User manual

DA14580 SmartTag reference application

UM-B-018

Abstract

This document describes the architecture and the implementation details of the Dialog Bluetooth® Smart Proximity reference application running on the DA14580 SmartTag reference design. The SmartTags proximity monitoring application running on iOS systems is presented as well.

Contents

Figures

Tables

1 Terms and definitions

BLE	Bluetooth Low Energy (now: Bluetooth Smart)
DIS	Device Information Service
EDIV	Encrypted DIVersifier
FSM	Finite State Machine
GAP	Generic Access Profile
IRQ	Interrupt ReQuest
LTK	Long Term Key
PWM	Pulse Width Modulation
RSSI	Received Signal Strength Indicator
Rand	Random number
SoC	System on a Chip
SPOTA	Software Patch Over The Air
SUOTA	Software Update Over The Air

2 References

1. PXP_SPEC, Proximity Profile, Bluetooth SIG.
2. DA14580-01 datasheet, Dialog Semiconductor
3. UM-B-014, DA14580 Bluetooth Smart Development kit - Expert, User manual, Dialog Semiconductor
4. UM-B-003, Software development guide, User manual, Dialog Semiconductor
5. UM-B-015, DA14580 Software architecture, User manual, Dialog Semiconductor
6. UM-B-006, DA14580 Sleep mode configuration, User manual, Dialog Semiconductor
7. RW-BLE-GAP-IS, GAP Interface Specification, Riviera Waves.
8. RW-BLE-PRF-BAS-IS, Battery Service Interface Specification, Riviera Waves.
9. RW-BLE-PRF-PXP-IS, Proximity Profile Interface Specification, Riviera Waves.
10. UM-B-010, DA14580 Proximity application, User manual, Dialog Semiconductor
11. UM-B-007, DA14580 Software Patching Over The Air, User manual, Dialog Semiconductor
12. UM-B-012, DA14580 Creation of a secondary boot loader, User manual, Dialog Semiconductor
13. UM-B-019, DA14580 Beacon Reference Application, Dialog Semiconductor
14. UM-B-004, DA14580 Peripheral drivers, User manual, Dialog Semiconductor
15. UM-B-005, DA14580 Peripheral examples, User manual, Dialog Semiconductor

3 Introduction

The Dialog Bluetooth® Smart proximity tag reference design and software distribution provides an ideal starting point to develop a proximity tag application with the shortest time-to-market and lowest development cost and effort. The design comes with a complete software solution of the full proximity application and profiles source codes. Dialog also provides a full-featured iOS application to manage the proximity tag's settings, trigger alerts, check the battery status and play a fun FindMe game, all in source code.

The SmartTag reference application is functioning in the Bluetooth Low Energy Proximity Reporter Role defined in the Bluetooth specification [1]. The Proximity profile defines the behaviour of a Bluetooth device when it moves away from a peer device, which covers the use case where a connection loss causes an immediate alert. This alert is used to notify the user that the devices have become separated.

The SmartTag application is designed to run on Dialog's DA14580 Proximity reference design PCB as shown in Figure 1.

As a starting point the developer is suggested to get familiar with the DA14580 datasheet [2], software and hardware and first read the details about the development kit [3]. Software development guide [4] and software architecture document [5].

Also the UM-B-010 User manual of the DA14580 Proximity application [10] provides a good introduction to the SmartTag reference design.

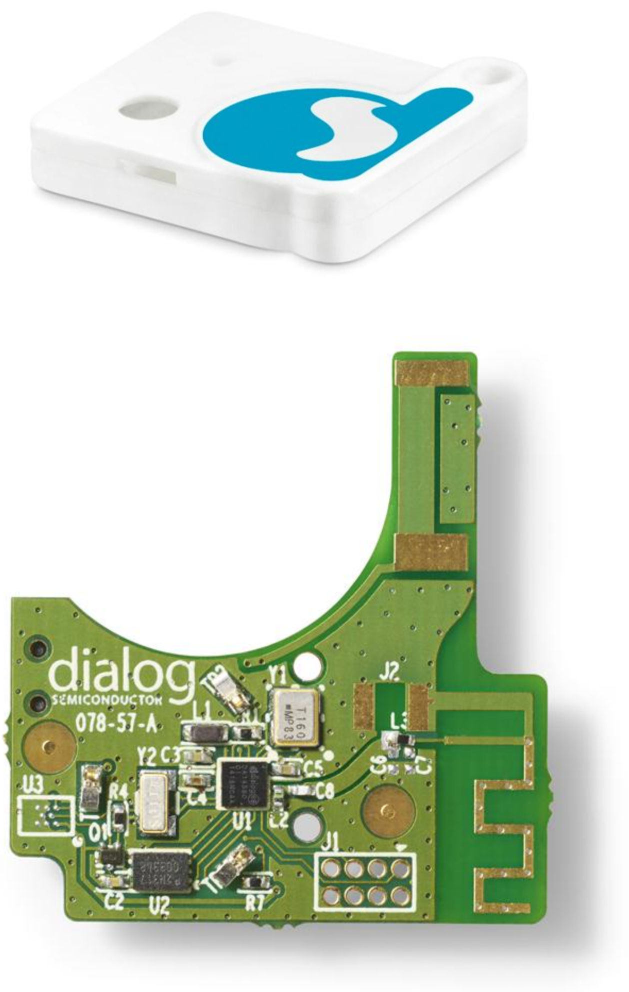


Figure 1: SmartTag reference design

4 System features

The SmartTag reference design offers all the required functionality including a LED and buzzer to signal link-loss or find-me alerts, as well as a push button to silence the alarm.

The SmartTag reference hardware serves as a flexible vehicle to demonstrate the capabilities of the DA14580 Bluetooth Smart device. It clearly shows the key differentiators: system cost, power consumption and application size.

The SmartTag reference design has the following key features:

- Long battery life on a 225 mAh CR2032 battery:
  - 1 year and 8 months at 500 ms connection interval
  - 3 years and 8 months at 2500 ms advertising interval
- Minimum configuration needs only 9 external components (3x C, 1x L, 1x R, 1x 16 MHz crystal, 1x LED, 1x buzzer, 1x push-button)
- SPI FLASH memory that enables support for Software Update Over The Air (SUOTA)
- Small area reference application: 9 x 9 mm^2 (excluding buzzer and antenna)
- Below 5.30 mA peak current in active Tx/Rx mode and less than 2.1 μA in Extended sleep mode. Note: Deep sleep mode is not supported when SPI FLASH is used for SUOTA services.
- Single button (light-touch small height SMT)
- Single colour LED (2 mm orange)
- Piezoceramic buzzer for audible alert indication
- High efficiency printed antenna
- Debug/programming connector
- 2-layer PCB, resin filled and copper plated vias on the WLCSP internal pads
- DA14580 in WLCSP34 package
- Single coin cell CR2032
- Optional power switch

5 Hardware overview

5.1 Hardware reference design deliverables

The schematic of the SmartTag reference design is shown in Figure 2. Figure 3 shows the layout and component view. Table 1 presents the Bill of Materials with indication for optional components.

Schematic, layout, Gerber files and Bill Of Material of the SmartTag are available from the Dialog support portal.

