Application Note

DA14585/586 Interfacing with External Memory

AN-B-055

Abstract

This Application Note describes how to use an external memory (I2C EEPROM or SPI Flash) with the DA14585/586. The main differences between the internal OTP and an external memory are also presented.

Contents

Figures

Tables

1 Terms and definitions

BLE	Bluetooth® Low Energy
CLI	Command Line Interface
DK	Development Kit
EEPROM	Electrically Erasable Programmable Read-Only Memory
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
MCU	microcontroller unit
OTP	One-Time Programmable (memory)
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SW	Software
UART	Universal Asynchronous Receiver/Transceiver

2 References

[1] DA14585, Data sheet, Dialog Semiconductor

[2] UM-B-119 DA14585/586 SDK 6 Software Platform Reference

[3] UM-B-048, DA14585 Getting Started Guide with the Basic-Development Kit, Dialog Semiconductor

[4] UM-B-049, DA14585 Getting Started Guide with the PRO-Development Kit, Dialog Semiconductor

[5] UM-B-083 SmartSnippets™ Toolbox User Manual

3 Introduction

The DA14585/586 includes user memory of 64kB of OTP (One Time programmable) and 96kB of SRAM. The flexibility of the device allows the DA14585/586 to also use external SPI/I2C FLASH or EEPROM in a variety of different use cases. This includes booting from external memory, using external memory to store critical data parameters, provide flexibility for the DA14585/586 to support complete Over the Air image update, or booting from memory within an external MCU (externally hosted application). The OTP memory gives optimum performance in terms of low power and cost and coupled with the ability to use external FLASH or EEPROM, the DA14585/586 can support the most basic of configurations highly optimised for system cost right through to more feature rich configurations requiring image updates or multiple images. Please refer to [2] for the booting sequence of DA14585/586 from an external SPI or I2C memory. The following table sums up the pros and cons of using the internal OTP or an external memory (SPI or I2C).

Table 1: Pros and cons of using internal or external memory

	Advantages	Disadvantages
Internal memory (OTP)	No added cost	Can be programmed only once
	Minimum energy needed to load application into SRAM	Needs minimum 2.25V onVBAT3V to be programmed
	Already integrated in the DA14585. No extra board space	
		The memory size is fixed to 64kB
External memory (I2C or SPI)	The memory size can be increased. Flexible memory configuration	Added cost ($). Typically, a SPI Flash is cheaper than the I2C EEPROM.
	Can be programmed many times, which can be handy using SW design. Software updates over the air can be stored.	Added PCB surface + added PCB design time.
		Impact on coin cell battery life due to the high erase current
		Consume more energy during booting
		Mirror time (= loading SW from external memory to SRAM) can be very long

4 Hardware configuration of the external memory

The application code can be stored into an external SPI or I2C memory. The DA14585/586 ROM booter can only work up until 64kB of external EEPROM.

The DA14585/586 can boot from an external memory using the default ROM Booter **without the need to program the OTP**, see [2]. The schematics are shown on the next page according to the type of external memory which has to be connected to the DA14585/586. Note that, for isolation purposes and to limit the fault current in case of short circuit, series of 1kΩ may be used between the Tx_DA14585/Rx_DA14585 pins to UART-TTL board in production or bench.

NOTE
The DA14585/586 operates in two modes, namely the 'Normal Mode' and the 'Development/Calibration Mode'. The decision which mode the chip enters after power-up, is taken by the boot code residing in the ROM. When in Development mode, the Boot ROM code initializes all peripheral devices from which the DA14585/586 might download code: UART, SPI (both Master and Slave) and I2C. The code searches for a valid response on various combinations of I/Os one after the other. There is also the option that the user can define desired I/Os using a specific OTP field **(Boot specific)** for the SPI interface. See [2]

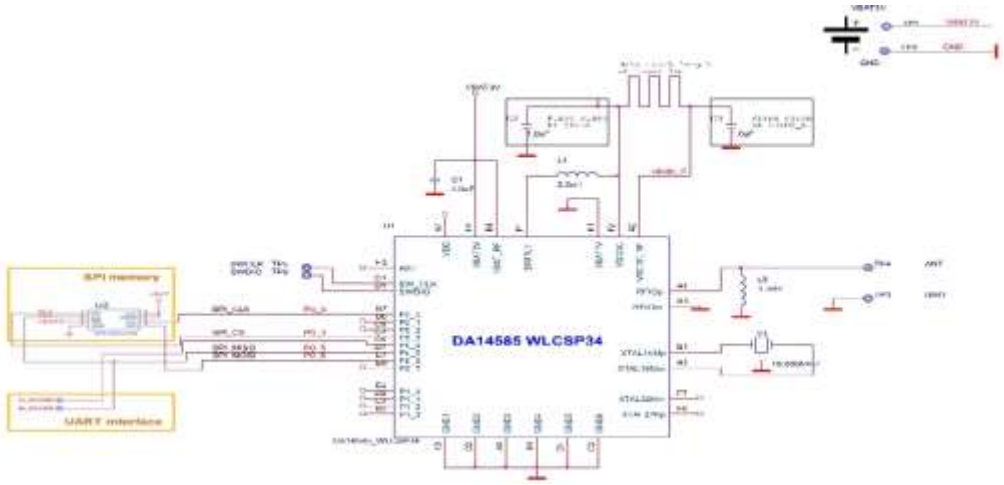

Figure 1: Booting from external SPI Flash memory with the DA14585 (blank OTP)

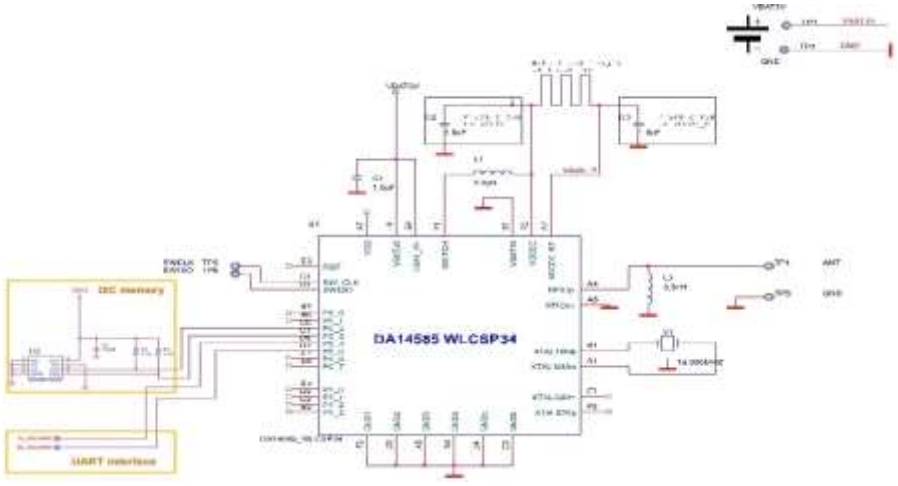

Figure 2: Booting from external I2C EEPROM memory with the DA14585 (blank OTP)

4.1 With the use of the DA14585 PRO Development kit

The DA14585/586 Development kit-PRO is populated with the **W25X20 CL** SPI memory IC (Memory size: 256 kB, Page size: 256 Bytes). Figure 3 shows the required jumper settings for accessing the SPI memory. Loading an image into the SPI Flash can be done using the UART or SWD interface from a PC tool, like SmartSnippets™ Toolbox or the Keil IDE. Please see [4] for more details about how to use this Kit-PRO.

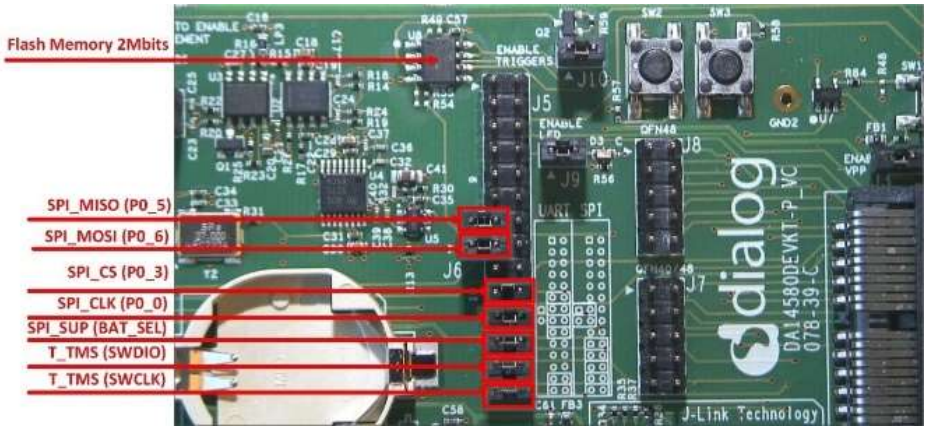

Figure 3: Development kit-PRO

4.2 With the use of the DA14585 BASIC Development kit

The DA14585 Development kit-Basic is populated with the **MX25R2035** SPI memory IC (Memory size: 2 MB, Page size: 256 Bytes). Figure 4 shows the required jumper settings for accessing the SPI memory. Loading an image into the SPI Flash can be done using the UART or SWD interface from a PC tool, like SmartSnippets™ Toolbox or the Keil IDE. Please see [3] for more details about how to use this Kit-Basic.

