the transmission of a buffer over UART is complete. This function checks if more buffers exist in dma_uart.tx_list_ready and reinitiates the transfer by calling dma_uart_write using a detached buffer from dma_uart.tx_list_ready.

High watermark (HWM) and low watermark (LWM) values are defined for each linked list to set a buffer utilization level at which a flow off or flow on signal is sent to the corresponding interface. Regarding the DMA UART TX interface the `TX_FLOW_OFF_LVL` and `TX_FLOW_ON_LVL` applied on `dma_uart.tx_list_ready` define when the `FLOW_ON` and `FLOW_OFF` will be send to the peer device. For the DMA UART RX interface the `RX_FLOW_ON_LVL` and `RX_FLOW_OFF_LVL` applied on `dma_uart.rx_list_ready` define when the RTS will be asserted or de-asserted.

The values of `TX_FLOW_OFF_LVL`, `TX_FLOW_ON_LVL`, `RX_FLOW_ON_LVL`, `RX_FLOW_OFF_LVL` can be trimmed from `dma_uart_sps.h`.

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.

5.2 Source files

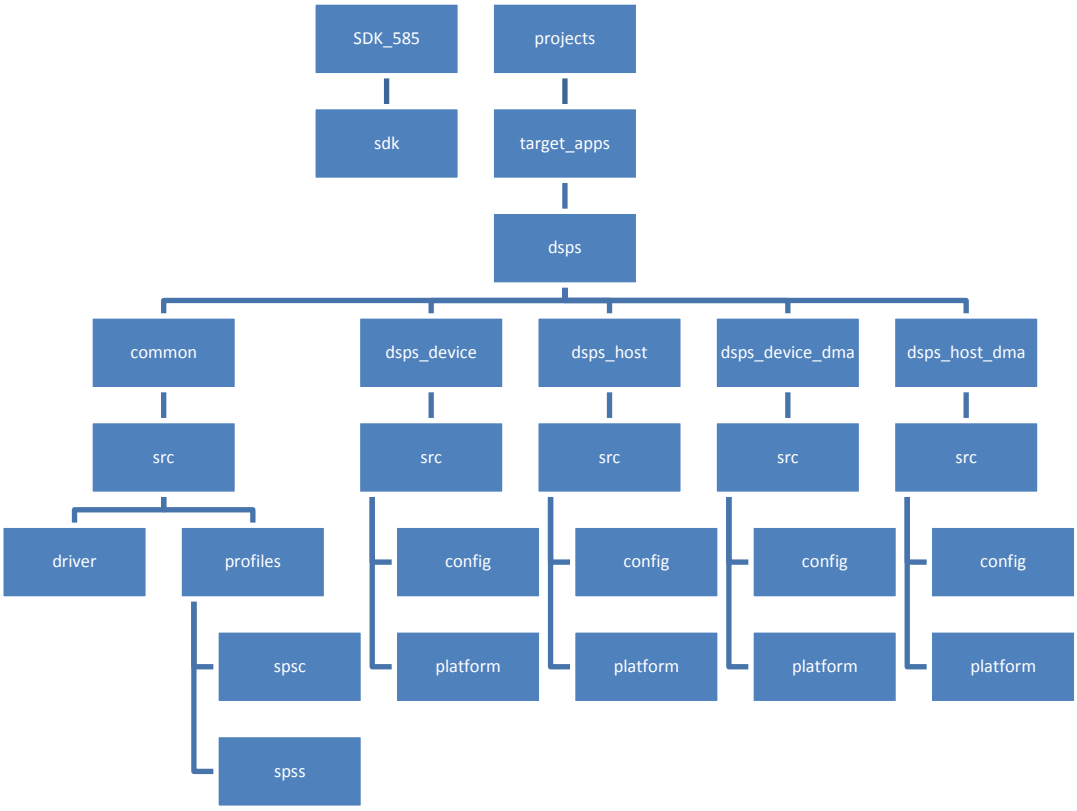

Figure 3: SRS application project overview

The main files of the SPS application project are listed in Table 5.

Table 5: SPS application files

File name (Note 4)	Description
.\dsps_device\src\user_sps_device.c .\dsps_host\src\user_sps_host.c .\dsps_device_dma\src\user_sps_device_dma.c .\dsps_host_dma\src\user_sps_host_dma.c	Application main functions and system callbacks.
.\dsps_device\src\user_spss.c .\dsps_device\src\user_spss_task.c .\dsps_device_dma\src\user_spss.c .\dsps_device_dma\src\user_spss_task.c	Application profile specific functions (Peripheral role).

File name (Note 4)	Description
.\dsps_host\src\user_spsc.c .\dsps_host\src\user_spsc_task.c .\dsps_host_dma\src\user_spsc.c .\dsps_host_dma\src\user_spsc_task.c	Application profile specific functions (Central role).
.\dsps_device\src\platform\user_periph_setup.c .\dsps_host\src\platform\user_periph_setup.c .\dsps_device_dma\src\platform\user_periph_setup.c .\dsps_host_dma\src\platform\user_periph_setup.c	Peripheral modules initialization, GPIO pins assignment.
.\common\src\user_scheduler.c	Application level UART control, FIFOs management, application level data transfer. Interrupt driven UART.
.\common\src\user_buffer.c	FIFO management functions. Interrupt driven UART.
.\common\src\user_sps_utils.c	Utilities used by the DSPS project header file
.\common\src\driver\uart_sps.c	Custom UART driver with hardware/software flow control. Interrupt driven UART.
.\common\src\user_sps_buffer_dma.c	Application level, linked list management and data transfer. DMA driven UART.
.\common\src\user_sps_schedule_dma.c	Sleep scheduling, asynchronous RX/TX operations and flow control check. DMA driven UART.
.\common\src\driver\dma_uart_sps.c	Custom DMA UART driver with hardware flow control.
.\common\src\profiles\spsc\sps_client.c .\common\src\profiles\spsc\sps_client_task.c	Serial Port Service, GATT client role.
.\common\src\profiles\spss\sps_server.c .\common\src\profiles\spss\sps_server_task.c	Serial Port Service, GATT server role.

Note 4 SPS files are in folder ‘projects\target_apps\dsps’ and all displayed paths are relative to this folder.

5.3 Header files

The application’s header files and their purpose are listed in Table 6.

Table 6: Header files of the SPS application

File name (Note 5)	Purpose
.\dsps_device\src\config\da1458x_config_advanced.h .\dsps_host\src\config\da1458x_config_advanced.h .\dsps_device_dma\src\config\da1458x_config_advanced.h .\dsps_host_dma\src\config\da1458x_config_advanced.h	Advanced compile configuration files.
.\dsps_device\src\config\da1458x_config_basic.h .\dsps_host\src\config\da1458x_config_basic.h .\dsps_device_dma\src\config\da1458x_config_basic.h .\dsps_host_dma\src\config\da1458x_config_basic.h	Basic compile configuration files.
.\dsps_device\src\config\user_callback_config.h .\dsps_host\src\config\user_callback_config.h .\dsps_device_dma\src\config\user_callback_config.h .\dsps_host_dma\src\config\user_callback_config.h	Callback functions configuration files.
.\dsps_device\src\config\user_config.h .\dsps_host\src\config\user_config.h .\dsps_device_dma\src\config\user_config.h .\dsps_host_dma\src\config\user_config.h	User configuration files.

File name (Note 5)	Purpose
.\dsps_device\src\config\user_modules_config.h .\dsps_host\src\config\user_modules_config.h dsps_device_dma\src\config\user_modules_config.h .\dsps_host_dma\src\config\user_modules_config.h	User modules configuration files.
.\dsps_device\src\config\user_periph_setup.h .\dsps_host\src\config\user_periph_setup.h .\dsps_device_dma\src\config\user_periph_setup.h .\dsps_host_dma\src\config\user_periph_setup.h	Peripherals setup header files.
.\dsps_device\src\config\user_profiles_config.h .\dsps_host\src\config\user_profiles_config.h .\dsps_device_dma\src\config\user_profiles_config.h .\dsps_host_dma\src\config\user_profiles_config.h	Configuration files for the profiles used in the application.
.\dsps_device\src\user_sps_device.h .\dsps_host\src\user_sps_host.h .\dsps_device_dma\src\user_sps_device.h .\dsps_host_dma\src\user_sps_host.h	Application main functions and system callbacks header files.
.\dsps_device\src\user_spss.h .\dsps_device\src\user_spss_task.h .\dsps_device_dma\src\user_spss.h .\dsps_device_dma\src\user_spss_task.h	Application profile specific functions (Peripheral role) header files.
.\sps_host\src\user_spsc.h .\sps_host\src\user_spsc_task.h .\sps_host_dma\src\user_spsc.h .\sps_host_dma\src\user_spsc_task.h	Application profile specific functions (Central role) header files.
.\common\src\user_scheduler.h	Scheduler header file. Interrupt driven project.
.\common\src\user_schedule_dma.h	Scheduler header file. DMA driven project.
.\common\src\user_buffer.h	FIFO management functions header file. Interrupt driven project.
.\common\src\user_sps_buffer_dma.h	Linked list management functions header file. DMA driven project.
.\common\src\user_sps_utils.h	Utilities used by the DSPS project header file
.\common\src\driver\uart_sps.h	Custom UART driver header file. Interrupt driven project.
.\common\src\driver\dma_uart_sps.h	Custom DMA UART driver header file. DMA UART driven project.
.\common\src\profiles\spsc\sps_client.h .\common\src\profiles\spsc\sps_client_task.h	Serial Port Service header file, GATT client role.
.\common\src\profiles\spss\sps_server.h .\common\src\profiles\spss\sps_server_task.h	Serial Port Service header file, GATT server role.