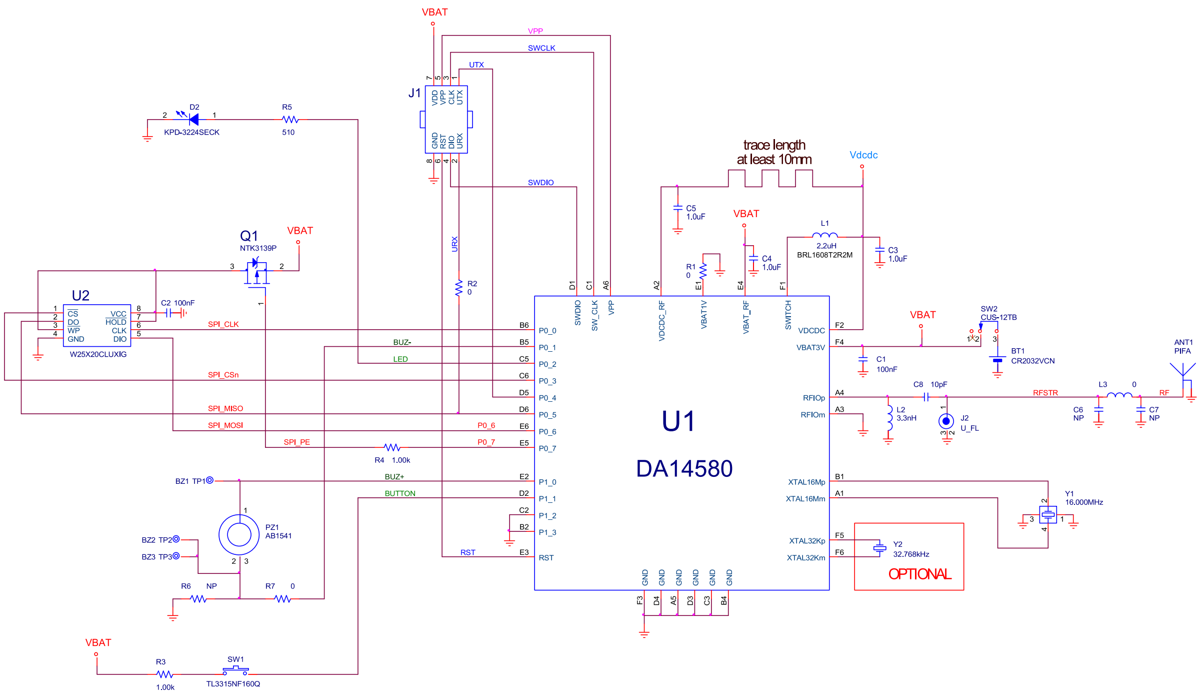


Figure 2: SmartTag reference schematic

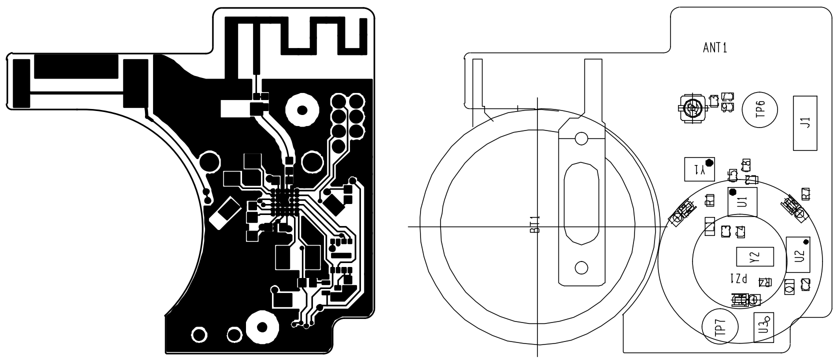


Figure 3: SmartTag Layout and component view

Table 1: SmartTag Bill of Materials

Reference	Quantity	Part name	Required for minimal functionality
U1	1	DA14580 Bluetooth Smart SoC (WLCSP34 package)	Y
Y1	1	CRYSTAL 16MHZ 10PF SMD	Y
C3 C4 C5	3	CAP CER 1.0UF 6.3V 10% X5R 0402	Y
L1	1	INDUCTOR, SHIELDED, 2.2UH, 5%	Y
R5	1	RES 510 OHM 1/10W 5% 0402 SMD	Y
D2	1	LED, SMD, DOME-LENS, ORANGE	Y
PZ1	1	PIEZO BUZZER DISC 15mm	Y
SW1	1	SWITCH TACTILE SPST-NO 0.05A 15V	Y
C1 C2	2	CAP CER 0.1UF 16V 10% X7R 0402	N
R1 R2 R7 L3	4	RES 0.0 OHM 1/16W 5% 0402 SMD	N
R3 R4	2	RES 1.0K OHM 1/16W 5% 0402 SMD	N
TP1 TP2 TP3	1	MICRO UNIVERSAL CONTACT Z 1.3MM	N
Q1	1	MOSFET P-CH 20V 780MA SOT-723	N
SW2	1	SWITCH SLIDE SPDT LOW PROF 4V	N
U2	1	SPI Flash 2.5V/3.3V 8ns 8-USON	N
Y2	1	CRYSTAL 32.768KHZ 12.5PF SMD	N
C8	1	CAP CERAMIC 10PF 50V NP0 0402	N
L2	1	INDUCTOR 3.3NH 300MA 0402	N

5.2 Signal allocation

The SmartTag hardware is based on the DA14580 in a WLCSP34 package. Pins that are located on the internal rows need microvias, which normally require a 4-layer PCB. This design uses an unusual PCB structure: small mechanical vias on internal pins, which are then resin-filled and copper plated.

The GPIO port allocation is outlined in Table 2.

Table 2: GPIO allocation

GPIO	Function
P0_0	SPI_CLK (SUPPORTS BOOTING FROM SPI FLASH)
P0_1	BUZZER (-)
P0_2	LED (ACTIVE HIGH, USED IN BURSTS OF 50ms typ.)
P0_3	SPI_CS (SUPPORTS BOOTING FROM SPI FLASH)
P0_4	UART_TX (DEBUG/PROGRAMMING OPTION)
P0_5	SPI_MISO (SUPPORTS BOOTING FROM SPI FLASH) / UART_RX
P0_6	SPI_MOSI (SUPPORTS BOOTING FROM SPI FLASH)
P0_7	SPI_PE (POWER ENABLE/DISABLE OPTION FOR THE SPI FLASH)
P1_0	BUZZER (+)
P1_1	BUTTON
P1_2	NOT USED – GROUNDED
P1_3	NOT USED – GROUNDED
P1_4	SWCLK (JTAG)
P1_5	SWDIO (JTAG)

6 Software features

6.1 Profiles and services

Besides the proximity reporter profile, the SmartTag application implements a number of profiles and services for monitoring and supporting additional features.

- Proximity profile, reporter role
  - Link Loss service
  - Immediate Alert service
  - Tx Power service
- Battery service, server role
- Data Information service, server role
- Find me profile, locator role
- SPOTA/SUOTA, server role

6.2 Alerts

The SmartTag application supports two types of alerts for user notifications (High level, Mild level), as described in Table 3.

Table 3: Alert types

Alert type	Description
High level	LED blinking with buzzer tone with the following pattern: LED: 150 ms on – 150 ms off. Buzzer: 150 ms on – 150 ms off, alternating 392 Hz (“G” note) and 440 Hz (“A” note) Triggered when peer device writes immediate alert with ‘High Alert’ or SmartTag disconnects from peer and Link Loss is set to ‘High Alert’.
Mild level	LED blinking with buzzer tone with the following pattern: LED: 500 ms on – 500 ms off Buzzer: 500 ms on – 500 ms off, 440 Hz (“A” note) Triggered when peer device writes immediate alert with ‘Mild Alert’ or SmartTag disconnects from peer and Link Loss is set to ‘Mild Alert’.

6.3 Advertising and sleep phases

SmartTag advertises in undirected mode with different intervals for specific advertising phases:

- **Advertising phase (200 ms interval):** For the first minute after start-up or disconnection.
- **Advertising phase (1000 ms interval):** For three minutes following the 200 ms interval phase.
- **Extended sleep phase:** After four minutes the SmartTag stops advertising and enters continuous Extended sleep mode.

In Advertising mode the SmartTag blinks the LED with the following pattern: 50 ms on – 1000 ms off.

The SmartTag application only supports Extended sleep mode. The supported sleep modes of the DA14580 are explained in [6].

6.4 Push-button interface

The actions that are triggered on a push-button event depend on the state of the SmartTag as described in Table 4:

Table 4: Push-button interface

State	Action on button press
Alert is active	Stop alert.
Continuous Extended sleep mode	Wake up SmartTag and start advertising.
Connected and 'find me' locator discovered, immediate alert service on peer device	Write alert characteristic of immediate alert service on peer device.
Advertising state	Long press (currently set to 3 s): bonding data are deleted from SPI FLASH memory. When long press is detected, an 880 Hz tone ("A" note, 5th octave) will be played for 125 ms.
All other states	None.

6.5 Security

According to the Bluetooth Core specification, the purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices, to be used for future authentication.

Since the SmartTag reference design does not have a keyboard or display, it only supports the 'Just Works pairing method. Upon completion of a successful bonding procedure the SmartTag stores the security information (LTK, EDIV and RAND which will be also referred as "bonding data") in the SPI FLASH memory for reuse on subsequent reconnections of the bonded device. For more information regarding the bonding procedure refer to [7], section 5.6.

When the SmartTag is in Advertising mode, the user can 'forget' a bonded central device by keeping the button pressed until a tone is heard. This indicates that the security information has been deleted from the SPI FLASH memory and a new central device can now pair with the SmartTag device.

Note: The SmartTag transmits a Security Request command to the central device in order to trigger the pairing procedure, upon receiving the connection request from the central device. However, the command could be ignored by the central device, when it does not wish to start a pairing procedure.

