# Firmware update via SD card

## XMC4000

## About this document

### Scope and purpose

This application note describes how to make use of the Alternate Boot Mode (ABM) to perform a firmware update via the SD card.

## Table of contents

# 1 Introduction

As the XMC4000 microcontroller is equipped with an internal flash memory, firmware can be erased and reprogrammed. This gives product manufacturers the opportunity to make progressive improvements to their firmware addressing software bugs or enhancing product performance.

This project is capable of performing flash programming during software run time.The latest firmware is stored in the SD card in the form of a binary file called "**firmware.bin**" which is read and programmed into the XMC4500 flash when a button is pressed.

This provides the following benefits

- Simple - no special GUI software or stack required
- Convenient and portable as no PC or cable connection is required
- Only single flash programming is required during production
- Applicable to all XMC4000 series with SD card support

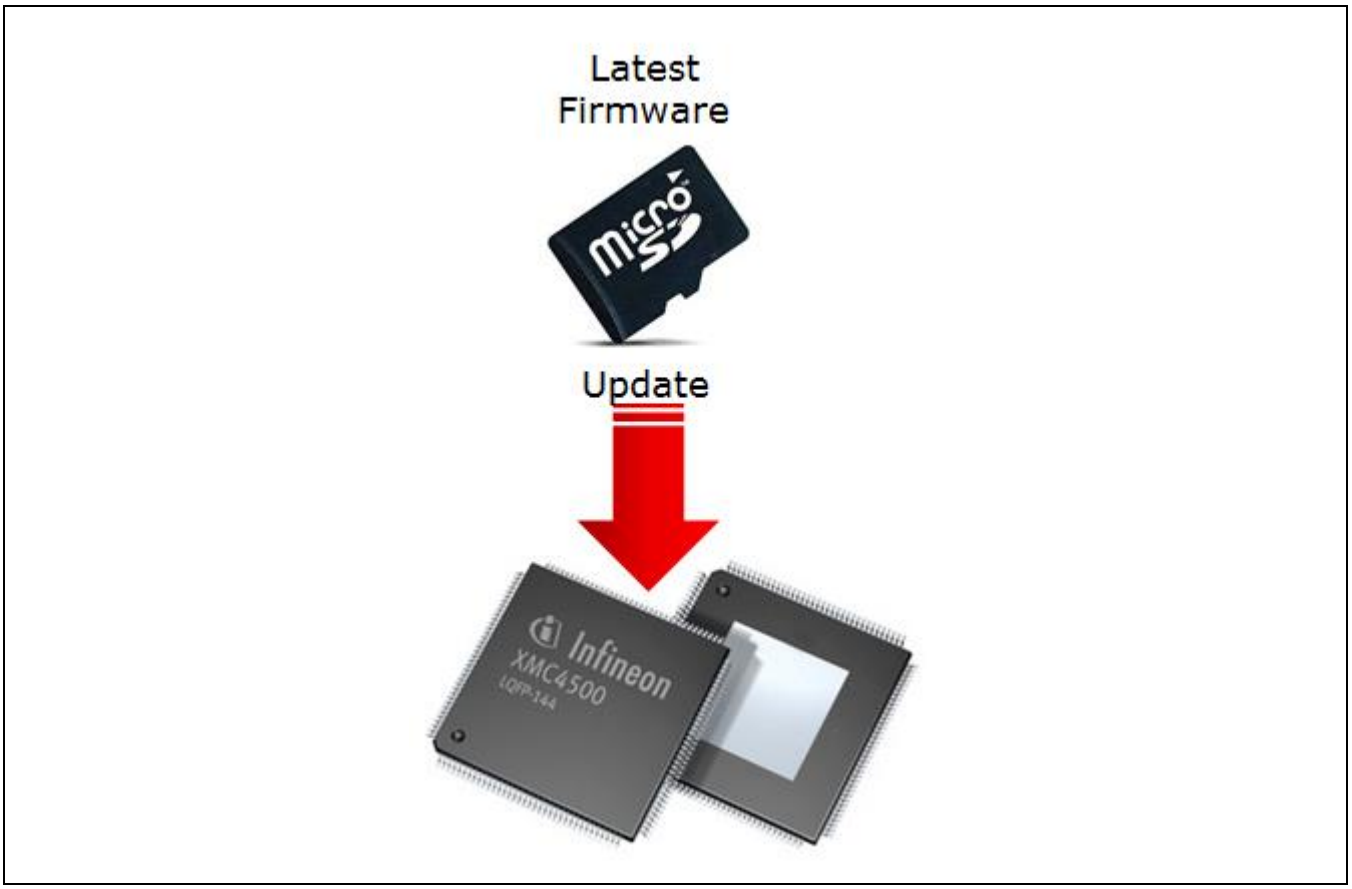


**Figure 1 Firmware update concept**

# 2 Implementation overview

The complete implementation of this firmware update via SD card involves creating the following 3 DAVE™4 projects with different objectives.

**DAVE™4 projects**

1. XMC45_FlashLoader project
   - › Reading SD card
   - › Flash program the XMC™ flash
   - › Create the ABM header
2. XMC45_Production_Firmware project
   - › Install ABM header
   - › Include both the application and flash loader software
   - › Conditional jump into flash programming during run time
   - › Production firmware package
3. XMC45_Update_Firmware A or B project
   - › Latest application firmware update