Note 5 SPS files are in folder ‘projects\target_apps\dsps’ and all displayed paths are relative to this folder.

6 Code overview and state machines

This section provides information about important functions of the application and a detailed description of the Finite State Machines used.

6.1 Application task state machine

6.1.1 Peripheral state machine

The FSM of the peripheral SPS application consists of the following states:

Table 7: Peripheral application task: FSM states

State	Status
APP_DISABLED	Server disabled, database not created
APP_DB_INIT	Database initialization in progress
APP_CONNECTABLE	Advertising
APP_CONNECTED	Connected to central

Figure 4 shows the FSM of the peripheral application.

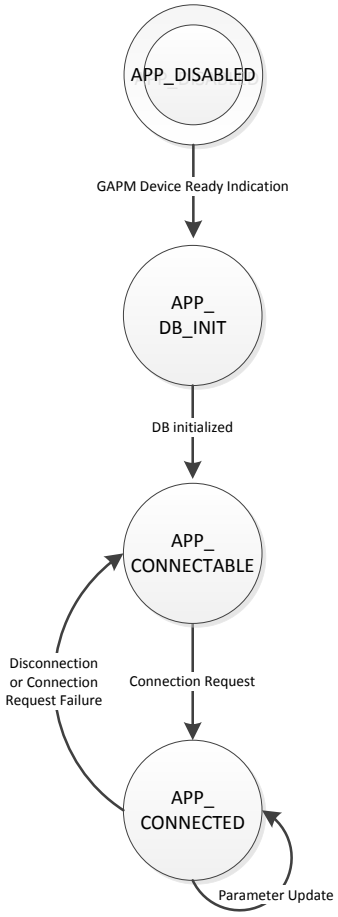

Figure 4: Peripheral application FSM

Table 8: State transition table of the peripheral application task FSM

State transition	Event
From >>> To	Action
APP_DISABLED	Reception of GAPM configuration completion indication.
APP_DB_INIT	Start database initialization of profile.
APP_DB_INIT	Database initialization procedure is completed.
APP_CONNECTABLE	Start server service.
APP_CONNECTABLE	Central initiates connection. Reception of GAPC_CONNECTION_REQ_IND message.
APP_CONNECTED	Start server profile.
APP_CONNECTED	Disconnect indication received (GAPC_DISCONNECT_IND).
APP_CONNECTABLE	Start advertising.
APP_CONNECTED	Parameter update request.
APP_CONNECTED	Update connection parameters

6.1.2 Central state machine

The FSM of the central SPS application consists of the following states:

Table 9: Central application task: FSM states

State	Status
APP_DISABLED	Client disabled, database not created
APP_DB_INIT	Database initialization in progress
APP_CONNECTABLE	Scanning
APP_CONNECTED	Connected to peripheral

Figure 5 shows the FSM of the central application.

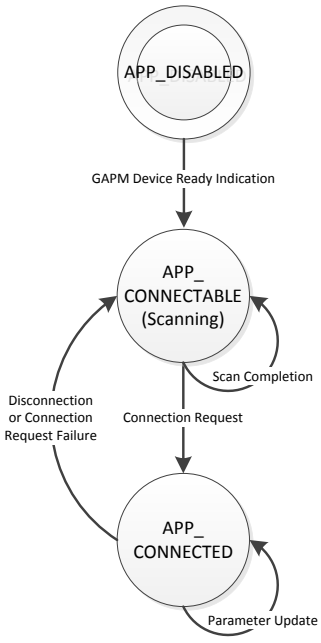

Figure 5: Central application FSM

Table 10: State transition table of the central application task FSM

State transition	Event
From >>> To	**Action**
APP_DISABLED	Reception of GAPM device ready indication.
APP_CONNECTABLE	Start client service.
APP_CONNECTABLE	Appropriate peripheral found. Reception of GAPC_CONNECTION_REQ_IND message.
APP_CONNECTED	Start server profile.
APP_CONNECTABLE	Scan procedure completed.
APP_CONNECTABLE	Restart scanning.
APP_CONNECTED	Disconnect indication received (GAPC_DISCONNECT_IND).
APP_CONNECTABLE	Start scanning.
APP_CONNECTED	Parameter update request.
APP_CONNECTED	Update connection parameters

6.2 Application callback functions

6.2.1 user_on_connection()

This function is called when a connection request (GAPC_CONNECTION_REQ_IND) is received from the GAPC. The peripheral application enables the profiles and services (default_app_on_connection() [2]). It also initializes an MTU exchange procedure and a parameter update. The central application disables connection timer, enables profiles and services and initializes an MTU exchange procedure.

6.2.2 user_on_disconnect()

This function is called when a disconnect indication (GAPC_DISCONNECT_IND) is received from the GAPC. The peripheral application restarts advertising (default_app_on_disconnect() [2]). The central application restarts scanning.

6.2.3 user_on_set_dev_config_complete() (Central only)

This function is called when the device configuration is complete (GAPM_SET_DEV_CONFIG) and starts scanning.

6.2.4 user_on_scanning_completed() (Central only)

This function is called when a scan completion event (GAPM_SCAN_ACTIVE, GAPM_SCAN_PASSIVE) is received from the GAPM. Depending on the reason of completion, it either initiates a connection to a peripheral device and starts connection timer (if scan was cancelled by application) or it restarts scanning.

6.2.5 user_on_adv_report_ind() (Central only)

This function is called when an advertising report (GAPM_ADV_REPORT_IND) is received. When the advertising peripheral device supports the DSPS profile (by containing the DSPS profile UUID in the advertise data) it sets connection parameters and cancels scan operation. Details of the Advertise Data structure can be found in paragraph 4.7.3 of [3] and [7].

6.2.6 user_on_connect_failed() (Central Only)

This function is called when a connection procedure fails to complete (GAPM_CONNECTION_DIRECT). The central application asserts a warning in debug mode and stops the connection timer.

6.3 Main loop callback functions

6.3.1 user_on_init()

This function is called once, during system initialization.

6.3.2 user_on_db_init_complete()

This function is called once on database initialization. For the interrupt driven project the ring buffers and the scheduler are initialized. The UART and its callbacks are initialized and set to the desired modes of operation. For the DMA driven project the linked lists are initialized.

6.3.3 user_on_system_powered()

This function is called periodically upon scheduling pending BLE events after device wake up. For the interrupt driven project It calls `user_ble_pull()` (section 6.5.3) to send data to the BLE interface if available and if extended sleep mode is used, it restores UART flow control state.

For the DMA driven project this function keeps the control state (`dma_uart.power_on_state`) of operation. On wakeup it initializes the flow control signals and the DMA and on the subsequent runs it asynchronously triggers transmit operations to BLE or UART DMA.

6.3.4 user_before_sleep()

This function is called periodically upon disable of interrupts and before system prepares for sleep. If extended sleep mode is used, it goes through a series of checks in order to decide if the system can go to sleep or not. It runs part of `rwip_sleep()` and also checks if cyclic buffers or linked lists and UART buffers are empty.

6.4 Other application functions

6.4.1 user_scan_start() (Central only)

This function allocates and sends a `GAPM_START_SCAN_CMD` message to initiate a scan operation. It is called in several cases when scanning is needed to start (sections 6.2.2, 6.2.3, 6.2.4).

6.4.2 user_gapm_cancel() (Central only)

This function allocates and sends a `GAPM_CANCEL_CMD` message to either cancel the connection request when it times out (section 6.2.4) or to cancel the ongoing scan procedure when a device supporting the DSPS profile has been found (section 6.2.5).

6.4.3 user_gattc_exc_mtu_cmd()

This function allocates and sends a `GATTC_EXC_MTU_CMD` message to initiate an MTU exchange between peer devices. It is called by `user_on_connection()` callback function on both GAP roles.

6.4.4 user_spsc_process_handler() (Central only)

This function is a wrapper for the `app_std_process_event()` to be called by the `user_process_catch()` (section 6.4.5) function which handles incoming application events.

6.4.5 user_process_catch()

This function handles all the messages unhandled by the SDK task handlers. The peripheral application handles only the DSPS profile application tasks. The central application handles the DSPS profile application tasks and also the main application tasks.

6.4.6 user_gapc_param_update_req_ind_handler() (Central only)

This function is called by the main application task handler on reception of `GAPC_PARAM_UPDATE_REQ_IND` and sends a parameter update confirmation message (`GAPC_PARAM_UPDATE_CFM`) with the new connection parameters.

6.5 Data management and flow control for interrupt driven project (user_sps_scheduler)

6.5.1 user_scheduler_init()

This function is called by `user_on_db_init_complete()` (section 6.3.1) callback and it allocates the cyclic buffers (section 6.6.1), it registers UART callbacks (sections 6.5.7, 6.5.9) and sets the UART flow control state.

6.5.2 user_scheduler_reinit()

This function is called by `periph_init()`, which is called periodically, every time the system wakes up. It is used to register UART callbacks (sections 6.5.7, 6.5.9).

6.5.3 user_ble_pull()

This function is called periodically by the `user_on_system_powered()` (section 6.3.3) to initiate a BLE transmission and by the `user_sps_server_data_tx_cfm_handler()` (section 6.9.6) (peripheral role) or the `user_sps_client_data_tx_cfm_handler()` (section 6.9.6) (central role) to continue transmission. It is also called by the `user_sps_server_enable_cfm_handler()` (section 6.9.5) (peripheral role) or the `user_sps_client_enable_cfm_handler()` (section 6.9.5) (central role) to initialize TX state.