6.6 Battery level

In Connected state the SmartTag software samples the battery level every 5 minutes and updates the value of the battery level characteristic. The notification capability of the characteristic is disabled at the beginning of each connection. The peer device must write the configuration attribute of the characteristic with the corresponding value to enable value update notifications.

6.7 Software Update Over The Air

The SmartTag software supports software updating over a Bluetooth Low Energy (BLE) link by using the SPOTA profile as described in [11]. The software images downloaded using the SPOTA profile are stored in the external SPI FLASH memory.

To support SUOTA, the secondary boot loader **must** be programmed in the OTP memory of the DA14580. The SmartTag software sets the SPI FLASH memory in power down mode to reduce the power consumption. The secondary boot loader in OTP will release the SPI FLASH memory from power down mode during booting.

The SmartSnippets tool can be used for programming the boot loader in OTP and for programming the product header and the dual images in SPI FLASH. The details of the specific memory map and the procedure to create and write an image that is compliant to the required memory map, are described in [12]. Information on the tools and hardware needed to program the SmartTag can be found in [13], section "Downloading/Programming".

Appendix A describes how to use a Dialog Semiconductor developed SUOTA application to update a SmartTag image.

7 Software architecture

7.1 Source files

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

Table 5: SmartTag application files

Group	File	Description
boot	system_ARMCM0.c boot_vectors.s hardfault_handler.c	Application start-up, ISR vectors, hard fault handler
arch	arch_main.c jump_table.c arch_sleep.c nmi_handler.c arch_system.c	Main loop, kernel scheduling, system setup, system clocks, common sysRAM/ROM code structure (jump_table), sleep modes API
	periph_setup.c	Peripheral modules initialisation, GPIO pins assignment.
driver	rf_580.c	Radio
	gpio.c	GPIO driver
	battery.c adc.c	Battery and analogue to digital convertor (ADC) drivers used for battery level sampling
	spi_flash.c	Flash driver over SPI interface
	spi.c	SPI interface driver
	pwm.c	Pulse width modulation driver
	systick.c	SysTick Timer driver
	wkupct_quadec.c	Wakeup module driver used in push button interface
nvds	nvds.c	Non-volatile data structure
rwble	rwble.c	BLE core interrupt service routines
	rwip.c	BLE sleep function
profiles	proxr.c proxr_task.c	Proximity Reporter profile
	diss.c diss_task.c	Device information service, server role
	findl.c findl_task.c	Find-me Locator profile
	bass.c bass_task.c	Battery service, server role
	spotar.c spotar_task.c	SUOTA/SPOTA receiver profile, server role

Group	File	Description
app	app.c app_task.c app_sec.c app_sec_task.c	BLE application skeleton. Basic BLE and security functionality and message handling.
	app_smarttag_proj.c	SmartTag application specific functionality. Database creation of supported profiles, advertise and sleep profile, push button interface, security features.
	app_smartrtag_utils.c	Functions to control the PWM engine and SPI FLASH read and store functions for the bonding data,
	app_proxr.c app_proxr_task.c	Application level proximity reporter functionality. Message composing and handling, alert user notifications.
	app_batt.c app_batt_task.c	Application level battery server functionality. Message composing and handling, battery level sampling.
	app_dis.c app_dis_task.c	Application level DIS server functionality. Message composing and handling.
	app_findme.c app_findme_task.c	Application level Find-me locator functionality. Message composing and handling, immediate alert service discovering and writing on peer device.
	app_spotar.c app_spotar_task.c	Application level SPOTA server functionality. Message composing and handling of SUOTA/SPOTA receiver protocols.

7.2 Header files

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

Table 6: Header files of the SmartTag application

File name	Purpose
da14580_config.h	Project Configuration file. It includes definitions that are global to all code and to the common scatter file as well.
da14580_scatter_config.h	Configuration file for the common scatter file.
da14580_stack_config.h	Configuration file for the Stack.
app_smarttag_proj.h	Contains application related definitions. It also exports externally used functions of app_smarttag_proj.c.
app_smarttag_utils.h	Exports function definitions for PWM engine control and bonding data manipulation.
app_proxr.h app_proxr_task.h	Contains prototypes of the externally used functions of proximity reporter profile related functionality.
app_bat.h app_bat_task.h	Contains prototypes of the externally used functions of battery service related functionality.
app_dis.h app_dis_task.h	Contains prototypes of the externally used functions of device information service related functionality.
app_findme.h app_findme_task.h	Contains prototypes of the externally used functions of find-me locator profile related functionality.
app_spotar.h app_spotar_task.h	Contains prototypes of the externally used functions of SPOTA/SUOTA receiver profile related functionality.

File name	Purpose
app.h	It exports the global variable app_env and functions in app.c that are used by other modules of the application.
app_task.h	Contains the definitions of the application's state with respect to the Bluetooth status and export global variables and functions that are defined in app_task.c to the rest of the code.
gpio.h	Contains the prototypes of the functions in gpio.c (GPIO driver) that are used by the rest of the code.
battery.h adc.h	Contain various definitions that are required by the battery and ADC driver and export globally accessed functions (defined in battery.c and adc.c).
wkupct_quadec.h	Contains various definitions that are required by the WKUP – QUADEC driver and its globally accessed functions (defined in wkupct_quadec.c).

8 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.

8.1 Application configuration parameters

The main parameters of the SmartTag application software, which can be adjusted to customise certain functionality of the application, are listed in Table 7.

Table 7: SmartTag application configurable parameters

Parameter	Description	Current Value
`APP_SPI_POWEROFF_DELAY`	Upon application initialisation, the timer `APP_FLASH_POWEROFF_TIMER` is set with this value to delay the SPI FLASH power down mode and allow the developer to use SmartSnippets Flash Programmer application to connect to the SmartTag device and re- program the SPI FLASH device.	5 s
`APP_SLEEP_DELAY`	Upon application initialisation, the function `app_set_startup_sleep_delay(APP_SLEEP_DELAY)` is called to modifie the system startup sleep delay. This delay will allow the developer to have an active JTAG interface and be able to use the debugger to connect to the SmartTag device.	5 s
`APP_FIRST_ADV_PHASE_DUR`	This parameter is used to set the `APP_ADV_TIMER` to control the advertising intervals. Refer to paragraph 6.3 for details.	60 s
`APP_FIRST_ADV_PHASE_INTVAL`	This parameter is used to set the advertising interval for the first advertising phase.	200 ms
`APP_SECOND_ADV_PHASE_DUR`	This parameter is used to set the `APP_ADV_TIMER` to control the advertising intervals. Refer to section 6.3 for details.	3 min
`APP_SECOND_ADV_PHASE_INTVAL`	This parameter is used to set the advertising interval for the second advertising phase.	1 s
`APP_BOND_DATA_FLASH_ADDR`	The SPI FLASH start address where the bonding data are stored.	0x3F004
`APP_ADV_BLINK_ON_DUR`	Controls the LED 'on' duration during advertising	50 ms
`APP_ADV_BLINK_OFF_DUR`	Controls the LED 'off' duration during advertising	1 s

8.2 Application task state machine

The FSM of the application task of SmartTag consists of the following states, see Table 8.

Table 8: Application task: FSM states

State	Status
APP_DISABLED	Application task initiated. Waiting for GAPM_DEVICE_READY_IND message.
APP_DB_INIT	Database initialisation in progress.
APP_CONNECTABLE	Advertising or Continuous Extended Sleep.
APP_CONNECTED	Device connected to Proximity Monitor
APP_SECURITY	BLE Security is activated. Device is paired or bonded with a Proximity Monitor.

Figure 4 graphically illustrates the FSM. The state transitions are described in Table 9.

