User manual

DA14580 Serial Port Service reference application

UM-B-038

Abstract

This document describes the architecture and the implementation details of the Dialog Bluetooth® Smart Serial Port Service application running on the DA14580 development kit and on Android or iOS.

Contents

Figures

Tables

1 Terms and definitions

BLE	Bluetooth Low Energy (Bluetooth Smart)
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-015, DA14580 Software Architecture, User manual, Dialog Semiconductor.

[2] UM-B-050, DA1458x Software development guide, User manual, Dialog Semiconductor.

[3] UM-B-051, DA1458x Software platform reference, User manual, Dialog Semiconductor.

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

[5] UM-B-025, DA14580 Bluetooth Smart Development Kit – Basic, User manual, Dialog Semiconductor.

[6] UM-B-034, DA14580 Bluetooth Smart Development Kit – Pro, User Manual, Dialog Semiconductor

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

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® Smart. The SPS software distribution (currently at version 5.150.2) includes the application and profile source codes. It is based on the 5.0.3.177 Software Development Kit.

Software has been developed for the DA14580 Development Kit – PRO and DA14580 Development Kit – Basic and for Android / iOS tablets or phones, allowing a serial port to be emulated using two DA14580 DKs or using a DA14580 DK and an Android / iOS device. The DA14580 DK can either function in the GAP central role or the peripheral role. The Android / 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.

To get familiar with the DA14580's software and hardware, the reader could also consult the DA14580 Software Development Guide [2], the Software Architecture [1], the Smart Development Kit – Basic user guide [5] and the Smart Development Kit – PRO user guide [6] documents.

The Android / iOS application, which is described in detail later in this document (see section 7), 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 / iOS device.

Figure 1: DA14580 Development Pro Kit

4 Software feature list

This section lists 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[1] in case of link loss

4.2 UART

- Hardware and Software[2] flow control are supported.
- Binary data transfer supported in hardware flow control mode
- UART baud rates: 115200, 57600, 38400, 19200, 9600

4.3 System

- iOS/ Android application.
- Extended sleep mode
- Transfer rates of from 40kbps up to 80kbps with 115200 baud rate.
- Memory Size of application image: 15 Kbytes

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

[2] Software flow control is partially supported in Extended sleep mode. Only incoming serial data can be controlled by DA14580 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 DA14580 until transmission stops.

5 Software architecture

5.1 Software features

5.1.1 GAP roles

The DA14580 SPS application supports both GAP central and peripheral roles.

In the DA14580 SPS software distribution the `\projects\target_apps\dsps\sps_device\Keil_5\sps_device.uvprojx` Keil project implements the peripheral device and the `\projects\target_apps\dsps\sps_host\Keil_5\sps_host.uvprojx` Keil project implements the central role device.

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

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 160-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 1.

Table 1: Device services/characteristics

Service/ Characteristic	UUID	Properties	Size (B)
SPS	0x0783b03e8535b5a07140a304d2495cb7	ATT_CHAR_PROP_RD	-
SPS_SERVER_TX	0x0783b03e8535b5a07140a304d2495cb8	ATT_CHAR_PROP_NTF	160
SPS_SERVER_RX	0x0783b03e8535b5a07140a304d2495cba	ATT_CHAR_PROP_WR_NO_RESP	160
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 DA14580 application task with the SPS server and client task.

Table 2: Server Service API messages

Message	Direction	Description
SPS_SERVER_CREATE_DB_REQ	IN	Creates the database of the service. Sent by the application at application initialisation.
SPS_SERVER_CREATE_DB_CFM	OUT	Confirmation of the database creation. Sent by the profile to the application at completion of the database creation.
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. 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 there is data available to transmit.

Message	Direction	Description
SPS_SERVER_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_SERVER_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_SERVER_RX_FLOW_CTRL_REQ	IN	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_SERVER_TX_FLOW_CTRL_IND	OUT	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 3: Client Service API messages

Message	Direction	Description
SPS_CLIENT_ENABLE_REQ	IN	Enables the client service. Sent by the application task at connection establishment.
SPS_CLIENT_ENABLE_CFM	OUT	Confirmation of the client service enable. Sent by the profile to the application at completion of client service enable task.
SPS_CLIENT_DISABLE_IND	OUT	Indication that the client service has been disabled.
SPS_CLIENT_DATA_TX_REQ	IN	Sends data to a peer device. 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	IN	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	OUT	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

The UART driver controls the UART peripheral module of the DA14580 and manages the serial connection between the DA14580 and the external host processor. It uses an interrupt based transmission and reception scheme requiring minimal 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. The selection of the UART baud rate is done by the setting the value of `UART_SPS_BAUDRATE` definition in the `uart_sps.h` header file to the required baud rate.

5.1.4 Data scheduling and flow control

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.