- When called by `user_on_system_powered()`, it checks if a transmission is ongoing by checking the flow state `tx_flow_en` and the `tx_busy_flag`. If flow is on and there is no ongoing procedure, then it checks cyclic buffer for data. If there is data available, it sends it to BLE interface only if available data is more than `TX_WAIT_LEVEL` or if buffer has been polled for more than `TX_WAIT_ROUNDS` times.
- When called by the `user_sps_server_data_tx_cfm_handler()` or the `user_sps_client_data_tx_cfm_handler()`, it initially checks to confirm that data has successfully been sent and if true then it frees it from cyclic buffer. It clears the `tx_busy_flag` and checks if `tx_flow_en` is `true` and that the available data in cyclic buffer is more than `TX_WAIT_LEVEL`. If it is, it sends the available data to the BLE interface and sets the `tx_busy_flag`.
- When called by the `user_sps_server_enable_cfm_handler()` or the `user_sps_client_enable_cfm_handler()` it behaves as an unsuccessful transmission and it clears the `tx_busy_flag`.

In all cases it checks if a flow on control signal should be send to the UART interface and sends it.

In order to maximize throughput, data should be sent after a certain amount has been collected. Two parameters are introduced to support this scheme, `tx_wait_level` and `TX_WAIT_ROUNDS`. There is a trade-off between throughput and responsiveness of the application which depends on the values of these parameters. `tx_wait_level` is calculated to be the ¾ of the MTU as exchanged with the central device and acquired from `gattc_mtu_changed_ind_handler`. `TX_WAIT_ROUNDS` should have a value that does not significantly deteriorate the responsiveness.

6.5.4 user_ble_push()

This function is called by the `user_sps_server_data_rx_ind_handler()` (section 6.9.7) (peripheral role) or the `user_sps_client_data_rx_ind_handler()` (section 6.9.7) (central role) when there is available data in the BLE interface. It pushes data in the `ble_to_periph_buffer` cyclic buffer and

checks whether the buffer level gets above the high watermark to send a flow off signal to the peer device.

6.5.5 user_periph_pull()

This function is called from the `uart_tx_callback()` (section 6.5.8) function which is called upon the end of the transmission of the previously sent data to the UART or by any function that needs to send data to the UART. It frees previously sent data from the `ble_to_periph_buffer` cyclic buffer and returns available data to send to the UART. It also checks whether the buffer level gets below the low watermark to send a flow on signal to the peer device.

6.5.6 user_periph_push()

This function is called from the `uart_rx_callback()` (section 6.5.7) function which is called upon reception of data from the UART. It confirms the amount of data that have been written into the `periph_to_ble_buffer` cyclic buffer and returns to the caller the data amount available and the pointer to the first element. It also checks whether the buffer level gets above the high watermark to send a flow off signal to the UART.

6.5.7 uart_rx_callback()

This function is the callback that handles the UART data reception. It always reinitiates a read procedure by calling the `uart_sps_read()` function. It is initially called by the `user_scheduler_init()` (section 6.5.1) function at system power up and `user_scheduler_reinit()` (section 6.5.2) function every time the system wakes up.

6.5.8 uart_tx_callback()

This function is the callback that handles the UART data transmission completion by sending the available data in cyclic buffer calling the `user_periph_pull()` (section 6.5.5). It also initiates a write procedure when called by `user_ble_push()` (section 6.5.4).

6.5.9 uart_flow_control_callback()

This function is the callback that handles the UART flow control reception by calling either `user_override_ble_xon()` (section 6.5.10) or by issuing a flow off signal to peer device using `user_send_ble_flow_ctrl` (section 6.9.4).

6.5.10 user_override_ble_xon()

This function sets UART flow control flags, checks the `ble_to_periph_buffer` level and sends a flow on signal to the connected BLE peer device. It overrides the automatic level detection and is called by the UART flow control callback (section 6.5.9).

6.5.11 user_rwip_sleep_check()

This function performs some of the pre-sleep checks in order to provide feedback if application will be able to enter extended sleep. It is called by the `user_sps_sleep_check()` (section 6.5.12) in a series of buffer and FIFO checks.

6.5.12 user_sps_sleep_check()

This function is performing a series of pre-sleep checks in order to determine if system is able to enter extended sleep. It checks cyclic buffers (section 6.6.5), UART FIFOs and also if a data stream is currently transferred through UART while trying to flow off it.

6.5.13 user_sps_sleep_restore()

This function restores UART RX flow control state to the previous state before sleep. It is necessary, as `user_sps_sleep_check()`(section 6.5.12) disables data flow before entering sleep state.

6.6 Cyclic buffer for interrupt driven project (user_buffer)

6.6.1 user_buffer_create()

This function allocates and initializes a cyclic buffer and sets the high and low watermark levels.

6.6.2 user_buffer_write_items() and user_buffer_read_items()

These functions push and pop the specified amount of data, if possible. When the available space is insufficient, data is written until the buffer is full. When the requested data is not available, only the available data is read.

6.6.3 user_buffer_read_address() and user_buffer_release_items()

As a pair, they are equivalent to the `user_buffer_read_items()` function with the difference that accessing and releasing of the slots of the buffer happen at a different time. `user_buffer_read_address()` returns a pointer to buffer data and `user_buffer_release_items()` frees the requested memory space.

6.6.4 user_buffer_write_check() and user_buffer_cfm_write()

As a pair, they are equivalent to the `user_buffer_write_items()` with one difference. The `user_buffer_write_check()` returns an available space that is not fragmented, so that another function can write directly to the buffer, without the risk of overwriting data. After the write is complete, `user_buffer_cfm_write()` must be used to confirm the exact amount of data that have been written.

6.6.5 user_buffer_item_count()

This function returns the number of items in the provided cyclic buffer. It is called by various functions (sections 6.5.3, 6.5.12) to check the data level in a certain buffer.

6.6.6 user_check_buffer_almost_full() and user_check_buffer_almost_empty()

These functions check the provided buffer's level against the high watermark and low watermark respectively return true only when certain level is reached once thus implementing a kind of hysteresis loop.

6.7 Data management and flow control for DMA driven project (user_sps_schedule_dma.c)

6.7.1 user_dma_sps_sleep_check()

This function is performing a series of pre-sleep checks in order to determine if system is able to enter extended sleep. It checks linked lists length, receive data timeout and transmit activity. If all conditions are met, then it deasserts RTS, waits for a character length interval and checks if a character has arrived. This is because reception of a character might have started just before the deassertion of the RTS signal. If the DMA UART Rx channel is empty the system is allowed to enter extended sleep.

6.7.2 user_check_dma_uart_rx_to()

This function is called asynchronously to check if there is UART DMA Rx timeout. Each time it runs the counters `rx_to_cnt` (counts rx inactivity) or `rx_zero_data_cnt` (counts empty buffer runs) are incremented or zeroed in order to signal DMA timeout handling. Specifically if `rx_to_cnt` exceeds `DMA_UART_RX_TO_ROUNDS` the `dma_uart_timeout` runs in order the receive buffer to be attached to `dma_uart.rx_list_ready` and the UART DMA Rx channel to be reinitialized with a newly allocated buffer.

6.7.3 user_check_set_flow_off( )

This function checks if the receive list queue has exceeded the limit `RX_FLOW_OFF_LVL` in order to deassert RTS.

6.7.4 bool user_check_set_flow_on()

This function checks if the receive list queue is bellow limit `RX_FLOW_ON_LVL` in order to assert RTS.

6.8 Linked Lists for DMA driven project (user_sps_buffer_dma.c)

6.8.1 user_init_queues()

This function initializes `dma_uart`. The `dma_uart` structure is the core container of system data that includes the linked lists heads and operation variables

6.8.2 user_ble_to_dma_uart()

Places a buffer from BLE to DMA UART TX queue (`dma_uart.tx_list_ready`).

6.8.3 user_dma_uart_to_ble()

This function is called to initiate the transmission of data over BLE. It checks the size of `dma_uart.rx_list_ready` and if it is not empty it detaches a buffer marked with `dma_uart.p_rx_ready_active` and forwarded to profile task.

6.8.4 user_dma_uart_to_ble_confirm()

On successful placement of a data buffer to BLE stack a confirmation message is send to user profile. This message is `SPS_CLIENT_DATA_TX_CFM` (central) or `SPS_SERVER_DATA_TX_CFM (device)` that also holds the confirmation of success or failure of the operation. On success the `user_dma_uart_to_ble_confirm` is triggered in order to send the next buffer by calling `user_dma_uart_to_ble` (section 6.8.3). When failure the `dma_uart.p_rx_ready_active` is resend for transmission.

6.9 Application task interface to Serial Port Service

6.9.1 user_spss_create_db() / user_spsc_create_db()

This function creates and sends an `GAPM_PROFILE_TASK_ADD_CMD` message to the GAP Manager (GAPM) to initialize the SPS server task `TASK_ID_SPS_SERVER` or the SPS client task `TASK_ID_SPS_CLIENT`.

6.9.2 user_spss_enable() / user_spsc_enable()

This function allocates and sends a `SPS_SERVER_ENABLE_REQ` (peripheral role) or a `SPS_CLIENT_ENABLE_REQ` (central role) message to the `TASK_ID_SPS_SERVER` or `TASK_ID_SPS_CLIENT` respectfully. It is registered as a callback in the `prf_funcs` structure and it is called by the SDK to enable the services.