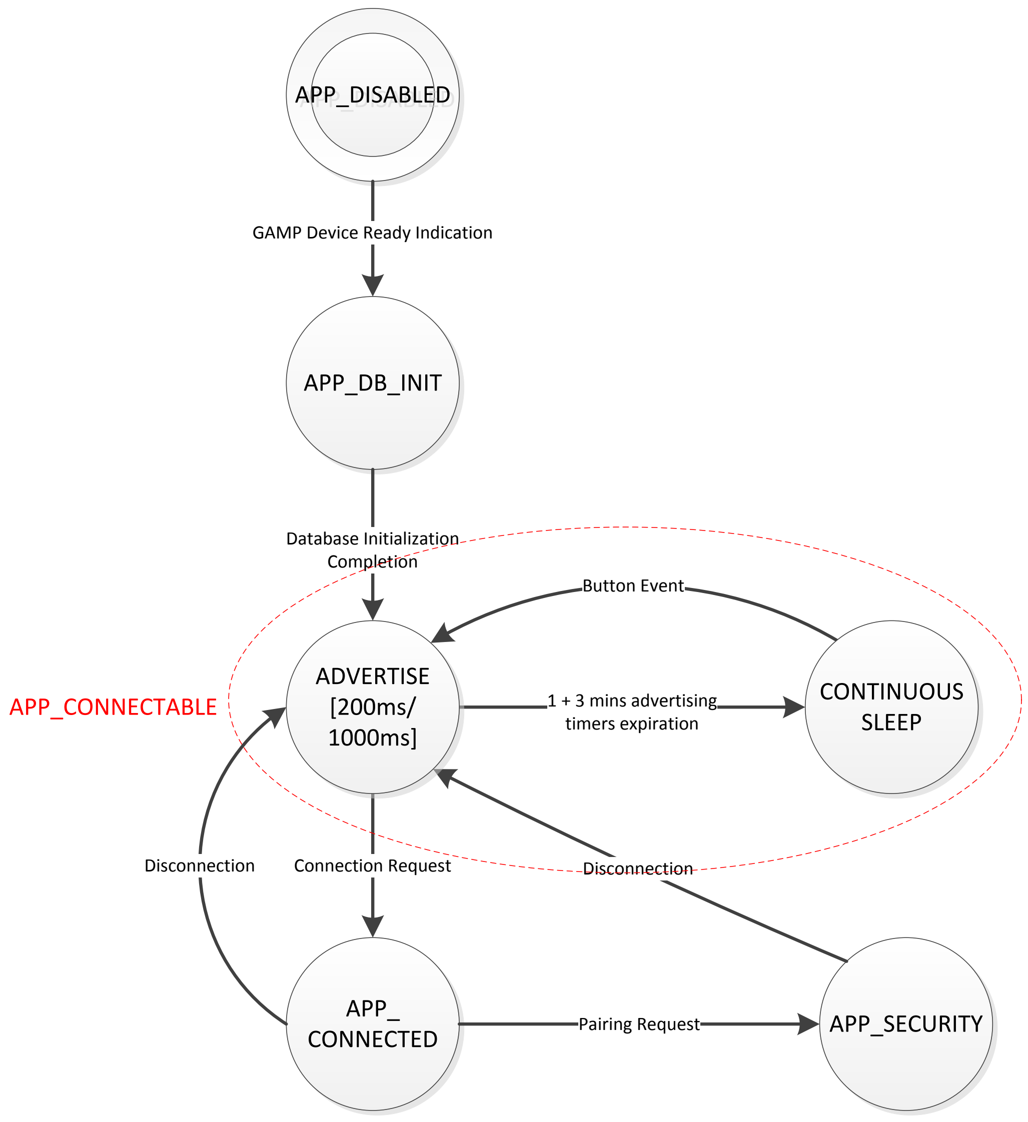


Figure 4: Application task FSM

Table 9: State transition table of the application task FSM

State transition		Event
From	**To**	**Action**
APP_DISABLED		GAPM_DEVICE_READY_IND message reception.
	APP_DB_INIT	Start database initialisation of supported profiles.
APP_DB_INIT		Database initialisation procedure is completed.
	ADVERTISE (APP_CONNECTABLE)	Start advertising in 200 ms advertising interval
ADVERTISE (APP_CONNECTABLE)		Timer for 1000 ms advertising interval expired.
	CONTINUOUS SLEEP (APP_CONNECTABLE)	Device stops advertising and/ or active alerts and switches to continuous Extended sleep mode.
CONTINUOUS SLEEP (APP_CONNECTABLE)		Button press event.
	ADVERTISE (APP_CONNECTABLE)	Device exits extended sleep mode and restarts advertising in 200 ms interval.
ADVERTISE (APP_CONNECTABLE)		Connection Request has been received.
	APP_CONNECTED	Enable profiles and start battery polling.
APP_CONNECTED		Pair request received.
	APP_SECURITY	Starts pairing procedure. If procedure completes successfully device remains connected, otherwise it disconnects from central.
APP_CONNECTED or APP_SECURITY		Device disconnects.
	ADVERTISE (APP_CONNECTABLE)	Start advertising in 200 ms advertising interval.

8.3 Entry points

8.3.1 app_init() and app_init_func()

This is the main entry point of the application task. Function `app_init()` creates and initialises the `TASK_APP` and calls `app_init_func()` to initialise low level hardware modules and parameters.

8.3.2 app_configuration_func() and app_db_init_func()

The function `app_configuration_func()` is called during device startup. It sets the GAP role (peripheral) and other parameters of the device.

When the configuration of the device is completed, the function `app_db_init_func()` is called to initialise the database of the supported profiles. Upon completion of the database initialisation of each service and profile, an `APP_MODULE_INIT_CMP_EVT` message is sent to the `TASK_APP`. Function `app_db_init_func()` is called by the message handler to create the database of the next profile in the list. When the creation of the databases is finished, the device will start advertising.

8.4 Advertising

8.4.1 app_adv_func()

This function starts or restarts the Advertising mode. It is called upon completion of the database initialisation, upon device disconnection and upon expiration of the `APP_ADV_TIMER` to start the advertising interval of the 1000 ms phase. Also, it initialises the `APP_ADV_TIMER` for the variable advertising interval feature and the `APP_ADV_BLINK_TIMER` for advertising mode LED blinking. Finally it constructs and sends `GAPM_START_ADVERTISE_CMD` to the GAPM_TASK in order to initiate the Advertising mode.

8.4.2 app_adv_timer_handler()

This function is called upon expiration of the `APP_ADV_TIMER`. Its main functionality is to program the correct advertising phase. For continuous Extended sleep phase the function disables all timers and alerts before setting the device in Extended sleep mode.

8.4.3 app_adv_blink_timer_handler()

This function handles the expiration of the `APP_ADV_BLINK_TIMER`, restarts the timer and inverts the state of the LED.

8.5 Connection

8.5.1 app_connection_func()

This function is called upon the reception of a connection request from a central role device. The SmartTag application stops the `APP_ADV_BLINK_TIMER`, enables the profiles and services, selects the proper sleep mode depending on the received connection interval and initiates polling of the battery level.

8.5.2 app_disconnect_func()

This function is called upon reception of a `GAPC_DISCONNECT_IND` message, which indicates that the connection does not exist anymore, to stop battery level polling and restart advertising.

8.6 Security

8.6.1 app_send_pairing_rsp_func()

This function is called during the pairing process. It informs the peer device about the security capabilities of the device. The SmartTag application uses 'Just works' mode with bonding capability. It also checks whether the SmartTag device is already paired with another device (it looks for bonding data stored in SPI FLASH memory). When the SmartTag device is already paired with another central device, it will not accept the new pairing request. SmartTag only supports bonding with one central device at a time.

8.6.2 app_send_ltk_exch_func()

This function is called upon reception of `the GAPC_BOND_REQ_IND` message with `request` set to `GAPC_LTK_EXCH` (Long Term Key exchange), to generate the LTK and send it to the host.

8.6.3 app_paired_func()

Called upon reception of the `GAPC_BOND_IND` message with status `GAPC_PAIRING_SUCCEED` to store the security info in to SPI FLASH memory and complete the connection establishment phase.

8.6.4 app_validate_encrypt_req_func ()

This function is called upon reception of the `GAPC_ENCRYPT_REQ_IND` message to validate whether the connecting host has to write bonding data or not. The parameters `RAND` and `EDIV` are used to check whether the SmartTag has already stored the bonding data in SPI FLASH memory. If not, the request is rejected.

8.7 Push button

In the application initialisation function `app_init_func()`, a wakeup interrupt (IRQ) on the allocated GPIO for the push button interface is enabled. This is done via the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver. The call back function `app_button_press_cb()` is registered to enable a wakeup interrupt when the button is pressed.

8.7.1 app_button_press_cb()

This function is the callback function of the application, called from the `WKUP_QUADEC_IRQn()` interrupt handler of the wakeup module driver, when the button is pressed. It checks the state of the application and triggers the required action, as described in Table 4. It also calls the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver, to register the `app_button_release_cb()` callback function and enable a wakeup interrupt when the button is released. Finally, the function sends the `APP_WAKEUP_MSG` message to the `TASK_APP` to start the `APP_BUTTON_PRESS_TIMER` to detect a long key press (for deleting the bonding data stored in the SPI FLASH memory).

