Application note

QSPI Loader for the DA14681

AN-B-045

Abstract

This document explains how to write a QSPI loader for the DA14681 using our SmartSnippets Studio platform. This will open the possibility to add support for Flash types that are not booting using the ROM booter from the DA14681.

Contents

Abstract **1**

Contents **2**

Tables **3**

Figures **3**

1 Terms and definitions **4**

2 References **4**

3 Introduction **5**

4 Software & Tools **6**

5 BootROM sequence **6**

6 How to support a new QSPI Flash model **9**

7 Example: How to support a QSPI Flash? **13**

7.1 HW configuration 13

7.2 Tool to operate with the QSPI Flash 14

7.3 QSPI Loader SW architecture 15

7.4 Testing the QSPI loader project 17

7.5 Generating the binary file of the QSPI loader project 21

8 Burning the OTP using our SmartSnippets toolbox or PLT **23**

Revision history **25**

Tables

Table 1: Three officially supported QSPI Flash Devices 5
Table 2: Byte code according to the instruction name 8
Table 3: content of the QSPIC_BURSTCMDA_REG 9
Table 4: content of the QSPIC_BURSTCMDB_REG 9
Table 5: QFIS FLASH Initialization Section (QFIS) 11
Table 6: Address for the QSPI Loader code 11
Table 7: QSPI Functions address 12

Figures

Figure 1: DA14681 block diagram 5
Figure 2: DA14681 Development kit PRO 6
Figure 3: SmartSnippets Studio 6
Figure 5: BootROM sequence 7
Figure 6: Sequence of the Fast read Quad I/O command 10
Figure 7: QSPI Flash correctly mounted on the DA14681 daughter boards 13
Figure 8: Final test bench to evaluate a new QSPI Flash 13
Figure 9: SmartSnippets toolbox 14
Figure 10: QSPI Programmer tab using our SmartSnippets Toolbox 14
Figure 11: Architecture of the QSPI loader project 15
Figure 12: C codes for each memory vendors 15
Figure 13: main.c file 16
Figure 14: Memory selected for booting 16
Figure 15: QSPI Flash content 17
Figure 16: Step 1 to debug the QSPI loader project 18
Figure 17: Step 2 to debug the QSPI loader project 18
Figure 18: Beginning of the Debugging procedure 18
Figure 19: Step 3 to debug the QSPI loader project 19
Figure 20: Memory map of the DA14681 19
Figure 21: Memory windows view 19
Figure 22: Step 4 to debug the QSPI loader project 20
Figure 23: Step 5 to debug the QSPI loader project 20
Figure 24: Actual binary file of the QSPI loader 22
Figure 25: Size of the actual binary file shown after compilation 22
Figure 26: QFIS parameters burnt in OTP using SmartSnippets toolbox 23
Figure 27: DA14681 advertising 24

1 Terms and definitions

FLASH	Non-volatile memory which can be electrically erased and reprogrammed.
GUI	Graphic User Interface
HW	HardWare
PLT	Production Line Tool
QFIS	QSPI FLASH Initialization Section
QSPI	Quad Serial Peripheral Interface
QSPIC	Quad Serial Peripheral Interface Controller
SDK	Software Development Kit
SW	SoftWare
XIP	Execute-In-Place

2 References

[1] DA14681 Datasheet, Dialog Semiconductor.

[2] UM-B-044 DA1468x Software Platform Reference, Dialog Semiconductor.

[3] UM-B-041 DA1458x/68x Production Line Tool Hardware and GUI, Dialog Semiconductor.

3 Introduction

The DA14681 which is based on an ARM Cortex-M0 CPU provides a flexible memory architecture, enabling code execution from embedded memory (RAM, ROM) or non-volatile memory (OTP or external FLASH memory).

This application note will describe how to make the DA14681 bootable from any Flash memories using the QSPI interface. This document only explains the creation of a program to bypass the ROM booter. Fully functional Flash support also needs updates of the drivers and programming scripts. This is explained section 10.2 Non Volatile Memory storage from [2].
The QSPI loader project is available from the DIALOG support website and must **be copied** in the path: `your_workspace/DA1468x_SDK_BTLE_v_x.x.x.xxx/utilities`. This project will be used to generate the actual QSPI loader binary file which will be burnt afterwards into the OTP of the DA14681.

The maximum size of the QSPI loader binary file is **2016 Bytes** see Figure 25].

A cache controller is used for code execution directly from OTP or external QSPI Flashes while DataRAM is used to store variables, stacks, heaps and application data. The QSPI controller supports single, dual and quad SPI.

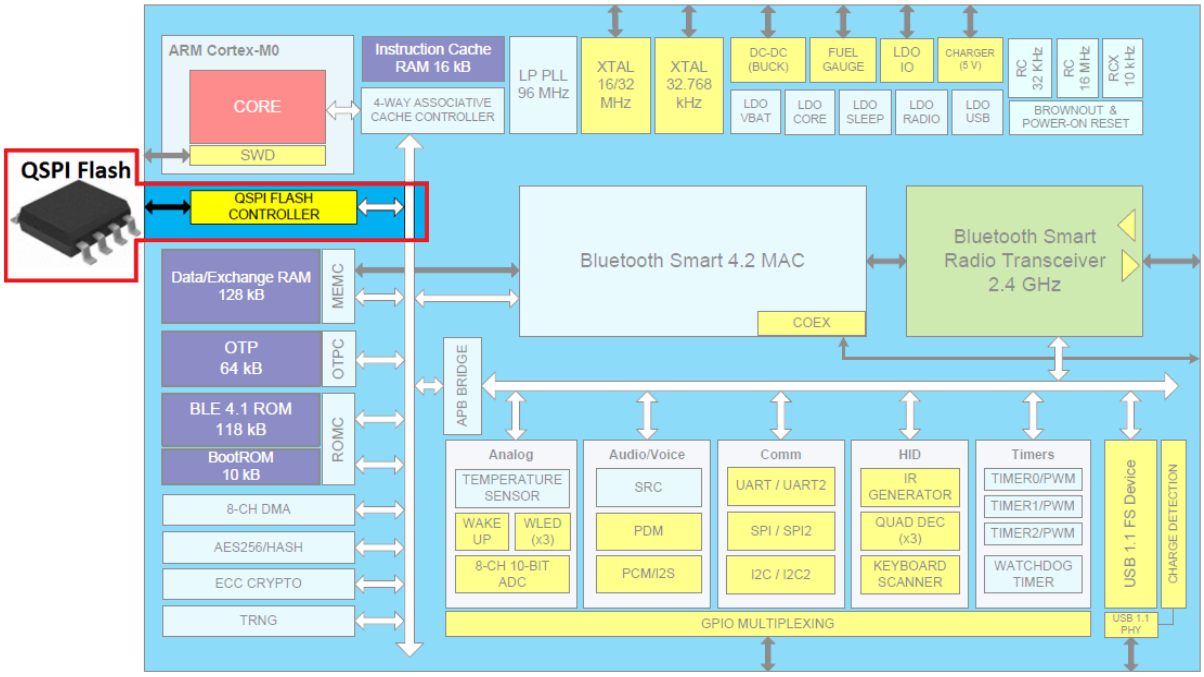


Figure 1: DA14681 block diagram

The DA14681 can boot and operate correctly using these three flash types mentioned in Table 1. Other flash devices with compatible boot sequence will also work without the need for a QFIS loader but might need updates in the driver or programming scripts (see section 10.2 from [2]).

Table 1: Three officially supported QSPI Flash Devices

Flash version	Flash vendor
GD25LQ80B, 8Mbits	GigaDevice
MX25U51245G, 512Mbits	MXIC
W25Q80EW, 8Mbits	winbond

4 Software & Tools

The tools and software needed to perform the tests are the following:

A DIALOG DA14681 Development kit Pro. The DIALOG website indicates our distributors and partners.

This includes the following:

- Mother board PRO
- USB cable/ Coin cell battery
- DA14681 daughterboard

Figure 2: DA14681 Development kit PRO

The SmartSnippets Studio (which includes SmartSnippets Toolbox) which can run on both Windows and Linux can be downloaded from our portal in the Software & Tools section:

https://support.dialog-semiconductor.com/

Figure 3: SmartSnippets Studio

A python script is used when the compilation is done to generate the actual hex file of the QSPI loader. Make sure to install Python 3.5 or above version from: https://www.python.org/downloads/

5 BootROM sequence

The Figure 4] shows the booting sequence of the DA14681. We will be focusing on the QFIS loader block highlighted at the bottom left corner in the Figure 4]. It will be used in order to boot from a specific QSPI Flash model. The QFIS loader will be used to copy the QSPI Flash content into the RAM of the DA14681.