6.9.3 user_send_ble_data()

It is used only in interrupt driven project. This function allocates and sends a `SPS_SERVER_DATA_TX_REQ` (peripheral role) or a `SPS_CLIENT_DATA_TX_REQ` (central role) message to the `TASK_ID_SPS_SERVER` or `TASK_ID_SPS_CLIENT` respectfully. It is called by the `user_ble_pull()` (section 6.5.3) to send data to the BLE interface and consequently to the peer device.

6.9.4 user_send_ble_flow_ctrl()

This function sends a SPS_SERVER_RX_FLOW_CTRL_REQ message (peripheral role) or a SPS_CLIENT_RX_FLOW_CTRL_REQ message (central role) to the TASK_ID_SPS_SERVER or TASK_ID_SPS_CLIENT respectfully, in order to transmit the selected flow control state to the peer device.

6.9.5 user_sps_server_enable_cfm_handler() and user_sps_client_enable_cfm_handler()

These functions handle the SPS_SERVER_ENABLE_CFM (peripheral role) or the SPS_CLIENT_ENABLE_CFM (central role) message which confirms that profile has been enabled. It calls the user_ble_pull() (interrupt driven project, section 6.5.3) function or the user_dma_uart_to_ble() (section 6.8.3) function (DMA driven project) to initialize the TX state.

6.9.6 user_sps_server_data_tx_cfm_handler() and user_sps_client_data_tx_cfm_handler()

These functions handle the SPS_SERVER_DATA_TX_CFM (peripheral role) or the SPS_CLIENT_DATA_TX_CFM (central role) message which confirms whether data transmission is successful or not. When used in interrupt driven project It calls user_ble_pull() (section 6.5.3) function with proper arguments. When used in DMA driven project it calls user_dma_uart_to_ble_confirm() (section 6.8.4) function with proper arguments.

6.9.7 user_sps_server_data_write_ind_handler() and user_sps_client_data_rx_ind_handler()

These functions handle the SPS_SERVER_DATA_WRITE_IND (peripheral role) or the SPS_CLIENT_DATA_RX_IND (central role) message which indicates that data have been received by the BLE interface. For the interrupt driven project It calls user_ble_push() (section 6.5.4) to pass the data that were received. For the DMA driven project it calls user_ble_to_dma_uart() to place a buffer at the end of dma_uart.tx_list_ready.

6.9.8 user_sps_server_tx_flow_ctrl_ind_handler() and user_sps_client_tx_flow_ctrl_ind_handler()

These functions handle the SPS_SERVER_TX_FLOW_CTRL_IND (peripheral role) or the SPS_CLIENT_TX_FLOW_CTRL_IND (central role) message which indicates that a flow control signal has been received by the peer device. It is mainly used for debug since sps_env->tx_flow_en that holds the flow control state is set on the reception of GATT message inside gattc_write_req_ind_handler (device) or gattc_event_ind_handler (host).

6.9.9 user_sps_server_error_ind_handler() (Peripheral Only)

This function handles the SPS_SERVER_ERROR_IND message that indicates an error in the SPS server profile. It asserts a warning if the debug mode is enabled.

6.10 Serial Port Service

The Serial Port Service consists of two parts: a server (peripheral device) and a client (central or host device). It exposes three characteristics as displayed in Table 2.

Server TX Data and Server RX Data characteristics are used to transfer data between the two connected devices (device - host). From the server's perspective, the Server TX Data is used for outgoing data by notifying the client whenever there is data available. The Server RX Data is used by the client to write data to the device's database by using the GATT 'write no response' method.

The Flow control characteristic is used by both sides to control the flow of data in both directions. The server notifies the client for the Server RX channel and the client writes flow control characters controlling the Server TX channel. The value 0x01 is considered as a 'flow on' signal and the value 0x02 as a 'flow off' signal.

Both profile roles have a common way to transmit data. Data transmission is invoked by reception of a `SPS_SERVER_DATA_TX_REQ` message from the application layer when a GAP peripheral role is selected, or the reception of `SPS_CLIENT_DATA_TX_REQ` when in GAP central role. This happens only when there is no ongoing transmission, verified by checking the state of the `tx_busy_flag` or the state of `dma_uart.p_rx_ready_active` for the DMA driven project.

The profile sends data only if the TX flow control flag is enabled, otherwise the data is stored until TX data flow control is re-enabled. After reception of confirmation that data have been put to send, a `SPS_SERVER_DATA_TX_CFM` (peripheral role) or a `SPS_CLIENT_DATA_TX_CFM` (central role) message is sent to the `TASK_APP` informing about the status of the request, so that a new data transmission is scheduled (section 6.9.6).

Transmission occurs either by using the function `sps_server_send_data()` at the server side or the `sps_client_gatt_write_data_tx()` function at the client side.

A 'normalization' of the size to be sent is proposed at the application level to increase throughput. The requested size `tx_size` is calculated based on the MTU size as acquired by `gattc_mtu_changed_ind_handler()` and the `tx_wait_level` is calculated at the ¾ of the `tx_size`.

For the interrupt driven project the following mechanism applies. After the initial transmission, when a completion of the transmission arrives, a new transmission is scheduled depending on the amount of data in the `periph_to_ble_buffer`. A low limit check is performed with the `tx_wait_level` and the procedure is repeated. When the criteria are not met, no transmission is scheduled and the application has to reinitiate a transmission when there are data available.

Upon the reception of data, the GATTC message handler allocates and sends a `SPS_SERVER_DATA_WRITE_IND` (peripheral role) or a `SPS_CLIENT_DATA_RX_IND` (central role) to the `TASK_APP`, which inserts data in the `ble_to_periph_buffer`, updates the BLE flow control flags and initiates the UART TX.

For the DMA driven project the following mechanism applies. After the initial transmission, when a completion of the transmission arrives, a new transmission is scheduled depending on the existence of ready buffers in the `dma_uart.rx_list_ready`.

Upon the reception of data, the GATTC message handler allocates and sends a `SPS_SERVER_DATA_WRITE_IND` (peripheral role) or a `SPS_CLIENT_DATA_RX_IND` (central role) to the `TASK_APP`, which inserts the buffer in the `dma_uart.tx_list_ready` and triggers a DMA UART transmit if the UART TX is inactive by calling `dma_uart_tx_async()`.

When a flow control signal is received by the GATTC (from the `gattc_write_req_ind_handler()` function of the server or from the `gattc_event_ind_handler()` of the client) it updates the profile TX flow control flag and also informs the application by sending the `SPS_SERVER_TX_FLOW_CTRL_IND` (peripheral role) or `SPS_CLIENT_TX_FLOW_CTRL_IND` (central role) message to the `TASK_APP`.

6.11 Sequence diagrams

In the sequence diagrams of this section the function call sequence for transferring data received on UART interface to BLE and vice versa.

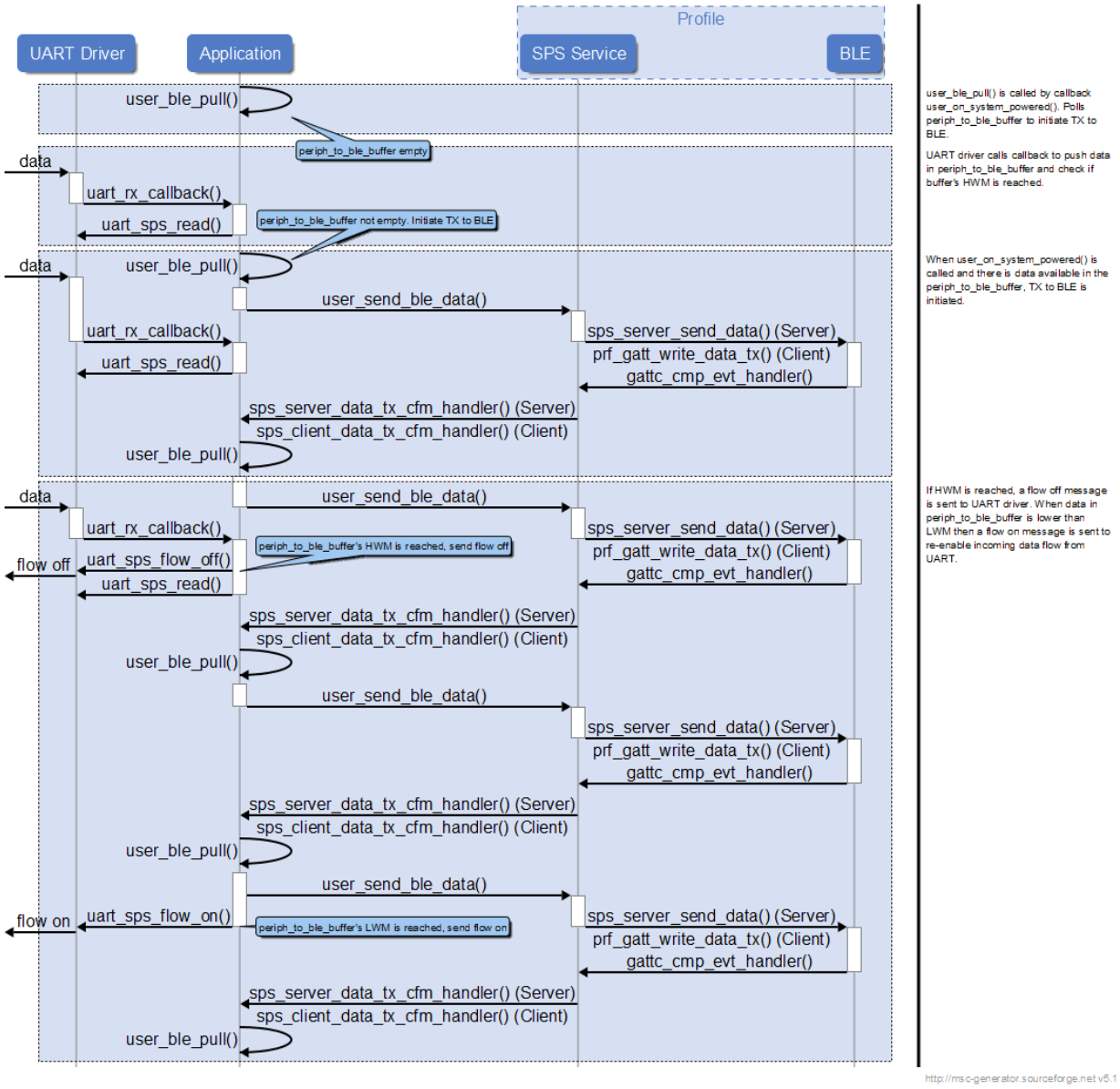

Figure 6: UART to BLE data transfer sequence diagram. Interrupt driven UART.

