User manual

DA14580/581/583 Creation of a secondary bootloader

UM-B-012

Abstract

This document provides an overview of the booting sequence of the DA14580/581/583 and it describes the implementation steps for the development of a secondary bootloader application. An extension of the secondary bootloader to support a dual image booting scheme is also presented.

Contents

Figures

Tables

1 Terms and definitions

CRC	Cyclic Redundancy Check
EEPROM	Electrically Erasable Programmable Memory
GPIO	General Purpose Input Output
LE	Little Endian
NVDS	Non-Volatile Data Storage
OTP	One Time Programmable (memory)
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SUOTA	Software Update Over The Air
SW	SoftWare
UART	Universal Asynchronous Receiver/Transmitter
URX	UART Receive port
UTX	UART Transmit port

2 References

[1] DA14580 Data sheet, Dialog Semiconductor.

[2] AN-B-001, DA1458x Booting from serial interfaces, Application note, Dialog Semiconductor.

[3] UM-B-014, DA14580/581 Bluetooth® Smart Development Kit - Expert, User manual, Dialog Semiconductor.

[4] UM-B-004, DA14580/581/583 Peripheral drivers, User manual, Dialog Semiconductor.

[5] UM-B-010, DA14580/581/583 Proximity application, User manual, Dialog Semiconductor.

[6] AN-B-023, DA14580 Interfacing with external memory, Application note, Dialog Semiconductor.

[7] AN-B-003, DA14580 Software Patching over the Air (SPOTA), Application note, Dialog Semiconductor.

[8] UM-B-007, DA14580/581 Software Patching over the Air (SPotA), User manual, Dialog Semiconductor.

[9] DA14581 Data sheet, Dialog Semiconductor.

[10] DA14583 Data sheet, Dialog Semiconductor.

3 Introduction

Firstly this document provides an overview of the booting sequence of DA14580/581/583. It is also recommended reading AN-B-001 [2] which describes the booting procedures supported by the DA1458x ROM code.

Secondly the steps for creating a secondary bootloader application are described. The secondary bootloader allows the DA14580/581 to boot from an external SPI Flash memory, an I2C EEPROM or a UART interface. It can be used to replace the ROM bootloader in case a faster booting sequence is needed. The DA14583 is shipped with a secondary bootloader burnt in OTP which supports booting from a UART interface or the internal SPI flash memory or booting to an advanced bootloader that may be optionally written in a separate OTP memory area.

Thirdly an extension of the secondary bootloader based on dual images is also presented in this document. The dual image bootloader is a building block for a Software Update Over The Air (SUOTA) enabled application (see [7] and [8]). On DA14583 SUOTA can be supported by using a dual image bootloader as the advanced bootloader.

Finally the secondary bootloader code structure, the steps for testing it and a methodology for measuring the booting time are presented.

4 How the DA14580/581 boots

The DA14580/581 operates in two modes, namely Normal Mode and Development/Calibration Mode, hereafter addressed as DevMode. See [1] and [2].

At power up or reset of the DA14580/581, the primary boot code (ROM code) will check whether the OTP memory has been programmed by checking if the “Application Programmed Flag #1” and the “Application Programmed Flag #2” are equal to 0x1234A5A5 and 0xA5A51234 respectively. When this is the case the DA14580/581 enters Normal Mode. It then proceeds with mirroring the OTP contents to System RAM and program execution.

When the OTP memory is not programmed, the DA14580/581 enters DevMode. It scans a predefined number of pins to communicate with external devices, using the three interfaces available on chip:

- UART
- SPI
- I2C

The application start-up time in DevMode is longer than Normal Mode due to the fact that in DevMode all available interfaces are sequentially scanned. The start-up time in Normal Mode is a few milliseconds, while it is several tenths of milliseconds in DevMode (see [2]).

For applications that need to be loaded from an external interface and at the same time require a short boot time, a dedicated *secondary bootloader* can be developed that will skip the long sequence of scanning interfaces. Such a secondary bootloader will start communicating directly on a selected interface to download the application software into SRAM. This secondary bootloader must reside in OTP, making it operate in Normal Mode instead of DevMode.

5 How the DA14583 boots

The DA14583 [10] comes with a factory burned OTP containing the DA14583 specific bootloader which supports:

- loading an application from several UART interfaces
- loading an application from the on-chip SPI flash memory
- passing control to an advanced bootloader that can be optionally burned in the OTP memory by the customer.

At DA14583 power up or reset the primary BootROM code detects that the DA14583 bootloader is programmed in the OTP memory by checking if the “Application Programmed Flag #1” and the “Application Programmed Flag #2” are equal to 0x1234A5A5 and 0xA5A51234. Because DA14583 comes with the secondary loader pre-programmed these two application flags are set and the BootROM code will proceed with mirroring the OTP contents to System RAM and execution of the DA14583 bootloader (the procedure is identical to booting a DA14580/581 as explained in section 4).

The DA14583 bootloader first checks if an advanced bootloader is also present in the OTP memory. If this is the case then it passes control to the advanced bootloader otherwise it proceeds to sequential scanning of the same four UART interfaces as described in application note AN-B-001 [2]. If an application binary is successfully loaded from one of the UART interfaces then control is passed to the application otherwise the DA14583 bootloader proceeds to check the internal SPI slave flash memory according to AN-B-001 [2]. If an application binary is successfully loaded from the internal flash then control is passed to the application otherwise the DA14583 bootloader restarts sequential scanning of the UART interfaces and then the internal SPI flash interface.

Example code of the secondary loader is available in the SDK and described in section 6.

5.1 Booting to an advanced bootloader

The DA14583 secondary bootloader detects if an advanced bootloader is present by checking the 32-bit word at OTP header offset 0x7F10 [10]. This field designates the length and OTP offset of the advanced bootloader binary image that may also be burned in OTP memory.

Table 1: DA14583 advanced bootloader configuration field

OTP Offset	Parameter	Description	Example value	Example value meaning	Example value as stored in OTP
0x47F10	*Advanced Bootloader Offset and Length*	*Byte[3] = length MSB,* *Byte[2] = length LSB,* *Byte[1] = offset MSB,* *Byte[0] = offset LSB*	0x12345ABC	length = 0x1234 offset = 0x5ABC	0x47F10 = 0xBC 0x47F11 = 0x5A 0x47F12 = 0x34 0x47F13 = 0x12

The DA14583 bootloader first reads the size and OTP offset of the advanced bootloader and then performs the following checks:

1. Check that the advanced bootloader image offset is non-zero.
2. Check that the advanced bootloader image size is non-zero.
3. Check that offset >= 0x2000 and (offset + size) < 0x7000.

If all checks succeed then the DA14583 bootloader loads the advanced bootloader image at SysRAM address 0x20000000 and executes the image. Otherwise it proceeds to the next step which is described in the next section.

5.2 Booting from a UART interface

The DA14583 secondary bootloader scans the UART configurations of Table 2 sequentially and according to AN-B-001 [2].

Table 2: DA14583 bootloader UART configurations

UART Configuration	UART TX Pin	UART RX Pin	Baud Rate
0	P00	P01	57600 bps
1	P02	P03	115200 bps
2	P04	P05	57600 bps
3	P06	P07	9600 bps

The DA14583 bootloader executes the following steps:

1. For i = 0..3 do
   a. Select the i-th UART configuration.
   b. If UART RX pin is high
      i. Try to download an application image according to the AN-B-001 UART protocol.
      ii. If success then run the application
2. Proceed to check the on-chip SPI flash memory as described in the next section.

5.3 Booting from the on-chip SPI flash memory

The DA14583 bootloader tries to load an application image from the on-chip SPI flash memory according to the following steps:

1. Initialize SPI interface. The DA14583 pins which are assigned to the on-chip SPI flash memory are configured as follows:
   SPI CS: P2_3
   SPI CLK: P2_0
   SPI MOSI: P2_9
   SPI MISO: P2_4
2. Release the flash from power down.
3. Check if a valid AN-B-001 header exists at flash offset 0 (also see section 6.2).
4. If true then
   Load the application image from flash memory
   Set the flash in power down
   Run the application
5. Otherwise
   Set the flash in power down
   Restart the UART booting phase described in the previous section.