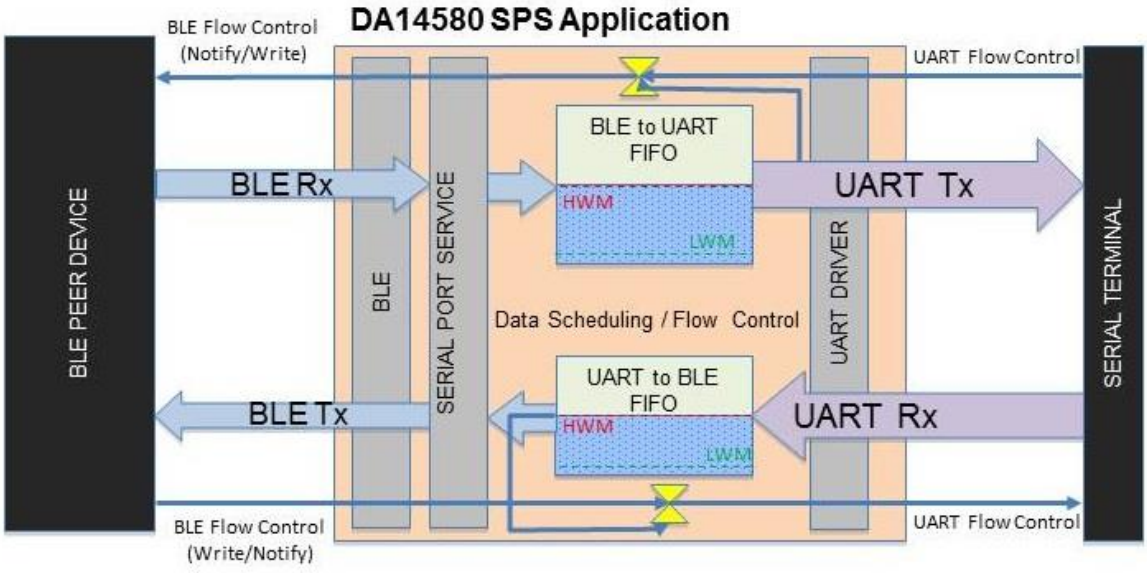

Figure 2: Data and flow control

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`. The scheduling of data transmission is performed 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` and when there is no data transfer ongoing it initiates a transfer. It schedules transmission of the data 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 subsequent 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. When the UART interface path is already busy, the check of pending data in the FIFO is performed again upon completion of the active UART transmit process.

The DA14580 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 to set the buffer utilisation level at which a flow off or flow on signal is sent to the corresponding interface. Flow control signalling on the BLE interface is also triggered when the corresponding flow control signal is received from the UART interface.

5.2 Source files

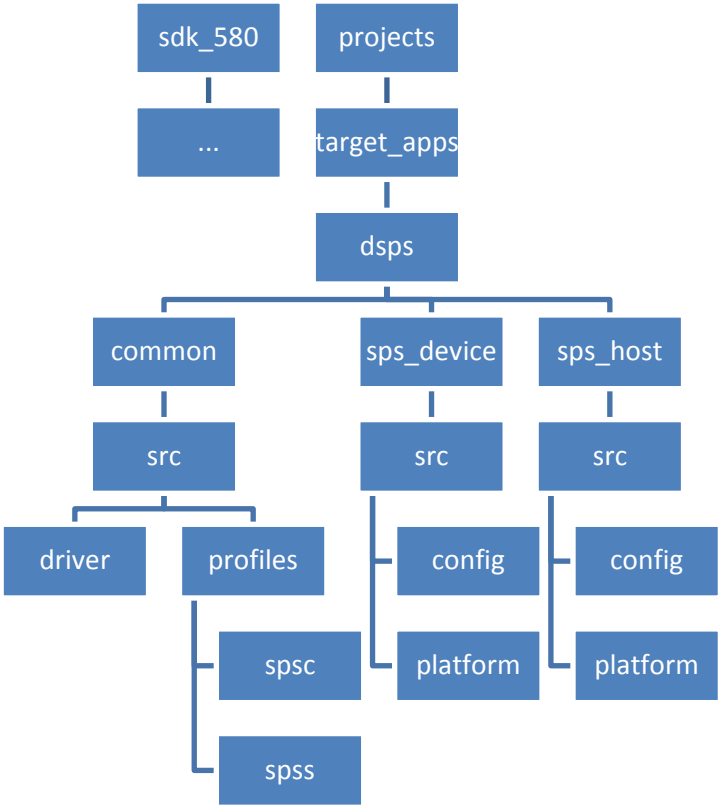

Figure 3: SRS application project overview

The main files of the SPS application project are listed in Table 4. As mentioned earlier, SDK version is 5.0.3.177 and the described DSPS reference version is 5.150.2.

Table 4: SPS application files[3]

File name	Description
.\sps_device\src\user_sps_device.c .\sps_host\src\user_sps_host.c	Application main functions and system callbacks.
.\sps_device\src\user_spss.c .\sps_device\src\user_spss_task.c	Application profile specific functions (Peripheral role).
.\sps_host\src\user_spsc.c .\sps_host\src\user_spsc_task.c	Application profile specific functions (Central role).
.\sps_device\src\platform\user_periph_setup.c .\sps_host\src\platform\user_periph_setup.c	Peripheral modules initialisation, GPIO pins assignment.
.\common\src\user_scheduler.c	Application level UART control, FIFOs management, application level data transfer
.\common\src\user_buffer.c	FIFO management functions.
.\common\src\driver\uart_sps.c	Custom UART driver with hardware/software 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.

[3] 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 5.

Table 5: Header files of the SPS application[4]

File name	Purpose
.\sps_device\src\config\da1458x_config_advanced.h .\sps_host\src\config\da1458x_config_advanced.h	Advanced compile configuration files.
.\sps_device\src\config\da1458x_config_basic.h .\sps_host\src\config\da1458x_config_basic.h	Basic compile configuration files.
.\sps_device\src\config\user_callback_config.h .\sps_host\src\config\user_callback_config.h	Callback functions configuration files.
.\sps_device\src\config\user_config.h .\sps_host\src\config\user_config.h	User configuration files.
.\sps_device\src\config\user_modules_config.h .\sps_host\src\config\user_modules_config.h	User modules configuration files.
.\sps_device\src\config\user_periph_setup.h .\sps_host\src\config\user_periph_setup.h	Peripherals setup header files.
.\sps_device\src\config\user_profiles_config.h .\sps_host\src\config\user_profiles_config.h	Configuration files for the profiles used in the application.
.\sps_device\src\user_sps_device.h .\sps_host\src\user_sps_host.h	Application main functions and system callbacks header files.
.\sps_device\src\user_spss.h .\sps_device\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	Application profile specific functions (Central role) header files.
.\common\src\user_scheduler.h	Scheduler header file.
.\common\src\user_buffer.h	FIFO management functions header file.
.\common\src\driver\uart_sps.h	Custom UART driver header file.
.\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.

[4] 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 6: Peripheral application task: FSM states

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

Figure 4 shows the FSM of the peripheral application.