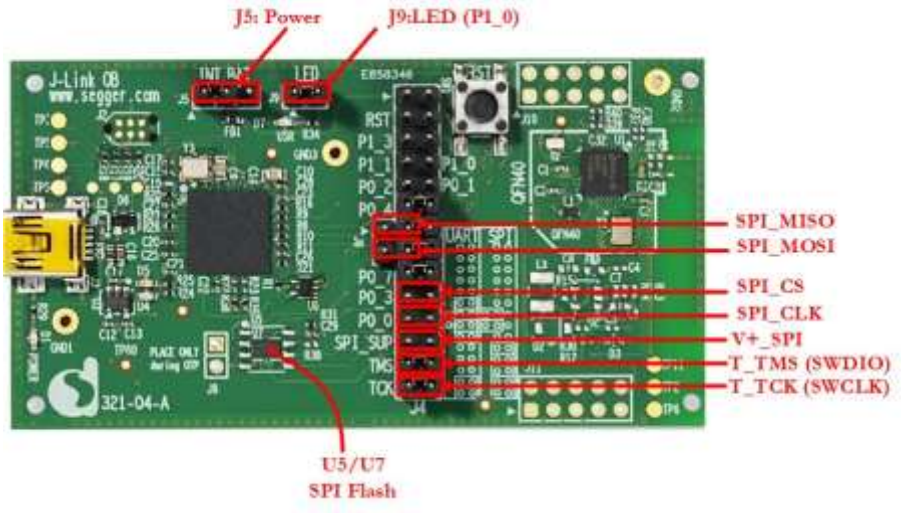

Figure 4: Development kit-Basic

4.3 Hardware Configuration

The following table must be taken into account when UART and SPI ports are used at the same time. If reading from UART is necessary then as the UART RX default pin conflicts with the SPI MISO pin (P0_5), a separate pin will have to be used for UART RX. In this case a jumper wire is needed to connect UART Rx as shown in Table 2.

Table 2: SPI Flash Memory Example Jumper Configuration with UART2 RX

GPIO	Function	DA14585/586 DK-Basic	DA14585/586 DK-Pro
P0_4	UART2 TX	Connect J4.11 - J4.12	Connect J5.11 - J5.12
P0_5	SPI DI	Connect J6.1 - J4.13	Connect J6.1 - J5.13
P0_6	SPI DO	Connect J6.2 – J4.15	Connect J6.2 – J5.15
P0_7	UART2 RX	Connect J4.17 - J4.14 (with a jumper wire)	Connect J5.17 - J5.14 (with a jumper wire)
P0_3	SPI CS	Connect J4.19 - J4.20	Connect J5.19 – J5.20
P0_0	SPI CLK	Connect J4.21 - J4.22	Connect J5.21 – J5.22
VBAT	SPI_SUPPLY	Connect J4.23 - J4.24	Connect J5.23 – J5.24
T_TMS	SWD IO	Connect J4.25 - J4.26	Connect J5.25 – J5.26
T_TCK	SWD CLK	Connect J4.27 – J4.28	Connect J5.27 – J5.28

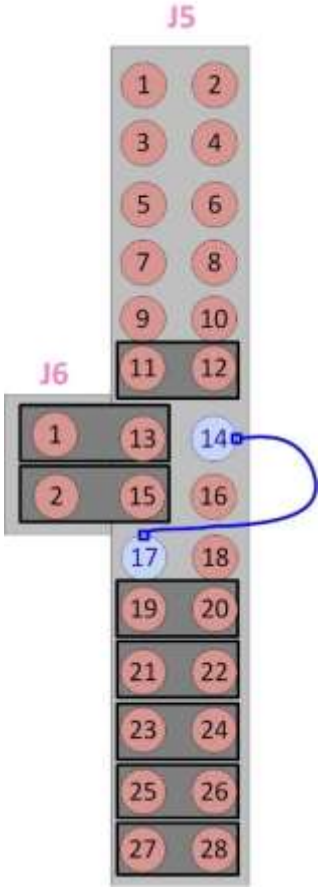

Figure 5: SPI Jumper Configuration with UART2 RX on Development kit-PRO

5 Tools & Support for programming OTP and external memory

SmartSnippets™ is the appropriated tool to download your application into the external memory very easily. Navigate to the DA14585/586 product page on Dialog's web site https://www.dialog-semiconductor.com/products/connectivity/bluetooth-low-energy/smartbond-da14585-and-da14586 and download the latest 5.0.x version of SmartSnippets™ Toolbox.
SmartSnippets™ Studio is a royalty-free software development platform for Smartbond™ devices. It is a framework of tools comes with the popular open source IDE Eclipse CDT [4].
SmartSnippets™ Toolbox covers all software development requirements, including Programming and loading of firmware into SPI/I2C Flash [4]. SmartSnippets™ Toolbox will be used only for this purpose.

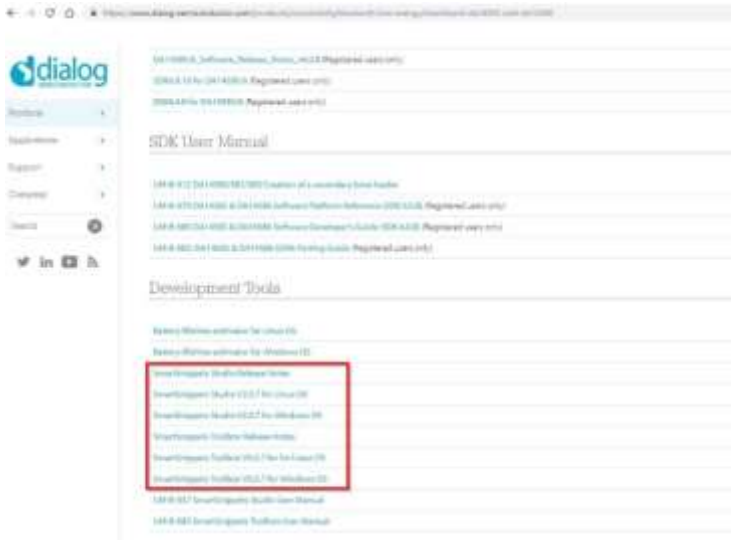

Figure 6: Download SmartSnippets™ Toolbox

There are two possibilities to download the image into the external memory (by either choosing UART or JTAG) as shown in Figure 7. The SmartSnippets™ tool is targeting the main activities of programming and reading the power performance of the DA14585. It enables:

- Programming the internal OTP with the actual application compiled image
- Downloading an application binary file to the external SPI/I2C memory over UART or JTAG
- Accurate examination of the power profile and how it is affected by application software (can be only used with the development kit)
- Downloading a SW image to SRAM over UART or JTAG and execute.

The SmartSnippets™ tool makes maximum use of the available features on the motherboard and thus allowing developers of Bluetooth smart applications to work without expensive and bulky equipment. The tool will provide full visibility on the chip activity, which is crucial in developing ultra-low power wireless applications. All the tests have been carried out with the **SmartSnippets™ Toolbox V5.0.8** For more information, please go to the **HELP** tab in the SmartSnippets™ GUI.

Figure 7: Mode to download the application

You can select either UART or JTAG modes, in Figure 7 the JTAG mode is selected **480068433** is the serial number that corresponds to the attached JTAG. This number is printed on a sticker on the development kit.

By default, the UART port must be connected to **P04/P05 as Tx/Rx** to download the application. However, this can be changed as mentioned in the next chapter. If the case of UART mode is chosen to download the application, make sure that the UART pins are connected according to the board setup as follow:

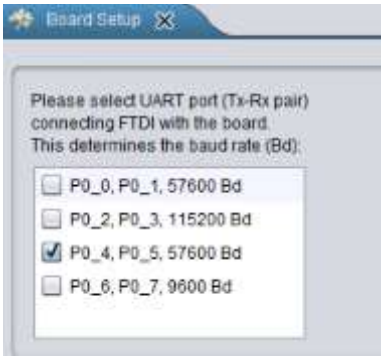

Figure 8: Board setup if UART is used

Having chosen the appropriated mode to download the application, then the type of memory must be selected from the top side of the GUI (SPI Flash Programmer or EEPROM programmer tab).

Figure 9: Memory programmers tab

The following steps show how to download an image into an external SPI memory over the UART. The shows the steps mentioned below.

1) Browse for the HEX file of the image needed to load into external memory.
2) Press the **CONNECT** button to establish a connection with the DA14585/586,
   ⇨ ***"Successfully downloaded firmware file to the Board"*** is displayed in the log windows.