6 DA14580/581/583 secondary bootloader application

In this section three use cases of the secondary bootloader are described:

6. Booting form the UART interface.
7. Booting from an external I2C EEPROM.
8. Booting from an external SPI slave Flash memory.

The DA14580/581 ROM bootloader also supports booting from an external SPI master device. In this case the DA14580/581 acts as an SPI slave. The current document does **not** describe a secondary bootloader for this booting method.

The secondary bootloader concept is also applicable to DA14583 where it can play the role of the advanced bootloader.

The secondary bootloader application code is located in the SDK under *\tools\secondary_bootloader*.

6.1 Booting from UART

Booting from UART can be enabled by defining `UART_SUPPORTED` in header file *bootloader.h*. This option can be used together with one of the other two booting options (external SPI slave Flash memory or I2C EEPROM). Under such a configuration DA14580/581 is able to:

- Run an application stored in the external non-volatile memory, or
- Boot from an external non-volatile memory using a dual-image bootloader.
- Download an external memory programming application over UART. Hence the *SmartSnippets* toolkit can be used to program the external SPI Flash memory or I2C EEPROM.

The secondary bootloader application reads the UART RX pin status using `GPIO_GetPinStatus()` and when it is logic HIGH, it starts the booting procedure from the UART interface.

The protocol for booting from the UART is the same as that of ROM boot code (see [2]) and it is implemented in function `FwDownload()` in file *uart_booter.c*. The event sequence is shown in Table 3.

The booting sequence starts with the DA14580/581 UART TX pin transmitting 0x02 (Start TX, STX). The external device is expected to respond with 0x01 (Start of Header, SOH), followed by two bytes that define the length of the code to be downloaded (LS byte first). The DA14580/581 responds with 0x06 (ACK), when three bytes were received and SOH was identified, or with 0x15 (NACK) in case of an error.

At this point the connection has been successfully established and the application code will start to be downloaded. The next N bytes are received and placed into the System RAM.

Following the completion of the required code bytes, the boot code will calculate the CRC and send it over the UART. The CRC is calculated by XORing every successive byte with the previous value. The initial CRC value is 0x00. The booting sequence ends successfully by reading a 0x06 (ACK) at the UART RX pin. In case of a CRC error a 0x15 (NACK) will be returned.

Table 3: Transmission sequence

DA14580/581	Data direction	External device
0x02 (Start TX, STX)	→	
	←	0x01 (Start of Header, SOH)
	←	LEN_LSB
	←	LEN_MSB
0x06 (ACK) or 0x15 (NACK)	→	

Copy the data to System RAM	←	#bytes sent:
		N = LEN_MSB * 8 + LEN_LSB

Calculate & Send the CRC	→	Read/Check the CRC
Branch to 0x20000000 (System RAM)	←	0x06 (ACK) or 0x15 (NACK)

During the final step of the bootloader code, the UART GPIO pins are initialised to their default state. The last action is to execute the application code that has been loaded in System RAM. Depending on the base address of the application code, either address 0 is mapped to ROM and a branch is performed to System RAM or address 0 is mapped to System RAM and a SW reset is applied.

6.2 Booting from SPI Flash memory

The secondary bootloader application initialises the SPI interface and the SPI Flash memory, when no UART is detected and the options `SPI_FLASH_SUPPORTED` and `SUPPORT_AN_B_001` are defined in the header file *bootloader.h*. Then it checks whether the header described in Table 7 of AN-B-001 [2] is present, by checking the first two bytes of the header for the signature (0x70, 0x50). See Table 4. When a valid header is detected, the bootloader copies a number of bytes equal to the code size into System RAM and starts the user application.

Table 4: SPI header

Byte #	Field
0	Signature (0x70)
1	Signature (0x50)
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Code size (MS byte)
7	Code size (LS byte)
8 to …	Code data […]

6.3 Booting from I2C/EEPROM

The secondary bootloader application initialises the I2C interface and the I2C/EEPROM memory, when no UART is detected and the options `EEPROM_FLASH_SUPPORTED` and `SUPPORT_AN_B_001` are defined in the header file *bootloader.h*. Then it checks whether the header described in Table 9 of AN-B-001 [2] is present by checking the first two bytes of the header for the signature (0x70, 0x50). See Table 5. When a valid header is detected it copies a number of bytes equal to the code size into System RAM. Finally, the bootloader calculates the CRC checksum of the code data according to [2]. When the calculated checksum matches the CRC field of the header, it starts the user application.

Table 5: EEPROM header

Byte #	Field
0	Signature(0x70)
1	Signature (0x50)
2	Code size (MS byte)
3	Code size (LS byte)
4	CRC
5 to 31	Empty bytes
32 to …	Code data […]

7 DA14580/581/583 dual image bootloader

The dual image bootloader is an extension of the secondary bootloader that supports a dual image booting scheme, which is used in Software Update Over The Air (SUOTA) applications for updating the product firmware in the field.

The dual image bootloader specifies how two application images can be stored in an external SPI Flash memory or I2C EEPROM and defines how the current image is determined. It also supports checksum verification and decryption of an AES encrypted image.

The secondary bootloader application is configured to operate in dual image mode by undefining the option `SUPPORT_AN_B_001` in the header file *bootloader.h*. Application image decryption support is enabled by defining the option `AES_ENCRYPTED_IMAGE_SUPPORTED` to 1 in the same header file.

The dual image bootloader concept is also applicable to DA14583 where it can play the role of the advanced bootloader.

7.1 Non-volatile memory map

The non-volatile memory map to meet the needs of the dual image bootloader is shown in Figure 1.

The first part is the dual image bootloader itself. This part is mandatory only when the dual image bootloader is required to run from the external non-volatile memory, e.g. when the DA14580/581 OTP is not programmed. In this case, a header according to AN-B-001 [2] must be prepended before the bootloader image. When the dual image bootloader is stored in OTP memory for faster booting, this part should be omitted.

The images with the corresponding headers are stored at offset #1 and offset #2, which are defined in the product header. The product header is suggested to be programmed in the last sector of the non-volatile memory.

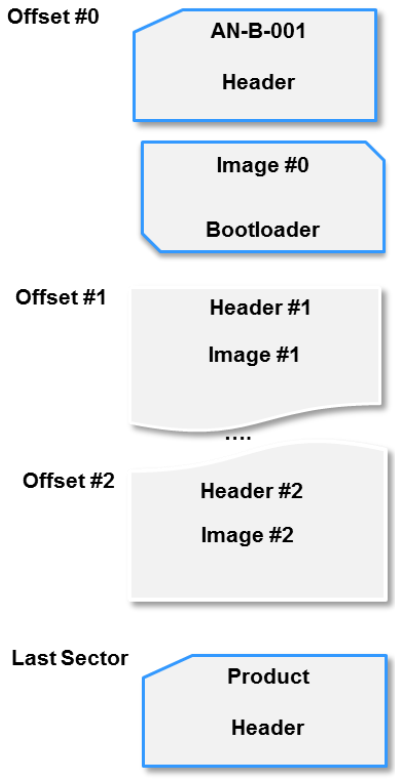


Figure 1: Non-volatile memory map

Note 1 The secondary bootloader code provided in the SDK assumes that the product header is programmed at offset 0x1F000.

Note 2 The bootloader is stored either in the first sector of the non-volatile memory according to AN-B-001 or in OTP Flash memory.

The **product header** defines the offsets of the two firmware images stored in the non-volatile memory. It is programmed on the production line and the corresponding Flash sector may be write-protected when this is supported by the Flash memory characteristics.