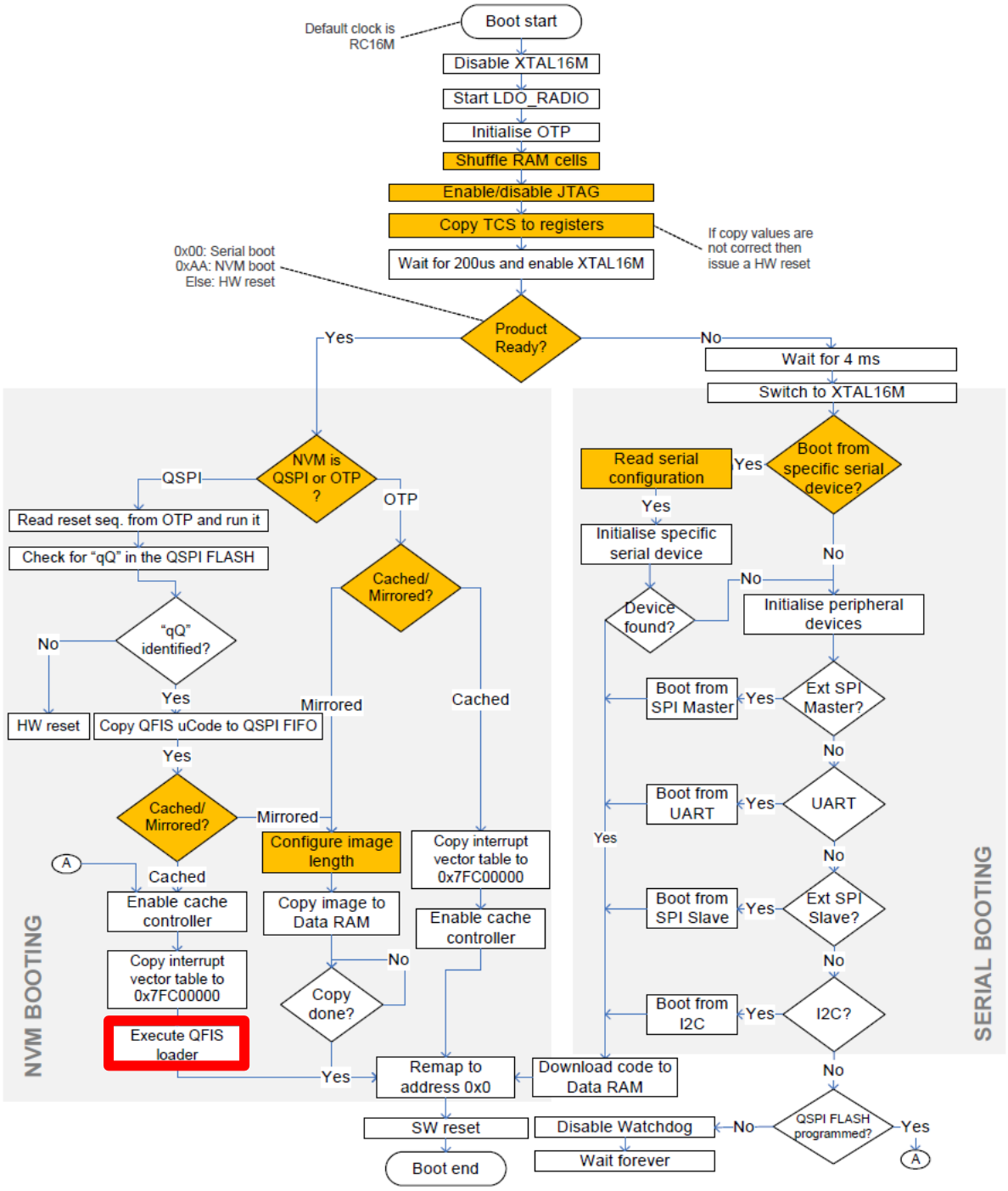


Figure 4: BootROM sequence

The BootROM of the DA14681 automatically takes care of the instructions mentioned below:

IMPORTANT NOTE
The QSPI loader project also follows the same sequence which can be modified (especially the properties of the Fast Read Quad I/O (FAST_READ) command) to make the DA14681 bootable from a specific QSPI Flash.

- Reset Quad I/O (RSTQIO) of the QSPI Flash:

  The Reset Quad I/O instruction resets the device to 1-bit Standard SPI operation.

- Reset-Enable (RSTEN) & Reset-Memory (RST) the QSPI Flash:

  The Reset operation is used as a system (software) reset that puts the device in normal operating Ready mode. This operation consists of two commands: Reset-Enable (RSTEN) and Reset (RST).

- Release from Deep Power-Down and Read Device ID (RDI):

  Once the device has entered the Deep Power-down mode, all instructions are ignored except the Release from Deep Power-down and Read Device ID (RDI) instruction. Executing this instruction takes the device out of the Deep Power-down mode.

The Table 2 summarizes the reset sequence of the BootROM including the Byte code for each single RESET commands.

Table 2: Byte code according to the instruction name

Instruction name	Byte 1 Code
Reset Quad I/O or Fast Read Enhance Mode (RSTQIO)	0xFF
Reset-Enable (RSTEN)	0x66
Reset-Memory (RST)	0x99
Release from Deep Power-Down, and read Device ID (RDI)	0xAB

IMPORTANT NOTE
If the Flash which needs to boot from the DA14681 does have the same Byte codes, the reset sequence does not need to be implemented again in the QSPI loader project.

- The Auto mode which is used to execute from FLASH.

  The Auto Mode is used to execute in QSPI Flash cached mode. The Auto Mode is up-to 32 Mbyte transparent Code access for XIP (Execute In Place) and Data access with 3-byte and 4-byte addressing modes.

  XIP mode allows the memory to be read by sending an address to the device and then receiving the data on one, two, or four pins in parallel, depending on the customer requirements. It is a method of executing programs directly from long term storage rather than copying it into RAM. XIP mode offers maximum flexibility to the application, saves instruction overhead, and reduces random access time.

  In the case of Auto Mode of operation the QSPIC generates a sequence of control signals in SPI BUS. This sequence of control signals is analysed to the following **phases**:

  - Instruction phase
  - Address phase
  - Extra Byte phase
  - Dummy clocks phase
  - Read data phase

These phases are programmed via the registers:

- QSPIC_BURSTCMDA_REG (command register A to read in Auto Mode)
- QSPIC_BURSTCMDB_REG (command register B to read in Auto Mode)

- The properties of Fast Read Quad I/O (FAST_READ) command are configured via the register QSPIC_BURSTCMDA_REG. This command must be used to boot from the FLASH.

If the reset sequence of the QSPI Flash does totally match the one described just above, only the 2 following registers need to be correctly programmed in the QSPI loader project.

- QSPIC_BURSTCMDA_REG (more information can be found from [1], section 37 Registers)
- QSPIC_BURSTCMDB_REG (more information can be found from [1], section 37 Registers)

The Table 3 & Table 4 summarize the content of QSPIC_BURSTCMDA_REG & QSPIC_BURSTCMDA_REG respectively.

Table 3: content of the QSPIC_BURSTCMDA_REG

Content of QSPIC_BURSTCMDA_REG	Bits position
Command value (IncBurst, Single)	[7:0]
Command value (WrapBurst)	[15:8]
Extra Byte	[23:16]
Command Transmit Mode	[25:24]
Address Transmit Mode	[27:26]
Extra Byte Transmit Mode	[29:28]
Dummy Bytes Transmit Mode	[31:30]

Table 4: content of the QSPIC_BURSTCMDB_REG

Content of QSPIC_BURSTCMDB_REG	Bits position
Read Data Receive Mode	[1:0]
Extra Byte Enable	[2]
Extra Half Byte Disable Out	[3]
Num of Dummy Bytes	[5:4]
Command Mode	[6]
Wrap Mode	[7]
Wrap Length	[9:8]
Wrap Size	[11:10]
CS High Min Number of CLKs	[14:12]

6 How to support a new QSPI Flash model

The QSPI Flash models from other vendors can be used by doing the following steps.
The Table 5, Table 6 & Table 7 are extracted from [1], section 3.3.

STEP 1:

The QSPI_Loader project from the path: `\utilities\QSPI_loader` using SmartSnippets Studio must be modified according to the QSPI Flash specification to generate the correct QSPI Loader binary file.

As mentioned in the previous section, the 2 below registers have to be correctly configured using the QSPI Loader project.

- `QSPIC_BURSTCMDA_REG`
- `QSPIC_BURSTCMDB_REG`

To do so, the first thing to do is to look at the timing diagram of the Fast Read Quad I/O (FAST_READ) instruction of the Flash vendor which looks like the following sequence:

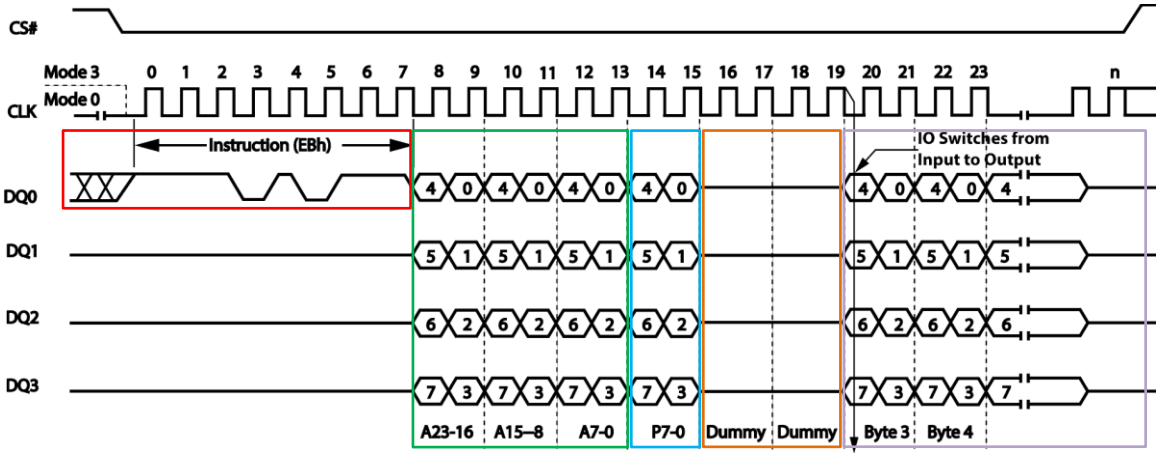


Figure 5: Sequence of the Fast read Quad I/O command

Instruction:

An instruction (in our case: Fast read Quad I/O (0xEB)) is sent to the Flash memory specifying the type of operation to be performed. This is done in SINGLE SPI mode.