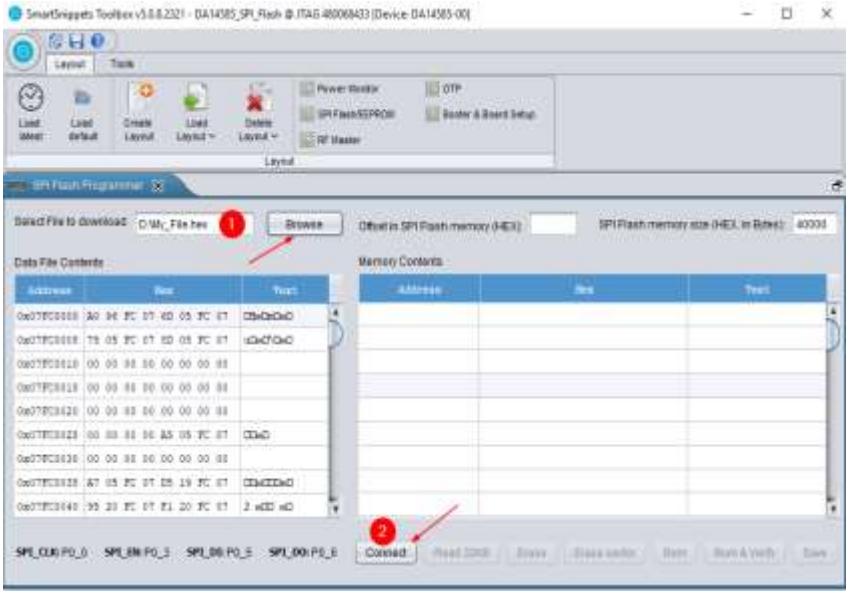

Figure 10: Browse/Connect Buttons to select the .hex and connect to the DA14585/586

When doing the programming, production or design parameters can be stored into the external memory, so some specific sectors can be erased, that's why it is highly recommended to before writing any data into the SPI memory, make sure to **ERASE** the memory first by clicking on the Erase sector button has shown below in Figure 11.

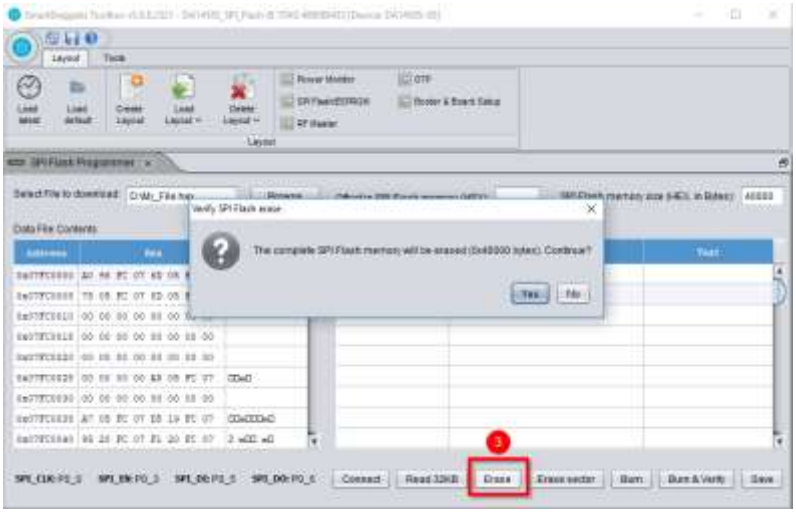

Figure 11: Erases the entire SPI Flash Memory

3) Specify in hexadecimal the memory size, by default use a 256 KBytes memory size (0x40000).

4) Set the starting address where the image has to be stored. Blank means starting address 0.

Figure 12: offset text filed and SPI flash size

5) Press the **BURN** button.

6) If the BURN operation has been chosen, choose YES (to boot from the external memory) when the pop-up window appears.

⇨ ***"Memory burning completed successfully"*** is displayed in the log windows.

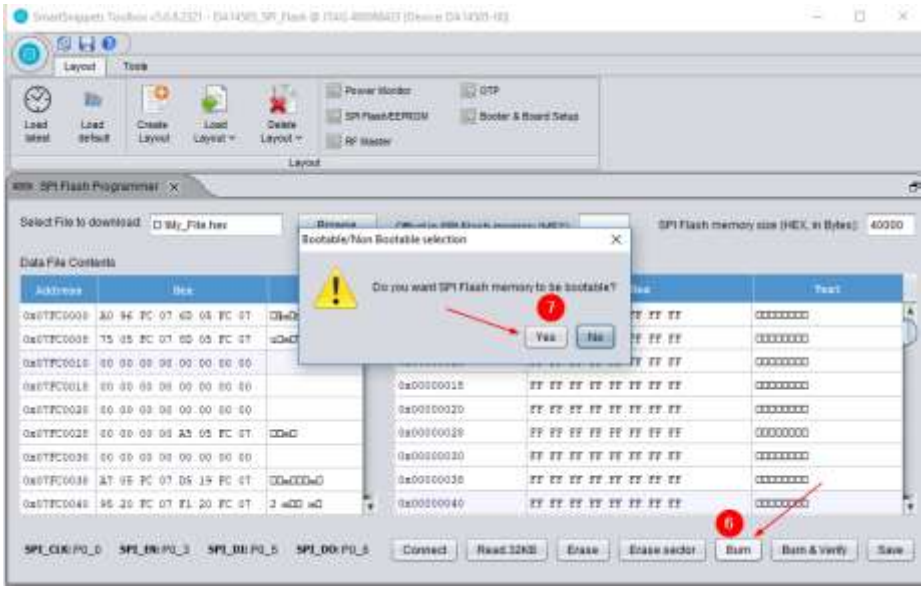

Figure 13: Burn button to burn the SPI Flash

NOTE
When the bootable option is selected, a special header is added before the data and the data is written starting at the selected offset. If the option to make the Flash image bootable is not selected, the SmartSnippets™ will not add this header at offset 0 of the SPI flash memory. Then the secondary bootloader is needed and the maximum booting time can be measured. Appendix 6 in [2] describes the booting procedures supported by the DA14585/586 ROM code.

5.1 Procedure to change the configuration of SmartSnippets™ Toolbox

5.1.1 SmartSnippets™ Toolbox settings

This section shows how to modify the default SmartSnippets™ Toolbox settings. The default parameters are the followings:

The booting port for the **I2C memory** can have the following combination.

- ✓ P0_0 & P0_1 as SCL & SDA. Available UART connection are:
  - STEP 4: P0_2, P0_3 as UART_TX and UART_RX (@Baudrate 115200 Bd)
  - STEP 5: P0_4, P0_5 as UART_TX and UART_RX (@Baudrate 57600 Bd)
  - STEP 6: P0_6, P0_7 as UART_TX and UART_RX (@Baudrate 9600 Bd)

- ✓ P0_2 & P0_3 as SCL & SDA. Available UART connection are:
  - STEP 3: P0_1, P0_2 as UART_TX and UART_RX (@Baudrate 57600 Bd)
  - STEP 5: P0_4, P0_5 as UART_TX and UART_RX (@Baudrate 57600 Bd)
  - STEP 6: P0_6, P0_7 as UART_TX and UART_RX (@Baudrate 9600 Bd)

- ✓ P0_4 & P0_5 as SCL & SDA. Available UART connection are:
  - STEP 3: P0_1, P0_2 as UART_TX and UART_RX (@Baudrate 57600 Bd)
  - STEP 4: P0_2, P0_3 as UART_TX and UART_RX (@Baudrate 115200 Bd)
  - STEP 6: P0_6, P0_7 as UART_TX and UART_RX (@Baudrate 9600 Bd)

- ✓ P0_6 & P0_7 as SCL & SDA. Available UART connection are:
  - STEP 3: P0_1, P0_2 as UART_TX and UART_RX (@Baudrate 57600 Bd)
  - STEP 4: P0_2, P0_3 as UART_TX and UART_RX (@Baudrate 115200 Bd)
  - STEP 5: P0_4, P0_5 as UART_TX and UART_RX (@Baudrate 9600 Bd)

The only booting port for the **SPI memory** (without the use of the secondary boot loader) is:

- ✓ P0_0, P0_3, P0_5, P0_6 as SCK, CS, MISO & MOSI

When in Development mode, the BootROM code initializes all peripheral devices from which the DA14585/586 might download code: UART, SPI (both Master and Slave) and I2C. For fast booting, and after peripheral devices initialization, the BootROM check if the OTP is blank, for that the Boot specific flag = 0XAA (JTAG is disabled) is checked from OTP.

The mapping of the serial interface to each pin is shown in Table 3. The booting sequence of DA14585/586 is shown in Table 20 and Table 21 of [2].