8.7.2 app_button_release_cb()

This function is the callback function of the application, called from the `WKUP_QUADEC_IRQn()` interrupt handler of the wakeup module driver when the button is released. It calls the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver, to register the `app_button_press_cb()` callback function and enable a wakeup interrupt when the button is pressed. Finally, the function sends the `APP_WAKEUP_MSG` message to the `TASK_APP` to stop the `APP_BUTTON_PRESS_TIMER`.

8.7.3 app_wakeup_handler()

This function is called upon reception of the `APP_WAKEUP_MSG` and calls function `app_adv_func()` to start advertising. It also starts/stops the `APP_BUTTON_PRESS_TIMER`, depending on the button status (`app_button_status`).

8.8 Proximity reporter and alerts

8.8.1 app_proxr_enable()

This function enables the proximity reporter profile. It is called from function `app_connection_func()`.

8.8.2 proxr_alert_ind_handler()

This function is the message handler of the `PROXR_ALERT_IND` message. The proximity reporter profile sends this message to trigger an alert on the device. This function calls the functions `app_proxr_alert_start()` or `app_proxr_alert_stop()` to start or stop an alert, depending on the alert level received in `PROXR_ALERT_IND`.

8.8.3 app_proxr_alert_start()

This function initiates user alert indications. It updates the alert state parameters and depending on the alert level it starts the PWM engine by calling the `app_proxr_pwm_enable()` function to generate the alert melody. **Note:** The LED functionality is controlled from within the functions that program the PWM tones, as explained in section 8.9.

8.8.4 app_proxr_alert_stop()

This function stops user alert indications. It clears the alert state parameters, turns off the alert LED and stops the `APP_PXP_TIMER`.

8.9 PWM engine

Details on the PWM driver can be found in [14] and PWM examples are explained in detail in [15]. This section describes how the SmartTag uses the PWM0 and PWM1 (TIMER0) outputs to create the alert melodies.

8.9.1 app_proxr_pwm_enable()

This function initialises TIMER0:

- Enables the TIMER0 peripheral clock, by calling the `set_tmr_enable()` PWM API driver function.
- Calls `set_tmr_div()` to sets the TIMER0 clock division factor to 8 (16 MHz clock source)
- Calls `timer0_init()` to initialise PWM with the desired PWM mode, TIMER0 'on' time division option and clock source selection settings. In this example, the timer tick period is configured to:

  (1/16 MHz) * 8 (clock division) * 10 (`TIM0_CLK_DIV_BY_10`) = **5 μs**
- Sets the TIMER0 'on', 'high' and 'low' times, by calling function `timer0_set()`.
- Registers a callback function for `SWTIM_IRQn` interrupt by calling `timer0_register_callback()`. The callback function pointer is an input parameter to this function. In the SmartTag application three different melodies/tones are needed, which are handled by the following callback functions:
  - o `high_alert_pwm_callback()`: programs the high alert melody
  - o `mild_alert_pwm_callback()`: programs the mild alert melody
  - o `button_pwm_callback()`: programs the button long press tone
- Enables the `SWTIM_IRQn`, by calling the `timer0_enable_irq()` function.

- Starts TIMER0 by calling the `timer0_start()` function.

8.9.2 high_alert_pwm_callback()

This is a callback function for the `SWTIM_IRQn` interrupt that handles the high alert melody. The following parameters configure the melody:

- The melody is defined in the constant array `alert_high_notes[]`. In this array the developer can define a new sequence of notes and pauses. Note values are the frequency of the musical notes. For example, '880' means 880 Hz which is the "A" note of the 5th octave.
- Each time this callback function is called, a note or a pause from the array will be programmed to the PWM engine by calling the functions `timer0_set_pwm_high_counter()` and `timer0_set_pwm_low_counter()`. The duration of the note/pause is determined by the parameter value `ALERT_HIGH_DURATION` that is passed to the function `timer0_set_pwm_on_counter()`. At the end of this duration, an interrupt will be triggered and the callback function will be executed again to program the next note/pause from the array.

In order for the LED to blink synchronously with the melody, the LED is controlled within this function. When a pause is programmed, the LED is turned off. When a note is played, the LED is turned on.

8.9.3 mild_alert_pwm_callback()

This is a callback function for the `SWTIM_IRQn` interrupt that handles the mild alert melody. This function is similar to the `high_alert_pwm_callback()` function, described in the previous section, with one extra parameter:

- `ALERT_MILD_EXTRA_DELAY`: This parameter determines how many times the function will be called whithout programming a note/pause. This is needed in case a note/pause duration needs to be quite long and cannot be set by the `timer0_set_pwm_on_counter()` range.

The notes/pauses for the mild alert melody are defined in the constant array `alert_mild_notes[]`. The duration of the note/pause is determined by the parameter value `ALERT_MILD_DURATION`.

8.9.4 button_pwm_callback()

This is a callback function for the `SWTIM_IRQn` interrupt that handles the long button press alert. This alert is a single tone alert (`button_press_notes = PWM_TONE_A_5TH`) and the TIMER0 'on' counter is not programmed again, because no more interrupts are needed to program more notes/pauses.

8.10 Battery service and battery level sampling

8.10.1 app_batt_enable()

This function enables the battery service. It is called in function `app_connection_func()`.

8.10.2 app_batt_poll_start()

This function starts the `APP_BATT_TIMER` to initiate periodic sampling of the battery level.

8.10.3 app_batt_poll_stop()

This function stops the `APP_BATT_TIMER` to terminate periodic sampling of the battery level.

8.10.4 app_batt_timer_handler()

This function handles the expiration of the `APP_BATT_TIMER` timer. It calls function `app_batt_lvl()` to sample the battery level and restarts the `APP_BATT_TIMER` timer.

8.10.5 app_batt_lvl()

This function calls the API function `battery_get_lvl()` of the battery driver to sample the current battery level. It updates the value of the battery level characteristic, when the current sample differs from the previous one.

9 Power consumption

Table 10 shows the power consumption of the SmartTag device for Extended sleep mode and different application events.

Table 10: Power consumption

Mode/event	Average current (µA)	Time (ms)	Charge (µC)
Extended sleep mode	2.1	-	-
Advertising	935	12.42	11.622
Connection (500 ms interval)	752	9.381	7.060
Connection (2500 ms interval)	1199	11.276	13.526
High level alert	2961	150	444.19

The power profiler graphs for the events presented in Table 10 are shown in the following sections.

9.1 Advertising

The graph of the current drawn during an advertising event in all advertising channels is shown in Figure 5.

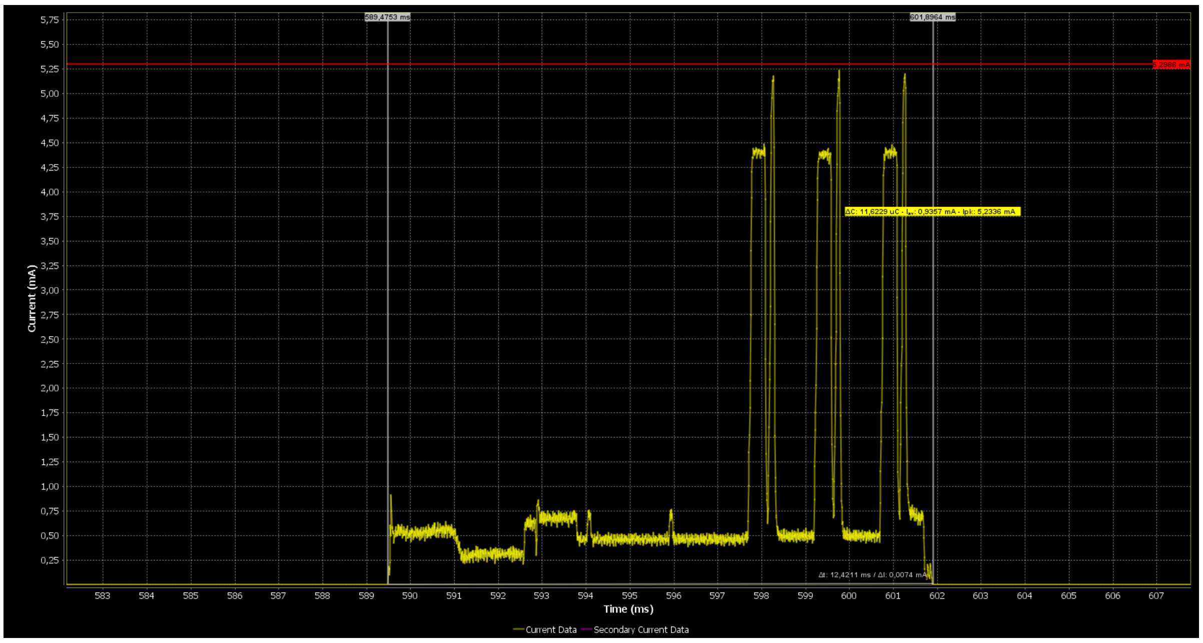


Figure 5: Power consumption of advertising event

9.2 Connection events

Graphs of the current drawn during a connection event are shown in Figure 6 for a 500 ms connection interval and in Figure 7 for a 2500 ms connection interval. For longer intervals the RF receiver must be active for a longer time to detect polling from the master, due to clock inaccuracy.