IMPORTANT NOTE
In case the QSPI Flash has a Command transmit mode which is using a Quad SPI mode, the Enable Quad Peripheral mode (EQPI) instruction must be enabled before the reset sequence. This can be done using the following command: `SetWord8(QSPIC_WRITEDATA_REG, 0xXX);  //0xXX is the EQPI value of the Flash vendor` This will enable the flash device to support Full Quad SPI Mode.

Address:

An address is sent to the Flash, specifying the address of the data to be read or written. This is done in QUAD SPI mode. Address is automatically set in the QSPI loader project. No need to take care of that.

Extra-Byte (or Performance enhance indicator):

Extra-Byte field supported by the QSPI SPI interface offers more flexibility. It must be used with the Fast read Quad I/O command and is generally used for controlling the mode of operation (e.g. to keep the memory in Execute-In-Place mode).

Dummy bytes:

The dummy-cycle phase is needed in some cases when operating at high clock frequencies. This phase allows to ensure “turnaround” time for changing the data signals from output mode to input mode. In our case, the SPI clock frequency when booting is set to 8 MHz. Therefore, 2 dummy bytes are enough.

Data:

The data is sent or received from or to the QSPI memory using QUAD SPI mode.
In our case, it will be the reading of the content of the QSPI Flash.

STEP 2:

The QSPI loader binary file generated from the QSPI_Loader project has to be burnt in the OTP of the DA14681 at offset: 0xF818. This is part of the QFIS FLASH Initialization Section (see [1], section 3.3 SYSTEM CONFIGURATION).

Table 5: QFIS FLASH Initialization Section (QFIS)

Address	Size (B)	Field name	Description
0x7F8F818	2016	Contains all QSPI related code segments	

0x7F8F818 is the start address of the customer functions. You are free to select an address in this area. But it makes sense to start at the beginning, unless you have to put other QSPI functions there as well, like the reset function.

STEP 3:

Another part of the OTP must be programmed which is at the address 0x7F8F808.
In this section, the length and the address of the QSPI loader must be programmed. This is part of the QFIS FLASH Initialization Section.

Table 6: Address for the QSPI Loader code

Address	Size (B)	Field name	Description
0x7F8F808	8	Address for the QSPI Loader code	B7-B5: Section length (Bytes) B3-0: Address

STEP 4:

The last part of the OTP which must be programmed is at the address 0x7F8EA48 which contains the QSPI functions. In this section, only the Bit2 has to be set to 1.

Table 7: QSPI Functions address

Address	Size (B)	Field name	Description
0x7F8EA48	8	QSPI Functions	Bit0 0: Reset Function of QSPI FLASH is in BootROM 1: Reset Function of QSPI FLASH is in OTP Bit1 0: Find ‘qQ’ Function of QSPI FLASH is in BootROM 1: Find ‘qQ’ Function of QSPI FLASH is in OTP Bit2 0: QSPI Loader of QSPI FLASH is in BootROM **1: QSPI Loader of QSPI FLASH is in OTP**

7 Example: How to support a QSPI Flash?

In this section, we will see how to support a new QSPI FLASH. We will call it: XYZ_Flash.

7.1 HW configuration

The combination of the DA14681 mother & daughter boards can be used to perform some tests with a new QSPI Flash. Make sure you replace the current QSPI Flash mounted on the DA14681 with the new QSPI Flash as shown in Figure 6.
The power supply of the Flash delivered from the DA14681 (QSPI_VDDIO pin) is 1.80V.

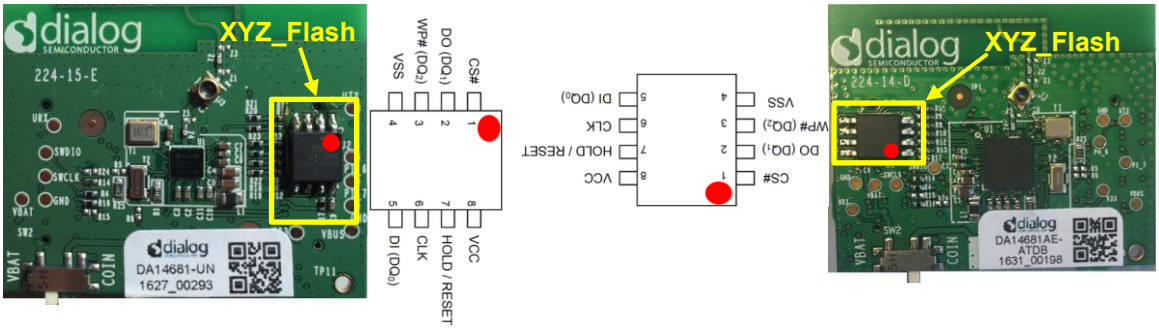


Figure 6: QSPI Flash correctly mounted on the DA14681 daughter boards

IMPORTANT NOTE
Our SmartSnippets toolbox supports the QSPI Flashes with both HOLD and RESET signal on pin #7 (IO3) to program, read & erase the Flash content. It is using the single SPI mode because the majority of the SPI Flashes are using the same operational codes for program, read & erase.

The final setup with the correct jumper positions on the mother board should looks like the following:

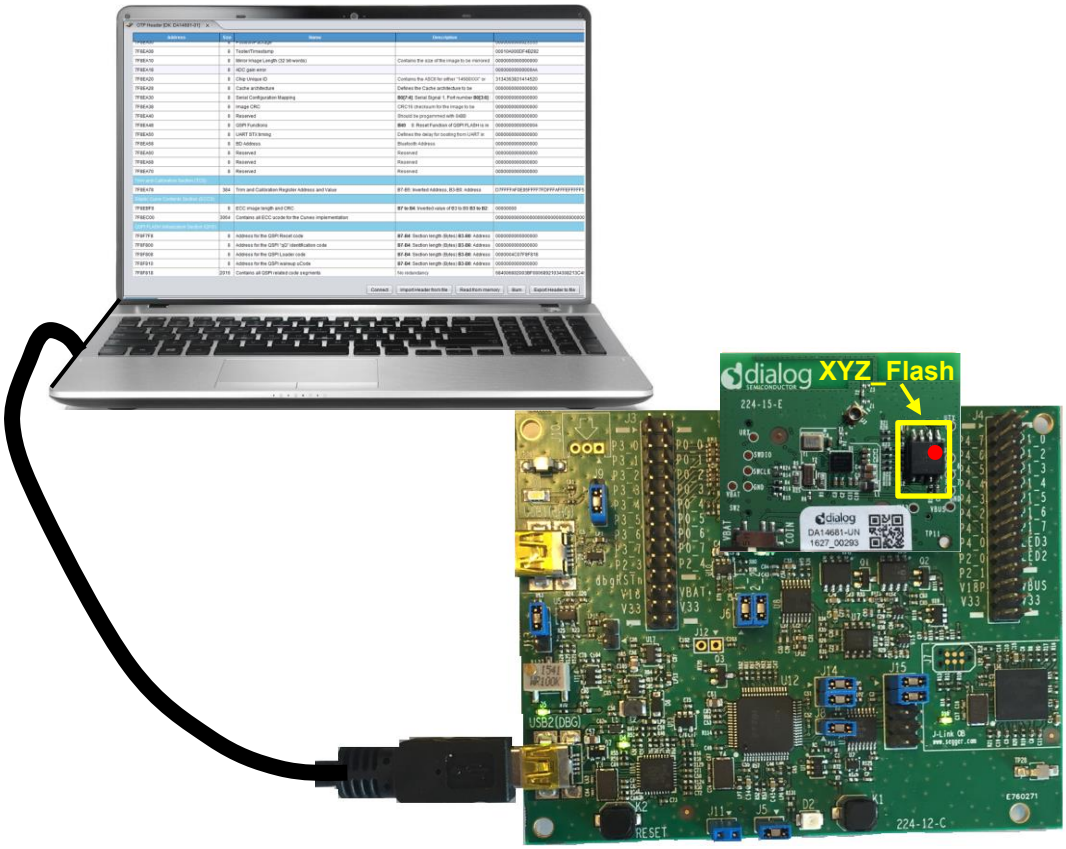


Figure 7: Final test bench to evaluate a new QSPI Flash

7.2 Tool to operate with the QSPI Flash

Our Smart Snippets toolbox can be used to Read / Erase / Program the QSPI Flash.