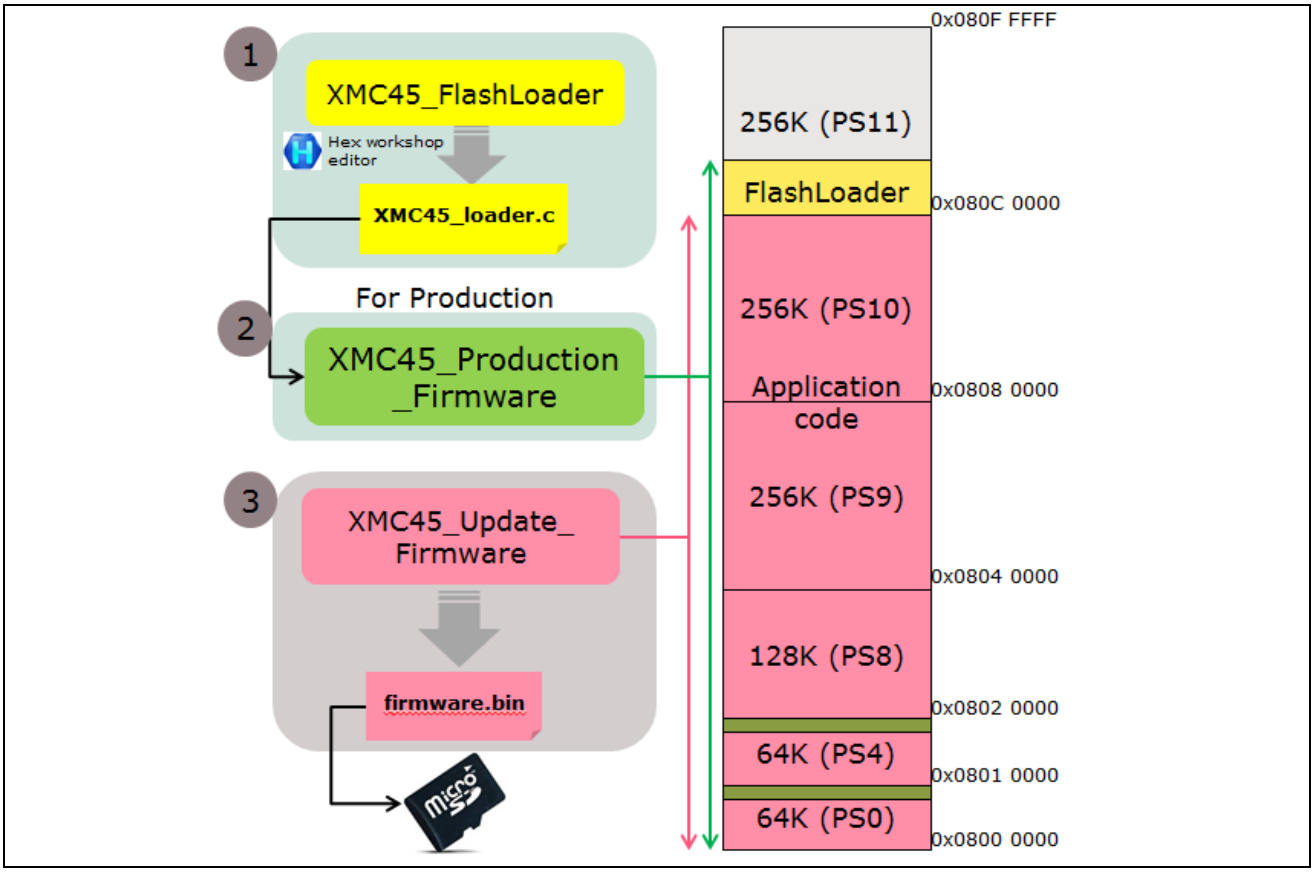


Figure 2 Implementation overview

## 2.1 Required tools

### Hardware

- XMC4500 relax kit
- SD card
- SD card adaptor
- USB cable
- PLS UAD2 debugger (optional)

### Software installation

- PLS UDE debugger (optional)
- DAVE4 version 4.2.6
- Hex workshop - http://www.hexworkshop.com/

### Preparation

Please setup the hardware connection as shown below and copy the required files to the windows desktop.