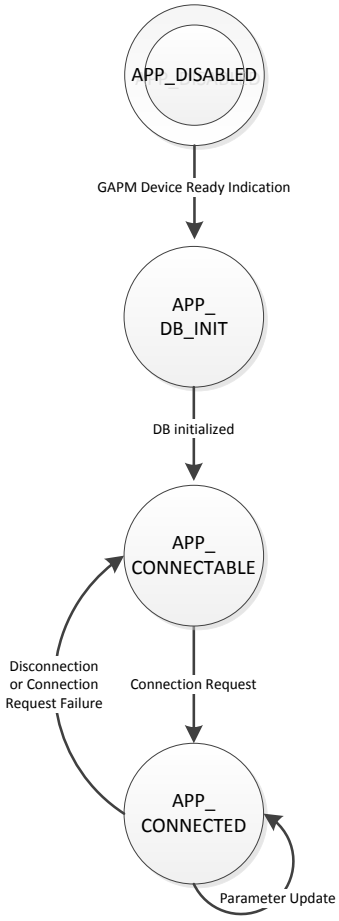

Figure 4: Peripheral application FSM

Table 7: 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 initialisation of profile.
APP_DB_INIT	Database initialisation 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 8: Central application task: FSM states

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

Figure 5 shows the FSM of the central application.

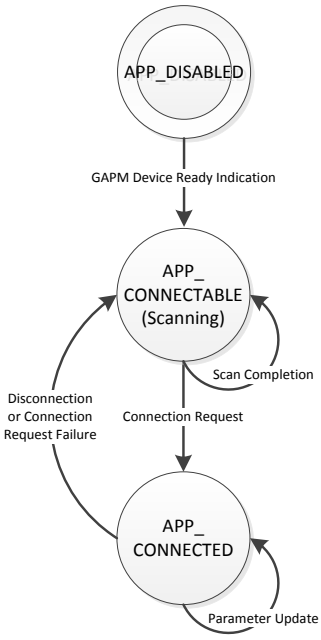

Figure 5: Central application FSM

Table 9: 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() [3]). It also initialises an MTU exchange procedure and a parameter update. The central application disables connection timer, enables profiles and services and initialises 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() [3]). 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 [4].

6.2.6 user_on_connect_failed()

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

6.3 Main loop callback functions

6.3.1 user_on_init()

This function is called once, during system initialisation. In both GAP roles, it allocates the cyclic buffers, it registers UART callbacks and sets the UART flow control state. It also sets the configured sleep mode.

6.3.2 user_on_sytem_powered()

This function is called periodically upon scheduling pending BLE events after device wake up. 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.

6.3.3 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 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 described above (sections 6.2.1, 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_main_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 gapc_param_update_req_ind_handler() (Central only)

This function is called by the main application task handler and it allocates and sends a parameter update confirmation message (GAPC_PARAM_UPDATE_CFM) to decline the peer requested parameter update.

6.5 Data management and flow (user_sps_scheduler)

6.5.1 user_scheduler_init()

This function is called by user_on_init() (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 6.3.2) to initiate a BLE transmission and by the user_sps_server_data_tx_cfm_handler() (section 6.7.7) (peripheral role) or the user_sps_client_data_tx_cfm_handler() (section 6.7.7) (central role) to continue transmission. It is also called by the user_sps_server_enable_cfm_handler() (section 6.7.6) (peripheral role) or the user_sps_client_enable_cfm_handler() (section 6.7.6) (central role) to initialise TX state.

- When called by user_on_system_powered(), it checks if a transmission is ongoing by checking the tx_busy_flag. If 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 available data in cyclic buffer is more than TX_WAIT_LEVEL. If it is, it sends the available data to the BLE interface, 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 maximise 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. This is a trade-off between throughput and responsiveness of the application which depends on the values of these parameters. TX_WAIT_LEVEL should be chosen to give an integer number of packets and 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 6.7.8) (peripheral role) or the user_sps_client_data_rx_ind_handler() (section 6.7.8) (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 6.5.11) or `user_override_ble_xoff()` (section 6.5.10) to enable or disable the BLE incoming data flow.

6.5.10 user_override_ble_xoff()

This function sets the UART flow control flags and sends a flow off signal to the connected BLE device. It overrides the automatic buffer level checking and is called by the UART flow control callback (section 6.5.9).

6.5.11 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).

6.5.12 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 6.5.13) in a series of buffer and FIFO checks.

6.5.13 user_sps_sleep_check()

This function is performs 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.

6.5.14 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 6.5.13) disables data flow before entering sleep state.

6.6 Cyclic buffer (user_buffer)

6.6.1 user_buffer_create()

This function allocates and initialises 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, 6.5.13) 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 Application task interface to Serial Port Service

6.7.1 user_sps_db_create() (Peripheral only)

This function creates and sends an `SPS_SERVER_CREATE_DB_REQ` message to the server profile (`TASK_SPS_SERVER`) to allocate and initialise the database. It is registered as a callback in the `user_prf_funcs` struct and it is called by the SDK.

6.7.2 user_sps_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_SPS_SERVER` or `TASK_SPS_CLIENT` respectfully. It is registered as a callback in the `user_prf_funcs` struct and it is called by the SDK to enable the service.

6.7.3 user_send_ble_data()

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_SPS_SERVER or TASK_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.

6.7.4 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_SPS_SERVER or TASK_SPS_CLIENT respectfully, in order to transmit the selected flow control state to the peer device.

6.7.5 user_sps_create_db_cfm_handler() (Peripheral only)

This function handles the SPS_SERVER_CREATE_DB_CFM message which confirms that database has been successfully created. It allocates and sends an APP_MODULE_INIT_CMP_EVT message to the TASK_APP to allow database creation to be completed.

6.7.6 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() (section 6.5.3) function to initialise the TX state.

6.7.7 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. It calls user_ble_pull() (section 6.5.3) with proper arguments.

6.7.8 user_sps_server_data_rx_ind_handler() and user_sps_client_data_rx_ind_handler()

These functions handle the SPS_SERVER_DATA_RX_IND (peripheral role) or the SPS_CLIENT_DATA_RX_IND (central role) message which indicates that data have been received by the BLE interface. It calls user_ble_push() (section 6.5.4) to pass the data that were received.

6.7.9 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.

6.7.10 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.

6.8 Serial Port Service

The Serial Port Service consists of two parts: a server and a client. It exposes three characteristics as displayed in Table 1.

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`.

The profile sends data only if the TX flow control flag is enabled, otherwise it stores data 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.