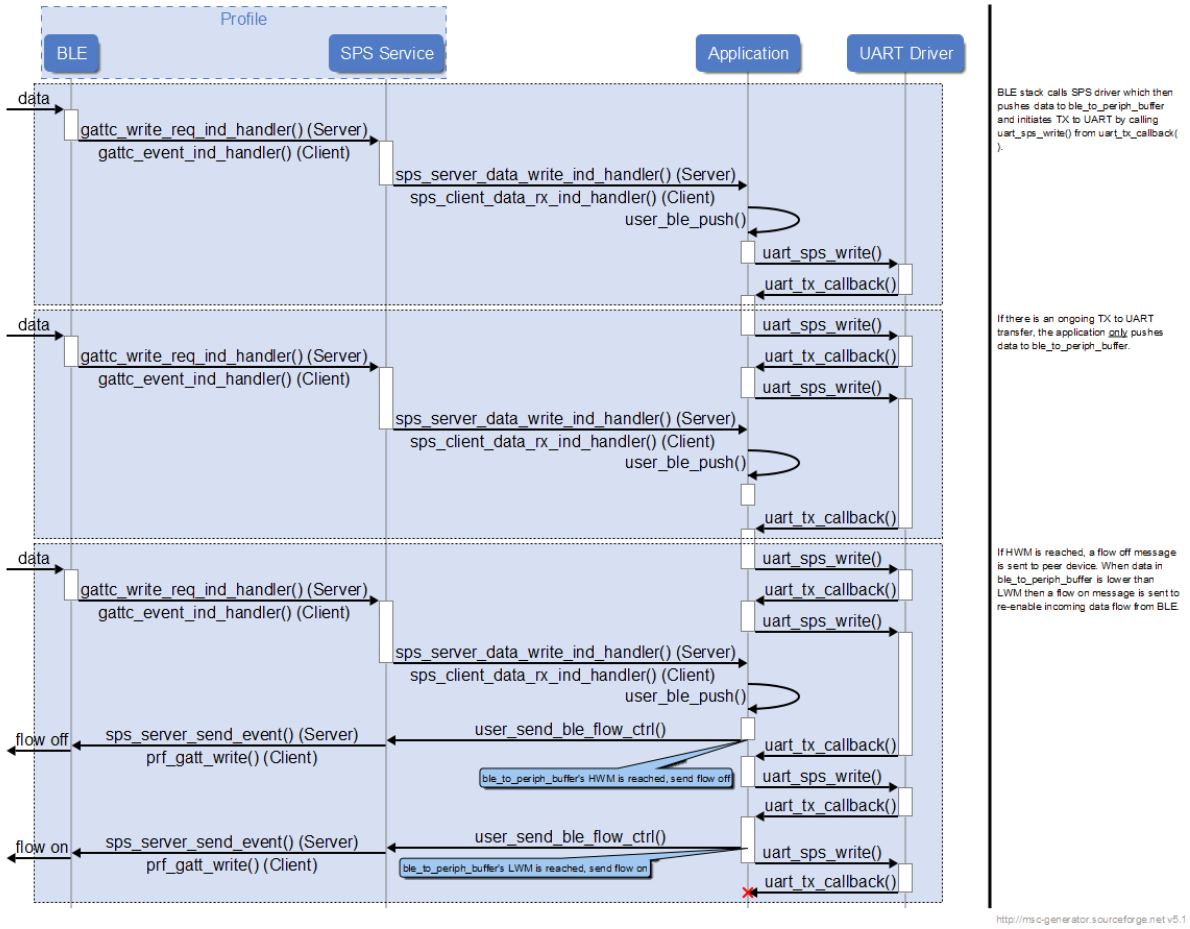

Figure 7: BLE to UART data transfer sequence diagram. Interrupt driven UART.

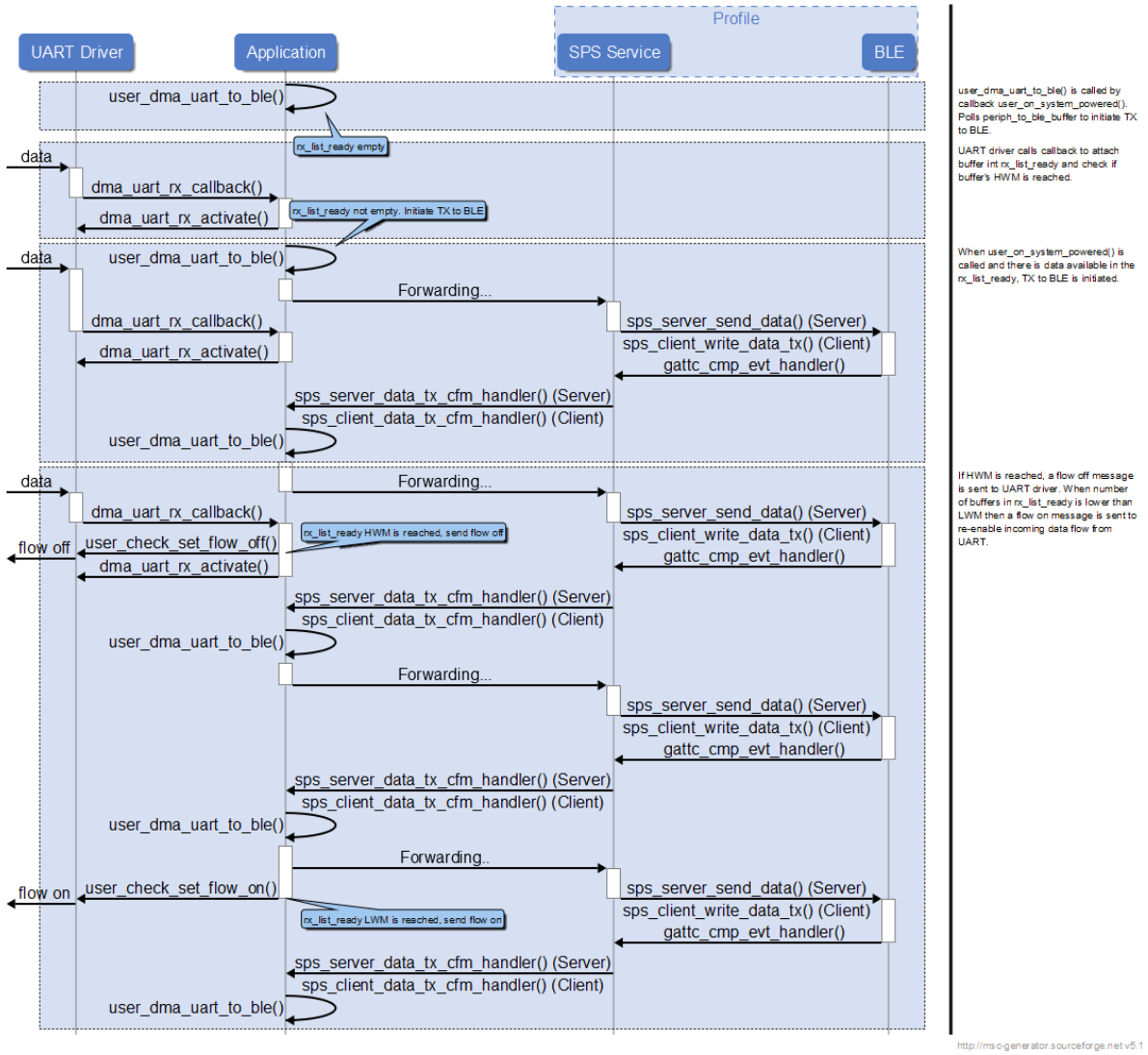

Figure 8: UART to BLE data transfer sequence diagram. DMA driven UART.