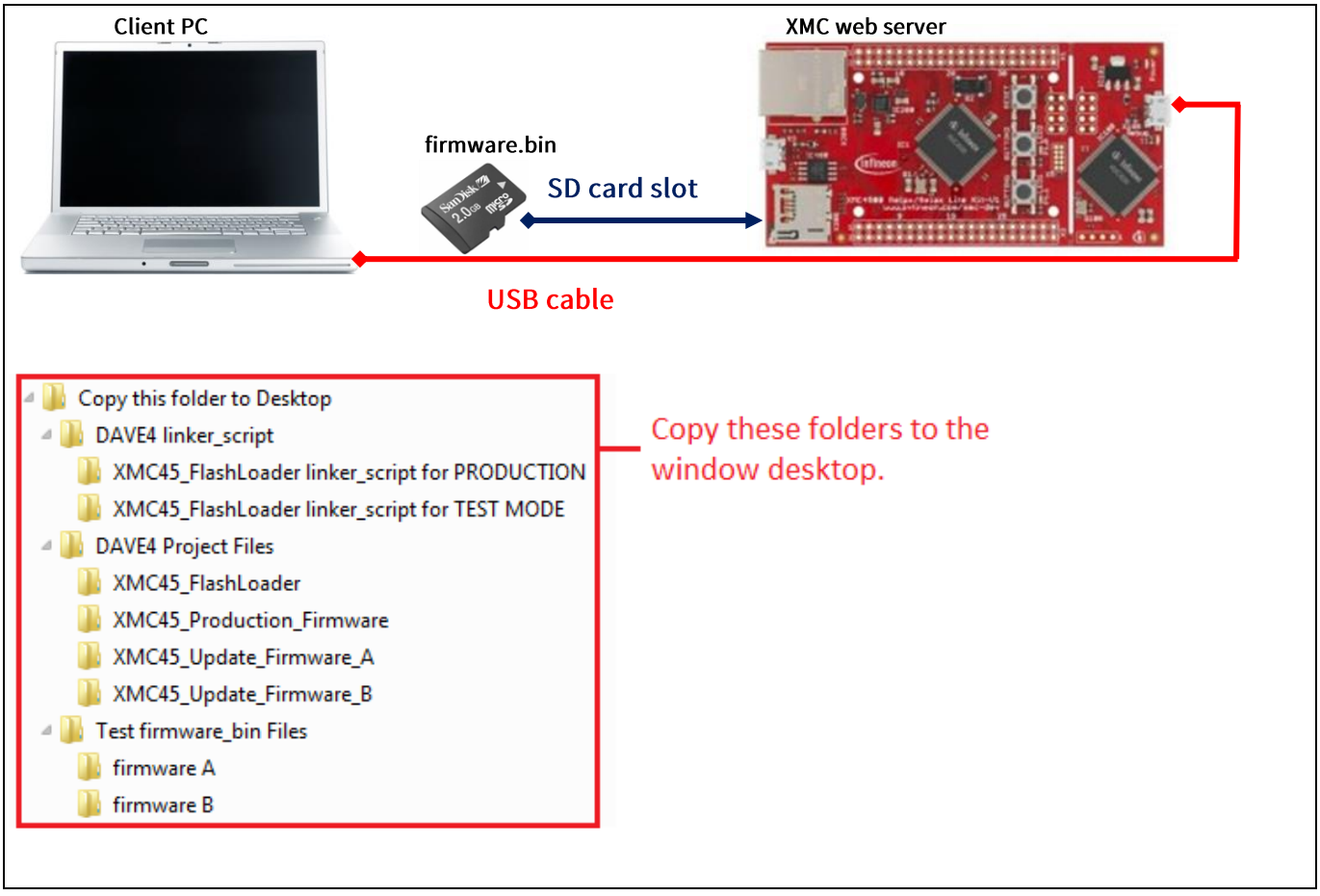


Figure 3 Hardware connection and projects files

## 2.2 Alternate Boot Mode (ABM)

As normal boot mode does not support the reading of SD cards, we shall make use of ABM for this application.

An application (eg.Flash loader) located at a user defined location on the flash is given control by the Startup Software (SSW). The SSW, after completing its execution, evaluates the ABM header stored at a defined address on the flash which, in turn, provides the location of the application placed at a user defined address. Two such applications can be programmed into the flash and, thus, two ABMs are supported. An invalid header results in the SSW aborting further execution and launching the CPU into safe mode. A PORST is required to exit the safe mode of operation.

The ABM0 and ABM1 header is the last 32 bytes (Example $0C00FFE0_H$ and $0C01FFE0_H$ for XMC4500) of the first and second 64 KB physical sectors.

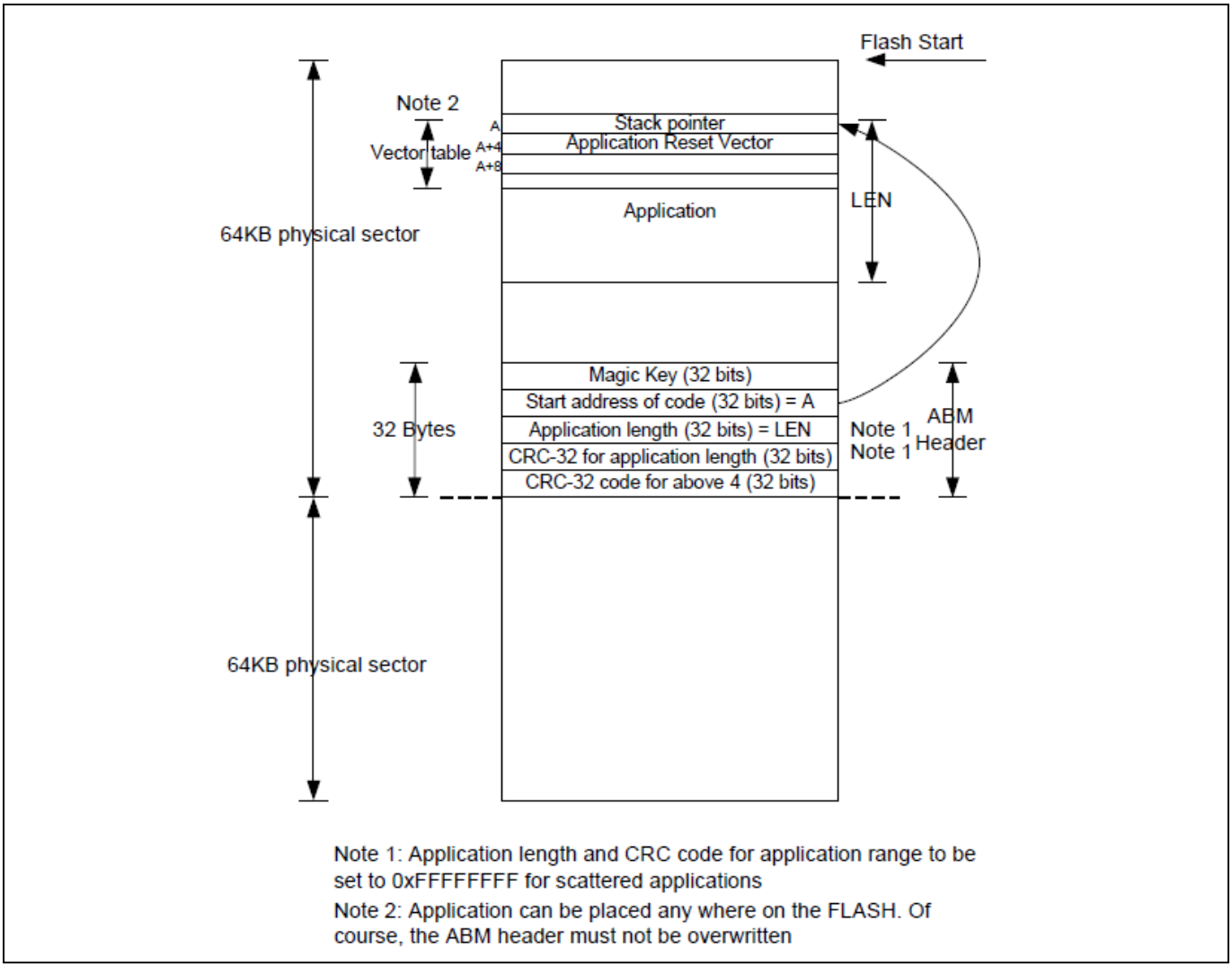


**Figure 4 ABM header**

*Note: Please note that the ABM header can be generated in an **XMC45_FlashLoader** project using the **TEST_MODE** (see flash loader test mode). Any changes in the ABM header field (eg. start address etc) requires regenerating the ABM header as the CRC value will be incorrect.*

## 2.3 Entry to Alternate Boot Mode (ABM)

While the application code is running, the software can gain entry into Alternate Boot Mode (ABM) (entry into flash loader software) by programming the **SWCON** of the startup configuration register with **boot from alternate Flash Address 1,** followed by a system reset as shown below.