Transmission occurs either by using the functions `attmdb_att_set_value()` and `prf_server_send_event()` at the server side or the `prf_gatt_write()` function at the client side.

A 'normalisation' of the size to be sent is proposed at the application level to increase throughput. Initially, the requested size is limited by the `TX_SIZE` and then it gets quantised to a value that fills an integer number of packets. 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_RX_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.

When a flow control signal is received by the GATTC (from the `gattc_write_cmd_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_CLIENT_TX_FLOW_CTRL_IND` (peripheral role) or `SPS_CLIENT_TX_FLOW_CTRL_IND` (central role) message to the `TASK_APP`.

6.9 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.

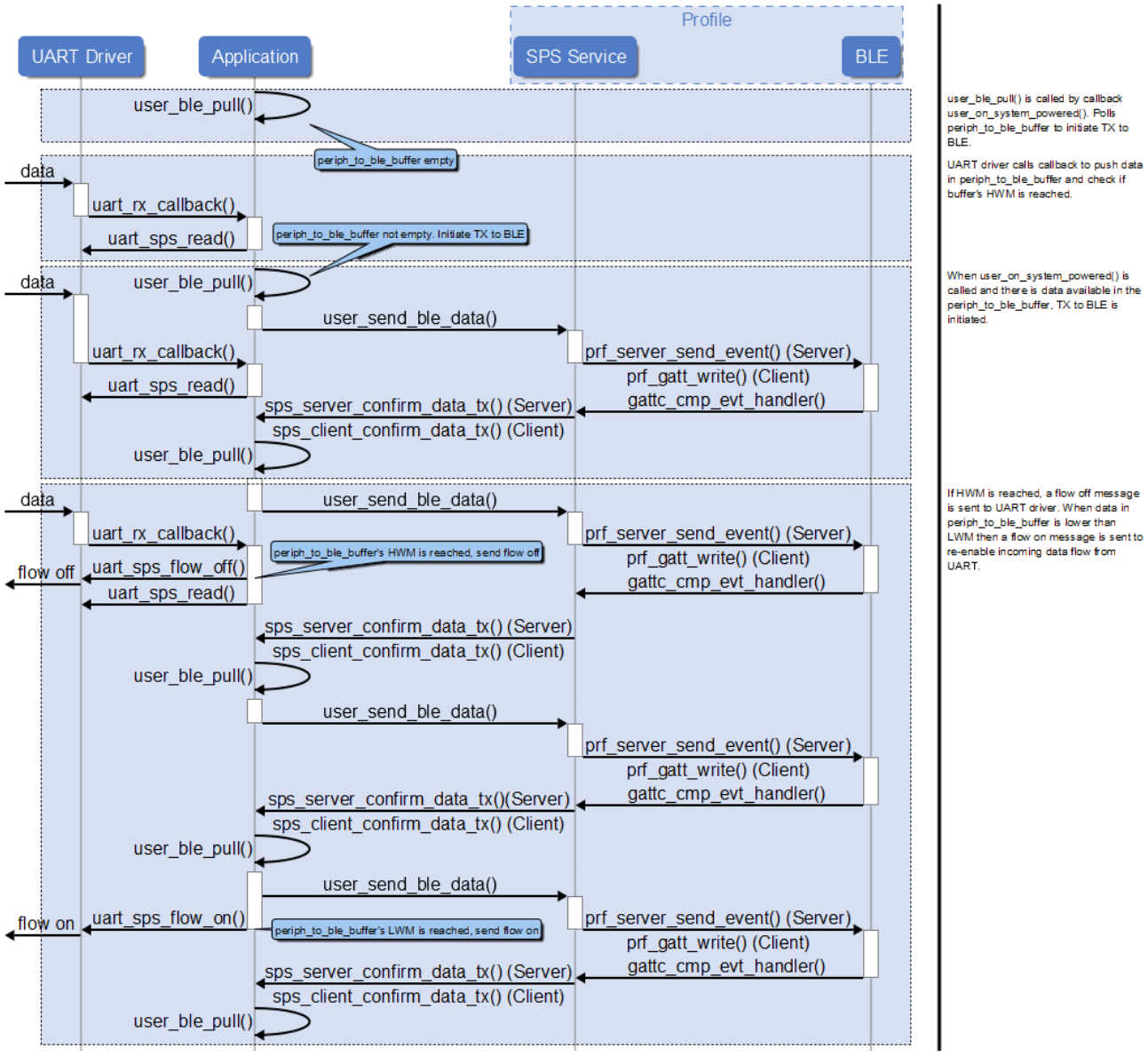

Figure 6: UART to BLE data transfer sequence diagram

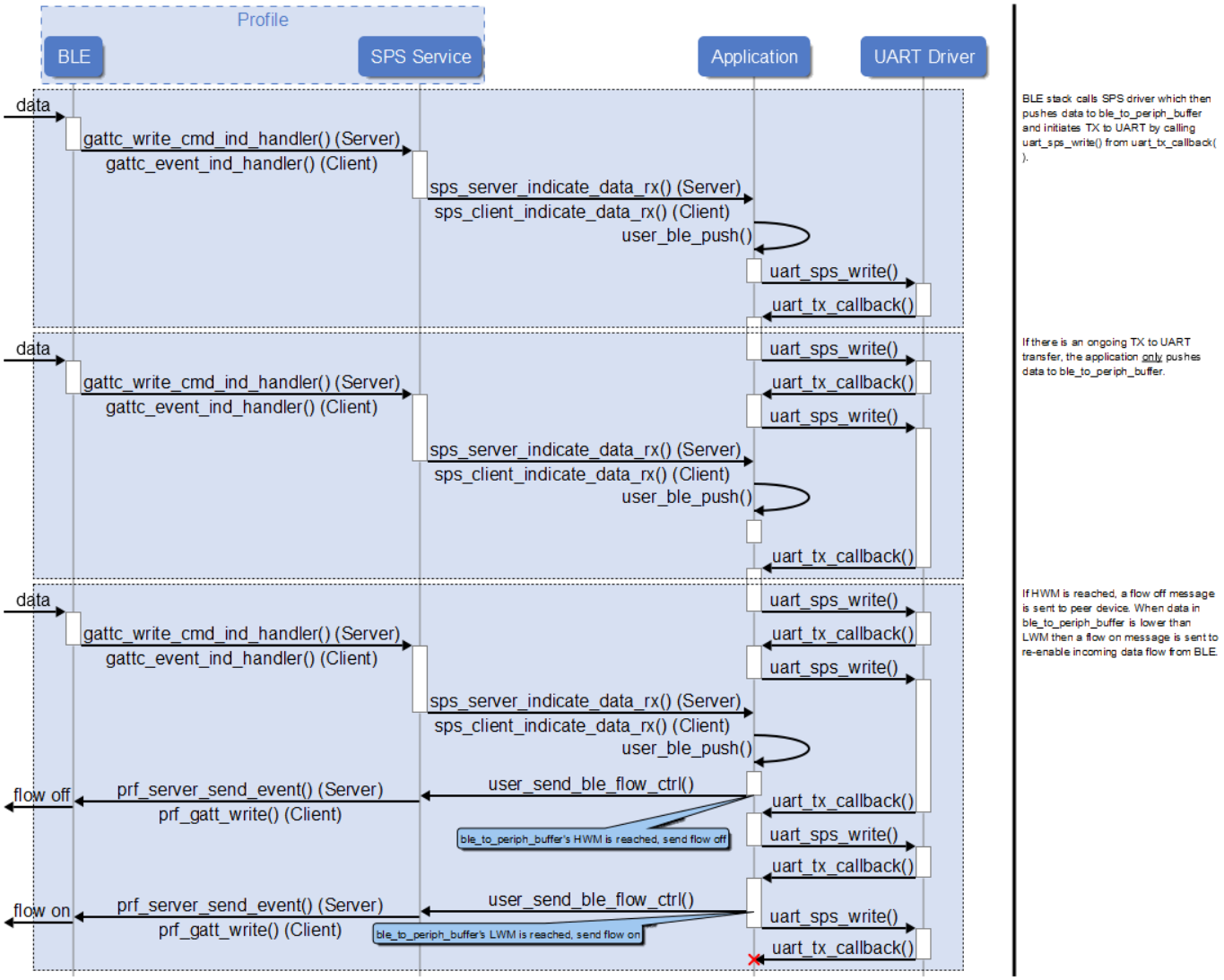

Figure 7: BLE to UART data transfer sequence diagram

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. The application has been tested with Android versions 4.4.2 to 5.0 and iOS 7.0 to 8.1.1.

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 8: DSPS Icon

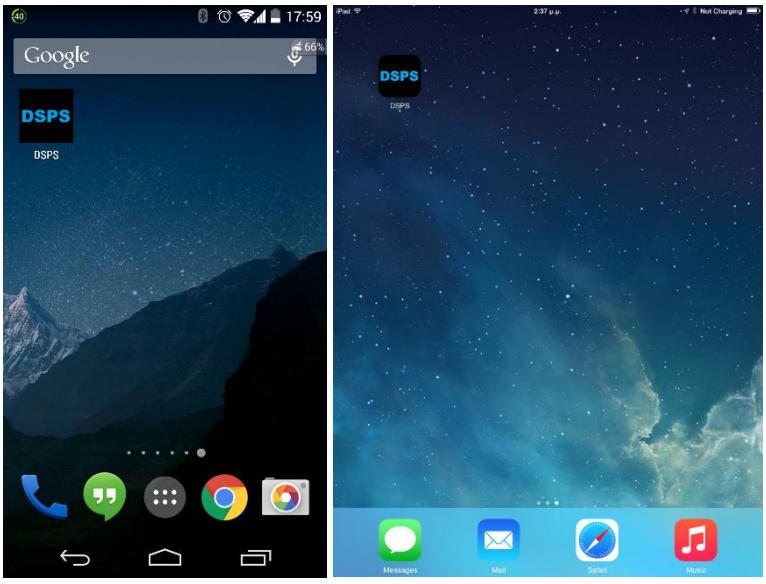