The product header contains the following fields (see Table 6):

- *signature (0x70, 0x52):* A magic number identifying the product header.
- *version:* Two bytes reserved for the versioning of the product header.
- *offset #1:* Defines the address of the first image stored in non-volatile memory. Four bytes in Little Endian format.
- *offset #2:* Defines the address of the second image stored in non-volatile memory. Four bytes in Little Endian format.
- *BD address:* Contains the public BD address for DA14583 devices. Six bytes in Little Endian format. The DA14583 platform initialization code sets this as the public address of the BLE stack.

Table 6: Product header format

Byte #	Field
0	signature (0x70)
1	signature (0x52)
2 to 3	version
4 to 7	offset #1 (LE format)
8 to 11	offset #2 (LE format)
12 to 17	BD address
17 to …	reserved (Note 3)

Note 3 The product header may contain more information or configuration settings, such as NVDS data, XTAL16 trim settings. The use of these parameters is explained in [6].

The **image header** includes the following fields (see Table 7):

- *signature (0x70, 0x51):* A magic number identifying the image header.
- *validflag:* Indicates whether the image is valid (0xAA: valid, other values: invalid)
- *imageid:* An increment counter which identifies the active image.
- *code_size:* Defines the size of the firmware image (in bytes).
- *CRC:* Contains the CRC32 checksum calculated over the image data.
- *version:* A 16-byte string used for the image version.
- *timestamp:* Defines the image creation time based on seconds since standard epoch of 1/1/1970.
- *encryption:* Indicates whether the image is encrypted or not (0: not encrypted, 1: encrypted).

Table 7: Image header format

Byte #	Field
0	signature (0x70)
1	signature (0x51)
2	validflag
3	imageid
4 to 7	code_size (LE format)
8 to 11	CRC (32-bits, LE format)
12 to 27	version (16 byte string)
28-31	timestamp (LE format)
32	encryption

7.2 Booting sequence

During the booting phase, the dual image bootloader performs the following actions:

9. Check the product header and read the offsets of the two images stored in the non-volatile memory.
10. Read the contents of the two image header to find the valid image with the highest *imageid* and load it into System RAM.
11. Decrypt the loaded image when it is encrypted.
12. Calculate the CRC32 checksum of the code data and verify that it matches the value of the CRC header field.
13. Execute the loaded image.

7.3 How to prepare the non-volatile memory using only *SmartSnippets*

The *SmartSnippets* tool can be used for writing the bootloader to non-volatile memory or OTP and the product header and the dual images to non-volatile memory.

For example, the SPI memory preparation is done in three steps, which are described below:

1. Program the product header (0x1F000):
   a. Create a text file to describe the product header as shown below:

```
2       Signature       MagicNumber
2       Version         VersionNumber
4       Offset1         Offset_image_1
4       Offset2         Offset_image_2
```

   b. Load it using the Memory header option of the *SmartSnippets* toolkit and enter the following values in the product header fields:

```
Signature: 0x70, 0x52
Version: 1234
Offset1: 00800000 (it corresponds to offset 0x8000)
Offset2: 00300100 (it corresponds to offset 0x13000)
```

   c. Write these values at offset 0x1F000 of the external SPI Flash memory. For more information on how to create the product header refer to the HELP menu of the *SmartSnippets* toolkit.
2. Program the image binaries:
   a. Build the application and convert the HEX to BIN file. Use the *mkimage* tool (located in *\tools\mkimage* of the SDK distribution) to convert the BIN to IMG file. For more information on using this tool, refer to Appendix A or run *mkimage* without arguments.
   b. Load the *image.img* file using the SPI Flash Programmer option of the *SmartSnippets* toolkit and write it to SPI Flash memory at address 0x8000 (image #1) and 0x13000 ((image #2).
3. Program the bootloader:
   a. Build the secondary_bootloader in dual image mode and write the HEX file at address 0x0 using the SPI Flash Programmer option of the *SmartSnippets* toolkit. Select ‘Yes’ to the question whether to make the SPI memory bootable.

7.4 How to prepare the non-volatile memory using *mkimage*

The *mkimage* tool can be used to prepare the entire contents of the external non-volatile memory as a binary file. This file can then be written to the non-volatile memory using the *SmartSnippets* toolkit. More information on the *mkimage* tool can be found in Appendix A .

The steps to prepare an SPI Flash memory are as follows, assuming that the application firmware image is contained in file *app.bin*:

1. Prepare the application image file *app.img* with:

   ```
   mkimage.exe app.bin app.h app.img enc
   ```

   The file *application.h* is the application version header file that must be formatted as follows:

   ```
   #define DA14580_SW_VERSION "v_1.0"
   #define DA14580_SW_VERSION_DATE "2014-10-3 12:34 "
   #define DA14580_SW_VERSION_STATUS "REPOSITORY VERSION"
   ```

   The `enc` command line flag indicates that the application firmware will be encrypted.

2. Construct the entire SPI Flash memory contents (file *mem.bin*) with:

   ```
   mkimage.exe multi spi loader.bin app.img 0x8000 app.img 0x13000 0x1F000 mem.bin
   ```

   The command places the product header at offset 0x1F000, image #1 at 0x8000 and image #2 at 0x13000. Both images contain the same application firmware.

   The file *loader.bin* is the dual image loader itself and it can be omitted when it is already written in OTP memory. When file *loader.bin* is included, it is prepended with an AN-B-001 header [2] at offset 0 of the output *mem.bin* file.

3. Write the file *mem.bin* into SPI Flash memory at offset 0 using SmartSnippets. Select “No” to the question whether to make the SPI memory bootable, as it is already bootable.

8 Application description

8.1 File structure

The file structure of the secondary_bootloader project is shown in Figure 2.

System initialisation files are located in folder *tools/secondary_bootloader/startup*:

- *startup_CMSDK_CM0.s:* The startup file for the ARM Cortex-M0. Contains the stack and heap configuration and the vector table.
- *system_CMSDK.c:* Contains the functions needed to initialise the system and update the `SystemFrequency` variable.
- *bootloader.sct:* The scatter-loading description file.
- *sysram.ini:* The Keil debugger initialisation script.

Application (.c) files are located in the folder *tools/secondary_bootloader/src*:

- *main.c:* Contains the main function, the system initialisation function and the main loop of the application.
- *bootloader.c:* Contains the functions for booting from SPI and EEPROM and the implementation of the dual image bootloader.
- *uart_booter.c:* Contains the functions for booting from UART.
- *timer.c:* Contains the timer functions. A software timer is used for the timeouts required by the UART boot protocol.
- *spi_commands.c:* Contains the additional commands for accessing the SPI Flash memory.
- *crc32.c:* Contains the CRC32 checksum calculation algorithm.
- *sw_aes.c:* Contains the software implementation of the AES encryption. The encryption key and IV are hardcoded in the application code.

Application (.h) files are located in the folder *tools/secondary_bootloader/includes*:

- *periph_setup.h:* Contains the configuration settings for the peripherals (UART, SPI, SPI Flash) used by the secondary bootloader application.
- *bootloader.h:* Contains the application configuration settings. For details see section 8.2.

Driver (.c) files for the peripheral interfaces are located in folder *dk_apps/src/plf/refip/src/driver.* Detailed information about the drivers can be found in [4].

- *spi.c:* Driver for the SPI interface.
- *spi_flash.c:* Driver for an external SPI Flash memory.
- *gpio.c:* Driver for the GPIO interface.