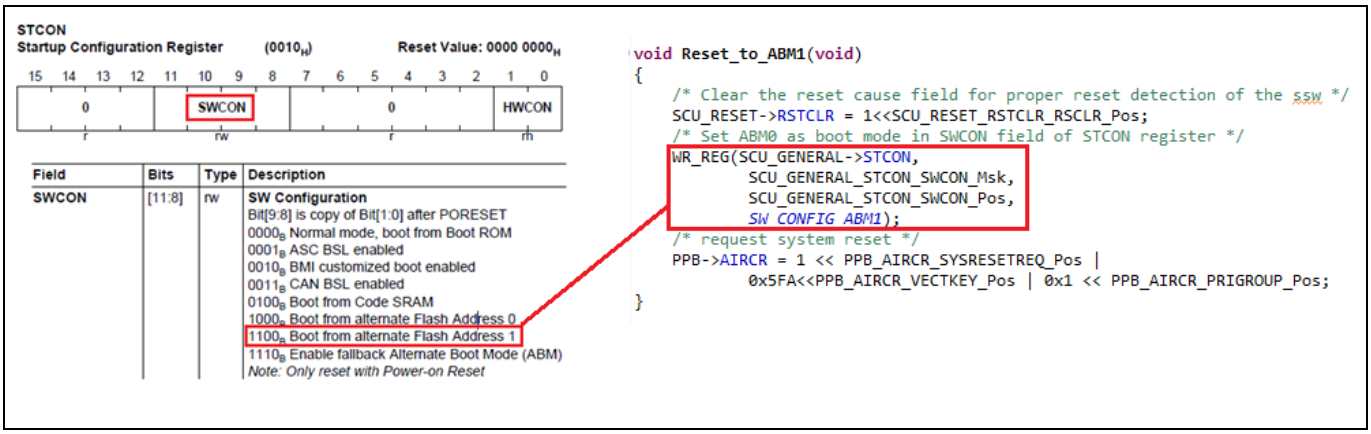


Figure 5 Entry into Alternate Boot Mode (ABM)

## 2.4 Exit from Alternate Boot Mode (ABM)

After flash programing, which is executed by the flash loader software in Alternate Boot Mode (ABM), the user can exit the Alternate Boot Mode (ABM) to run the application software by programming the bit field **SWCON** of the startup configuration register with **normal mode, boot from boot ROM,** followed by a system reset as shown below.

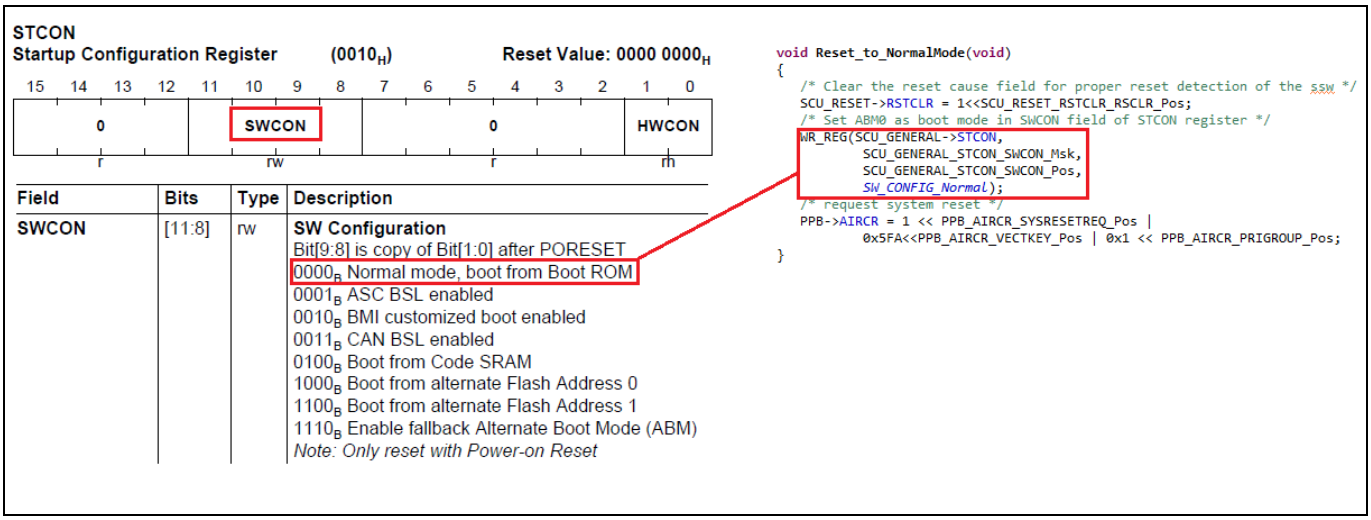


Figure 6 Exit from Alternate Boot Mode (ABM)

# 3 Project for flash loader

To begin, we shall create a project for the flash loader that will be located at address location 0x080C0000. The main objectives of this project are to read the SD Card and to program the XMC4000 flash, followed by RESETing to normal mode.