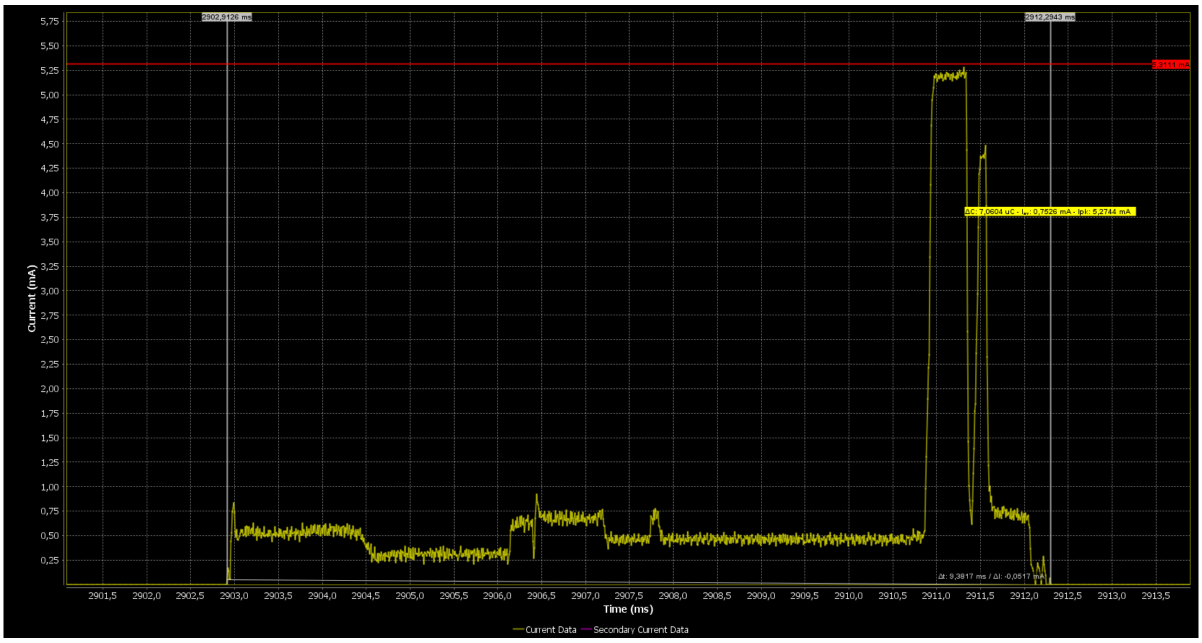


Figure 6: Power consumption of connection events with a 500 ms interval

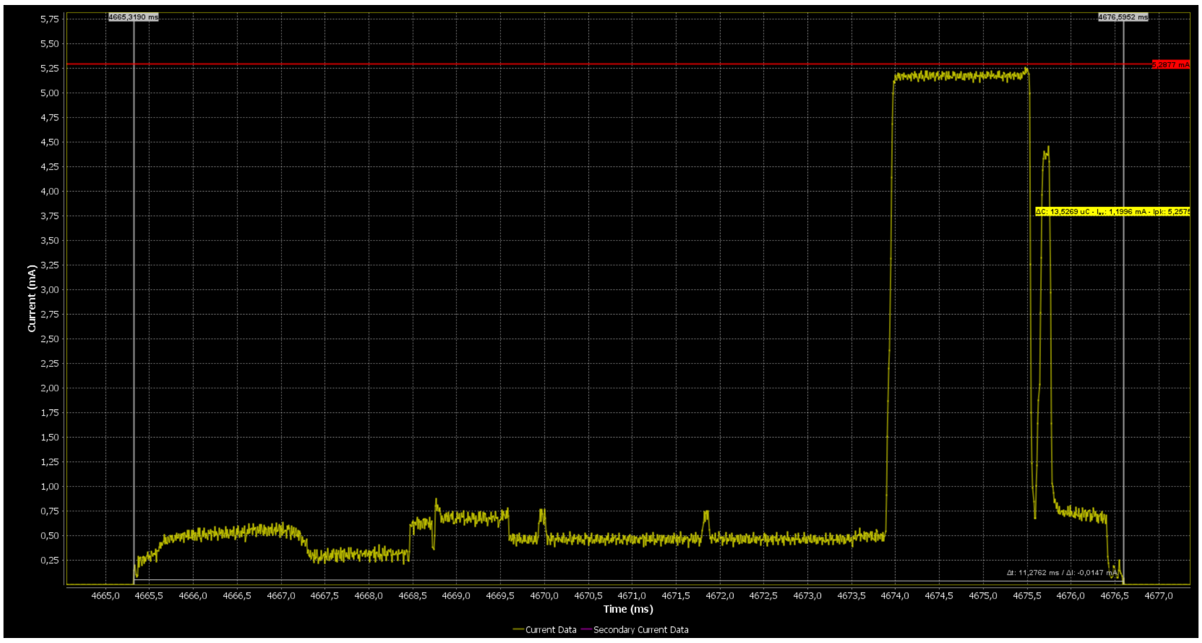


Figure 7: Power consumption of connection events with a 2500 ms interval

9.3 High level alert

The graph of the current drawn during the active period of a high level alert is shown in Figure 8. During this period LED is on and the buzzer tone is active. The duration of the alert event is 150 ms. The base current of ~3 mA corresponds to the power consumed by the LED and the processor. The current peaks correspond to the PWM generated tone, that is played repeatedly on the buzzer.

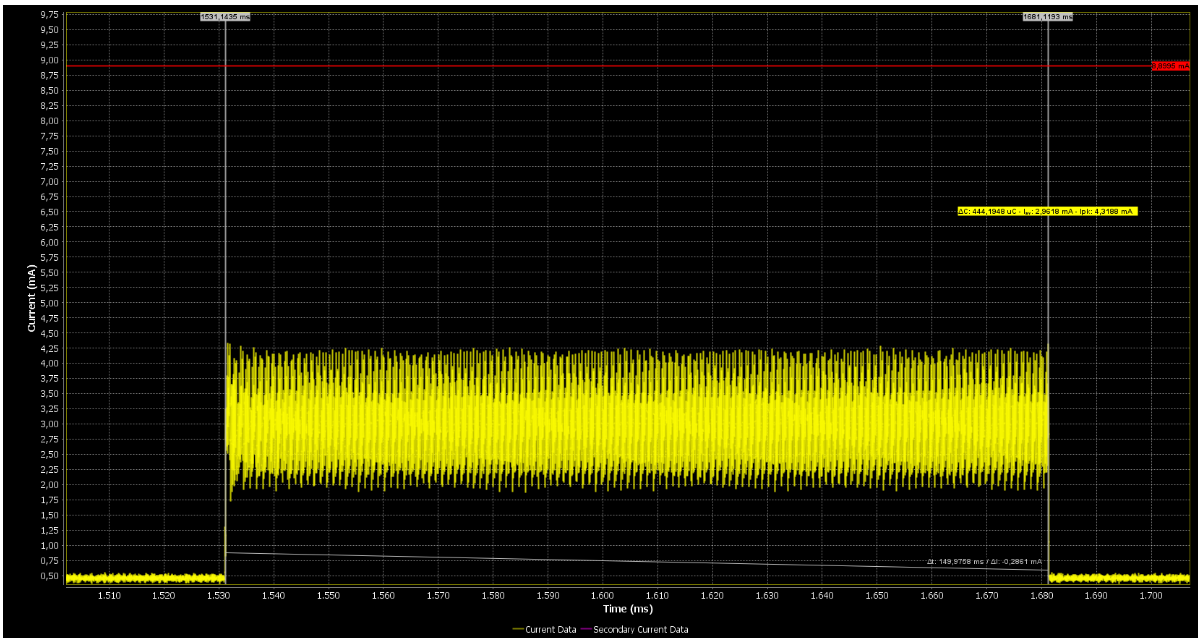


Figure 8: Power consumption during a high level alert event

10 SmartTags iOS application

10.1 Overview

The SmartTags proximity reference application comes with a proximity monitor demo application for iOS systems. SmartTags can discover, connect and control SmartTag devices within the Bluetooth RF range of the iOS device.