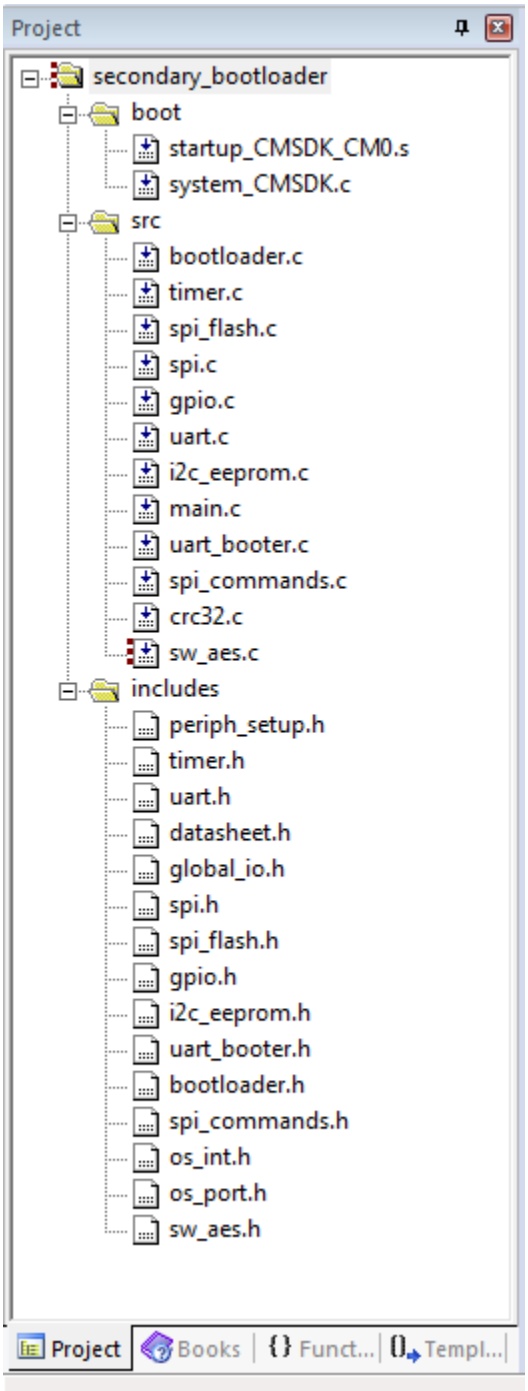


Figure 2: File structure

8.2 Compilation and configuration settings

The main compilation and configurations settings are included in the header files *bootloader.h* and *periph_setup.h*.

- `AES_ENCRYPTED_IMAGE_SUPPORTED`: This setting must be defined only when the image for the dual image bootloader is encrypted. Must be disabled for the secondary bootloader.
- `UART_SUPPORTED`: This setting defines whether the UART is enabled for firmware downloading. It is supported by both applications: secondary and dual image bootloader.
- `SPI_FLASH_SUPPORTED`, `EEPROM_FLASH_SUPPORTED`: These settings define the external Flash memory type that is supported by the product. Only one must be defined.

- `SUPPORT_AN_B_001`: This setting defines that the application will be compiled as secondary bootloader.

The configuration settings for the peripherals are contained in header file *periph_setup.h*.

```
// Select EEPROM characteristics
#define I2C_EEPROM_SIZE   0x20000         // EEPROM size in bytes
#define I2C_EEPROM_PAGE   256             // EEPROM's page size in bytes
#define I2C_SLAVE_ADDRESS 0x50            // Set slave device address
#define I2C_SPEED_MODE    I2C_FAST        // 1: standard mode (100 kbits/s), 2: fast
                                             mode (400 kbits/s)
#define I2C_ADDRESS_MODE  I2C_7BIT_ADDR   // 0: 7-bit addressing, 1: 10-bit addressing
#define I2C_ADDRESS_SIZE  I2C_2BYTES_ADDR // 0: 8-bit memory address, 1: 16-bit memory
                                             address, 3: 24-bit memory address
// SPI Flash settings
// SPI Flash Manufacturer and ID
#define W25X10CL_MANF_DEV_ID (0xEF10)     // W25X10CL Manufacturer and ID
#define W25X20CL_MANF_DEV_ID (0xEF11)     // W25X10CL Manufacturer and ID
// SPI Flash options
#define W25X10CL_SIZE 131072              // SPI Flash memory size in bytes
#define W25X20CL_SIZE 262144              // SPI Flash memory size in bytes
#define W25X10CL_PAGE 256                 // SPI Flash memory page size in bytes
#define W25X20CL_PAGE 256                 // SPI Flash memory page size in bytes
#define SPI_FLASH_DEFAULT_SIZE 131072     // SPI Flash memory size in bytes
#define SPI_FLASH_DEFAULT_PAGE 256        // SPI Flash memory page size in bytes
//SPI initialisation parameters
#define SPI_WORD_MODE  SPI_8BIT_MODE
#define SPI_SMN_MODE   SPI_MASTER_MODE
#define SPI_POL_MODE   SPI_CLK_INIT_HIGH
#define SPI_PHA_MODE   SPI_PHASE_1
#define SPI_MINT_EN    SPI_NO_MINT
#define SPI_CLK_DIV    SPI_XTAL_DIV_2
// UART GPIOs assignment
#define UART_GPIO_PORT  GPIO_PORT_0
#define UART_TX_PIN     GPIO_PIN_4
#define UART_RX_PIN     GPIO_PIN_5
#define UART_BAUDRATE   baudrate_57K6
// SPI GPIO assignment
#define SPI_GPIO_PORT   GPIO_PORT_0
#define SPI_CS_PIN      GPIO_PIN_3
#define SPI_CLK_PIN     GPIO_PIN_0
#define SPI_DO_PIN      GPIO_PIN_6
#define SPI_DI_PIN      GPIO_PIN_5
// EEPROM GPIO assignment
#define I2C_GPIO_PORT   GPIO_PORT_0
#define I2C_SCL_PIN     GPIO_PIN_2
#define I2C_SDA_PIN     GPIO_PIN_3
```

W25X10CL SPI Flash memory devices are supported. The W25X10CL arrays are organised into 512 programmable pages of 256 bytes each. Up to 256 bytes can be programmed at a time. The W25X10CL has 32 erasable sectors of 4 kB, 4 erasable 32 kB blocks and 2 erasable 64 kB blocks respectively. W25X20CL SPI Flash memory devices are also supported.

Other SPI Flash memory types can be supported by changing the above configuration settings (`SPI_FLASH_DEFAULT_SIZE`, `SPI_FLASH_DEFAULT_PAGE`, etc.).

GPIO Port 0 is used by default as it is supported by all DA14580/581 types (WLCSP34, QFN40 and QFN48 packages).

GPIO pins 4 and 5 are assigned to UART TX and RX respectively.

GPIO pins 0, 3, 5 and 6 are assigned to SPI CS, CLK, DI and DO respectively.

The conflict for GPIO pin 5 is solved by sequential access from the UART and SPI interfaces.

8.3 System initialisation

The secondary bootloader application executes in the retention memory, allowing the application code to be loaded into the System RAM. The secondary bootloader performs the following actions:

1. Switch to the 16 MHz crystal oscillator, when the system is not already running on XTAL16M.

```
if ((GetWord16(CLK_CTRL_REG) & RUNNING_AT_XTAL16M) == 0)
{
   while( (GetWord16(SYS_STAT_REG) & XTAL16_SETTLED) == 0 );
     // wait for XTAL16 to settle
    SetBits16(CLK_CTRL_REG , SYS_CLK_SEL ,0);
     // switch to XTAL16
    while( (GetWord16(CLK_CTRL_REG) & RUNNING_AT_XTAL16M) == 0 );
     // wait for actual switching
}
```

2. Set the system clock and memory configuration as shown below:

```
SetWord16(CLK_AMBA_REG, 0x00); //fastest
SetBits32(GP_CONTROL_REG, EM_MAP, 7);
SetBits16(PMU_CTRL_REG, RETENTION_MODE, 0xF);
```

3. In the main function, the secondary loader disables the Watch dog timer, sets all the peripherals in active mode and waits until the system is ready:

```
SetWord16(SET_FREEZE_REG,FRZ_WDOG);                 // disable Watch Dog
SetBits16(PMU_CTRL_REG, PERIPH_SLEEP,0);            // exit peripheral power down
while (!(GetWord16(SYS_STAT_REG) & PER_IS_UP));     // power up peripherals domain
```

4. Boot from UART when the UART booting option is enabled and the UART RX pin is logic HIGH. Otherwise boot from SPI or I2C.

9 Getting started

This section describes how to program the secondary bootloader into the OTP memory, program an application example (integrated processor Proximity Reporter) into the SPI Flash memory and measure the system booting time. A comparison with a normal booter (ROM booter in Development Mode) is also provided.