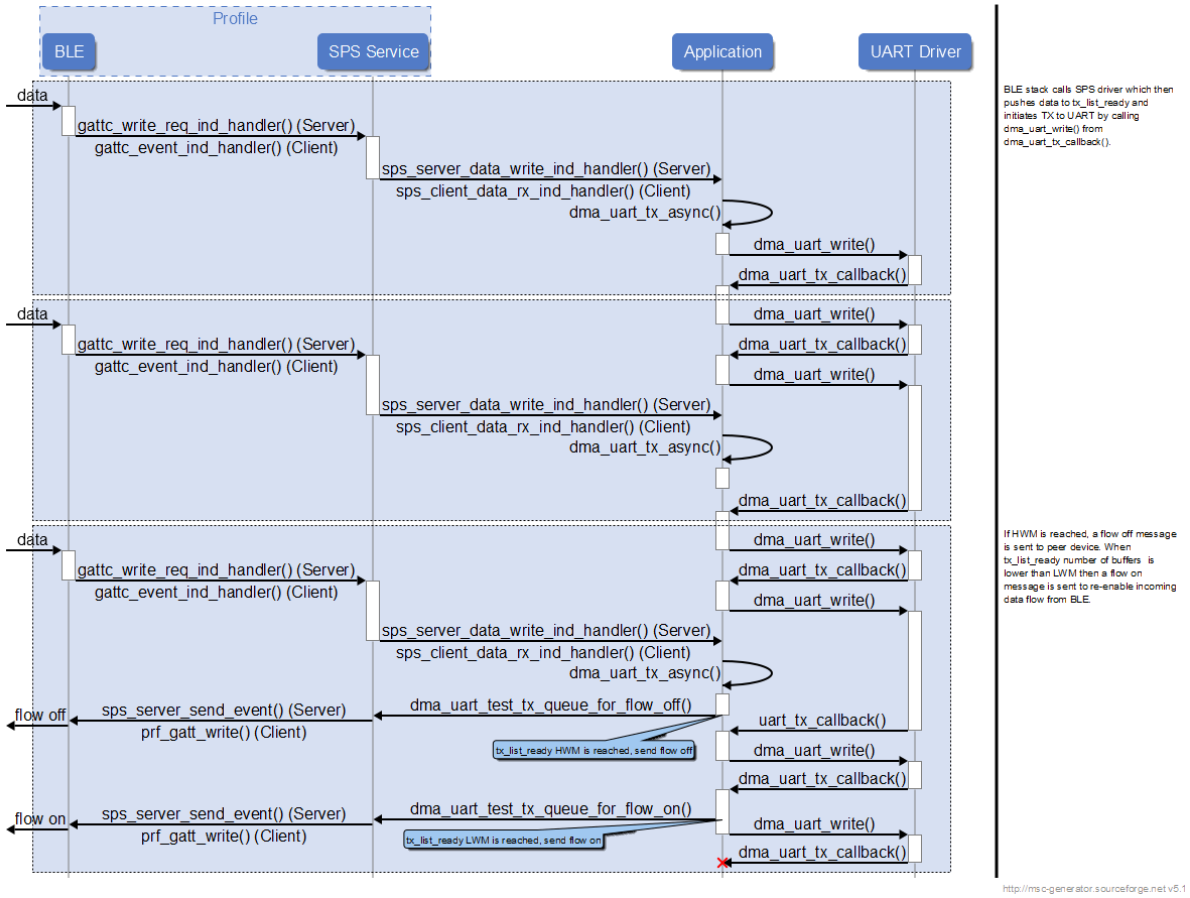

Figure 9: BLE to UART data transfer sequence diagram. DMA driven UART.

7 DSPS Android and iOS application

7.1 Overview

The Dialog Serial Port Service (DSPS) reference application comes with an SPS demo application for Android and iOS systems. The DSPS 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
- Reception of data (ASCII or HEX) from a peer device with flow control
- Transmission of data to a peer device, either once or repeatedly
- Transmission of file to a peer device with flow control

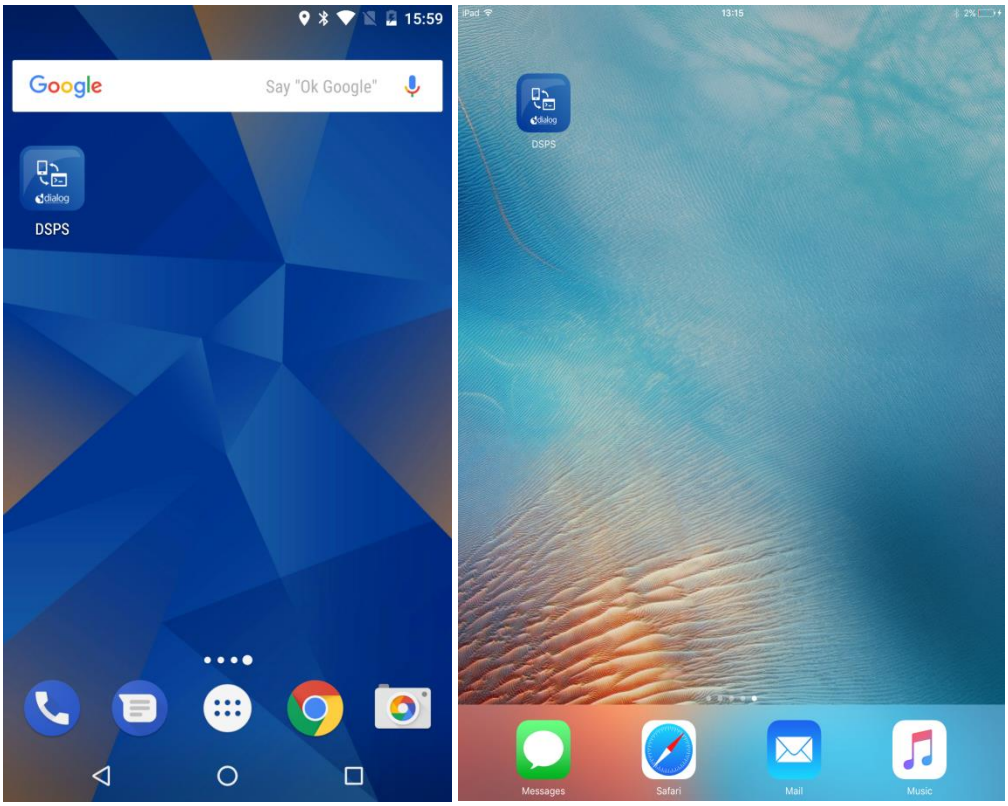

Figure 10: DSPS icon – Android – iOS

7.2 Installation

7.2.1 Android application

The Android application can be found in Google's 'Play Store' and easily installed from there as any other android application. To find the application, user can search for 'Dialog DSPS' (Figure 11). Then user has to open the applications description and press the 'Install' button. During installation the permission to access device resources will be requested as shown in Figure 12. The permission should be granted. After installation is complete, application can be open either from the 'Open' button of the 'Play store' or from the Android application drawer.

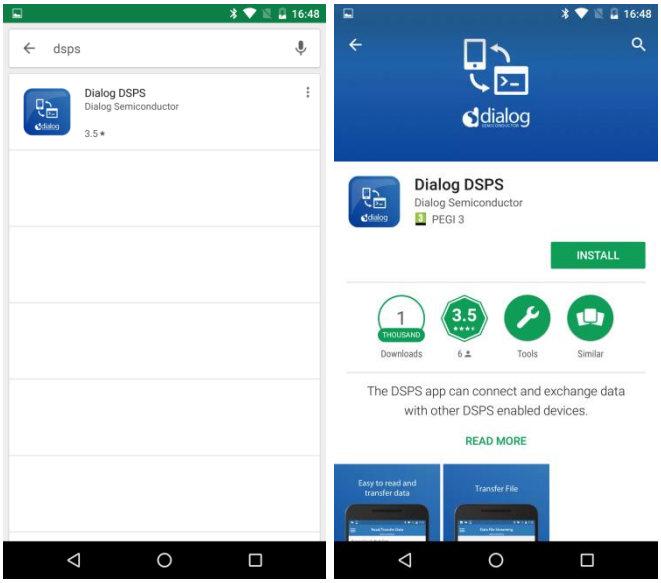

Figure 11: DSPS on Play Store

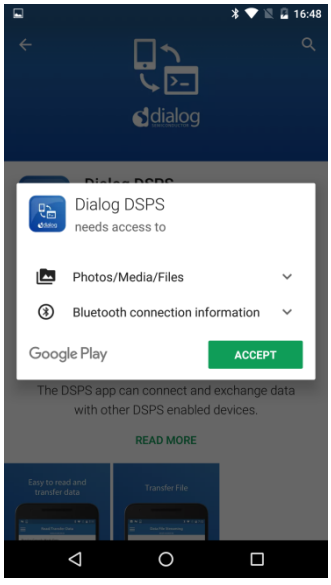

Figure 12: Installing the DSPS application

7.2.2 iOS application

The iOS application can be found in Apple's 'App Store' and easily installed from there like any other iOS application. To find the application, the user can search for 'DSPS' or 'Dialog DSPS'. Then the user has to open the applications description and press the cloud button (Figure 13) to download and

install it. After installation is complete, the application can be opened either using the ‘Open’ button of the ‘App store’ or from the iOS desktop.

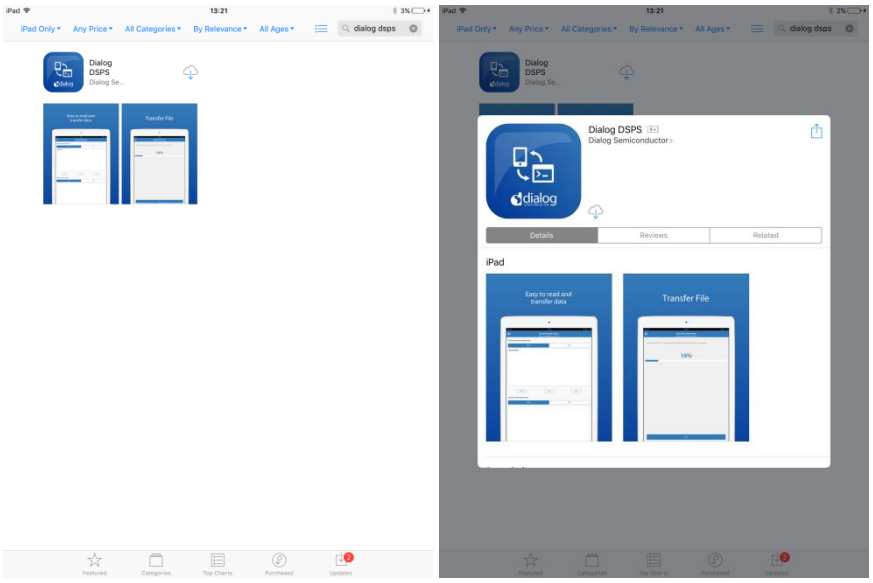

Figure 13: DSPS on App Store

7.3 Device list

The user must click on the DSPS icon to start the application. The scan screen will be shown (Figure 14) and the user has to press the SCAN button. While scanning, all devices found are displayed (device name and Bluetooth address). The complete list of all discovered devices is listed on the screen when the search has finished. To refresh the list of devices found, the user can press the button SCAN again. The user must click on the name of the desired device to connect to it.