Table 3: DA14585/586 Pin assignment and booting sequence

<table>
<tr><th>Pin</th><th>Step</th><th>Booting from external SPI Master</th><th>Step</th><th>Booting from external SPI Master</th><th>Step</th><th>UART</th><th>Step</th><th>Booting from external SPI Slave</th><th>Step</th><th>I2C</th></tr>
<tr><td>P0_0</td><td rowspan="8">1</td><td>SCK</td><td rowspan="8">2</td><td>SCK</td><td rowspan="2">3</td><td>TX</td><td rowspan="8">7</td><td>SCK</td><td rowspan="2">8</td><td>SCL</td></tr>
<tr><td>P0_1</td><td></td><td>CS</td><td>RX</td><td></td><td>SDA</td></tr>
<tr><td>P0_2</td><td></td><td>MISO</td><td rowspan="2">4</td><td>TX</td><td></td><td rowspan="2">9</td><td>SCL</td></tr>
<tr><td>P0_3</td><td>CS</td><td>MOSI</td><td>RX</td><td>CS</td><td>SDA</td></tr>
<tr><td>P0_4</td><td></td><td></td><td rowspan="2">5</td><td>TX</td><td></td><td rowspan="2">10</td><td>SCL</td></tr>
<tr><td>P0_5</td><td>MOSI</td><td></td><td>RX</td><td>MISO</td><td>SDA</td></tr>
<tr><td>P0_6</td><td>MISO</td><td></td><td rowspan="2">6</td><td>TX</td><td>MOSI</td><td rowspan="2">11</td><td>SCL</td></tr>
<tr><td>P0_7</td><td></td><td></td><td>RX</td><td></td><td>SDA</td></tr>
</table>

5.1.2 Flash programmer

The **flash programmer** is an application which is provided with the SDK package *DA14585_SDK_version\utilities\flash_programmer*. The flash programmer is used for uploading and reading back the application code. After booting the flash programmer application communicates over UART or JTAG interface with the host application allowing it to read or write application code to the FLASH, EEPROM or OTP memory. Default operations the user does not need to modify anything. The default version is selected by SmartSnippets™ Toolbox. As an application example, If the user would like to support new SPI/I2C Flash then the following parameters can be changed from the flash_programmer project, refer to section 7.1.

- ✓ Memory port and UART port

- ✓ Clock speed
- ✓ Memory size
- ✓ Memory page size

Here are the steps to change the default parameters:

1) Open & compile the flash programmer project found in the path:

   DA14585_SDK_version\ utilities \flash_programmer\ flashprogrammer.uvproj.

2) Search for the ***periph_setup.h*** file. From this file, the port, clock frequency, memory size and memory page size can be changed. Figure 14 shows how to add new I2C EEPROM characteristics to support the **AT24C256**.

Figure 14: user_periph_setup.h configuration file

3) Under KEIL project, Press the Rebuild all target (UART/JTAG) files button in order to generate the updated hex file, you can choose *programmer_jtag.*

4) When compiling the KEIL project a new *flash_programmer.bin /jtag_programmer.bin* is generated under `utilities\flash_programmer\Out_jtag` directory.

5) Take this new binary and replace the old binary in this folder `C:\DiaSemi\SmartSnippets™Studio2.0.8\Toolbox\common_resources\SupportPackages\DA14585-586\toolbox_resources\common` (where SmartSnippets™ has been installed)

6) Power Off/Power On the DK and open the EEPROM programmer Interface from the SmartSnippets™ and start connecting/reading/writing sequences.

6 Power consumption

After the application image has been copied in the SRAM of DA14585/586, a system with an external flash or EEPROM will have almost the same power consumption as a system running with the internal OTP because the external memory is not needed any more. The external memory can be put into ultra-deep power down mode (=200nA). Thus, the main difference of the energy consumption between the internal OTP and an external memory is caused by copying the image into the SRAM of the DA14585.

All the tests have been carried out with BLE All In One application located under: `projects\target_apps\ble_examples\ble_app_all_in_one`). The code size of this application is about **32.56 kB with a power supply of 3V**.

6.1 Booting from external SPI memory in development mode

In this mode, the OTP is blank. The energy consumption of a READING operation is going to be measured. The SPI Flash used is the P25Q10U (Puya Semiconductor)

- Default clock frequency of the SPI bus: 8.33 MHz, Power supply = 3V.

From theoretical point of view, the time needed to load 32.56 kB of data from the external SPI memory to the SRAM would be:

$$t = \left[(\text{Number of bytes}) \times \left(\frac{1}{8\ MHz} \times 8.33 + 3.76\ \mu s\right)\right] + 64.3ms$$

$$t = [(32560) \times (1.04\ \mu s + 3.76\ \mu s)] + 80.3ms$$

$$\boldsymbol{t = 238ms}$$

In practise, the time needed is around 238.7 ms (Refer to the Power profile view below)

The amount of charge required during the mirroring is about 250 μC.

NOTE
During the booting time the flash is not on powerdown mode, hence this given energy is including the energy of the Flash itself.

Below we are measuring the total timing for SPI memory mirroring into SRAM, Three Logic analyser traces are provided, the figure below shows the $3.76\ \mu s$ which is the time between two SPI clock cycles. The other one is showing the SPI clock frequency 8.33 Mhz. The last one is showing the $80.3ms$ which is the given time between the RESET and start image downloading ($157ms$)

Below is the power profile view is also provided to show the total booting period, we get the same result as calculated using the Logical analyzer traces.

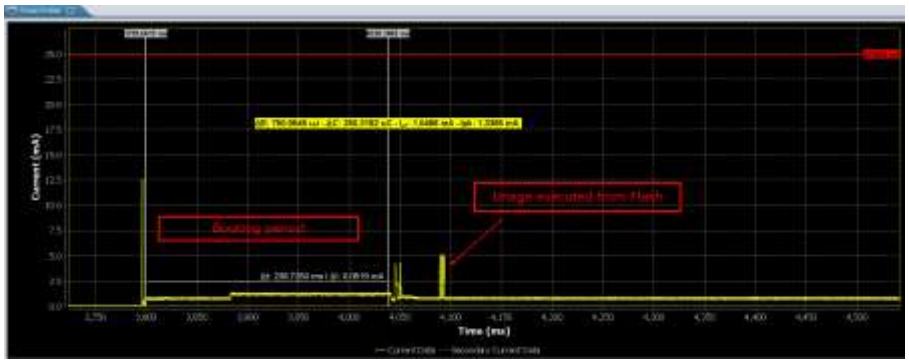

Figure 15: SPI memory mirroring into SRAM

6.2 Booting from external SPI memory with the secondary bootloader

In this mode, the secondary bootloader has been created and programmed in OTP to. The secondary bootloader is needed to change the default boot sequence for fast booting. The secondary bootloader should be loaded from OTP to SRAM before starting booting from external SPI flash. The energy consumption of a READING operation is going to be measured.

The SPI Flash used is the P25Q10U (Puya Semiconductor)

The secondary bootloader has been burnt into OTP for the purpose of having a SPI boot up as fast as possible.

- Default clock frequency of the SPI bus: 8.33 MHz, Power supply = 3V.

From theoretical point of view, the time needed to load 32.56 kB of data from the external SPI memory to the SRAM would be:

$$t = (\text{Number of bytes}) \times (\frac{1}{8\ MHz} \times 8.33 \times +0.18\ \mu s)$$

$$t = (32560) \times (1.04\ \mu s + 0.16\ \mu s)$$

$$\boldsymbol{t = 39\ ms}$$

In practise, the time needed is around **38 ms**.

NOTE
As in section 6.1 we are providing the power profile view from the SmartSnippets™ and logic analyser traces to show how we get the total booting timing.

The amount of charge required during the mirroring is about 87.63 µC. (With W25X20CL we have seen 143.75 µC of charge). During the booting time the flash is not on powerdown mode, hence this given energy is including the energy of the Flash itself.