The *SmartSnippets* toolkit provides tools for external SPI Flash and OTP memory programming.

9.1 Building the application and secondary bootloader images

Build the integrated processor Proximity Reporter application image according to the "DA14580/581/583 Proximity application" user manual [5].

Build the secondary bootloader image for DA14580/581 or the advanced bootloader image for DA14583 according the following steps:

1. Open the Secondary Bootloader project:
   - For Keil 4:
     \tools\secondary_bootloader\secondary_bootloader.uvproj
   - For Keil 5:
     \tools\secondary_bootloader\secondary_bootloader.uvprojx.
2. Configure the project according to section 8.2.
3. Compile the project to generate the executable file *secondary_bootloader.hex*.

9.2 Writing application HEX file into SPI Flash memory

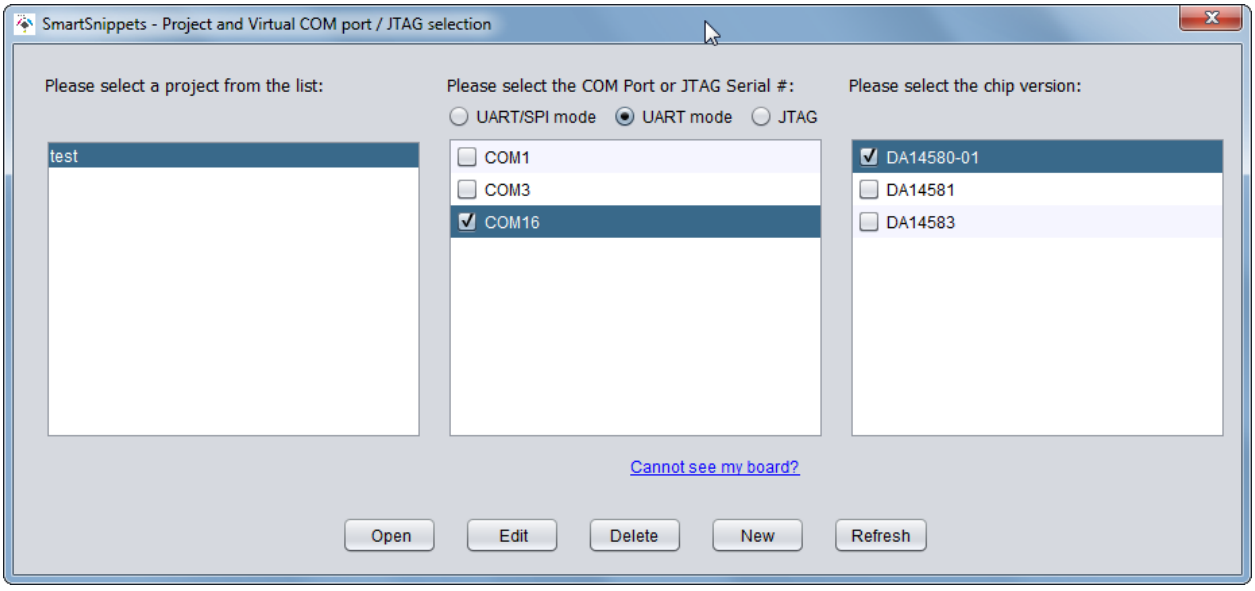
The *SmartSnippets* SPI Flash Programmer tool is used for downloading an application image file to:

- An external SPI Flash memory connected to DA14580/581.
- The on-chip SPI Flash memory of DA14583 (at least *SmartSnippets* v3.8 is required).

The following

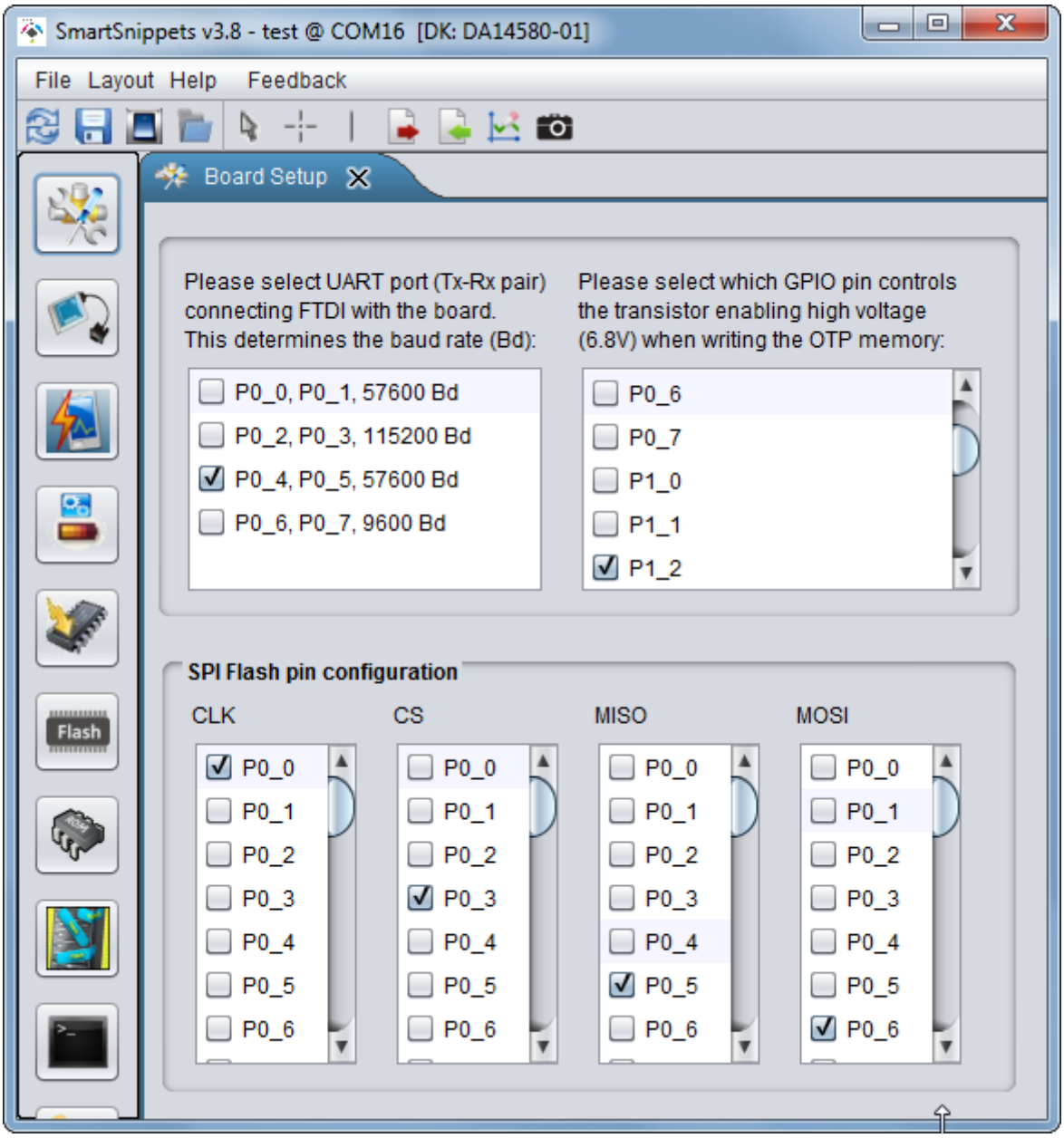
instructions demonstrate how to accomplish this using *SmartSnippets* v3.8 in UART mode.

1. Open *SmartSnippets* and select the chip version.



2. Open the "Board Setup" tool and select the appropriate UART and SPI flash pin configuration. Notice that the SPI flash pin configuration is automatically initialized to match:
   a. the SPI flash configuration used in Dialog development kits in case of a DA14580/581.

b. the on-chip SPI flash memory in case of a DA14583.