Features:

- Proximity reporter devices discovery
- Connection to discovered device
- RSSI polling of connected devices
- Immediate alert
- Distance alert based on measured RSSI value
- Link Loss alert
- Battery level monitoring
- Disconnection
- Warmer/Colder game based on the current RSSI value

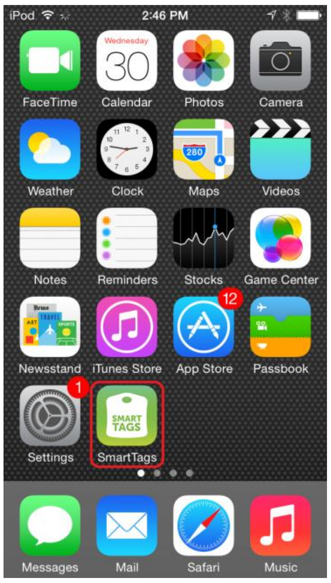
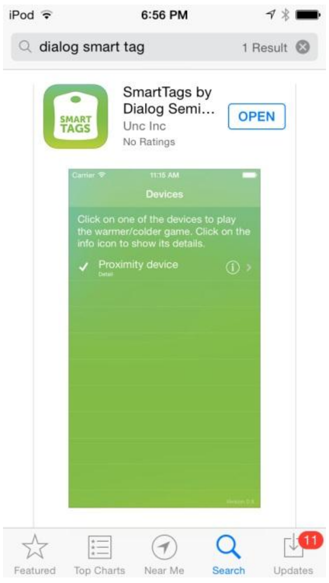


Figure 9: SmartTags icon

10.2 Installation

The SmartTags application is available in the App Store. A user can find the application by searching in the App Store for ‘Dialog SmartTags’. The search results will display the installation of SmartTags and the user can click on the download icon. The application will then be downloaded and installed without further user interaction.

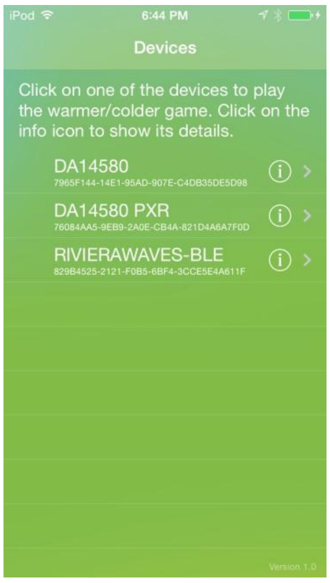


Click on one of the devices to play the warmer/colder game. Click on the info icon to show its details.

Figure 10: Installation of SmartTags

10.3 Device list

The user must click on the SmartTags icon to start the application. The list of all devices supporting the proximity reporter profile, including SmartTag, will be listed on the screen. When a device gets within the Bluetooth RF range of the iOS system while the application is running, it will be displayed automatically.



Click on one of the devices to play the warmer/colder game. Click on the info icon to show its details.

DA14580
7965F144-14E1-95AD-907E-C4DB35DE5D98

DA14580 PXR
76084AA5-9EB9-2A0E-CB4A-821D4A6A7F0D

RIVIERAWAVES-BLE
829B4525-2121-F0B5-6BF4-3CCE5E4A611F

Figure 11: Device list

10.4 Device details

The user must click on the info icon of the desired device to connect to it. The following controls are provided to the user for monitoring and control of the device:

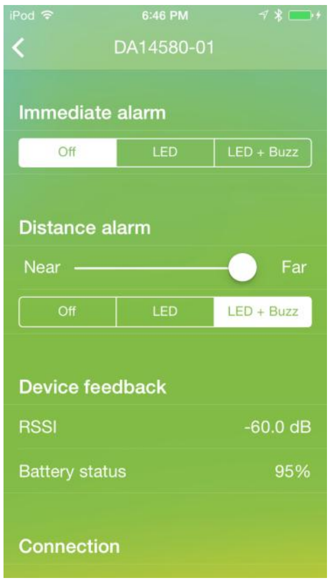


Figure 12: Device details

10.4.1 Immediate alarm

The user can trigger an immediate alert on the SmartTag device by clicking on ‘LED’ or ‘LED + Buzz’ button of the immediate alarm section. The alert can be terminated by clicking on the ‘Off’ button.

10.4.2 Distance alarm

Distance and link loss alerts are configured by this section’s controls. The SmartTag application polls the RSSI of the connected devices to estimate the distance between the iOS system and the SmartTag device.

The ‘Near/Far’ slider controls the RSSI threshold for triggering a distance alert. When the measured RSSI level of a connected device moves below the configured threshold or a device is disconnected, the SmartTags app will trigger an alert on both SmartTag and iOS devices. In case of an alert, the screen displays a message as shown in Figure 13. The user can terminate the alert on either device by clicking on ‘OK.

The alert level depends on the selected button (Off, LED, LED + Buzz). When the ‘Off button is selected, no alert will be triggered.

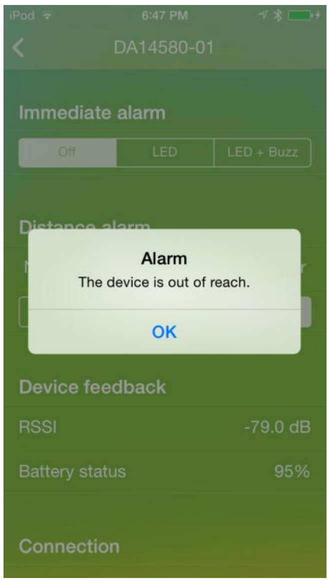


Figure 13: Distance alert window

10.4.3 Device feedback

In this section SmartTags provides status information for the device connected to the SmartTag proximity reporter. The current RSSI value and the battery level of the SmartTag are shown here.

10.4.4 Connection

The user can disconnect from the SmartTags application by clicking of ‘Disconnect’ or ‘Unpair’.

10.5 Warmer/Colder game

When the user clicks on the device name, the SmartTags application will start the Warmer/Colder game with the selected device. The application changes states and displays an image according to the RSSI based estimate of the distance of the SmartTag proximity reporter. The application changes to a warmer state when the SmartTag approaches the iOS system. The state will change to colder when the distance of the SmartTag increases.

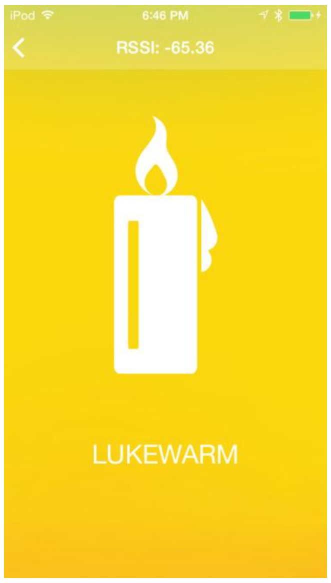


Figure 14: Warmer/Colder game

Appendix A Using the Android application for SUOTA

This appendix describes how to use the Dialog Semiconductor SUOTA Initiator application to update the software of a SmartTag device. There are two variants of the SUOTA Initiator application: one for the Android operating system and one for the iOS. The applications are available in the App Store and Play Store. A user can find the applications by searching in the App Store or Play Store for ‘Dialog SUOTA’. Since these applications have a similar user interface, only the Android application will be described here.

STEP 1: Prepare the SmartTag device

As described in section 6.7, a dual image secondary boot loader needs to be programmed into the OTP memory of the DA14580. Also, the initial SmartTag software image with the right header needs to be programmed to the device’s SPI FLASH using the FLASH Programmer. To verify that this step has been completed successfully, reset the SmartTag device and confirm that it boots up correctly and starts advertising (the LED will be flashing).

STEP 2: Make sure that the SmartTag device is not paired with another device

When the SmartTag device is in Advertising mode, the user can ‘forget’ a bonded central device by keeping the button pressed until a tone is heard. This indicates that the security information has been deleted from the SPI FLASH memory and a new central device can pair with the SmartTag device.

Note: When you delete the bonding data from the SmartTag SPI FLASH and the Android device was paired with the SmartTag device, the SmartTag device needs to be removed from the list of paired devices of the Android device (this is usually done via menu *Settings -> Bluetooth -> Forget Device*).

STEP 3: Install and start the application on the Android device

After successful installation, the following icon should appear under the installed applications menu. Click on the icon to start the application.