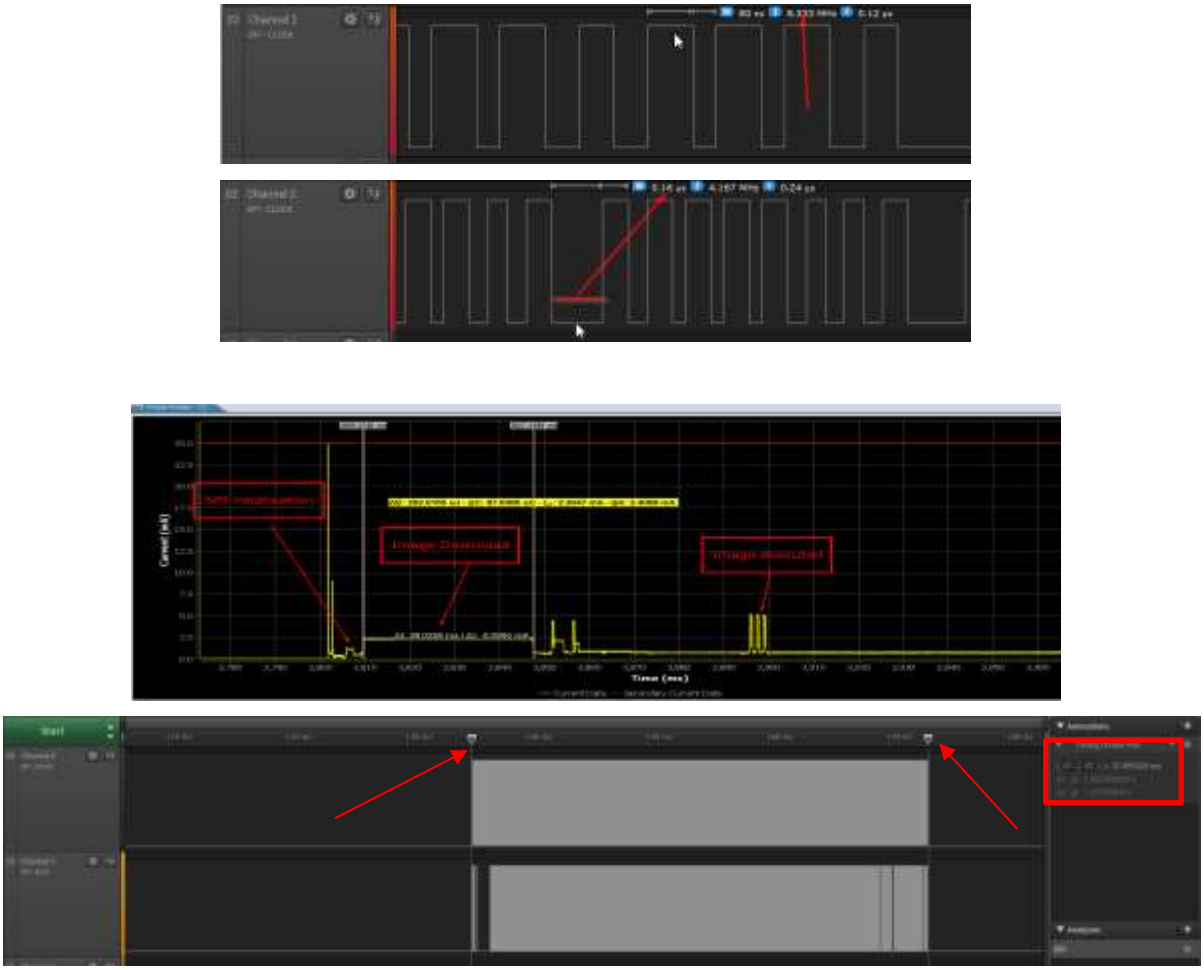

Figure 16: SPI memory mirroring into SRAM (with secondary bootloader)

6.3 Booting from external I2C memory in development mode

For hardware configurations using an external I2C EEPROM to store the Firmware image instead of a SPI Flash the boot time and energy is different. The energy consumption of a READING operation is going to be measured (the OTP is blank)

- Default speed of the I2C bus: 100 kbit/s, Power supply = 3V.

From theoretical point of view, the time needed to load 32.56 kB of data from the external I²C memory to the SRAM would be:

$$t = \frac{1}{100\ kBit/s} \times \frac{Image\ size\ (in\ byte)}{32\ bytes\ block} \times (297 + 27)$$

$$t = 10\ \mu s \times \frac{32\,560}{32} \times (297 + 27)$$

$$t = \mathbf{3.29\ sec}$$

In practise, the time needed is around 3.3 sec. Refer to Figure 17.

The amount of charge required during the mirroring is about 7710 μC. During the booting time the EEPROM is not on powerdown mode, hence this given energy is including the energy of the Flash itself.

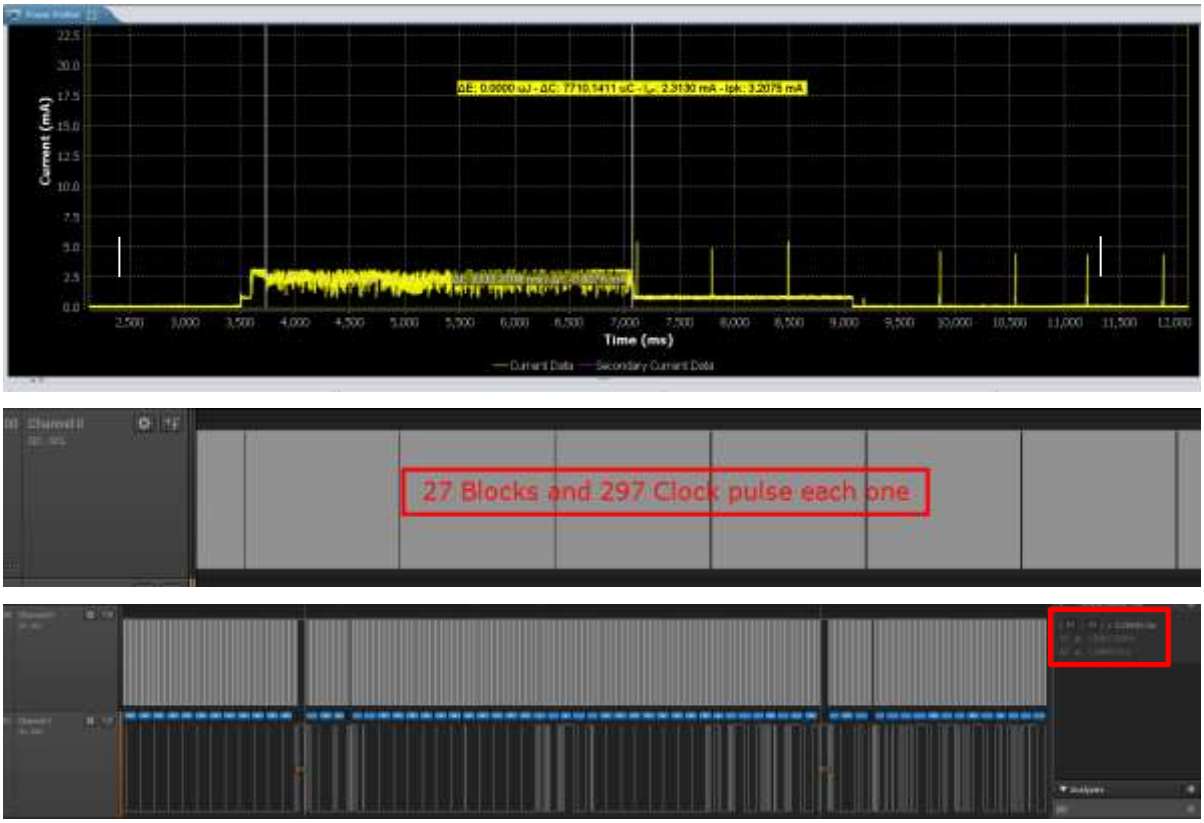

Figure 17: I2C memory mirroring into SRAM

6.4 Summary

Depending on the memory type from where the DA14585/586 has to boot, it can have quite some impact on the battery. The following setup summarizes the measurements.

Table 4: Energy vs time vs memory type

Setup		Time needed	Energy consumption (@ 3V)
Application of 32.56 kB loaded from OTP to SRAM		1.2 ms	2.3 µC
Application of 32.56 kB copied from SPI memory to SRAM	With blank OTP	238 ms	250 µC
	With secondary bootloader burnt into OTP	38 ms	84.66 µC
Application of 32.56 kB copied from I2C memory to SRAM	With blank OTP	3300 ms	7710 µC

7 Energy consumption during the memory programming

The energy consumption of a WRITING operation is going to be measured. The BLE All In One application located which is located under: `projects\target_apps\ble_examples\ble_app_all_in_one)` has been downloaded via JTAG using the SmartSnippets™ GUI for the three following cases. The code size of this application is about **32.56 kB with a power supply of 3V**.

7.1 External SPI memory

- SPI port: P0_0 = CLK, P0_3 = CS, P0_5 = MISO, P0_6 = MOSI

The SPI Flash used is the P25Q10U (Puya Semiconductor)

For this particular flash or some others, **the write protection** bit is not handled as it is not implemented in the current flash programmer version. Below the code snippet to resolve such an issue, this must run before trying to erase or write to the flash:

```
spi_set_bitmode(SPI_MODE_8BIT);
spi_transaction(0x06); //Write enable
spi_set_bitmode(SPI_MODE_16BIT);
//Write 00 to the status register, this disables all memory protection bits
spi_transaction(0x01 << 8);
```

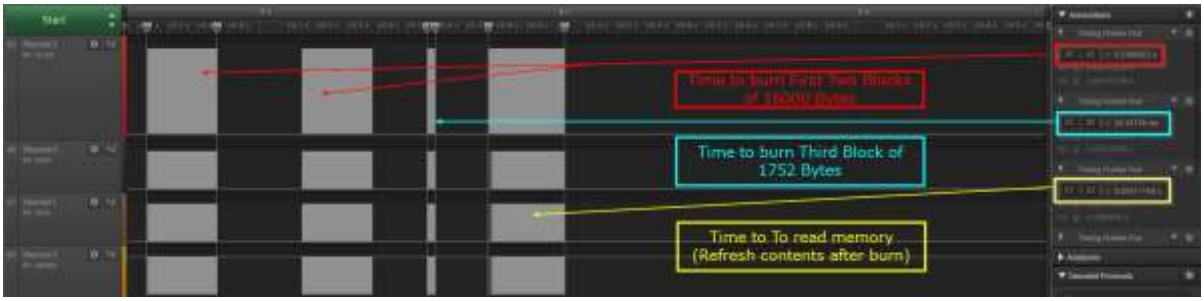

Figure 18: Writing operation time of the SPI memory

The total time to download the application into an external SPI memory is equal to **1.39 seconds**.