3. Open the “SPI Flash Programmer” tool (Figure 3), select the application image file and burn it at SPI flash memory offset 0. When asked whether to make the SPI Flash memory bootable, there are two options to consider depending on the chip version and bootloader configuration:
   a. **Bootable SPI Flash:** SmartSnippets will automatically add an AN-B-001 header at offset 0 of the SPI flash memory. The secondary bootloader will copy only the number of bytes defined in the SPI Flash header.

      Attention! It is mandatory to select this option when using a DA14583 with the default factory burned secondary bootloader since this bootloader requires that an AN-B-001 header exists at offset 0.
   b. **Non-bootable SPI Flash:** SmartSnippets will not add an AN-B-001 header at offset 0 of the SPI flash memory. The secondary bootloader will copy 32 kB from SPI Flash memory starting at offset 0x0. Using this setup the maximum booting time can be measured.

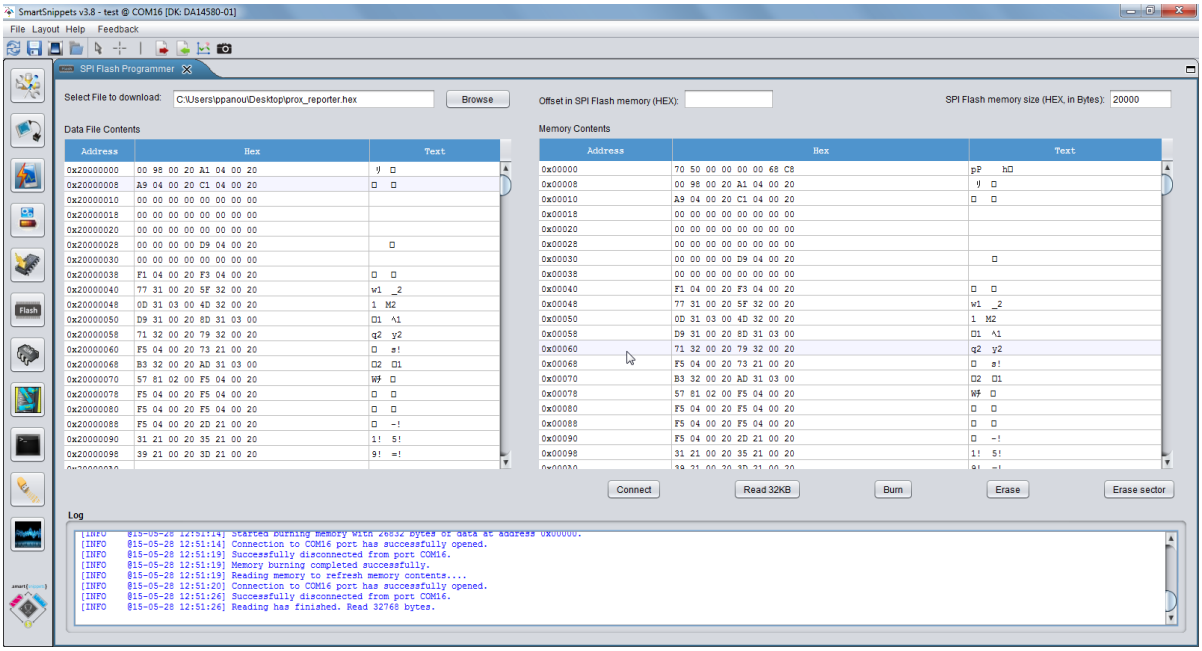


Figure 3: SPI Flash Programmer

9.3 Writing bootloader HEX file into OTP memory

The *SmartSnippets* OTP Programmer tool enables downloading the default firmware into the System RAM and writing a user-defined HEX or BIN file into the OTP memory. The tool can be used for: writing:

- A secondary bootloader in the OTP memory of a DA14580/581.
- An advanced bootloader in the OTP memory of a DA14583 (at least *SmartSnippets* v3.8 is required).

Figure 4 shows the main screen of the OTP Programmer.

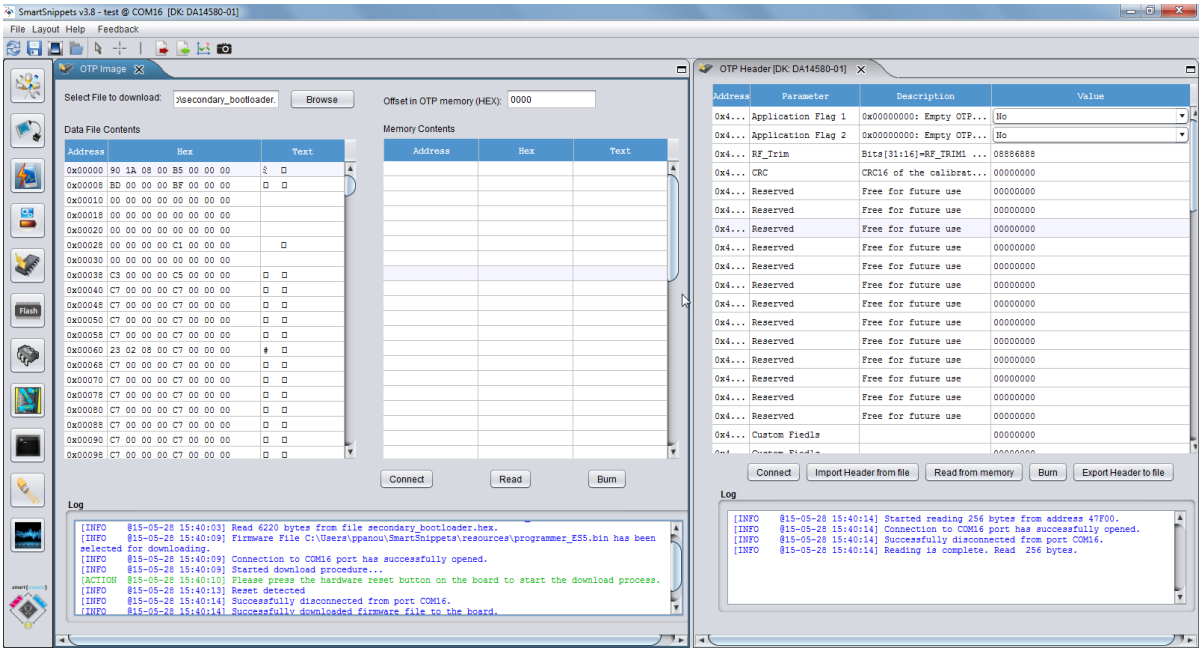


Figure 4: OTP Programmer

The following steps are required for writing the executable *secondary_bootloader.hex* into OTP memory using *SmartSnippets* v3.8 in UART mode:

1. Open *SmartSnippets* and select the chip version.
2. Open the “Board Setup” tool and select the appropriate UART configuration.
3. Open the “OTP Programmer” tool.
4. If we are writing a
   a. secondary bootloader for a **DA14580/581** then:
      i. Write the secondary_*bootloader.hex* into the OTP memory at offset 0.
      ii. Enable Application Flag 1 and Application Flag 2, set DMA Length and write the OTP header.
   b. advanced bootloader for **DA14583** then:
      i. Write the secondary_*bootloader.hex* into the OTP memory at an offset >= 0x2000 since the OTP offsets 0 - 0x1FFF are occupied by the factory burned DA14583 bootloader. SmartSnippets will automatically program the DA14583 specific “Advanced Bootloader Size and Offset” field [10] in the OTP header.

Note: When a Dialog hardware development kit [3] is used for OTP programming, make sure that jumpers J12 and J25 are populated to enable VPP control and UART communications, respectively. When a different hardware configuration is used, make sure that 6.8 V is applied to pin VPP.

9.4 Measuring the booting time

This section describes the procedure for measuring the booting time of an application stored in SPI Flash memory with the secondary bootloader , as described in this document. This booting time is compared with the time required with the normal bootloader stored in ROM.

The Power Profiler tool of the *SmartSnippets* toolset is used to measure the time between power up and the first advertising event.

The measurements have been carried out for both the DA14580 and the DA14581.