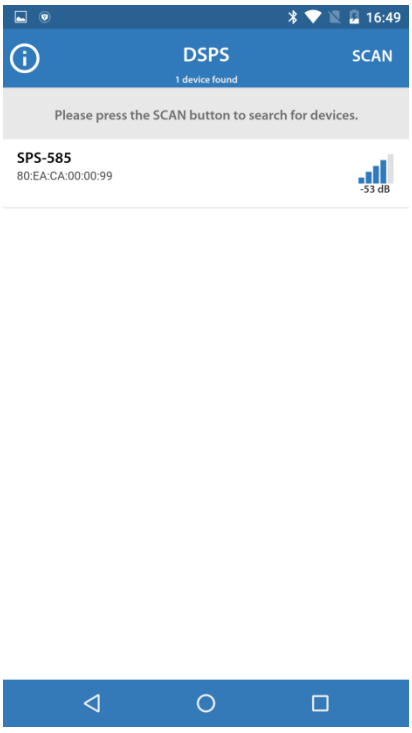

Figure 14: Scan Device Screen

7.4 Functionality tabs

When connected to a device, a side menu environment is shown to the user when the List Menu button is pressed, providing 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.

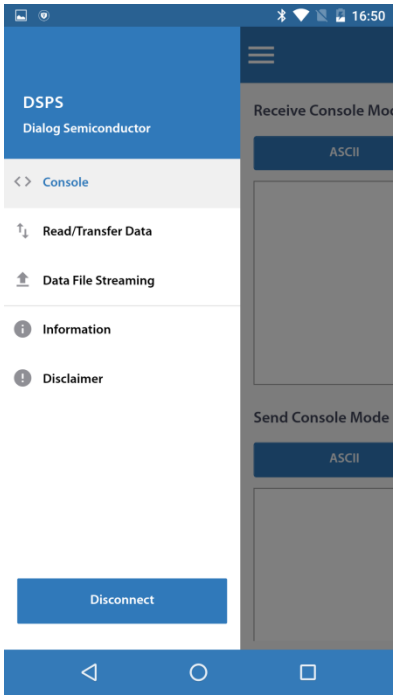

Figure 15: Select Operation Mode

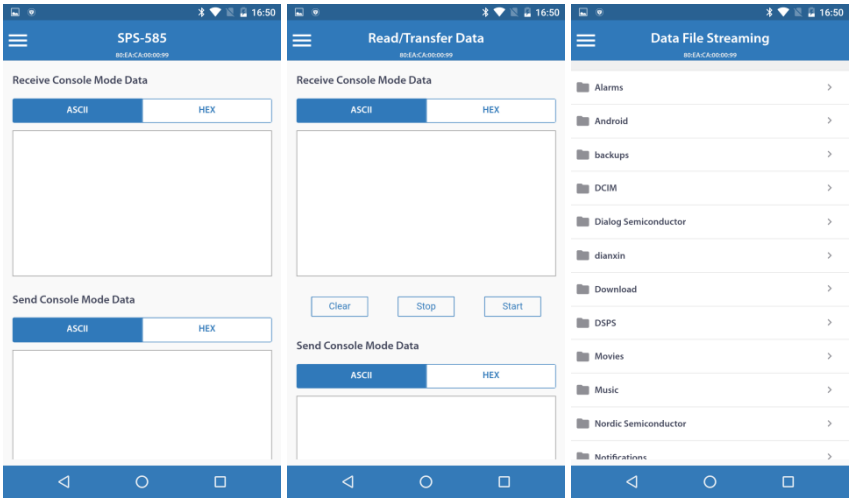

Figure 16: Application main screens

7.4.1 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.

7.4.2 Read/Transfer Mode

The Read/Transfer mode allows the user to send data to and receive data from the peer device. Initially, the reception 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 pressing the ‘Start’ or ‘Stop’ button respectively.

For the TX part, a text box is shown where data can be entered via the keyboard or pasted by long pressing the field and selecting ‘paste’. Pressing the ‘Send’ button will send the displayed data once or repeatedly (depending on the state of the ‘cyclic sending’ toggle switch) with the chosen interval.

7.4.3 Data File Streaming Mode

The user has also 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.

7.5 Information and Disclaimer Screens

The ‘Information’ screen displays contact information and the version of the application. See Figure 17. The Disclaimer screen contains the disclaimer notes.

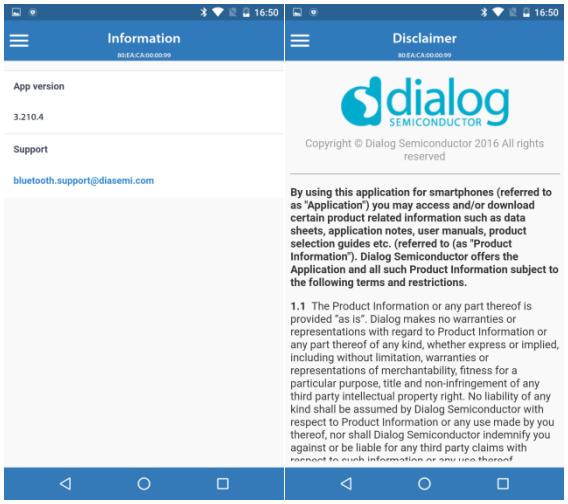

Figure 17: Information & Disclaimer Screens

8 Instructions for setting up a demonstration

The DA14585 pins that are used for serial communication are defined in `user_periph_setup.h` and may be changed according to user needs. If the user needs to get printed additional debug information from UART2 (`CFG_PRINTF` is defined in `da1458x_config_basic.h`) then the pins 0.1 (TX) and 0.2(RX) exposed to connectors J4 and J5 of Basic and Pro DK's respectively can be utilized by connecting a USB to Serial converter.

8.1 Hardware setup for Basic DK

In order to setup the Basic DK board there is an extra step that needs to be followed. Due to the lack of any flow control method on the Segger J-Link driver an external Serial to USB converter can be used, The instructions to use the FTDI TTL-232R TTL to USB Serial Converter cable [6] are described in this section. Connect the TTL-232R cable to Basic DK with the layout as shown Figure 18.

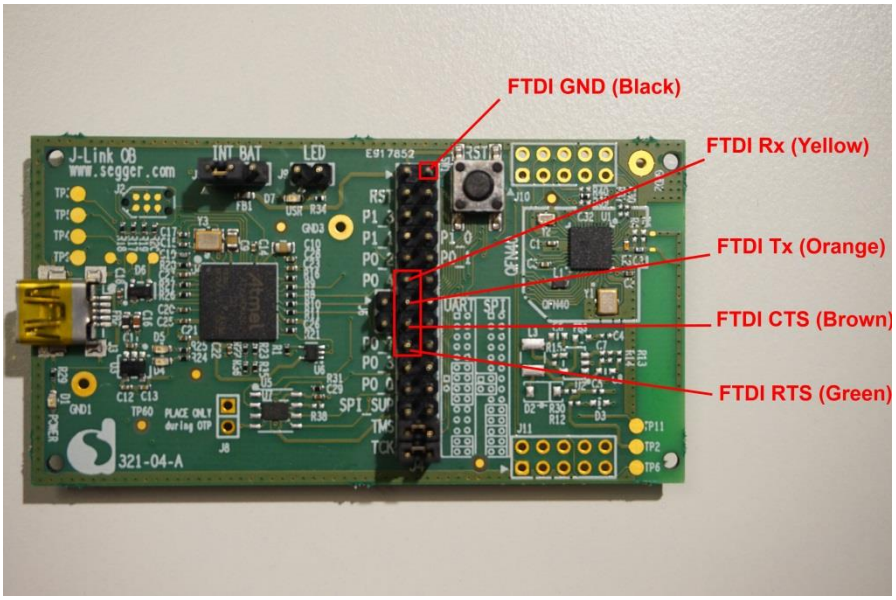

Figure 18: Basic DK pins that should be connected

Connect the:

- FTDI GND (black) to J4_2 (GND)
- FTDI TX (orange) to J4_13 (P0_5 - DA14585 RX)
- FTDI RX (yellow) to J4_11(P0_4 - DA14585 TX)
- FTDI CTS (brown) to J4_15 (P0_6 - DA14585 RTS)
- FTDI RTS (green) to J4_17(P0_7 - DA14585 CTS)

8.2 Hardware setup for PRO DK

The Pro DK board has an integrated FTDI chip with hardware flow control enabled. In order to use the feature the connections for J5 are indicated on the silkscreen next to J5 and should be applied as shown in Figure 19. For more information about the PRO DK, please refer to [5], section 3.7.2.