Figure 9: DSPS icon on Android – iOS

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 'DSPS Dialog'. Then user has to open the applications description and press the 'Install' button. After installation is complete, application can be open either from the 'Open' button of the 'Play store' or from the Android application drawer.

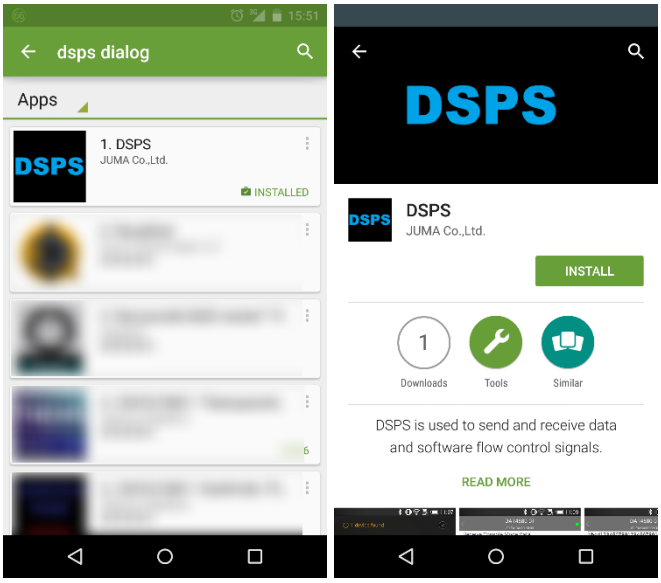

Figure 10: DSPS on Play Store

In case the user wants to install the application using the *.apk installer package file, the following steps must be executed:

1. Open a file browser and open the DSPS.apk installer package. The installer package file can be found under *binaries**host**android**dsps*\ directory of the release zip file.
2. When 'Installation of application from unknown sources' is not enabled, Android will inform the user that it must be enabled. The user must then click on the 'Settings' button and enable the 'Unknown Sources' option.
3. The Android application installer informs the user about the application privacy and device access requirements. When the user agrees, continue the installation by selecting 'Install'.
4. When the installation procedure has finished, a screen informs the user and provides the option to open the application.