Time to burn First and second blocks of 16000 Bytes: 0.23s

Time to burn third block of 1752 Bytes: 26.34 ms

Time to read the memory for contents refresh: 0.25 s

7.2 External I2C memory

- I2C port: P0_2 = SCL, P0_3 = SDA

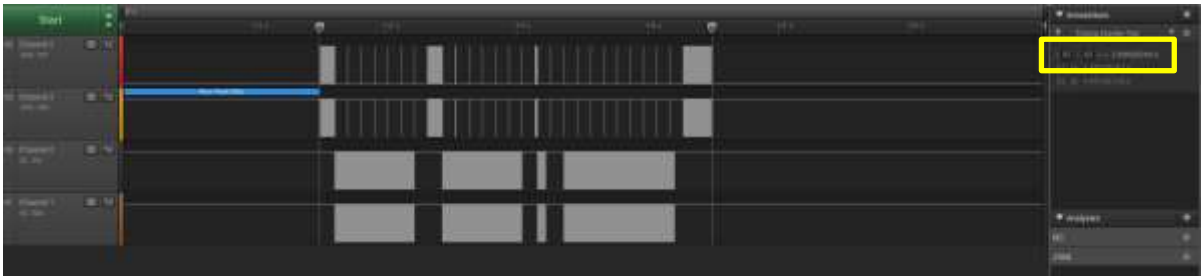

Figure 19: Writing operation time of the I2C memory

The total time to download the application into an external I2C memory is around to **3 seconds** from the figure above.

7.3 Internal OTP

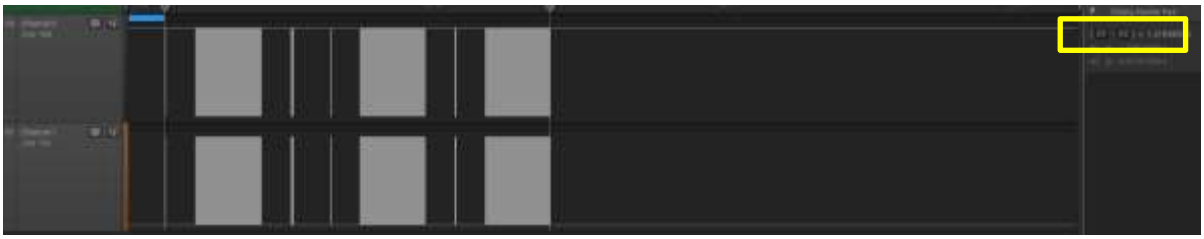

Figure 20: Time needed to load application into OTP

The total time is equal to **1.31 seconds**. This timing includes only programming, memory read and verify are not included.

7.4 Summary

Table 5: Time during the writing operation

Memory type	Time needed to load a 32.56 kB application with JTAG
External SPI flash	1.39 seconds
External I2C EEPROM	3 seconds
Internal OTP	1.31 seconds

8 How to handle the Specific data into the header section of the external memory

In this chapter, the steps needed to write/read values (non-volatile parameters) into/from an external memory (I2C or SPI) are described. Non-volatile parameters can be for instance Bluetooth address, XTAL trim value. Reading the non-volatile parameters should be defined in the application layer. This is done by calling a specific function to read out the value stored at a specific address. This is already taken care in the provided SDK.

SmartSnippets™ is equipped with a function to write a non-volatile parameter into external memory. Note that the explanation for an external SPI memory case only is mentioned in the example in the next section. A similar procedure can be used for an external I2C memory.

8.1 External SPI flash memory case

8.1.1 Reading operation:

It is assumed that the user knows how to handle the SPI drivers from the Peripheral Drivers [2]. The SPI driver must be included in your application.
The function which has to be used to read data at a specific address from the SPI memory is:

```
/**
 ****************************************************************************************
 * @brief Read data from a given starting address (up to the end of the flash)
 *
 * @param[in] *rd_data_ptr:  Points to the position the read data will be stored
 * @param[in] address:       Starting address of data to be read
 * @param[in] size:          Size of the data to be read
 *
 * @return  Number of read bytes or error code
 ****************************************************************************************
 */
int32_t spi_flash_read_data (uint8_t *rd_data_ptr, uint32_t address, uint32_t size)
```

Figure 21: Read bytes API from external SPI memory

8.1.2 Writing operation:

8.1.2.1 Using the SmartSnippets™ CLI

All the information/syntaxes about the CLI can be found in the **HELP** tab in the SmartSnippets™ GUI or by executing **SmartSnippets™ –help** in the CLI. In this example, the SPI connections are SCK (P0_0), CS (P0_3), MISO (P0_5) and MOSI (P0_6). However, if another pin assignment has to be used, some modifications have to be done in the flash programmer project see section 5.1.

Step 1, in order to use the CLI, refer to section 21 of [4], a binary file has to be downloaded into the SRAM of the DA14585/586 beforehand. There are two different bin files as shown below according to the way that the DA14585/586 is going to be connected by the PC (via UART or JTAG) using the CLI.

- *when using UART: flash_programmer.bin*
- *when using JTAG: jtag_programmer.bin*

Step 2, open the CLI (standard windows CMD shell) as shown here below:

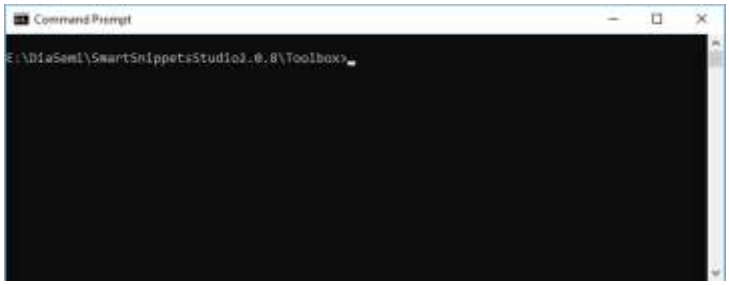

Figure 22: Describe how to open the CLI of SmartSnippets™

Run help command to displays the available commands and examples.

```
.\SmartSnippets™Toolbox.exe -help -chip DA14585-00
```

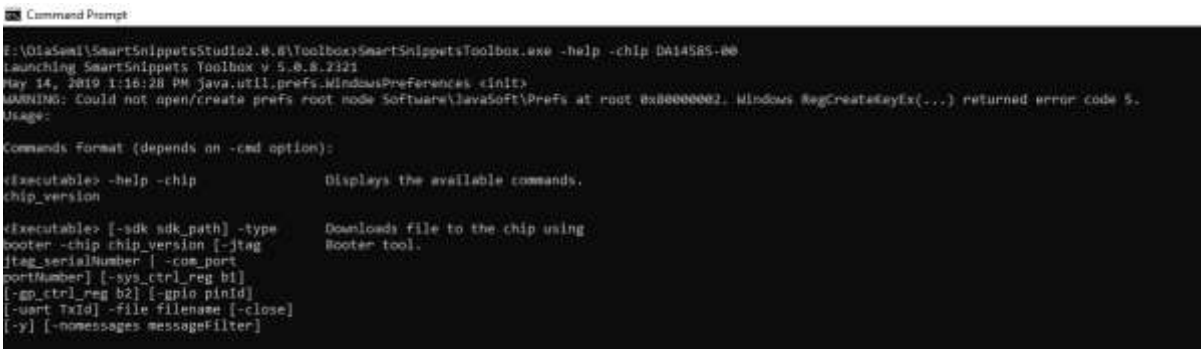

Figure 23: CLI help command

You should erase the flash before writing any data.

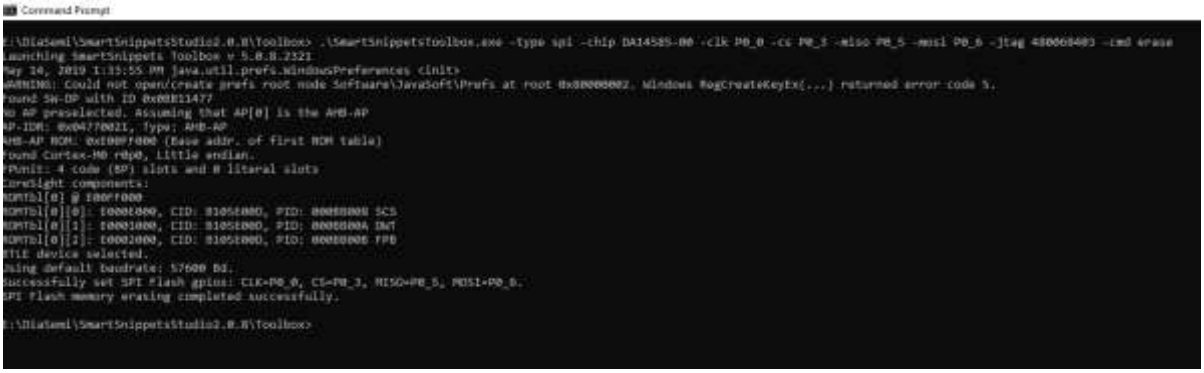

Figure 24: CLI SPI Flash Erase

Example: The following CLI writes data to a defined address, in the following screenshot the 0x1347 value is written at the address 0x93. This command is useful for example when you would like to write a Bluetooth device address to a specific address in the flash.

```
.\SmartSnippets™Toolbox.exe -type spi -chip DA14585-00 -clk P0_0 -cs P0_3
-miso P0_5 -mosi P0_6 -jtag 480068456 -cmd write_field -offset 0x93 -data
1347
```

The answers should be as follow:

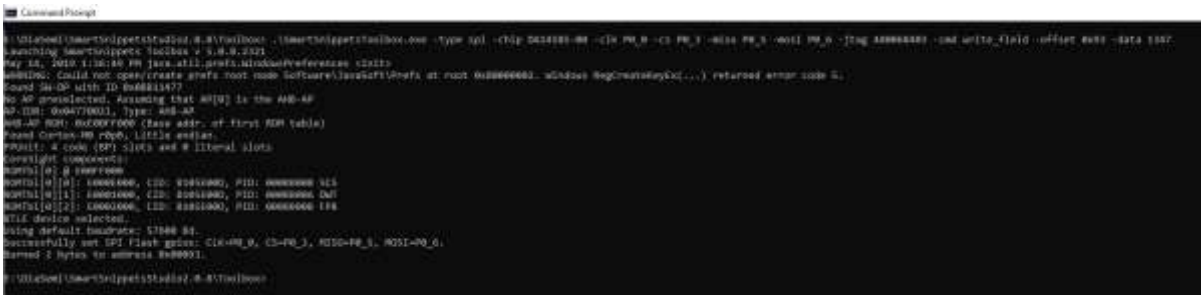

Figure 25: Answer from the DA14585 after receiving a command

You can see with Flash Programmer that the data are well written.