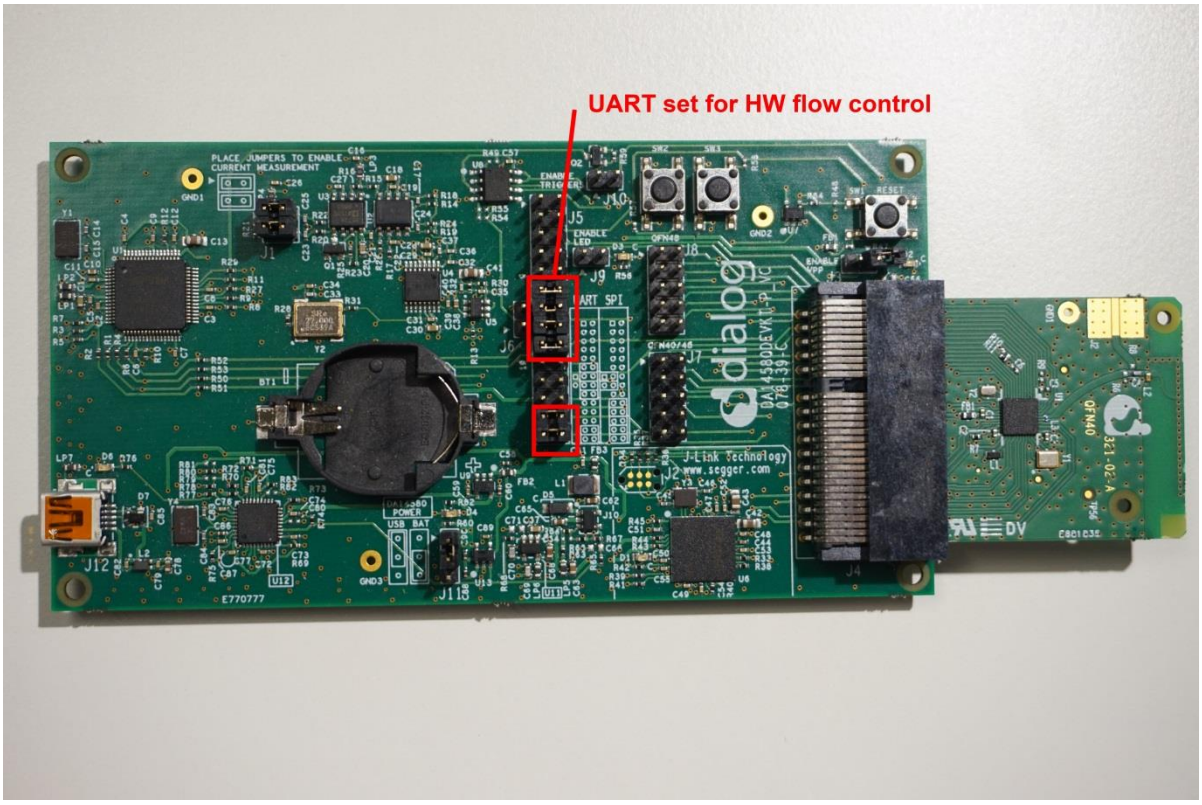

Figure 19: UART with HW flow control on PRO kit

8.3 DK to DK connection setup

1. Extract the DSPS reference application zip file.
2. Open `projects\target_apps\dsps\dsps_device\Keil_5\dsps_device.uvprojx` with Keil UVision.
3. Build project by pressing ‘F7’.
4. Select target device by pressing ‘Alt + F7’ then ‘Settings’ from the ‘Debug’ tab. Select appropriate SN and then ‘SW’ in the ‘Port’ field. Press ‘Ok’ to save and exit settings.
5. Start debugging session by pressing ‘Ctrl + F5’, run program with ‘F5’ and exit debugger mode by pressing ‘Ctrl + F5’ again.
6. Repeat the steps above for a second device with `…\projects\target_apps\dsps\dsps_host\Keil_5\dsps_host.uvprojx`, this time selecting the second device’s SN. Upon completion, the host device should have discovered and connected to the peripheral.
7. Any serial console application (e.g. RealTerm or Tera Term) can be used to transfer data to and from the DKs. Connect to the first of the two FTDI serial ports having set the correct settings:
   a. When TeraTerm is used on initialization the correct COM port should be chosen. If more than one com ports are active then the Windows Device Manager can be used to find the correct port numbers.
   b. From ‘Setup->Serial Port’ the COM port can be changed and set ‘Baud rate’ and ‘Flow control’ to the selected flow control method and baud rate as chosen in the `user_periph_setup.h` header file. In case of using the Basic DK, select the COM port of TTL-232R instead of the Segger’s COM port.
   c. Finally, press ‘Ok’ to accept the values and to continue.
8. Repeat the above steps for the second device. Upon completion, data can be transferred bidirectionally by either sending single characters or by using the ‘File->Send file’ function.

8.4 Android / iOS application to DK connection setup

1. As described in the steps to set up a DK to DK connection (see Section 8.3), extract, build and download the `dsps_device.uvprojx` project to the DK, using the debugger.
2. Open the UART connection from a console application, having set the correct parameters.
3. Install and open the '**DSPS**' Android / iOS application, as described in the application's user guide.
4. Scan for and connect to the device named '**DA1458x**'.

Upon connection, the transmission and reception of characters can be performed from the console and the application.

8.5 Run DA14585 application with SmartSnippets

1. Build DSPS project using the instructions above (Section 8.3, up to step 3).
2. Open SmartSnippets and connect to the device using UART/SPI mode or UART mode.
3. Open the 'Booter' tab and click 'Browse'
4. Using the dialog box find the produced *.hex file in directory `…\projects\target_apps\dsps\dsps_device\Keil_5\out_585\` for the peripheral device and `…\projects\target_apps\dsps\dsps_host\Keil_5\out_585\` for the central device.
5. Open it and then click 'Download' to download firmware to the connected device following the instructions in the 'Log' area.
6. Then repeat steps for secondary device
7. Close SmartSnippets and start a serial console application for each device as described previously.

9 SPS performance

The SPS performance analysis gives the following results regarding the service's throughput. The results depend heavily on the existence of DLE capability, the MTU and connection interval negotiated during the connection phase. The Android application's connect event length depends on the device running the application and thus the number of packets per connection event may differ substantially, giving different results.

Two groups of tests were performed, one group with DLE supporting devices and one group with DLE not supporting devices. For DLE the Android device used for the test is an LG Nexus phone with Android version 6.0.1 with 123 bytes payload size. The iOS device is an iPhone 7 Plus v11.0 with 251 bytes payload size. The devices not supporting DLE are an Android 4.4.4 ASUS Nexus tablet and an iPad with iOS 10.3.3. The UART baud rate 115200 bit/s was used for all performance measurements of the interrupt driven project. The UART baud rate 921600 bit/s was used for all performance measurements of the DMA driven project. When lower baud rates are used, the maximum throughput is limited by the selected baud rate.

Table 11: Performance results, DLE Devices. Interrupt driven projects.

Host			Android	iOS	DA14585
Connection parameters	MTU	octets	>130	>130	>130
	Connection Interval	ms	12.5	30	12.5
	Host max. write command size	B	128	128	231
Throughput measurements	Half Duplex				
	Central Tx	packets/conn event	2	4	2
		kB/s	11.0	11.6	11.2
		kbit/s	88.0	92.8	89.6
	Peripheral Tx	packets/conn event	2	5	2
		kB/s	11.5	10.0	11.2
		kbit/s	92.0	80.3	89.6
	Full Duplex				
	Central Tx	packets/conn event	3	2	3
		kB/s	10.9	7.1	11.2
		kbit/s	87.4	56.8	89.6
	Peripheral Tx	packets/conn event	3	2	3
		kB/s	11.5	9.5	11.2
		kbit/s	92.0	76.0	89.6

Table 12: Performance results, non DLE Devices. Interrupt driven projects.

Host			Android	iOS
Connection parameters	MTU	octets	>130	>130
	Connection Interval	ms	12.5	30
	Host max. write command size	B	128	128
Throughput measurements	Half Duplex			
	Central Tx	packets/conn event	5	4
		kB/s	7.7	5.9
		kbit/s	61.6	47.2
	Peripheral Tx	packets/conn event	5	5
		kB/s	10.0	5.9
		kbit/s	80.0	47.2
	Full Duplex			
	Central Tx	packets/conn event	5	2
		kB/s	6.25	4.6
		kbit/s	50.0	36.8
	Peripheral Tx	packets/conn event	5	2
		kB/s	9.8	5.9
		kbit/s	78.4	47.2

Figure 20: Performance results, DLE Devices. DMA driven projects.

Host			Android	iOS	DA14585
Connection parameters	**MTU**	octets	247	>130	>130
	Connection Interval	ms	30	30	30
	Host max. write command size	B	128	128	244
Throughput measurements	**Half Duplex**				
	Central Tx	packets/conn event	6	3	10
		kB/s	16.1	11.8	80.3
		kbit/s	129.0	94.3	642.4
	Peripheral Tx	packets/conn event	26	3	10
		kB/s	33.8	21.3	80.3
		kbit/s	270.3	170.2	642.4
	Full Duplex				
	Central Tx	packets/conn event	16	2	6
		kB/s	15.9	6.3	46.1
		kbit/s	127.2	50.8	368.7
	Peripheral Tx	packets/conn event	16	2	6
		kB/s	23.5	13.9	46.1
		kbit/s	188	111.1	368.7

Revision history

Revision	Date	Description
1.0	24-Nov-2017	Initial version.
1.1	25-Jan-2022	Updated logo, disclaimer, copyright.

Status definitions

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

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.