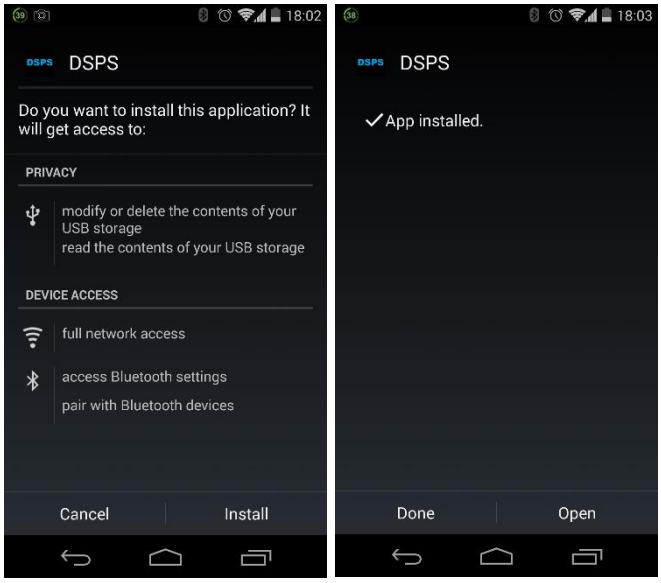

Figure 11: Installing the DSPS application

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 'DSPS Dialog'. Then the user has to open the applications description and press the cloud button 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.

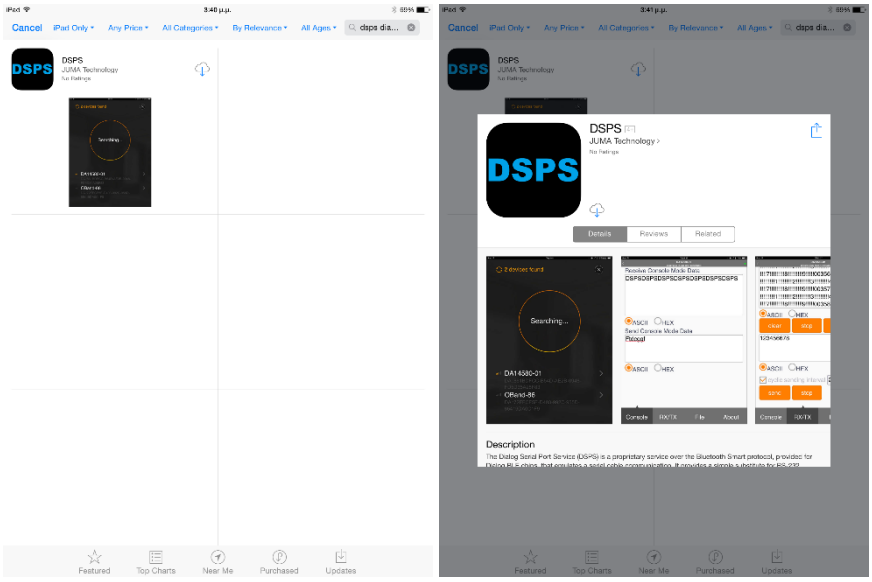

Figure 12: DSPS on App Store

In case the user wants to install the application using the *.ipa installer package file, the following steps must be executed:

1. Connect the iOS device to a PC with a USB cable.
2. Open the application file (.ipa) with iTunes and go to the connected device's tab. The installer package file can be found under *\binaries\host\iOS\dsps* directory of the release zip file.
3. Select 'Apps' from the 'Settings' list to show the DSPS application and press 'Install' to begin the installation procedure.

4. Next, the button changes to ‘Will Install’ and the user has to press the ‘Apply’ button to complete the installation.
5. After successful installation the ‘DSPS’ application icon will appear on the iOS desktop.

7.3 Device list

The user must click on the DSPS icon to start the application. While searching, 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 ‘search again’. The user must click on the name of the desired device to connect to it.

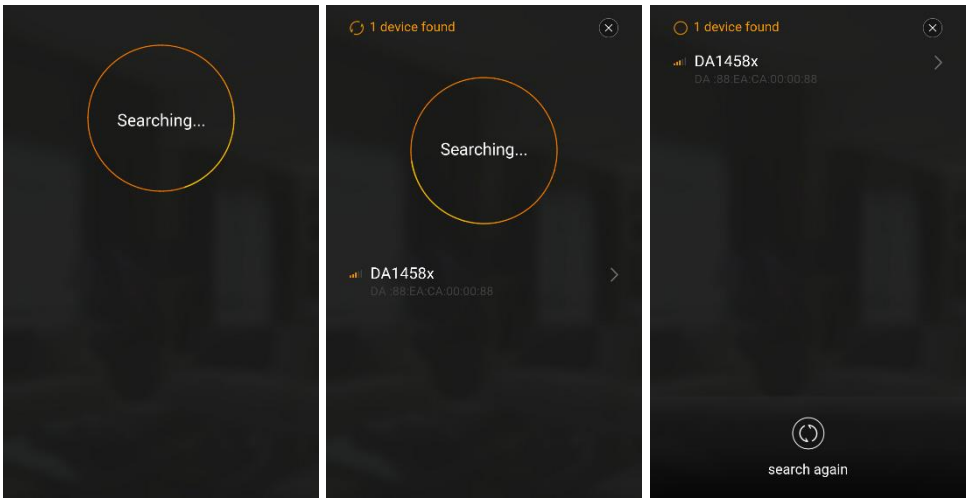

Figure 13: Search screen and Device list

7.4 Functionality tabs

When connected to a device, a tabbed environment is shown to the user, providing the functionality of the application. The top of the screen shows information about the currently connected device: such as the device name and the connection status, indicated by a green light that will turn grey when the device is disconnected.