9.4.1 DA14580

Figure 5 illustrates the booting time of the Proximity Reporter application with the secondary loader on a DA14580 device. The time required until the first advertisement is 158.8 ms. The data transfer from the SPI Flash takes 36 ms and the time required until the first application entry point is 145 ms.

Note: 32 kB data is transferred instead of the actual application data size.

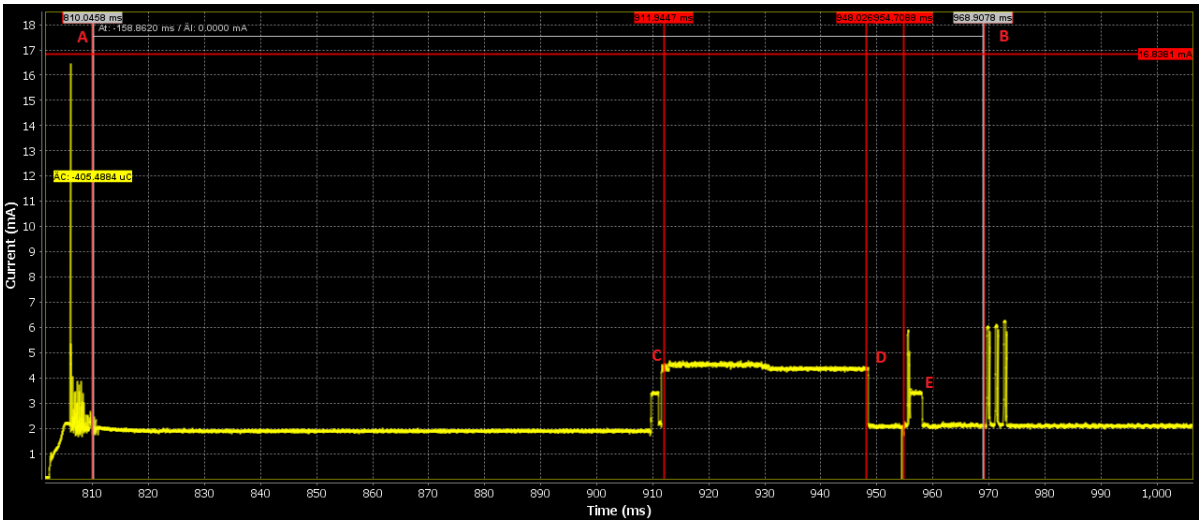


Figure 5: DA14580: Booting from SPI Flash memory using the secondary bootloader

Legend:

A	DA14580 power up
B	First advertising event
C, D	Start and end of application data transfer from SPI Flash memory to System RAM
E	First application entry point: `main()`

Figure 6 shows the booting time of the Proximity Reporter application with the normal ROM booting sequence. The time required until the first advertisement is 309.47 ms. The data transfer from the SPI Flash memory takes 100 ms and the time required until the first application entry point is 295 ms.

The secondary bootloader achieves a faster boot time, because it skips the scanning sequence of the Development Mode, while the SPI operation is optimised for the specific SPI Flash device used.

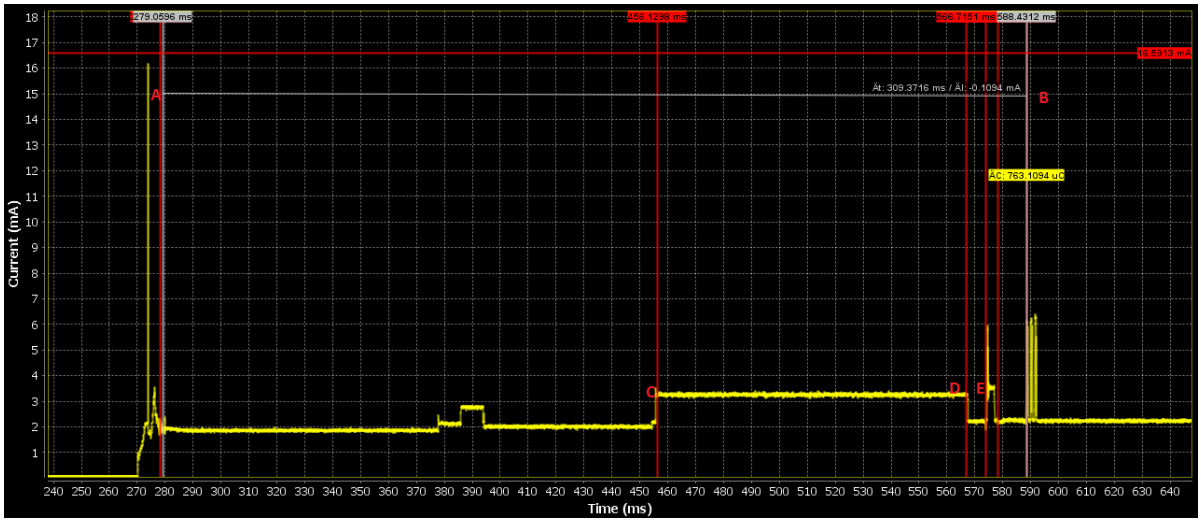


Figure 6: DA14580: Booting from SPI Flash memory using the ROM bootloader

9.4.2 DA14581

Figure 7 shows the booting sequence of the Proximity Reporter application when it runs from OTP memory on a DA14581 device. The booting time up to first advertising event is approximately 25 ms.

Figure 8 shows the booting sequence when running from an external SPI Flash memory using the secondary bootloader. The booting time up to first advertising event is approximately 56 ms.

Figure 9 shows the booting sequence when running from an external SPI Flash memory without the secondary bootloader. The booting time up to first advertising event is approximately 195 ms.