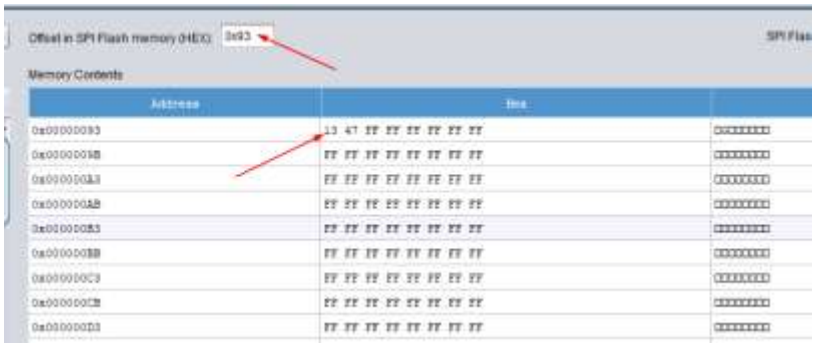

Figure 26: Programmed data seen on the Flash Programmer

NOTE
For these commands and others, different options, parameters and values that a user can use are described in the section 21 of [4].

8.1.2.2 Using the SmartSnippets™ GUI

From the Memory Header tab as shown below, it is possible to write a specific data (i.e Bluetooth address, crystal trimming) into the external memory.

Figure 27: Memory header tab

Then, a text file has to be created in order to specify at which address a value has to be stored. In this text file, three parameters must be added and separated with tabulation:

- ✓ Size (bytes)
- ✓ Type
- ✓ Parameter
- ✓ Description

Here is an example:

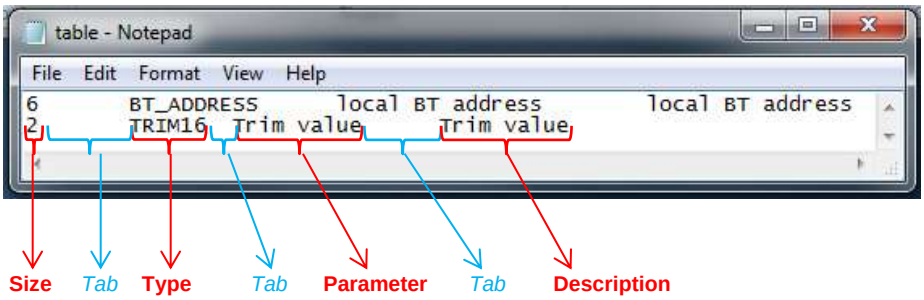

Figure 28: Text file format to write a customer header

Then, the text file has to be browsed and both address (Memory offset in hexadecimal) and data (Value) must be written manually in the GUI. At this point, the content of the header can be burnt into the external memory.

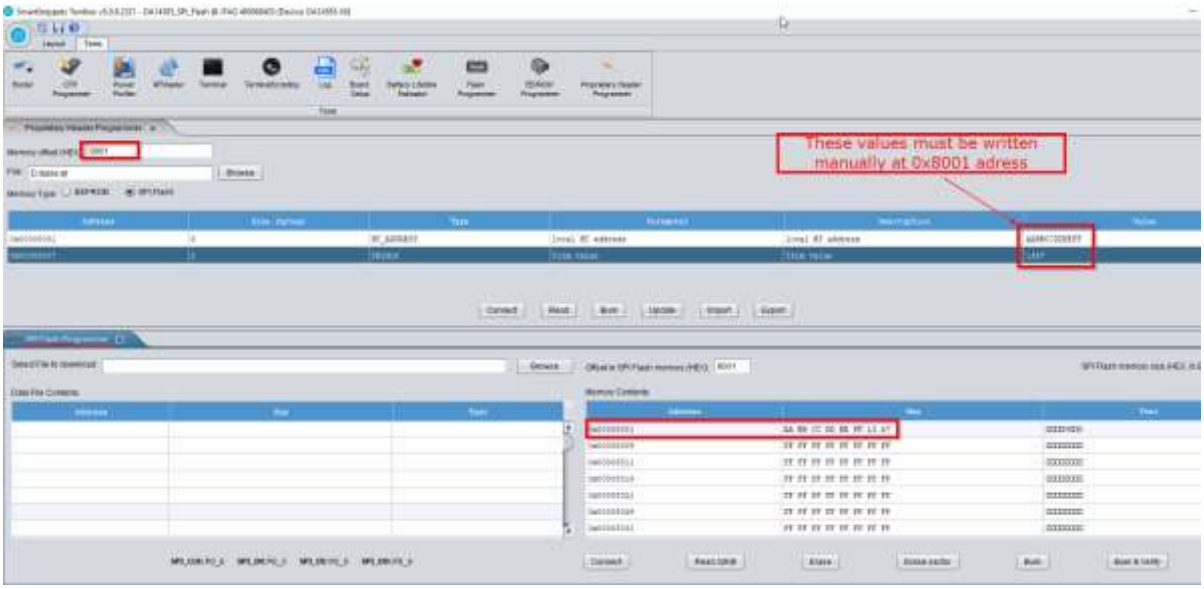

Figure 29: Memory header tab fulfilled

9 List of supported FLASH/EEPROM memories

The memories in Table 6 are currently supported by DIALOG. These memories support the standard Serial Peripheral Interface (SPI) and I2C and operate in the 3.3 V power supply.

Table 6: Memories supported

Memory type	Name	Memory size	Page size	Manufacturer name
SPI	AT25XE011	128 kB	256B	Adesto
SPI	AT25XE021A	256 kB	256B	Adesto
SPI	AT25XE041B	512 kB	256B	Adesto
SPI	AT25XE512C	64 kB	256B	Adesto
SPI	AT25XE041D	512 kB	256B	Adesto
SPI	AT25DN011	128 kB	256B	Adesto
SPI	AT25DF512C	64 kB	256B	Adesto
SPI	AT25DF011	128 kB	256B	Adesto
SPI	AT45DB041E	512 kB	256B	Adesto
SPI	P25Q10H	128 KB	256B	Puya
SPI	MX25R512F	64 kB	256B	Macronix
SPI	MX25R2035F	256 kB	256B	Macronix
SPI	GD25VQ20C	256 kB	256B	GigaDevice
SPI	GD25WD10CTIG	128 KB	256B	GigaDevice
SPI	GD25WD40CTIG	512K KB	256B	GigaDevice
SPI	GD25WD20CTIG	256 kB	256B	GigaDevice
SPI	W25X10CL	128 kB	256B	Windbond

SPI	W25X20CL	256 kB	256B	Windbond
I2C	M24M01-R	128 kB	256B	STMicroelectroncis

NOTE

This list is not extensive and other devices compliant to the requirement spec will work as well. Please be aware that the secondary bootloader allows the DA14585/586 to fast boot from an external SPI from other pins.

Table 7: Performance of the SPI memories supported

		W25X20CL	MX25V1006E	AT25XE011	P25Q10U	GD25VQ20C
Supply voltage (V)		2.3 – 3.6	2.35 – 3.6	1.65 – 3.6	1.65 – 3.6	2.3– 3.6
Deep power down current (µA)		1	2	0.4	0.1	1
		Measurements done @ 3.0V				
OTP BLANK	READING 32.56kB (µC)	385	312	941	250	270
	Time to read 32.56 kB (ms)	238	238	238	238	238
	Average current (mA)	2	1.5	4.8	1	1.5
Secondary bootloader burnt	READING 32.56kB (µC)	116.5	86	193	84.66	86
	Time to read 32.56 kB (ms)	38	38	38	38	38

NOTE

These provided values are for indication only. Actual performance should be obtained from the respective datasheet of each product.

10 Layout view of WLCSP + Flash

The following layout proposal for a DA14585 solution with an external Winbond USON SPI Flash is shown. The overall size is 11.5 mm by 9.5 mm. The schematic is shown in the Figure 31.