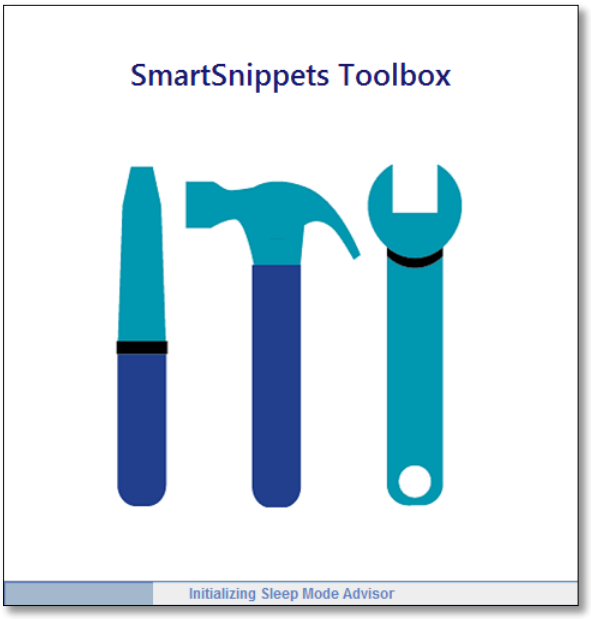


Figure 8: SmartSnippets toolbox

This tool can be downloaded from (with a windows machine):

http://support.dialog-semiconductor.com/resource/smartsnippetsstudiov12-windows-os

Or in case you have a Linux machine:

http://support.dialog-semiconductor.com/resource/smartsnippetsstudiov12-linux-os

To operate with the QSPI Flash, the QSPI Programmer tab must be used. This is show below. More information about how to use our tool can be found from the help tab.

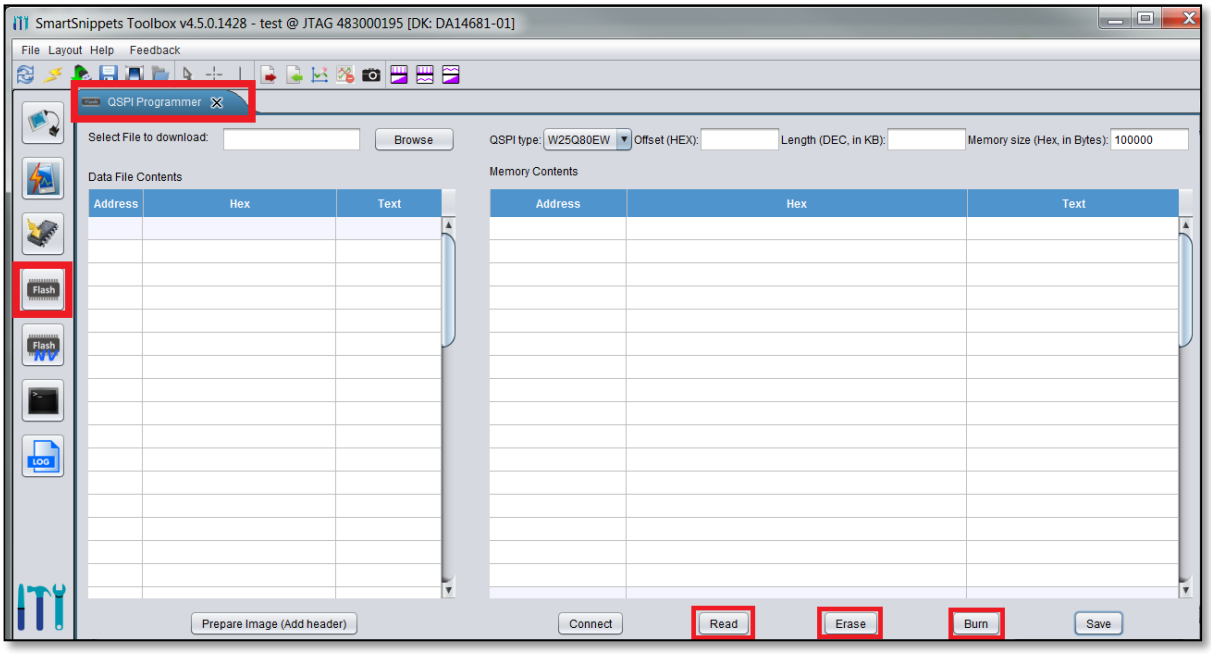


Figure 9: QSPI Programmer tab using our SmartSnippets Toolbox

7.3 QSPI Loader SW architecture

As already mentioned earlier, the QSPI Loader project **must be copied** in the path: `DA1468x_SDK_BTLE_v_x.x.x.xxx/utilities`.

This project must be imported using our SmartSnippets Studio.

Then, the proper programming macro in the `config.h` file needs to be defined, based on your flash memory selection for your application.
An example is shown in Figure 10, where WINBOND is used for demonstration.

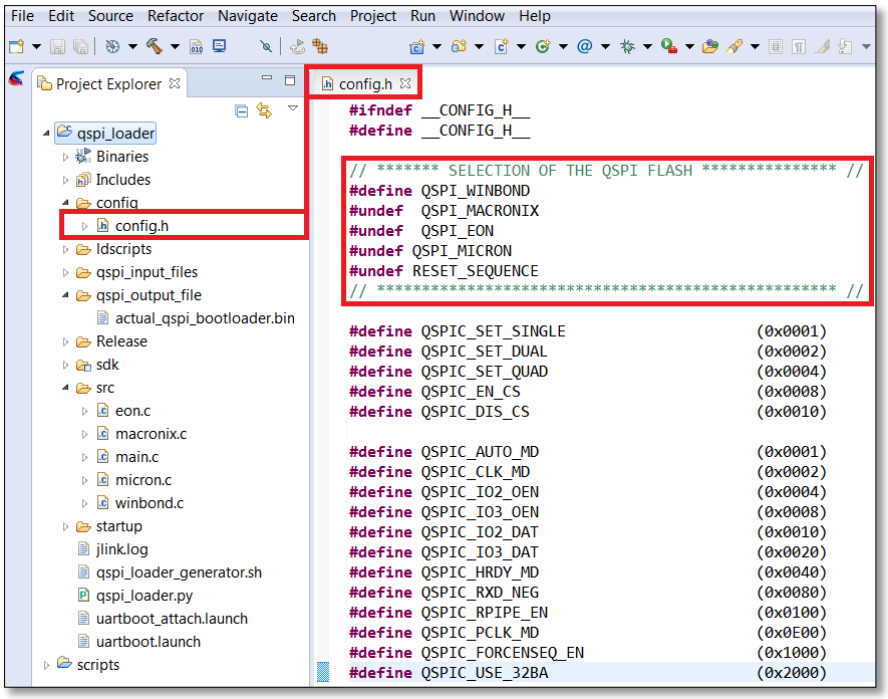


Figure 10: Architecture of the QSPI loader project

In the source folder, you can find the C files for each memory vendors.

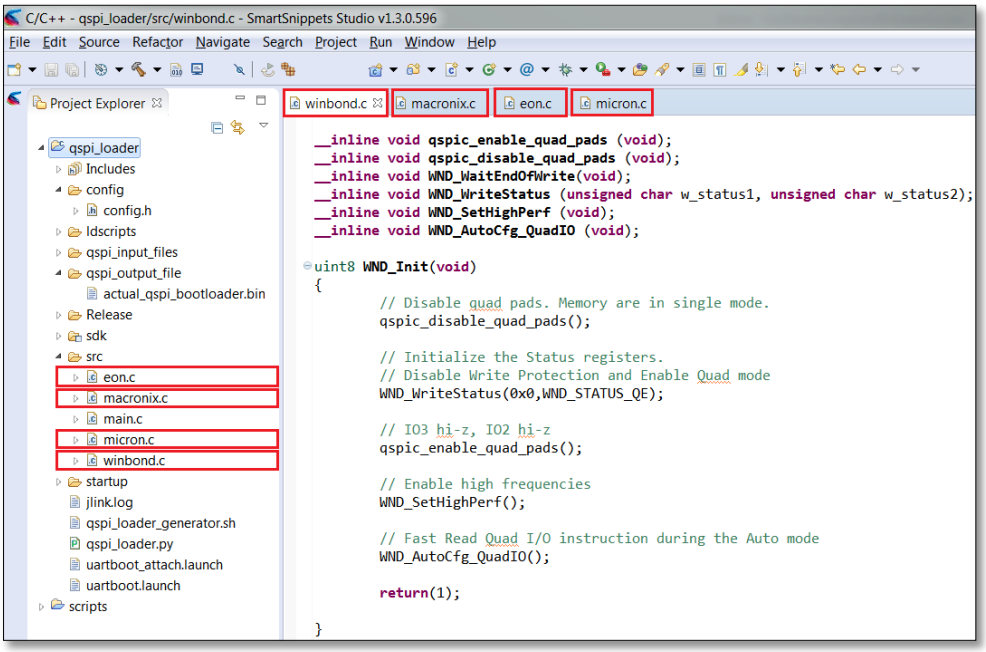


Figure 11: C codes for each memory vendors

In the `main.c` file, you can find the memory manufacturer IDs and the memory vendor which will be used for booting as shown in Figure 12.