The file “main.c” provides LED2 to indicate flash update completion or an error and the “button 2” is used as a reset to normal operation mode (running the application software).

The file “flash_loader.c” is used to read the SD Card for the binary file (firmware.bin) and interface with the flash write low level driver (flash.c).

The flash loader is also equipped with the capability to make the ABM header in the **TEST_MODE** setting.

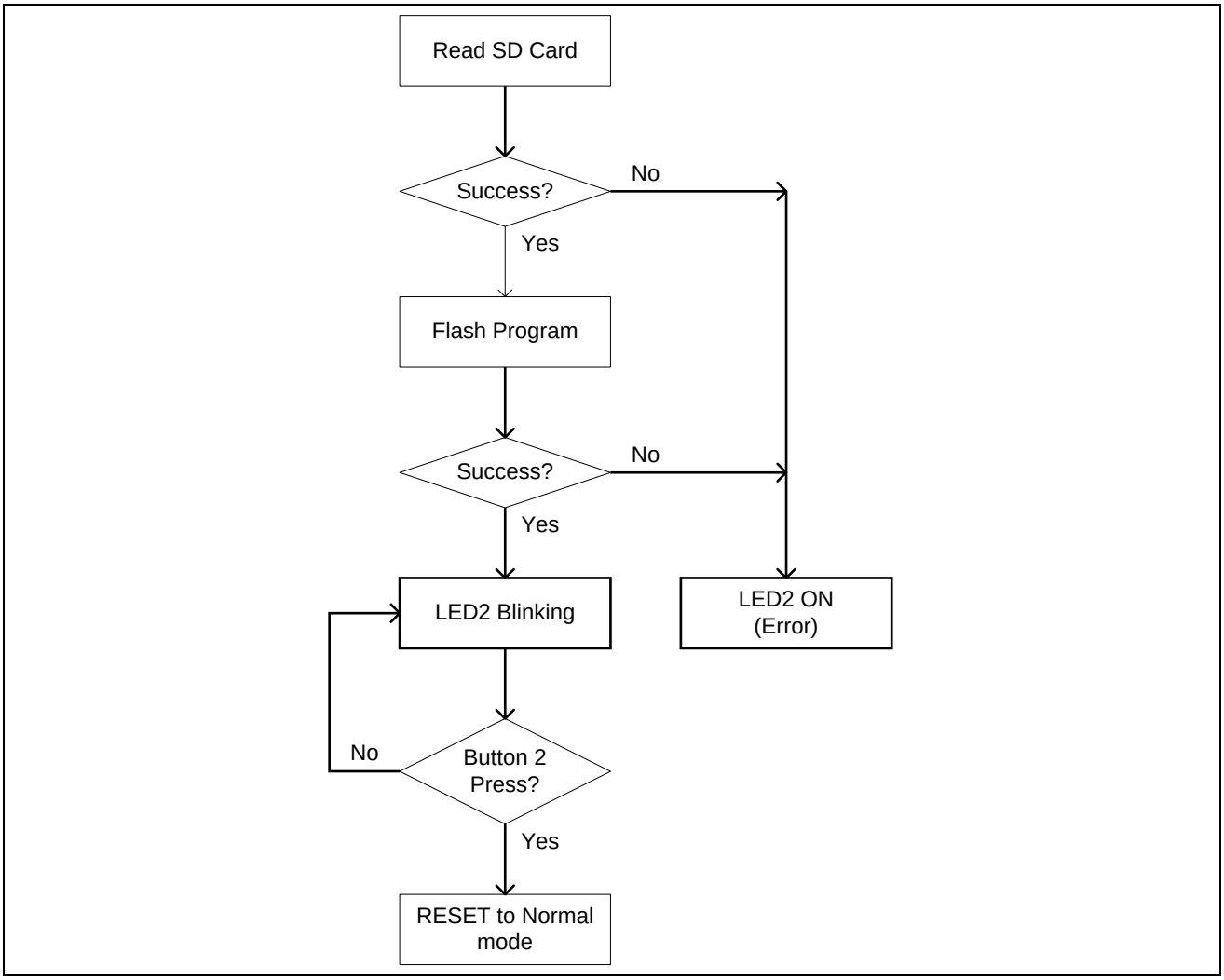


**Figure 7 XMC45_FlashLoader flow chart**

## 3.1 Creation of XMC45_FlashLoader project

Follow the procedures below to create project **XMC45_FlashLoader**

1. First create a XMC4500 DAVE™ CE project call "**XMC45_FlashLoader**".
2. From the "Add New App" window, add the "**FATFS**" app into the project as shown below.
3. At the "App dependency" window, click on the **FATFS** app.