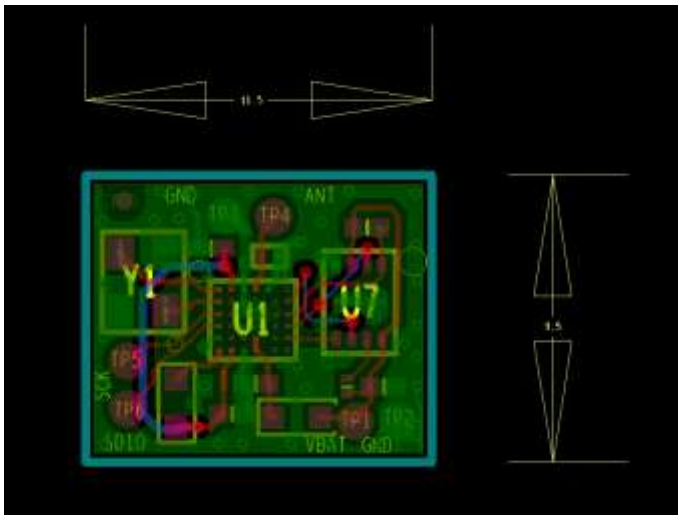

Figure 30: Layout of the DA14585 + external SPI memory

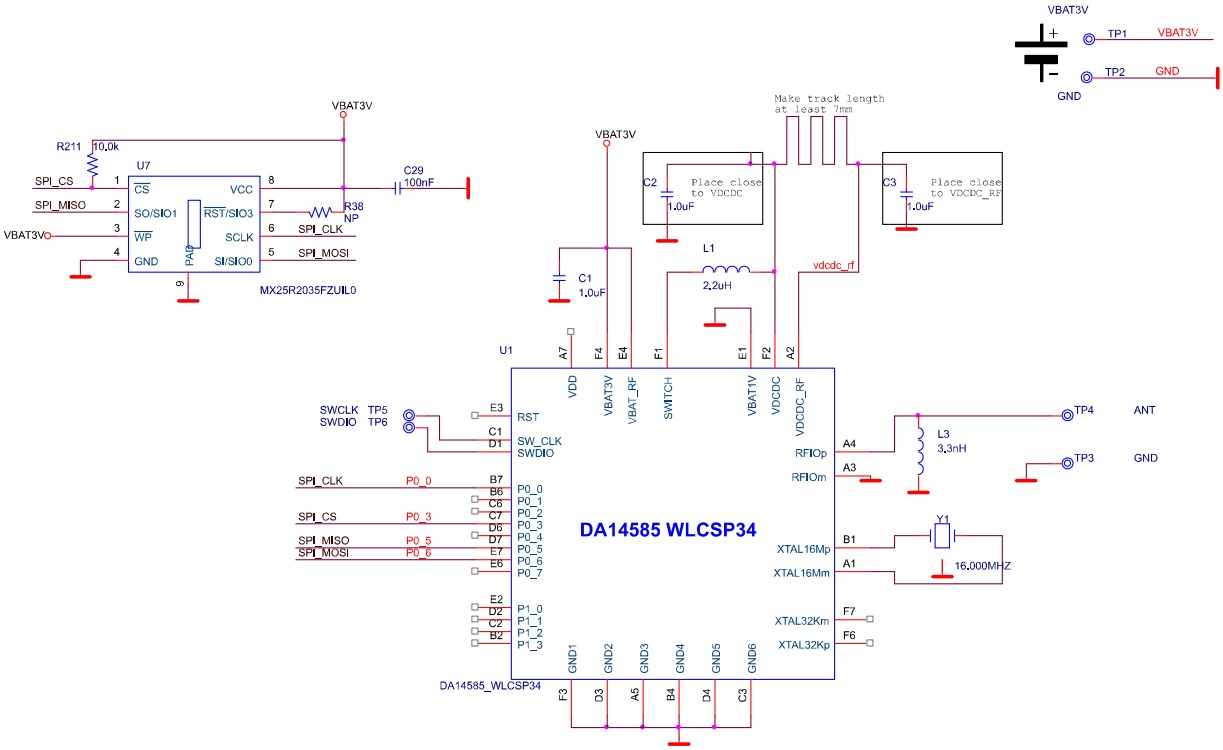

Figure 31: Schematic of the DA14585 + external SPI memory

11 Appendix

In this appendix we are describing some existing modules based on the DA14585 with internal flash such as **Inventek** and **AzureWave**.

11.1 Inventek ISM14585-L35-P8 module

The Inventek Systems ISM14585-L35 SiP (System in Package), is the smallest, lowest power and most integrated Bluetooth® 5.0 solution available. The ISM14585-L35 platform is the first BLE module from Inventek's **FLEXiBLE** product family. The ISM14585-L35 SiP is an embedded wireless Bluetooth Low Energy (BLE) IoT radio, based on the DA14585. The ISM14585-L35 offers designers all the benefits of the industry-leading DA14585 technology but with even greater flexibility to create more advanced applications from the smallest footprints and power budgets.

Table 8: ISM14585-L35 memory overview

Part Number	Memory Options			
	OTP	ROM	RAM	Flash
ISM14585-L35-P8	64kB	128kB	96kB	8 Mbit

Figure 32: ISM14585-L35 Development Kit

11.2 AzureWave AW-CU362

The AzureWave AW-CU362 is an advanced Stamp Bluetooth 5.0 module provides a highly cost-effective, flexible and easy to-use hardware/software device to build a new generation of connected, smart devices. These smart-connected devices enable device to deliver a broad-range of services to consumers including energy-management, demand-response, home automation and remote access. This allows a user to manage comfort and convenience, also run diagnostics and receive alerts and notifications, in addition to managing and controlling the device. Developers can leverage the rich connectivity features of these new smart devices to create a new generation of innovative new applications and services.

The device builds upon the success of the DA14585 device using the Dialog DA14585 and software. Adding new enhancements and capabilities. You can Refer to AW-CU362 + DA14585 web page on dialog web site.

Table 9: AW-CU362 memory overview

Part Number	Memory Options			
	OTP	ROM	RAM	Flash
AW-CU362	64kB	128kB	96kB	1 Mbit

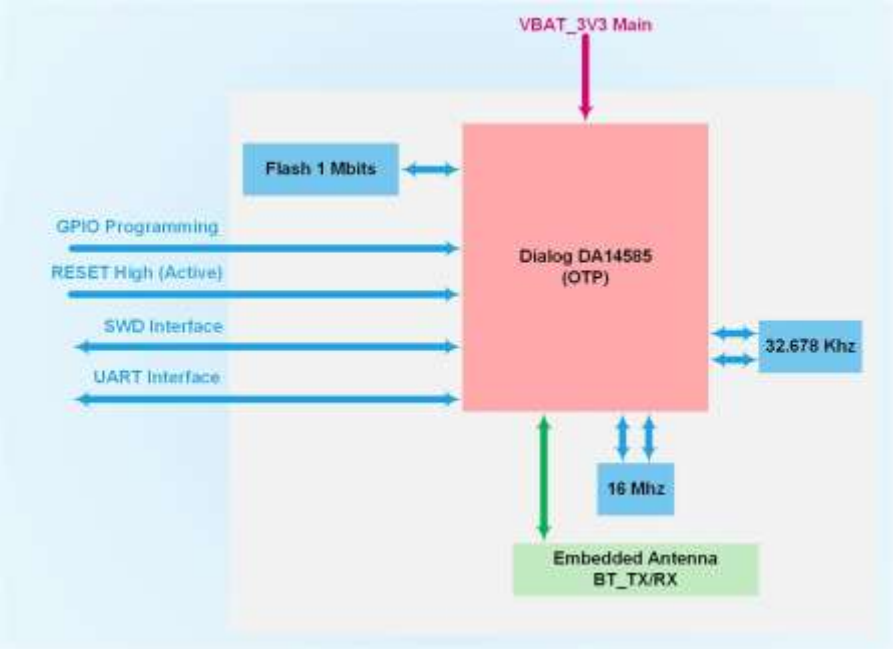

Figure 33: Block Diagram of AW-CU362

Revision history

Revision	Date	Description
1.0	13-March-2017	Initial version.
1.1	17-May-2018	Changes implemented: ● Minor text revisions
1.2	22-May-2019	Changes implemented: ● Update references and figures for DA14585/586 support ● Added Appendix (Inventek, AzureWave modules description)
1.3	11-Nov-2019	Changes implemented: ● Update the Table 6 to add support for AT45DB041E SPI Flash ● Update links on the References section
1.4	14-Jan-2020	Changes implemented: ● Update the Table 6 to add support for these SPI Flash - GD25WD10CTIG - GD25WD40CTIG - GD25WD20CTIG
1.5	24-Feb-2020	Changes implemented: ● Update the Table 6 to add support for these SPI Flash - AT25XE512C - AT25XE011 - AT25XE041D - AT25XE021A
1.6	04-Aug-2020	Changes implemented: ● Update the References section. ● Update the Table 6: update some wrong hyperlinks
1.7	20-Jan-2022	Updated logo, disclaimer, copyright.

Status Definitions

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

RoHS Compliance

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