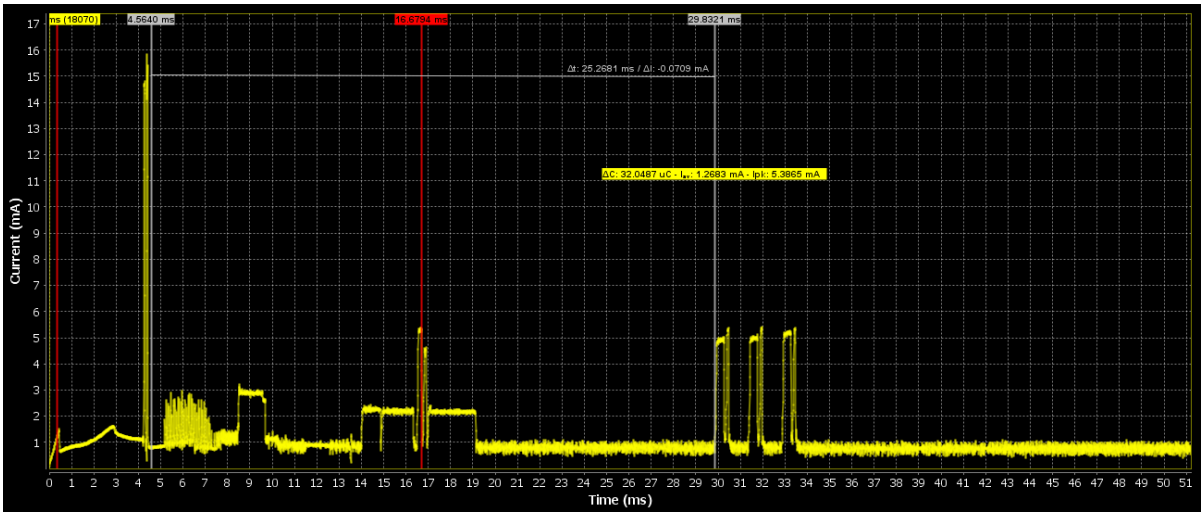


Figure 7: DA14581: Booting from OTP memory using the ROM bootloader

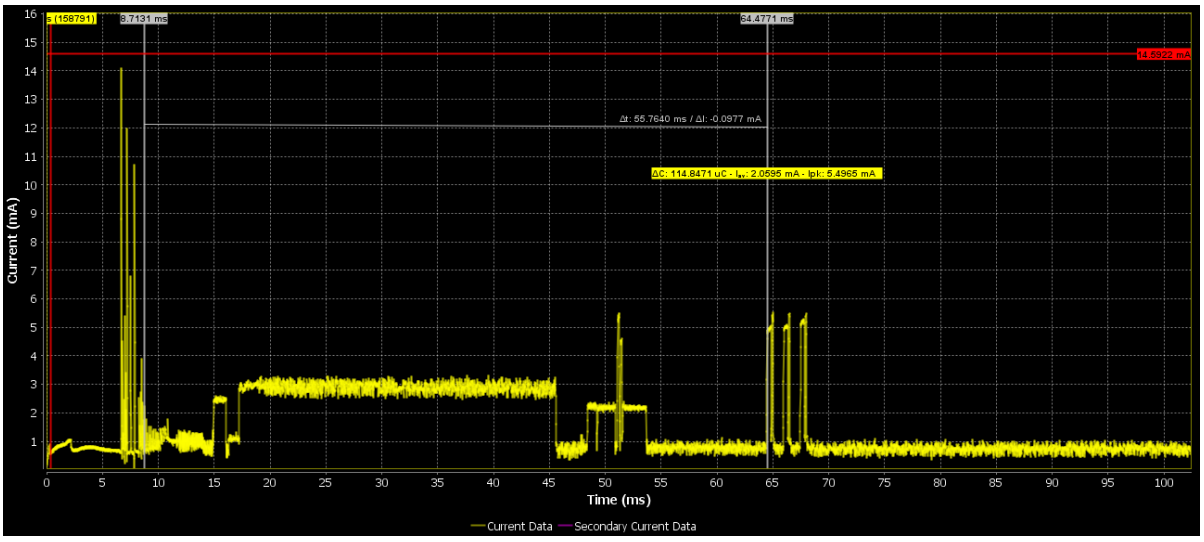


Figure 8: DA14581: Booting from SPI Flash memory using the secondary bootloader

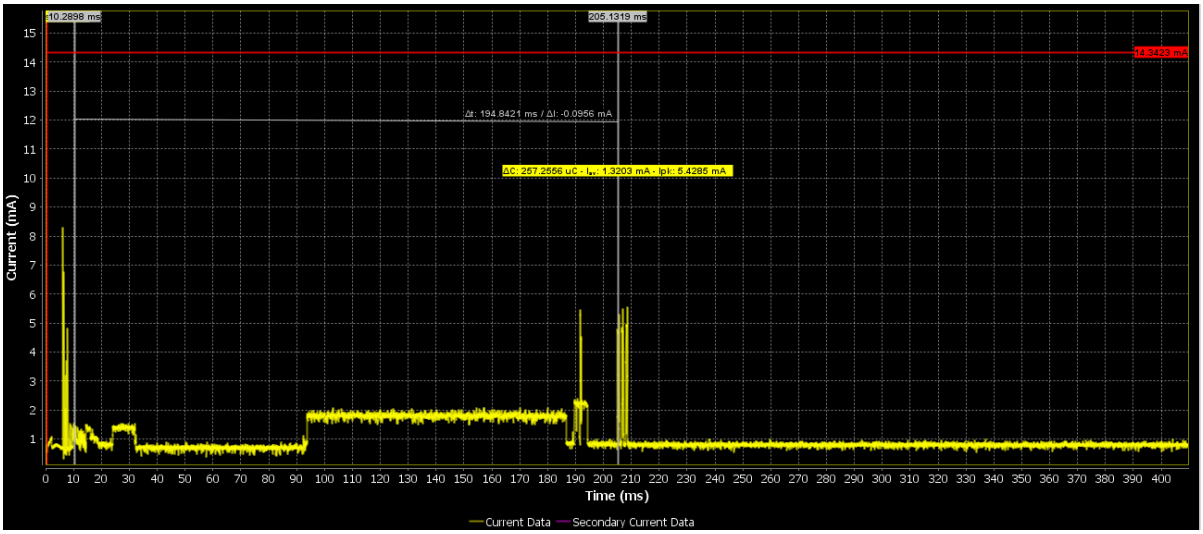


Figure 9: DA14581: Booting from SPI Flash memory using the ROM bootloader

Appendix A The mkimage tool

The *mkimage* tool is a command line Windows application for formatting a non-volatile memory according to the memory map specified by the dual image bootloader.

The tool supports two use cases:

- **Single image:** Create a binary application image file (.IMG), containing both the application image header and the application firmware.
- **Multi-part image:** Create the entire contents of the external non-volatile memory as a single multi-part binary image file (.IMG), that can be written into a non-volatile memory using the *SmartSnippets* toolkit.

The source code of the *mkimage* tool is located in folder *\tools\mkimage* in the SDK.

A.1 Creating an application image file

The command line syntax to create an application image file (.IMG) is the following:

```
mkimage.exe single <in_file> <version_file> <out_file> [enc [<key> <iv>]]
```

`single`	Instructs the tool to create the `<out_file>` application image (.IMG file).
`<infile>`	Specifies the raw application firmware binary file (BIN file).

`<version_file>`	C header file containing versioning, time stamping and housekeeping information for the image header. It must be formatted similar to the SDK header file *dk_apps/src/dialog/include/ble_580_sw_version.h*. It contains definitions such as:
	`#define DA14580_SW_VERSION "v_3.0.6.0"`
	`#define DA14580_SW_VERSION_DATE "2014-10-3 18:56 "`

Encryption of the raw binary image `<in_file>` may be enabled by including the `enc` option at the end of the command. The user may provide the encryption key (`<key>`) and initialisation vector (`<iv>`), as a string of 32 hexadecimal characters (without any prefix). When no values for `<key>` and `<iv>` are specified, the following default values are used:

```
Key:                        06A9214036B8A15B512E03D534120006
Initialisation vector:      3DAFBA429D9EB430B422DA802C9FAC41
```

The default values of the key and the initialisation vector are hardcoded in the secondary bootloader firmware.

A.2 Creating the entire contents of a non-volatile memory

The command line syntax to create a multi-part binary image file (.IMG) with the entire contents of a non-volatile memory is the following:

```
mkimage.exe multi spi|eeprom [<bloader>] <in_img1> <off1> <in_img2> <off2> <off3> [cfg
off4[,bdaddr]] <out_file>
```

`multi`	Instructs the tool to create the `<out_file>` binary image file (.IMG) with the entire contents of the SPI Flash memory (option: `spi`) or the I2C EEPROM (option: `eeprom`).

The multi-part image consists of:

1. AN-B-001 header plus the bootloader firmware `<bloader>` at offset 0, if `<bloader>` is provided.
2. `<img1>` (.IMG image) at offset `<off1>`
3. `<img2>` (.IMG image) at offset `<off2>`
4. Product header at offset `<off3>`

The cfg option configures the following product header fields:

- The application specific “Configuration Offset” is initialized from off4. If off4 is not provided then the “Configuration Offset” field shall be set to 0xFFFFFFFF.

- The BD Address is initialized from 'bdaddr'. If bdaddr is not provided then the BD Address field shall be set to FF:FF:FF:FF:FF:FF. If bdaddr is provided it is required that no space between off4, the comma character and the 'bdaddr' exists.

The offsets can be given either as decimal or as hexadecimal numbers.

The BD address 'bdaddr' can be given as XX:XX:XX:XX:XX:XX where X is a hex digit. E.g. 80:EA:CA:01:02:03.

Revision history

Revision	Date	Description
1.0	16-Jul-2014	Initial version.
2.0	28-Apr-2015	Major changes: added support for DA14581 and *mkimage* tool description.
3.0	8-June-2015	Added DA14583. Added new option in mkImage tool.

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

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

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

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

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

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, unless otherwise stated.

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)
Dialog Semiconductor PLC
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
Dialog Semiconductor Singapore
Phone: +65 64 849929

China
Dialog Semiconductor China
Phone: +86 21 5178 2561

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8291