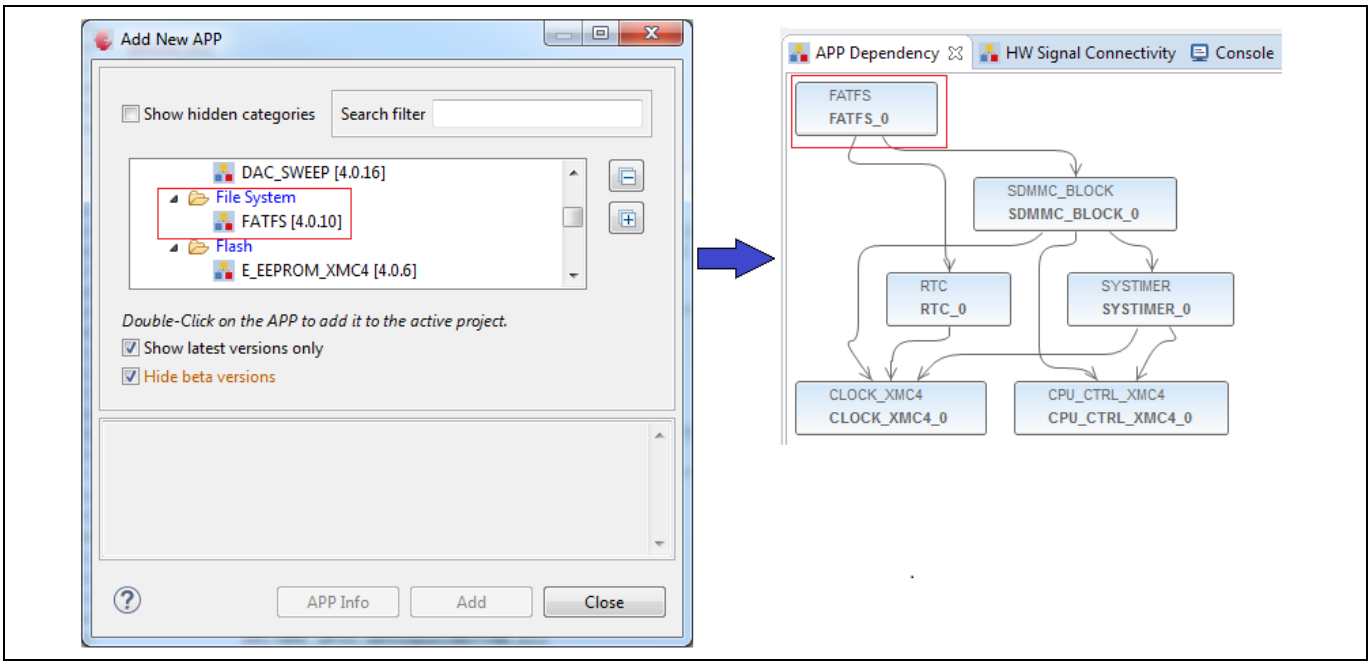


**Figure 8 Add FATFS apps into XMC45_FlashLoader project**

4. Configure the FATFS drive configuration using **SDMMC (SD mode)**
5. Finally, click on the code generation icon  .

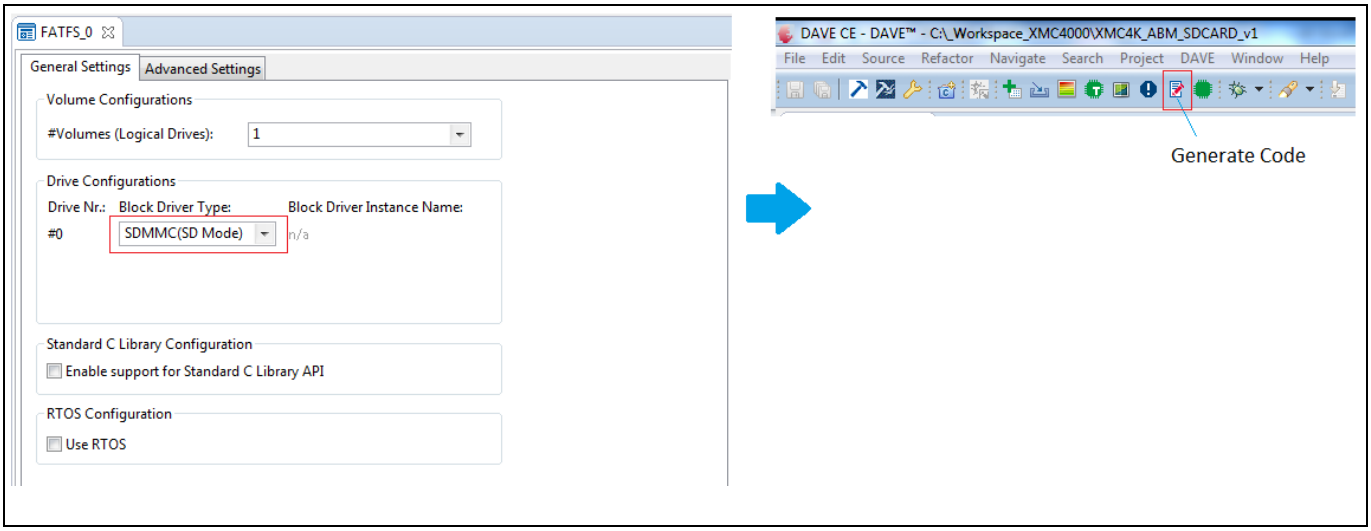


**Figure 9 FATFS configuration and code generation**

6. Copy all the files from folder **"..\DAVE4 Project Files\XMC45_FlashLoader"** into project **XMC45_FlashLoader** (Overwriting files main.c and linker_script.id).