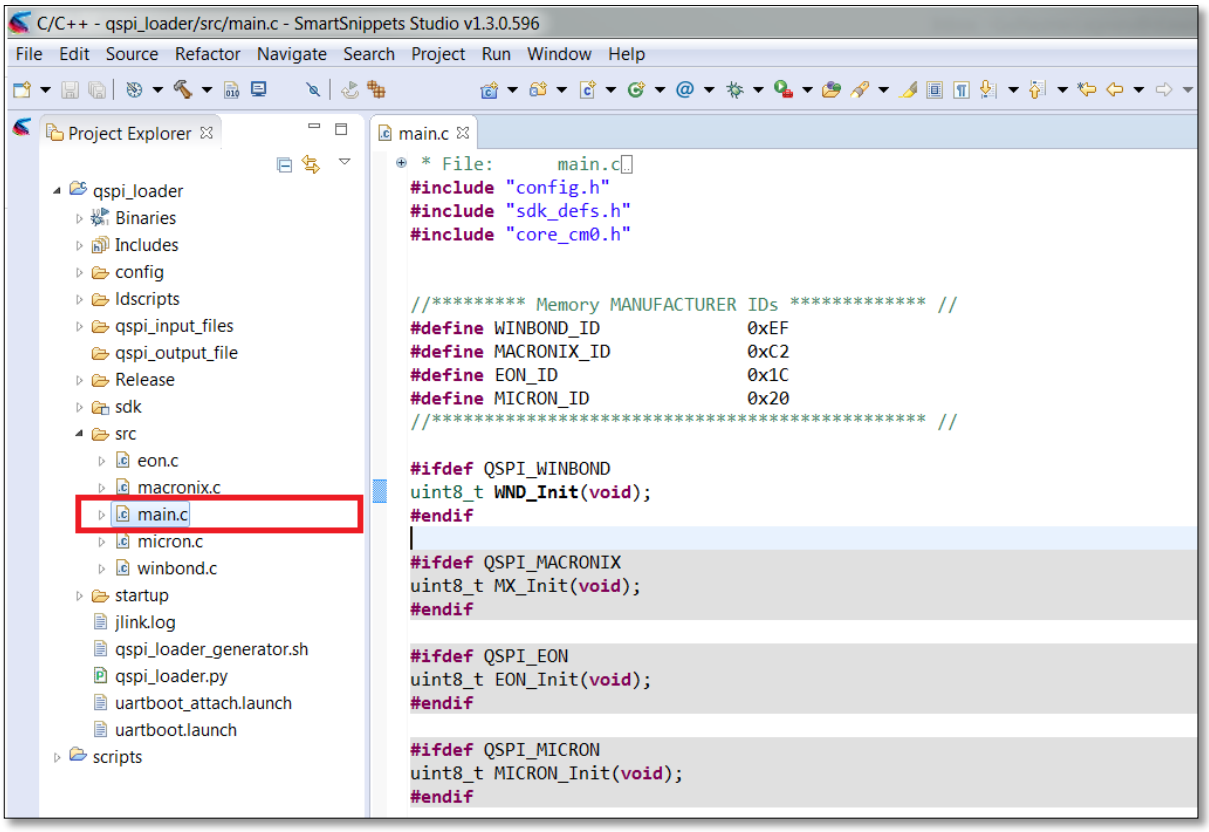


Figure 12: main.c file

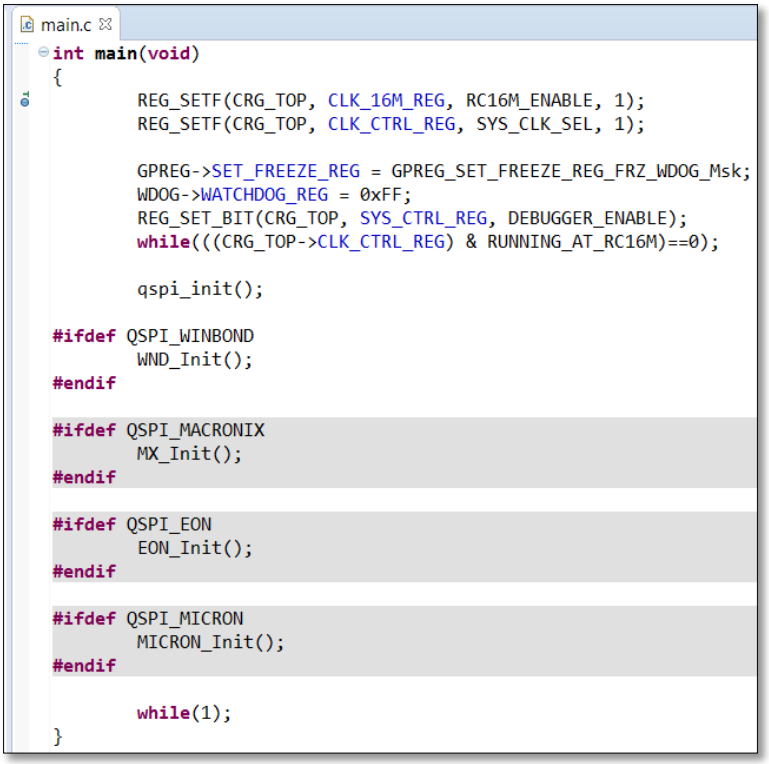


Figure 13: Memory selected for booting

7.4 Testing the QSPI loader project

For testing purposes, our SmartSnippets tool can be used to write any images in the QSPI Flash.

In order to know how to use our SmartSnippets tool, please refer to the HELP tab.

We first need to program the QSPI flash using the *.HEX file of the ble_adv demo project of the DA14681 with the purpose to debug the QSPI loader later on.

This gives us the following content:

Memory Contents

Address	Hex	Text
0x00000	71 51 00 00 80 01 1E 94	qQ ｦ ﾔ
0x00008	00 80 FC 07 BD 02 00 08	ｦ□ □
0x00010	59 03 00 08 59 03 00 08	Y Y
0x00018	00 00 00 00 00 00 00 00	
0x00020	00 00 00 00 00 00 00 00	
0x00028	00 00 00 00 00 00 00 00	
0x00030	00 00 00 00 65 03 00 08	e
0x00038	00 00 00 00 00 00 00 00	
0x00040	75 B9 00 08 59 03 00 08	u□ Y
0x00048	11 17 FD 07 E1 16 FD 07	□ £ □
0x00050	59 03 00 08 59 03 00 08	Y Y
0x00058	19 1D 00 08 59 03 00 08	Y
0x00060	45 C7 F0 07 59 03 00 08	E□□ Y
0x00068	E9 22 00 08 F9 22 00 08	ｪ" □"
0x00070	59 03 00 08 59 03 00 08	Y Y
0x00078	59 03 00 08 59 03 00 08	Y Y
0x00080	79 0F 00 08 59 03 00 08	y Y
0x00088	59 03 00 08 39 3D 00 08	Y 9=
0x00090	59 03 00 08 25 10 FD 07	Y % □
0x00098	59 03 00 08 59 03 00 08	Y Y
0x000A0	59 03 00 08 59 03 00 08	Y Y
0x000A8	59 03 00 08 31 23 00 08	Y 1#
0x000B0	19 0B 00 08 85 41 00 08	ｦA
0x000B8	7D 20 00 08 59 03 00 08	} Y
0x000C0	4D 35 00 08 59 03 00 08	M5 Y
0x000C8	31 01 00 08 31 01 00 08	1 1
0x000D0	31 01 00 08 31 01 00 08	1 1
0x000D8	31 01 00 08 31 01 00 08	1 1
0x000E0	31 01 00 08 31 01 00 08	1 1
0x000E8	31 01 00 08 31 01 00 08	1 1
0x000F0	31 01 00 08 31 01 00 08	1 1
0x000F8	31 01 00 08 31 01 00 08	1 1
0x00100	31 01 00 08 31 01 00 08	1 1

Figure 14: QSPI Flash content

IMPORTANT NOTE
Our SmartSnippets toolbox supports the QSPI Flashes with both HOLD and RESET signal on pin #7 (IO3) to program, read & erase the Flash content. It is using the single SPI mode because the majority of the SPI Flashes are using the same operational codes for program, read & erase.

The ble_adv demo now resides in flash but in order for this program to execute, the QSPI loader needs to be programmed in OTP. This is explained in section [8].

Now, going back to our SmartSnippet Studio platform, we can start debugging the QSPI loader project by going through the following steps:

Step 1: Click on the Build button and select Release.

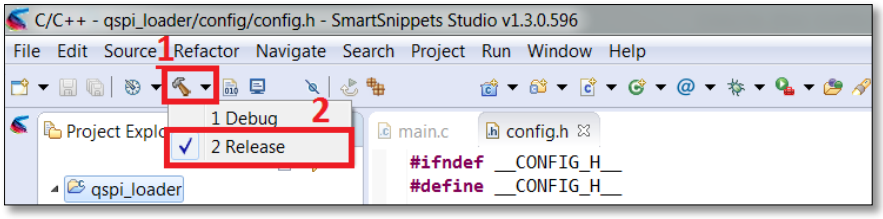


Figure 15: Step 1 to debug the QSPI loader project

Step 2: Click on the Debug button and download the code into the RAM of the DA14681:

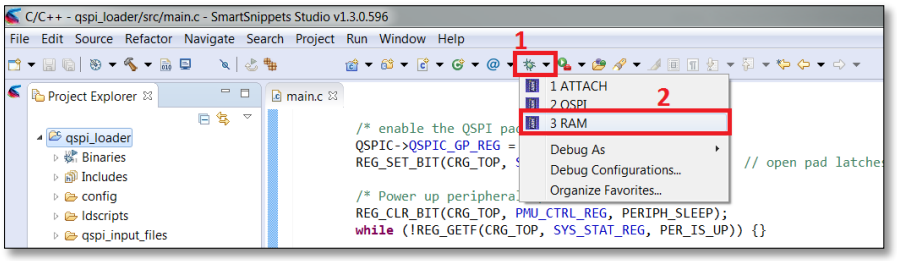


Figure 16: Step 2 to debug the QSPI loader project

At this point, you should have the following windows:

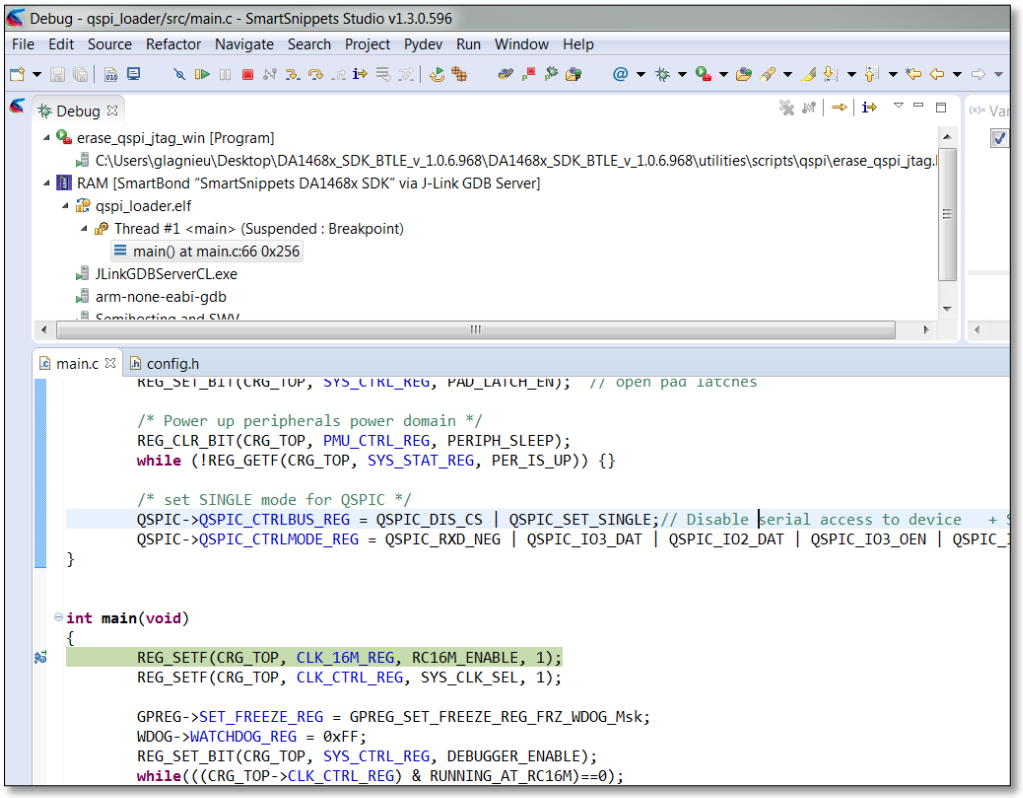


Figure 17: Beginning of the Debugging procedure

Step 3: A memory watch windows needs to be added. Address has to be set to: 0x8000000.

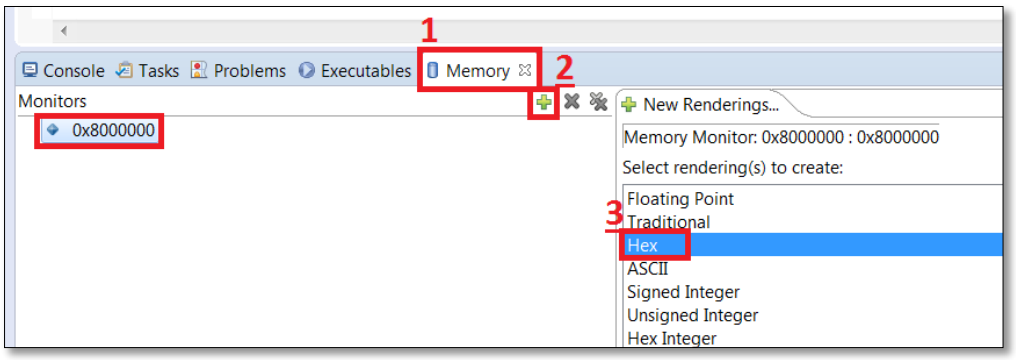


Figure 18: Step 3 to debug the QSPI loader project

Indeed, the aim of the QSPI loader project is to read the QSPI content. From the memory map of the DA14681, the reading of the QSPI flash content is done from the address 0x8000000 as shown in the Figure 19.

36 Memory map

This section contains a detailed view of the DA14681 memory map.

Table 61: Memory Map

Address	Description	Power Domain	AMBA
0x0	Remapped Device		
0x7F00000	ROM	SYS_PD	AHB
0x7F40000	OTPC	SYS_PD	AHB
0x7F80000	OTP	SYS_PD	AHB
0x7FC0000	DataRAM	SYS_PD	AHB
0x7FE0000	CacheRAM	SYS_PD	AHB
0x8000000	QSPI FLASH	SYS_PD	AHB
0xC000000	QSPIC	SYS_PD	AHB

Figure 19: Memory map of the DA14681

You should then have the following windows:

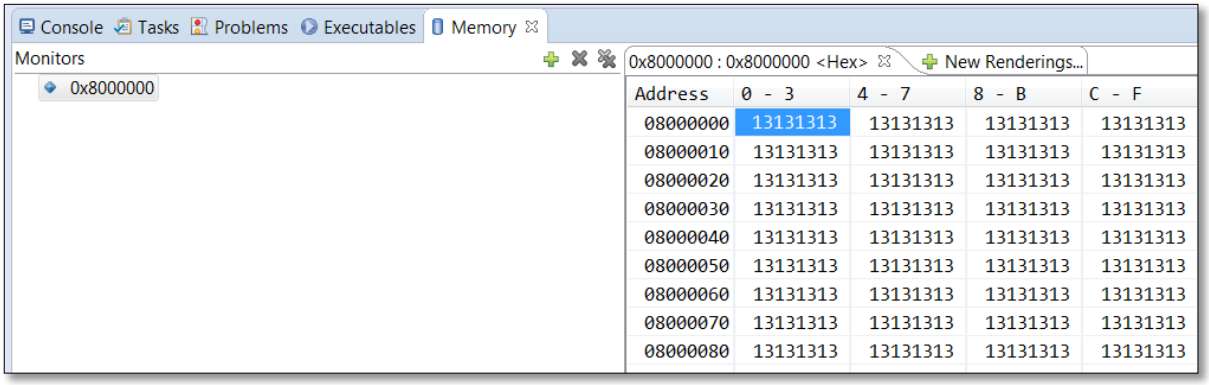


Figure 20: Memory windows view

Step 4: First, press the Resume button and after a short time (minimum 1 second), press the Suspend button.

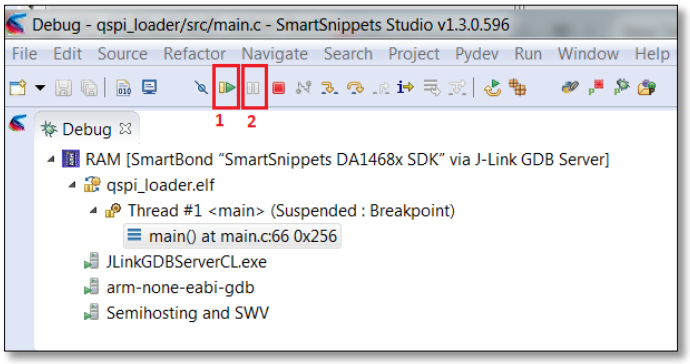


Figure 21: Step 4 to debug the QSPI loader project

Step 5: From now, you can check the content of the memory windows. Those are the data read in the QSPI Flash from the DA14681. You should read the same data shown when you read the memory content using our tool SmartSnippets toolbox.