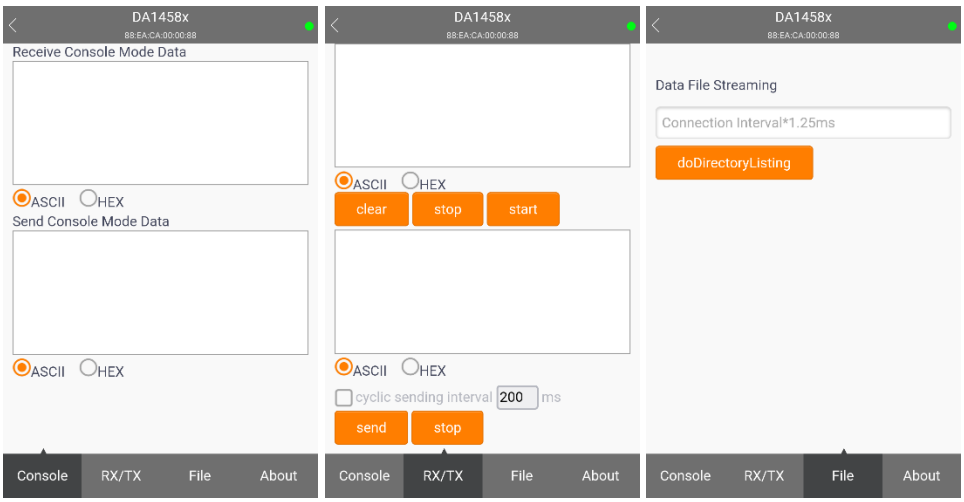

Figure 14: Application main screens

7.4.1 Console tab

In the Console tab similar controls can be found. In console 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 RX/TX tab

The RX/TX tab 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’ tick box) with the chosen interval.

7.4.3 File tab

The user has also the ability to send a file instead of individual strings of characters. This function can be accessed in the ‘File’ tab, where the user can browse the device’s file system to find the file to be transmitted. Before transmission, the connection interval has to be specified in the ‘interval’ field.

7.5 About tab

The ‘About’ tab displays contact information and the version of the application. See Figure 15.

Figure 15: About tab

8 Instructions for setting up a demonstration

8.1 Hardware setup for Basic DK

In order to setup the Basic DK board there are some extra steps that need to be followed. Due to the lack of any flow control method on the current 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 [7] are described in this section.

Connect the TTL-232R chip with the following layout:

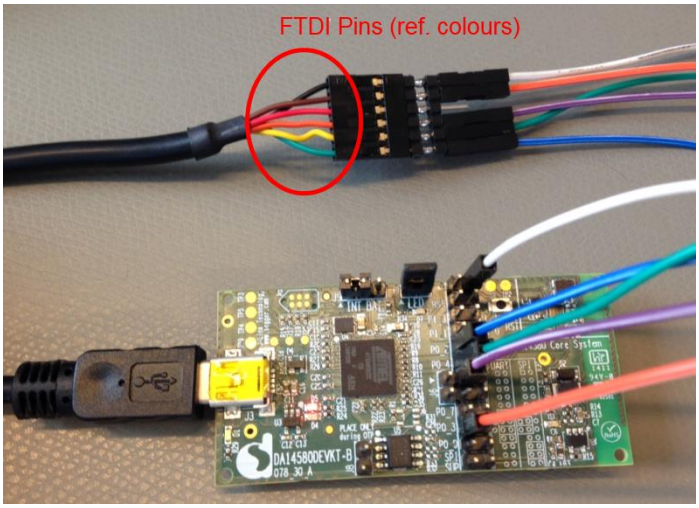

Figure 16: Basic DK connected to TTL-232R

Connect the:

- FTDI GND (black) to the J4_2 (GND)
- FTDI TX (orange) to the J4_13 (P0_5 - DA14580 RX)
- FTDI RX (yellow) to the J4_11 (P0_4 - DA14580 TX)
- FTDI CTS (brown) to the J4_19 (P0_3 - DA14580 RTS)
- FTDI RTS (green) to the J4_9 (P0_2 - DA14580 CTS)

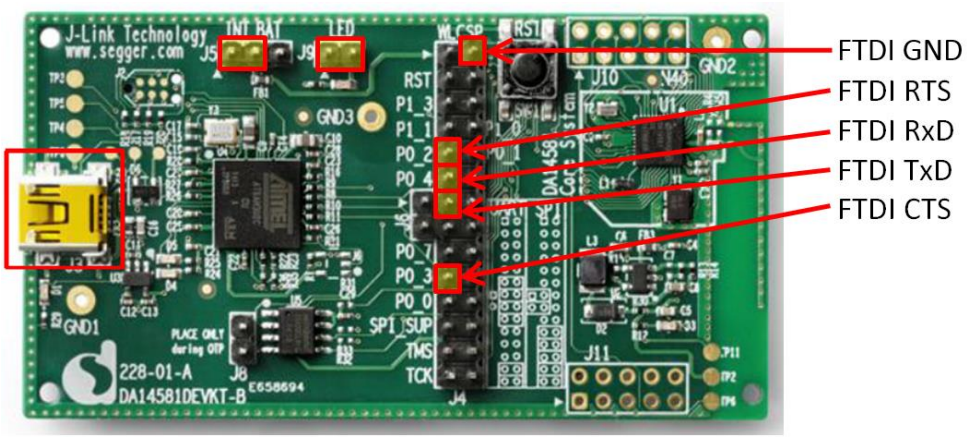

Figure 17: Basic DK pins that should be connected

8.2 Hardware setup for PRO DK

In order to setup the PRO DK board correctly to run the DSPS profile, the following connections must be done. For more information about the PRO DK, please refer to [6], section 3.3.2.