STEP 4: Initial menu, scan for advertising devices

Press the icon to initiate scanning. Assuming the SmartTag device is advertising, the device name and the Bluetooth Device address of the device will be displayed as shown in Figure 15.

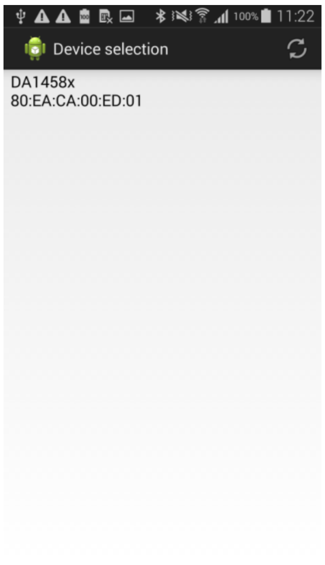
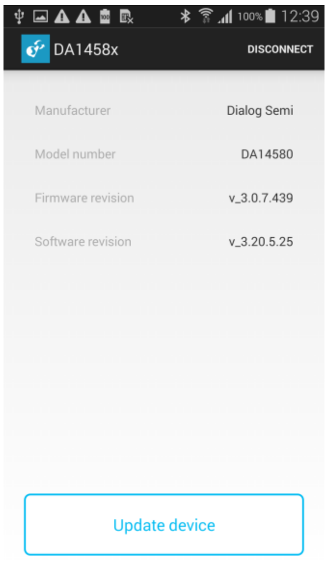


Figure 15: Device selection menu

STEP 5: Connect to the SmartTag device

Click on the SmartTag device to connect. Upon successful connection to the SmartTag device, the DIS information will be displayed on the screen as shown in Figure 16.



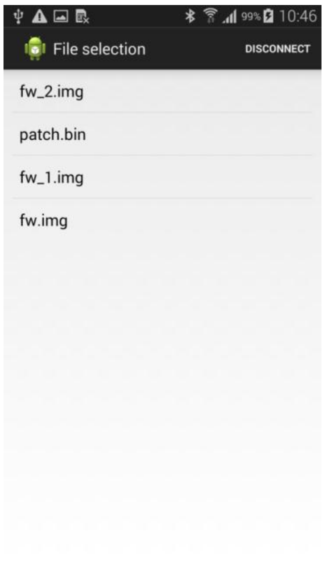
DA1458x DISCONNECT

Manufacturer	Dialog Semi
Model number	DA14580
Firmware revision	v_3.0.7.439
Software revision	v_3.20.5.25

Figure 16: DIS screen

STEP 6: Update SmartTag software image

Click on the 'Update device' button and a list of files will appear on the screen. In order for the file to appear in this 'File Selection' screen it has to be copied to the 'Suota' directory of the Android device. Connect the Android device via USB to the PC where the SmartTag images were created and copy these images under the 'Suota' directory. An example of the 'File Selection' screen is shown in Figure 17. Verify that the target image is listed on this screen.



fw_2.img
patch.bin
fw_1.img
fw.img

Figure 17: File selection screen

STEP 6: Set SUOTA parameters

As soon as the image is selected, the 'Parameter settings' screen appears. See Figure 18.

First, set the memory type. The image update procedure is only supported for non-volatile memory types of SPI (FLASH memory) and I2C (EEPROM). In this example SPI (FLASH) has been selected.

Then select the Image (memory) bank (see [12]):

1:	Use the first bank with start address as indicated in the Product Header
2:	Use the second bank with start address as indicated in the Product Header
0:	Burn the image into the bank that holds the oldest image

Next, define the GPIO pins of the memory device. In the SmartTag device the SPI FLASH GPIO configuration is as follows:

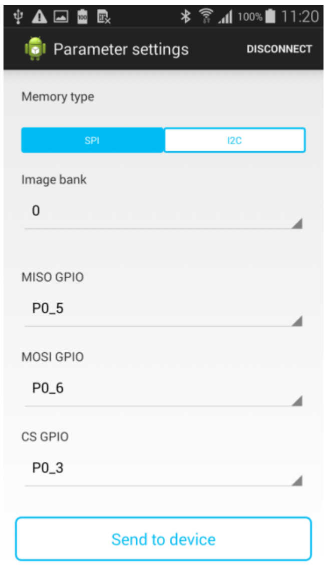
```
MISO   => P0_5
MOSI   => P0_6
CS     => P0_3
SCK    => P0_0
```

Finally, scroll down to choose the block size. A few points should be considered when this size is set.

- Firstly, it has to be larger than 64 bytes, which is the size of the image header.
- Secondly, it must be a multiple of 20 bytes, which is the maximum amount of data that can be written at once in the `SPOTA_PATCH_DATA` characteristic.
- Thirdly, it should not be larger than the SRAM buffer in the SUOTA Receiver implementation, which holds the image data received over the BLE link before burning it into non-volatile memory.

This example uses a block size of 240 bytes.

After all the parameters have been set, the user can click on the 'Send to device' button at the bottom of the screen.



Memory type: SPI [✓ SPI] I2C

Image bank: 0

MISO GPIO: P0_5

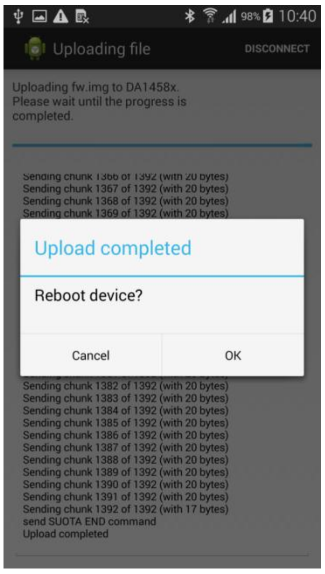
MOSI GPIO: P0_6

CS GPIO: P0_3

Figure 18: SUOTA parameter settings

STEP 7: Reboot the device

As soon as the ‘Send to device’ button is clicked, a log screen appears that shows the image data blocks sent over the BLE link. In case an error occurs, a pop up indication will inform the user. When no error occurs and the SmartTag device has received and programmed the image successfully, the following screen will appear, prompting the user to reset the SmartTag device.



Uploading fw.img to DA1458x.
Please wait until the progress is completed.

```
Sending chunk 1366 of 1392 (with 20 bytes)
Sending chunk 1367 of 1392 (with 20 bytes)
Sending chunk 1368 of 1392 (with 20 bytes)
Sending chunk 1369 of 1392 (with 20 bytes)
```

Upload completed

Reboot device?

```
Sending chunk 1382 of 1392 (with 20 bytes)
Sending chunk 1383 of 1392 (with 20 bytes)
Sending chunk 1384 of 1392 (with 20 bytes)
Sending chunk 1385 of 1392 (with 20 bytes)
Sending chunk 1386 of 1392 (with 20 bytes)
Sending chunk 1387 of 1392 (with 20 bytes)
Sending chunk 1388 of 1392 (with 20 bytes)
Sending chunk 1389 of 1392 (with 20 bytes)
Sending chunk 1390 of 1392 (with 20 bytes)
Sending chunk 1391 of 1392 (with 20 bytes)
Sending chunk 1392 of 1392 (with 17 bytes)
send SUOTA END command
Upload completed
```

Figure 19: Successful update screen

STEP 8: Verify that the new software is running on the SmartTag device

Repeat steps 4 and 5 to verify that the DIS screen shows the firmware and software version of the new software.

11 Revision history

Revision	Date	Description
1.0	09-May-2014	Initial version.
2.0	19-Dec-2014	New version of SmartTag reference design (SmartTag v2). ● Updated hardware information ● Added SUOTA/SPOTA service ● Added SUOTA android application (Appendix A)

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.

Disclaimer

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

United Kingdom (Headquarters)	North America	Singapore
Dialog Semiconductor PLC	*Dialog Semiconductor Inc.*	*Dialog Semiconductor Singapore*
Phone: +44 1793 757700	Phone: +1 408 845 8500	Phone: +65 64 849929
Germany	Japan	China
Dialog Semiconductor GmbH	*Dialog Semiconductor K. K.*	*Dialog Semiconductor China*
Phone: +49 7021 805-0	Phone: +81 3 5425 4567	Phone: +86 21 5178 2561
The Netherlands	Taiwan	Korea
Dialog Semiconductor B.V.	*Dialog Semiconductor Taiwan*	*Dialog Semiconductor Korea*
Phone: +31 73 640 8822	Phone: +886 281 786 222	Phone: +82 2 3469 8291
Email:	Web site:	
enquiry@diasemi.com	www.dialog-semiconductor.com	