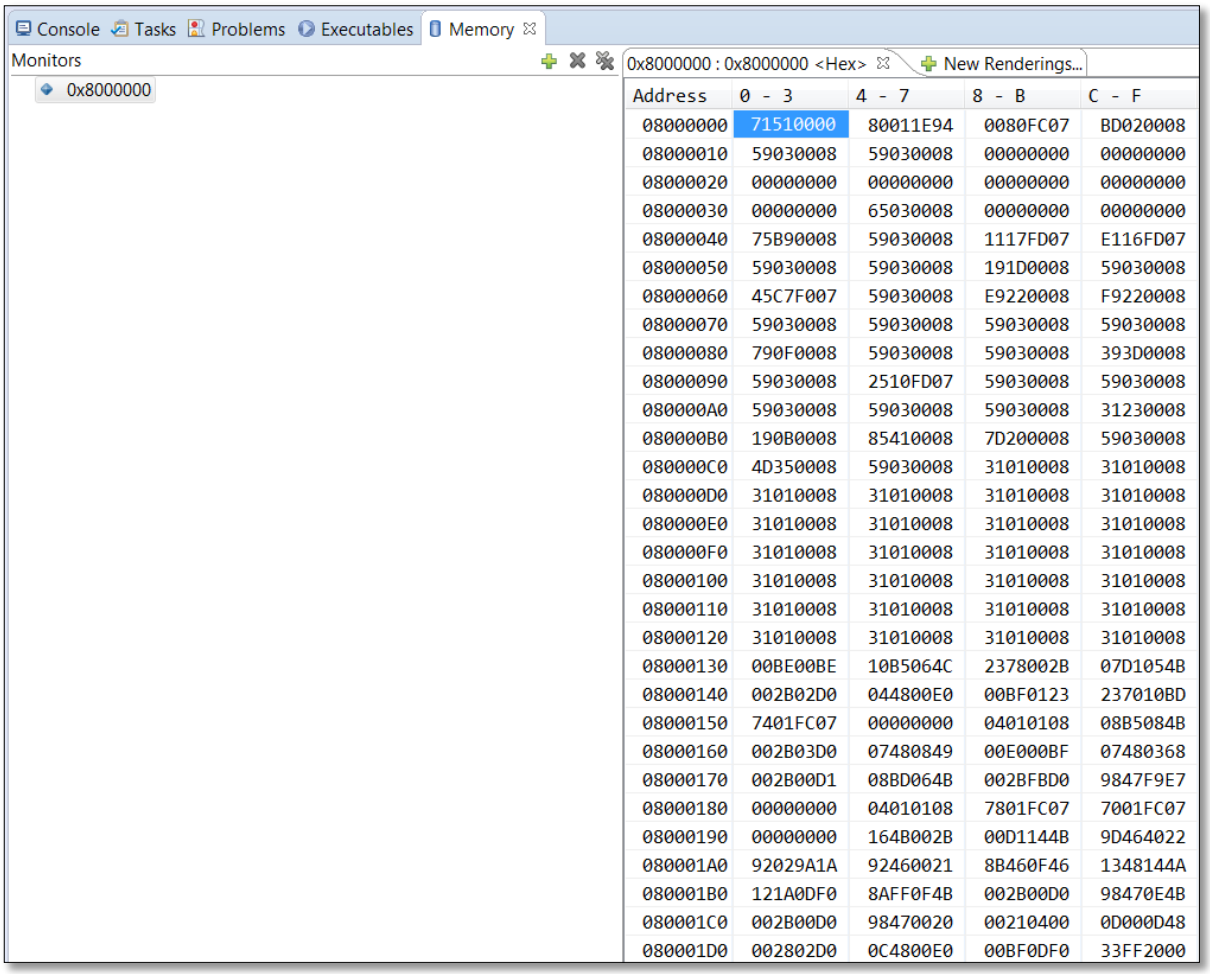


Figure 22: Step 5 to debug the QSPI loader project

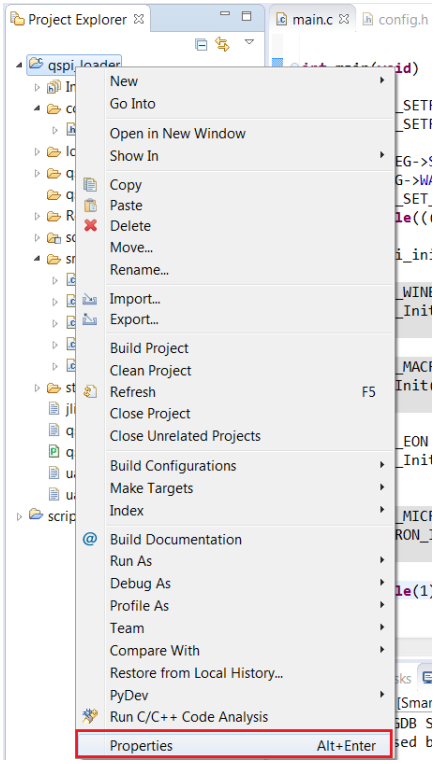
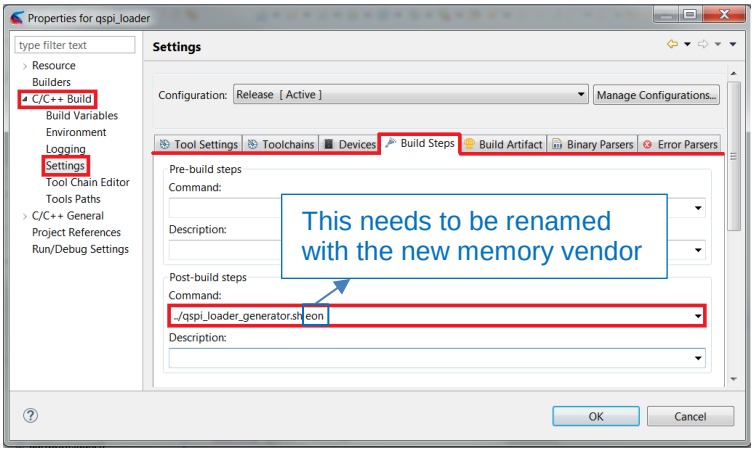
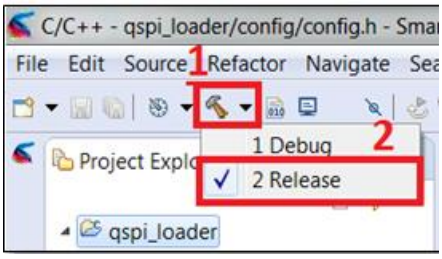
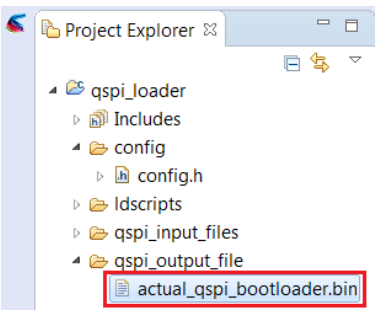
At this point, you have proven that the QSPI loader you have created works properly!

7.5 Generating the binary file of the QSPI loader project

Once you have verified that the QSPI loader works correctly, this is the time to generate the binary file.

First, having created you own QSPI loader, you need to mention the flash vendor in Post-build steps.

This can be done by going through the following steps:

<table>
<tr><td>STEP 1:
Right click on the project and then select Properties.</td><td>STEP 2:
Go to C/C++ Build, then Settings and choose the Build Steps tab.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>STEP 3:
Click on the Build button and select Release.</td><td>STEP 4:
The hex file is generated in the qspi_output_file folder. The binary file can be opened using NotePad++.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

After compiling the project, the python script has been added in the post-build steps to:

- Generate the actual binary file of the QSPI loader.
  This binary file will be burnt in the OTP of the DA14681 at the address: 0x7F8F818 (Address which contains all QSPI related code segments) or at the offset 0xF818.

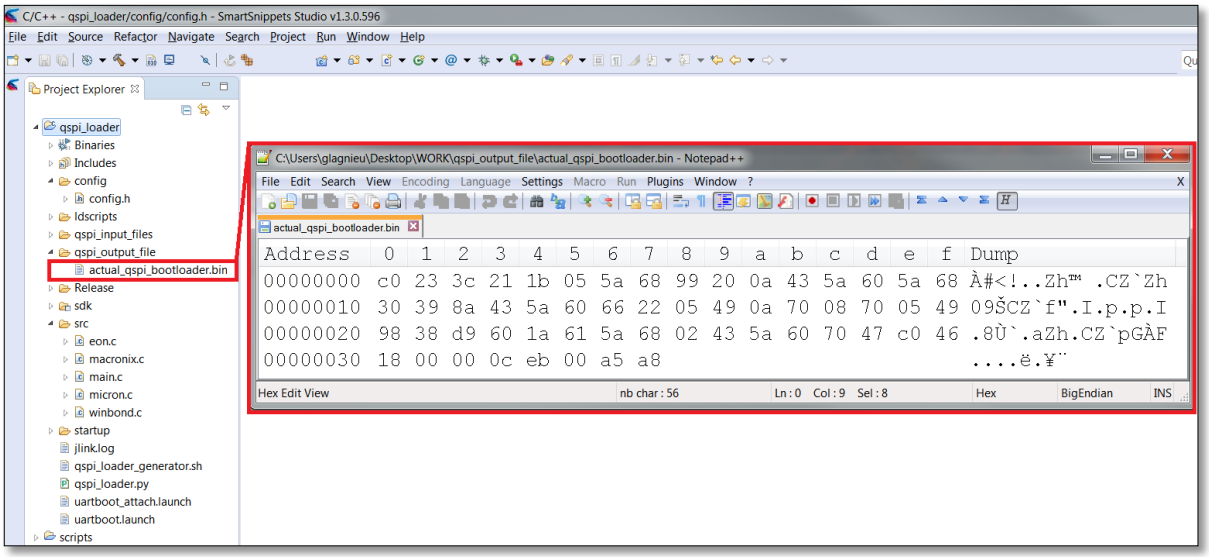


Figure 23: Actual binary file of the QSPI loader

- Show the size of the actual binary file.
  This size of the binary file will be burnt in the OTP of the DA14681 at the address: 0x78F808 (Address for the QSPI Loader code (Byte7-Byte5: Section length (Bytes)).