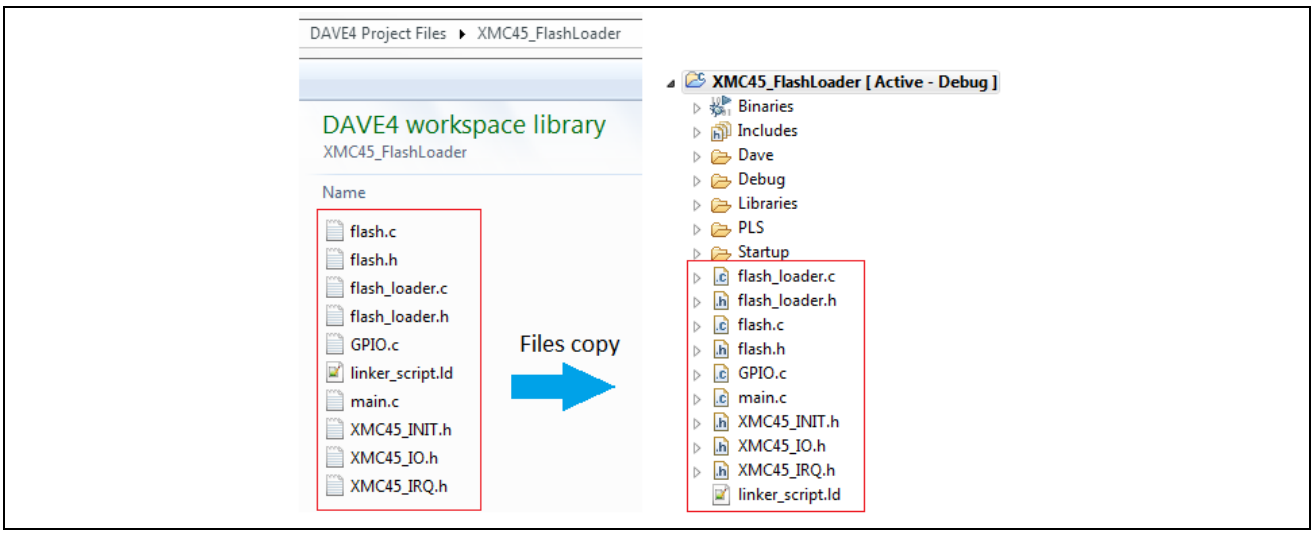


**Figure 10 Copy source files into project folder**

7. The copied **linker_script.id** file has been modified to map the flash loader software to flash memory location 0x080C0000. (see: flash loader script file handling).

8. To generate a binary file "**XMC45_FlashLoader.bin**" from this project, **right** click on the "**XMC45_FlashLoader**" project and select "**Properties**".

9. Select **C/C++ Build >> Settings >> ARM-GCC-Create Flash image**

10. Replace "**${OUTPUT_PREFIX}${OUTPUT}** " with "**XMC45_FlashLoader.bin**" in the command line as shown below and click on **Apply** button.

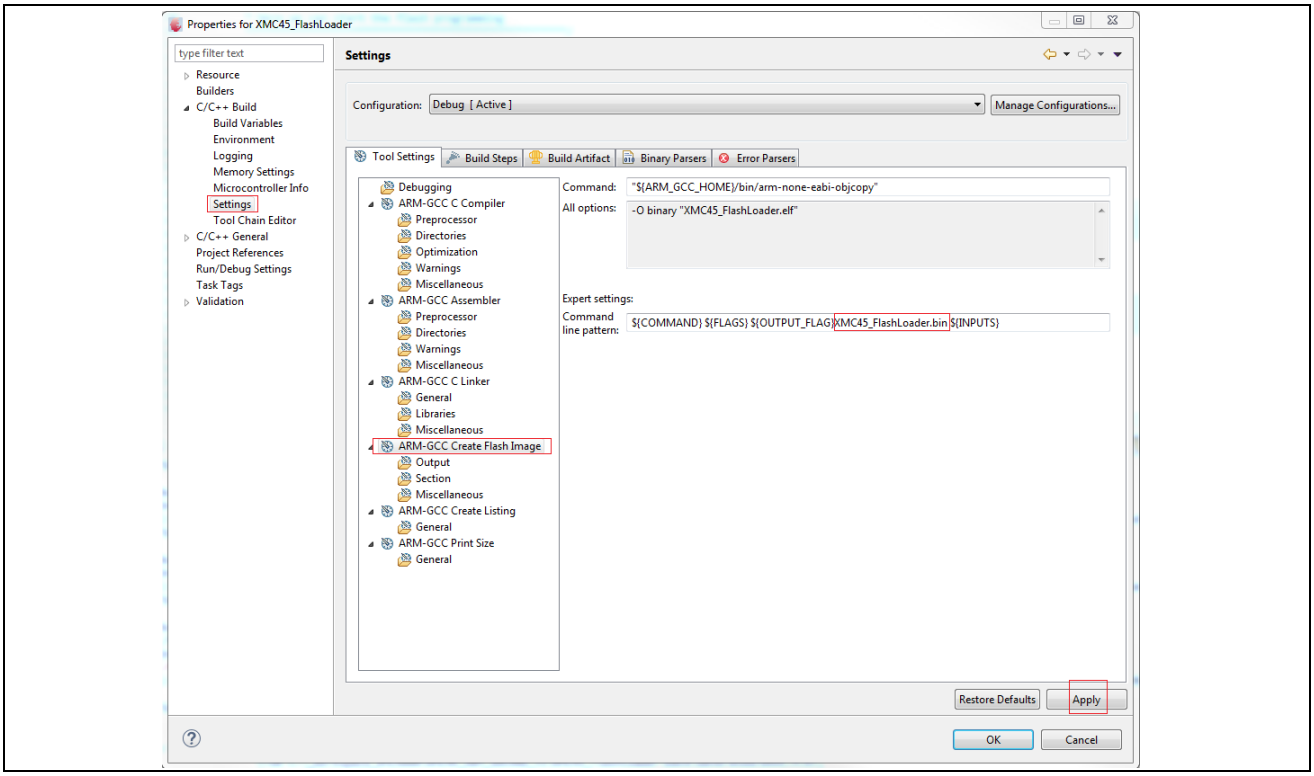


**Figure 11 Set bin file name as XMC45_Loader.bin**

11. Select “**Output**” and for output file format select **binary**.
12. Click on “**Apply**” and “**OK**” button to close.