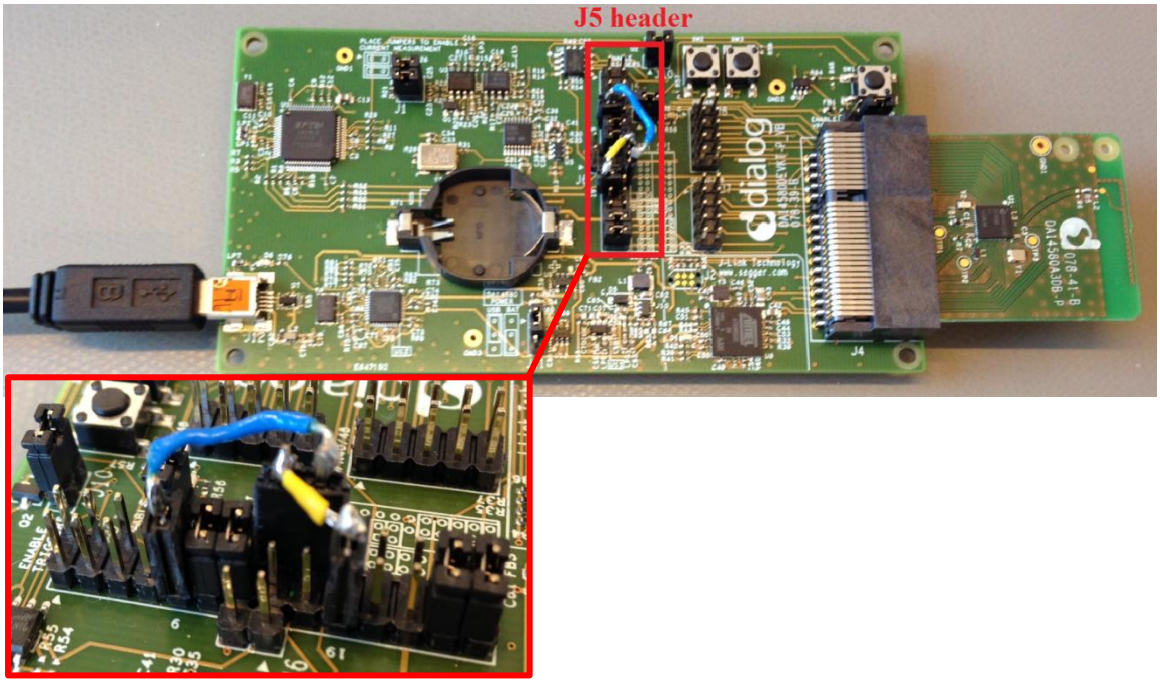

Figure 18: UART with HW flow control on PRO kit

Indeed, from the DSPS software, the location of the connections of the UART is shown below:

Table 10: UART pin out location

Parameter	Variable/macro	Source file
UART connection	UART1	Periph_setup.c

So, the connection on the J5 header must be the following:

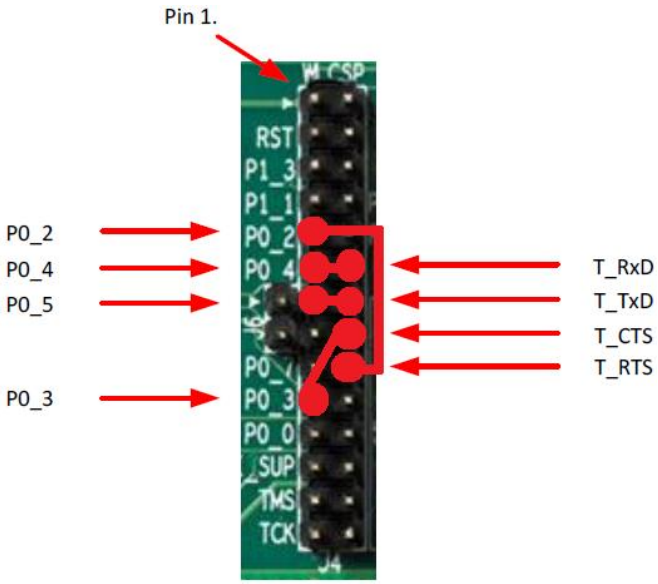

Figure 19: UART with HW flow control on J5 header

8.3 DK to DK connection setup

1. Extract the DSPS reference application zip file.

2. Open …\projects\target_apps\dsps\sps_device\Keil_5\sps_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\sps_host\Keil_5\sps_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. Hercules or TeraTerm) 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 Hercules is used, initially select tab ‘Serial’.
   b. Select the appropriate COM port in the ‘Name’ filed, set ‘Baud’ to ‘115200’ and ‘Handshake’ to the selected flow control method chosen in the user_periph_setup.h and user_sps.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 ‘Open’ to open the port.
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 ‘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 sps_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 DA14580 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\sps_device\Keil_5\out_580\ for the peripheral device and …\projects\target_apps\dsps\sps_host\Keil_5\out_580\ 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 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. The

Nexus 7 tablet with Android version 6.0.1 achieved the maximum data rate performance. The performance results are outlined in Table 11. The data rate performance has also been evaluated with Nexus 5 with Android versions 4.4.3, 6.0.1, Nexus 6P with Android version 6.0 and Galaxy S5 with Android version 4.4.2 smartphones.

The UART baud rate 115200 bit/s was used for all performance measurements in the following table. When lower baud rates are used, the maximum throughput is determined by the selected baud rate.

Table 11: Performance results (maximum values)

Host			Android	iOS	DA14580
Connection parameters	**MTU**	octets	>130	>130	>130
	Connection Interval	ms	12.5	30	12.5
	Host max. write command size	B	128	128	128
Throughput measurements	**Half Duplex**				
	Central Tx	packets/conn event	5	7	10
		kB/s	6.9	5.65	10.9
		kbit/s	56.1	44.2	88.9
	Peripheral Tx	packets/conn event	5	7	10
		kB/s	9.7	5.65	10.9
		kbit/s	79.5	44.2	88.9
	Full Duplex				
	Central Tx	packets/conn event	5	7	10
		kB/s	6.6	2.3	10.9
		kbit/s	54.3	18.5	88.9
	Peripheral Tx	packets/conn event	5	7	10
		kB/s	8.9	5.65	10.9
		kbit/s	72.9	44.2	88.9

Revision history

Revision	Date	Description
1.0	28-Oct-2014	Initial version.
1.1	14-Jan-2015	Added: ● Support of Extended sleep mode and H/W flow control. Support of multiple UART baud rates. ● User instructions for Basic DK and iOS application.
1.2	06-Feb-2015	● Updated figures 14 and 15. ● Added section 8.2 (Hardware setup for PRO Kit).
1.3	04-May-2015	Added section 10 (Known error).
2.0	3-Feb-2016	● Modified to demonstrate and make use of the new SDK release (5.0.x) features. ● Removed section 10 (Known error). Error has been resolved. ● Template updated to latest branding guidelines. ● Back page: contact information updated.
2.1	03-Feb-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.