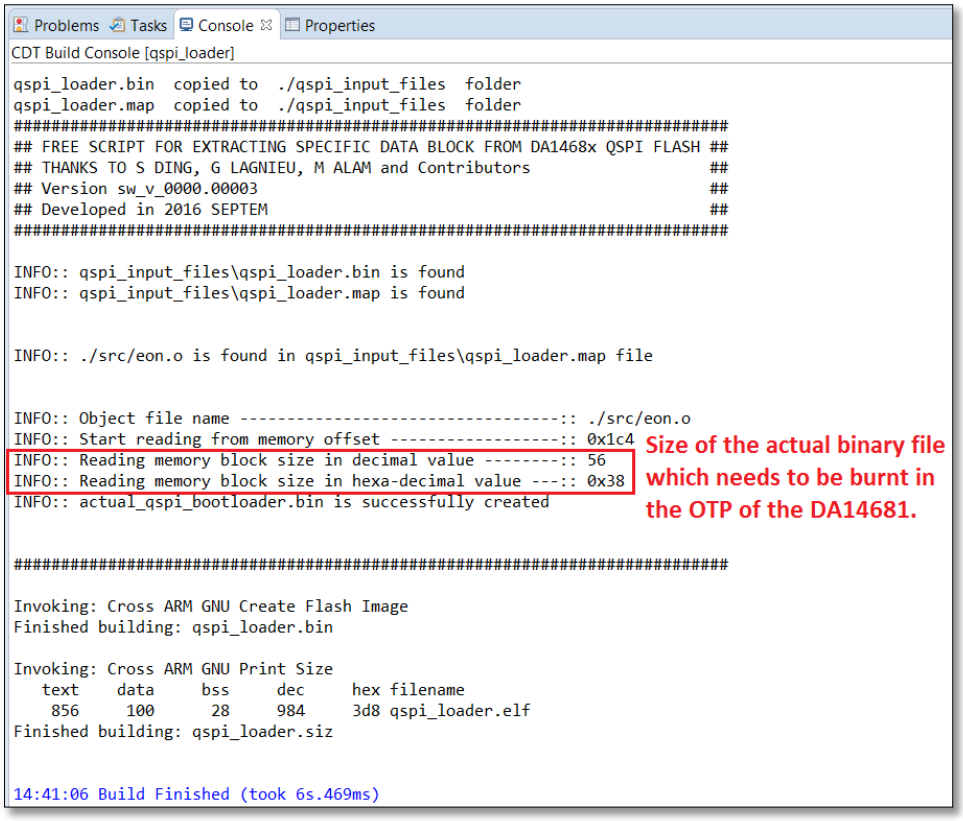


Figure 24: Size of the actual binary file shown after compilation

8 Burning the OTP using our SmartSnippets toolbox or PLT

To burn the QSPI loader in the OTP using the PLT tool, please refer to [3].

Having gone through all the steps listed in the section [6], you should have the following:

The parameter **QSPI Functions** (at address 0x7F8EA48) has to be set to 0x04 using the OTP header tab.

Bit0 = 0: Reset function of QSPI Flash is in BootROM

Bit1 = 0: Find “qQ” Function of QSPI FLASH is in BootROM

Bit2 = 1: QSPI loader of QSPI Flash is in OTP

IMPORTANT NOTE
You can only set this field (QSPI Functions) once. So if you want to add a different function to OTP, like for example the Reset function, you cannot enter this extra bit here, due to the ECC error it will produce.

The parameter **Address for the QSPI Loader code** (at address 0x7F8F808) has to be set to 0x4C07F8F818 using the OTP header tab.

B3-B0: Address.
This must be the starting address where the QSPI loader has been burnt (at address 0x07F8F818)

B7-B5: Section length (in Bytes).

From the Python script, when the output binary is generated, the size of the binary file is shown.

In our case, it is 0x4C (76 Bytes).

The parameter **Contains all QSPI related code segments** (at address 0x7F8F818 or offset 0xF818) should use the QSPI loader binary file.

Therefore, we should end up having the following:

Address	Size	Name	Description	Value
7F8EA48	8	QSPI Functions	**Bit0** 0: Reset Function of QSPI FLASH is in	0000000000000004
7F8EA50	8	UART STX timing	Defines the delay for booting from UART in	0000000000000000
7F8EA58	8	BD Address	Bluetooth Address	0000000000000000
7F8EA60	8	Reserved	Reserved	0000000000000000
7F8EA68	8	Reserved	Reserved	0000000000000000
7F8EA70	8	Reserved	Reserved	0000000000000000
Trim and Calibration Section (TCS)				
7F8EA78	384	Trim and Calibration Register Address and Value	B7-B5: Inverted Address, B3-B0: Address	D7FFFFAF0E95FFFF7
Elliptic Curve Contents Section (ECCS)				
7F8EBF8	8	ECC image length and CRC	**B7 to B4**: Inverted value of B3 to B0 **B3 to B2**:	00000000
7F8EC00	3064	Contains all ECC ucode for the Curves implementation		0000000000000000
QSPI FLASH Initialization Section (QFIS)				
7F8F7F8	8	Address for the QSPI Reset code	**B7-B4**: Section length (Bytes) **B3-B0**: Address	0000000000000000
7F8F800	8	Address for the QSPI "qQ" identification code	**B7-B4**: Section length (Bytes) **B3-B0**: Address	0000000000000000
7F8F808	8	Address for the QSPI Loader code	**B7-B4**: Section length (Bytes) **B3-B0**: Address	0000004C07F8F818
7F8F810	8	Address for the QSPI wakeup uCode	**B7-B4**: Section length (Bytes) **B3-B0**: Address	0000000000000000
7F8F818	2016	Contains all QSPI related code segments	No redundancy	684006802003BF000

Figure 25: QFIS parameters burnt in OTP using SmartSnippets toolbox

IMPORTANT NOTE
When burning the value of a parameters in the OTP, for ‘integer’ data-type fields, the least significant byte of a word is stored in the lowest address (little-endian).

Then after the burning of the OTP is completed, after having an HW Reset, we can see our board advertising.

So, the hex file from the ble_adv demo which was previously programmed in Flash is now executing.

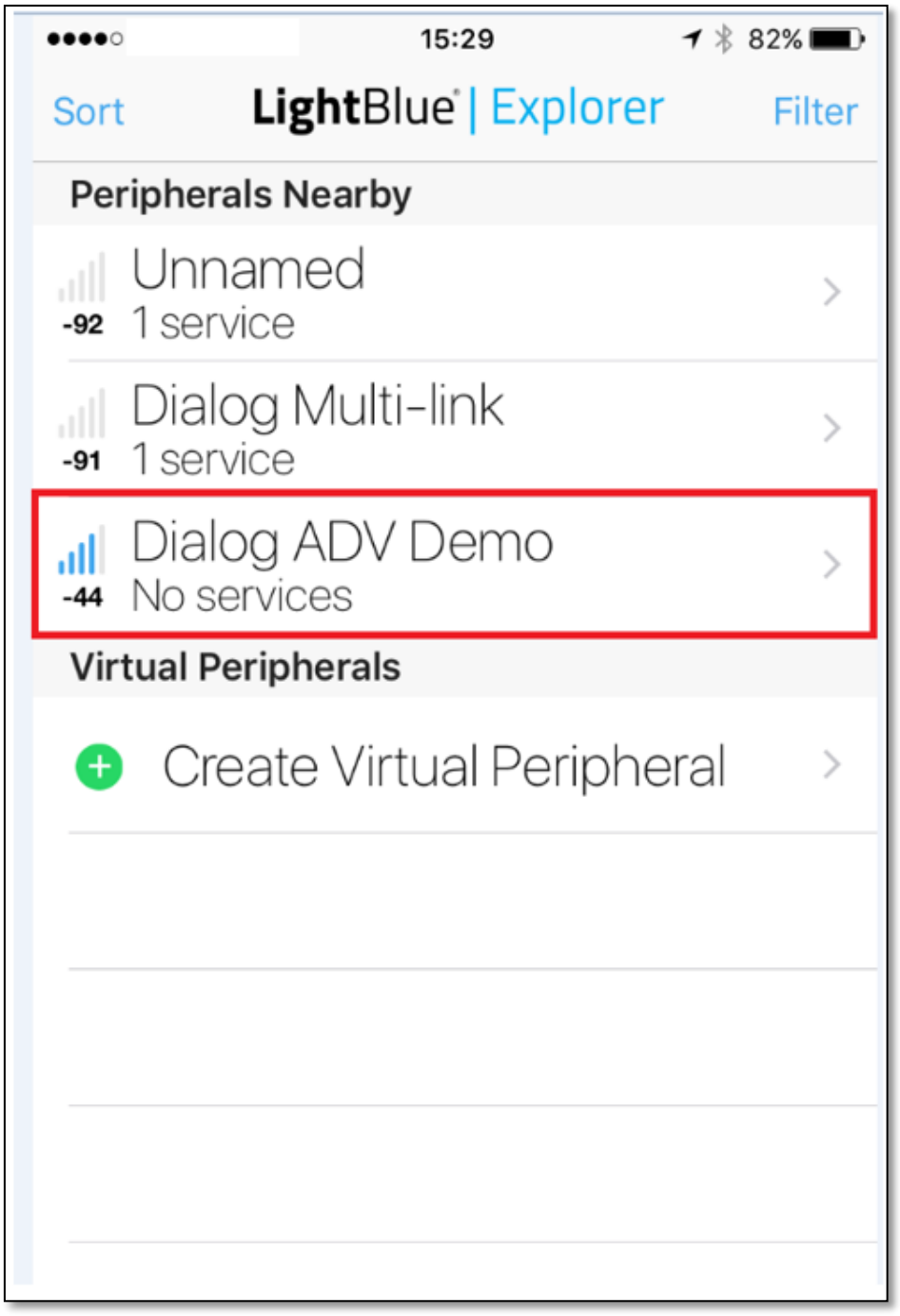


Figure 26: DA14681 advertising

Revision history

Revision	Date	Description
1.1	25-Feb-2022	Updated logo, disclaimer, copyright.
1.0	15-Dec-2016	Initial version.

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 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.

QSPI Loader for the DA14681

Important Notice and Disclaimer

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

https://www.renesas.com/contact/