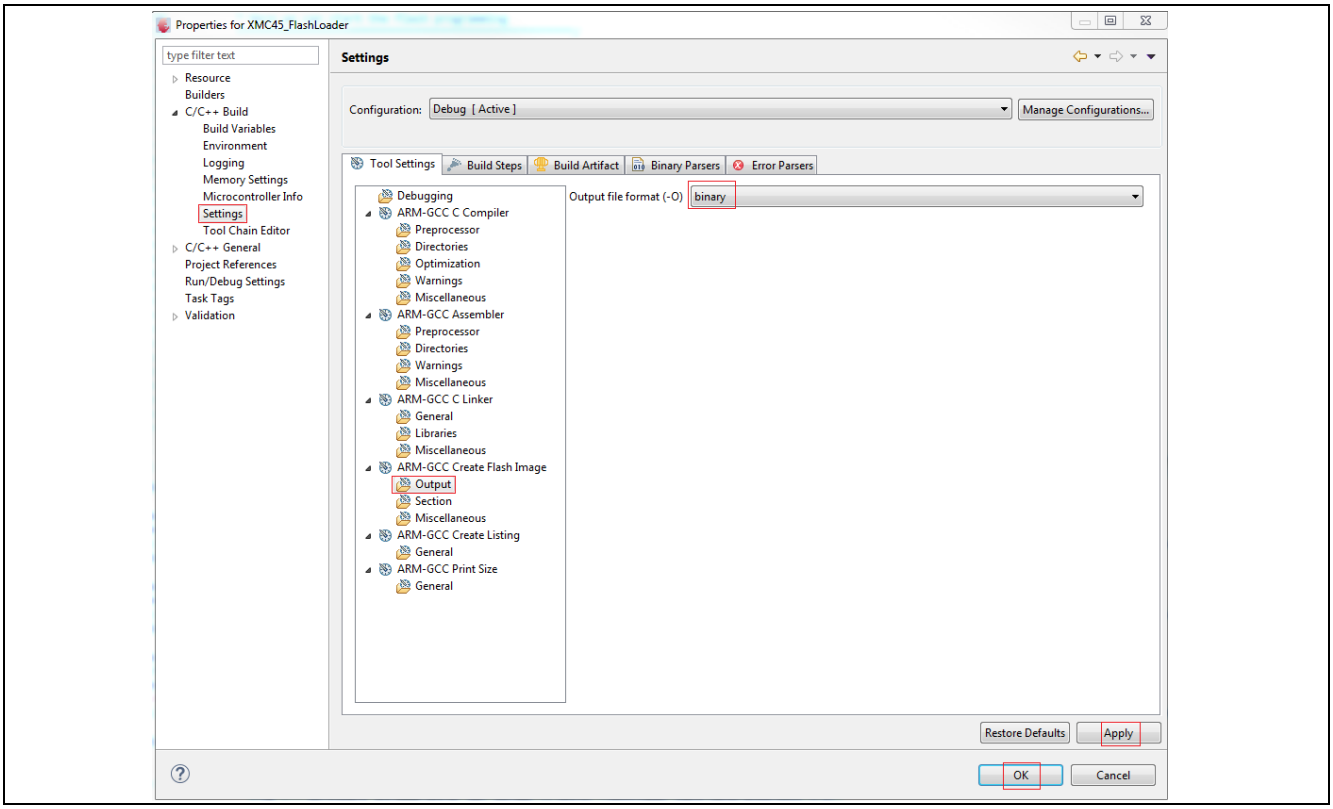


**Figure 12 Set output as binary**

13. Click on the compile button  to compile the software
14. Ensure that **XMC45_FlashLoader.bin** file is generated in the **Debug** folder.

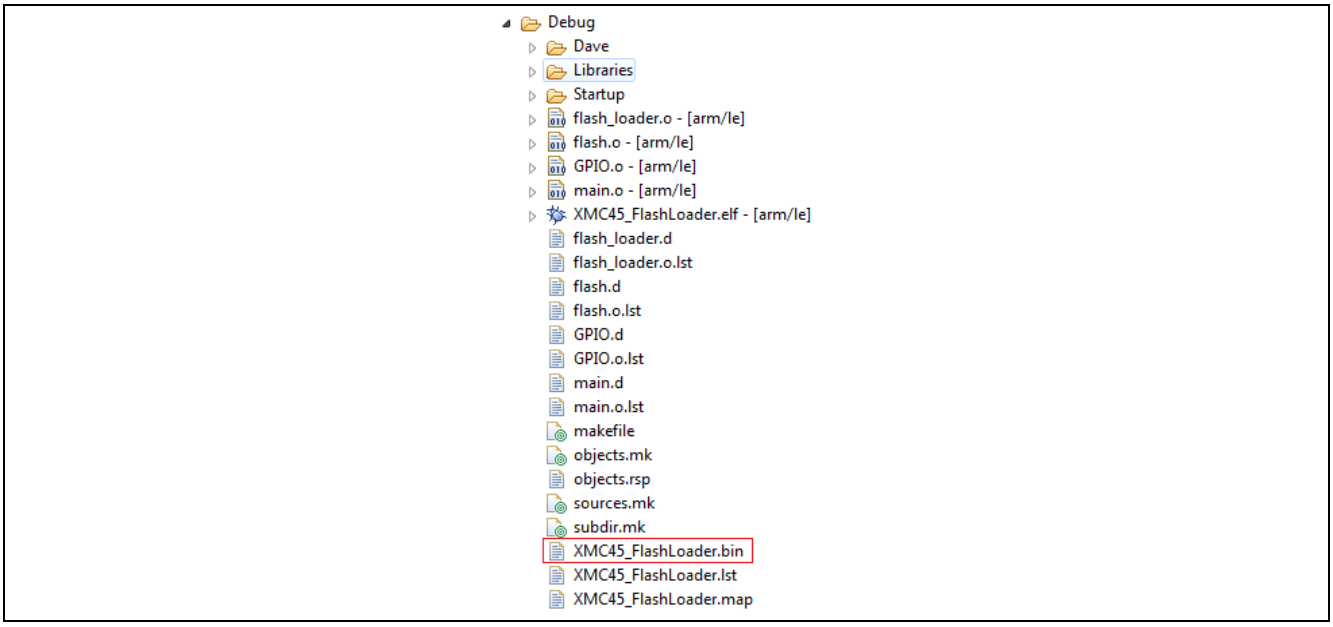


**Figure 13 Generated bin file**

## 3.2 Convert flash loader binary file to C source file

In order to merge the flash loader software into the production firmware as a single project, we need to transform the binary file to a C source file. This process can be performed with the "**Hex workshop**" software.

1. Activate the "**Hex workshop**" software
2. Click **File** >> **Open**
3. Browse to project debug folder "**../XMC45_FlashLoader/Debug**"
4. Select binary file "**XMC45_FLashLoader.bin**"
5. Click "**Open**" to complete the process

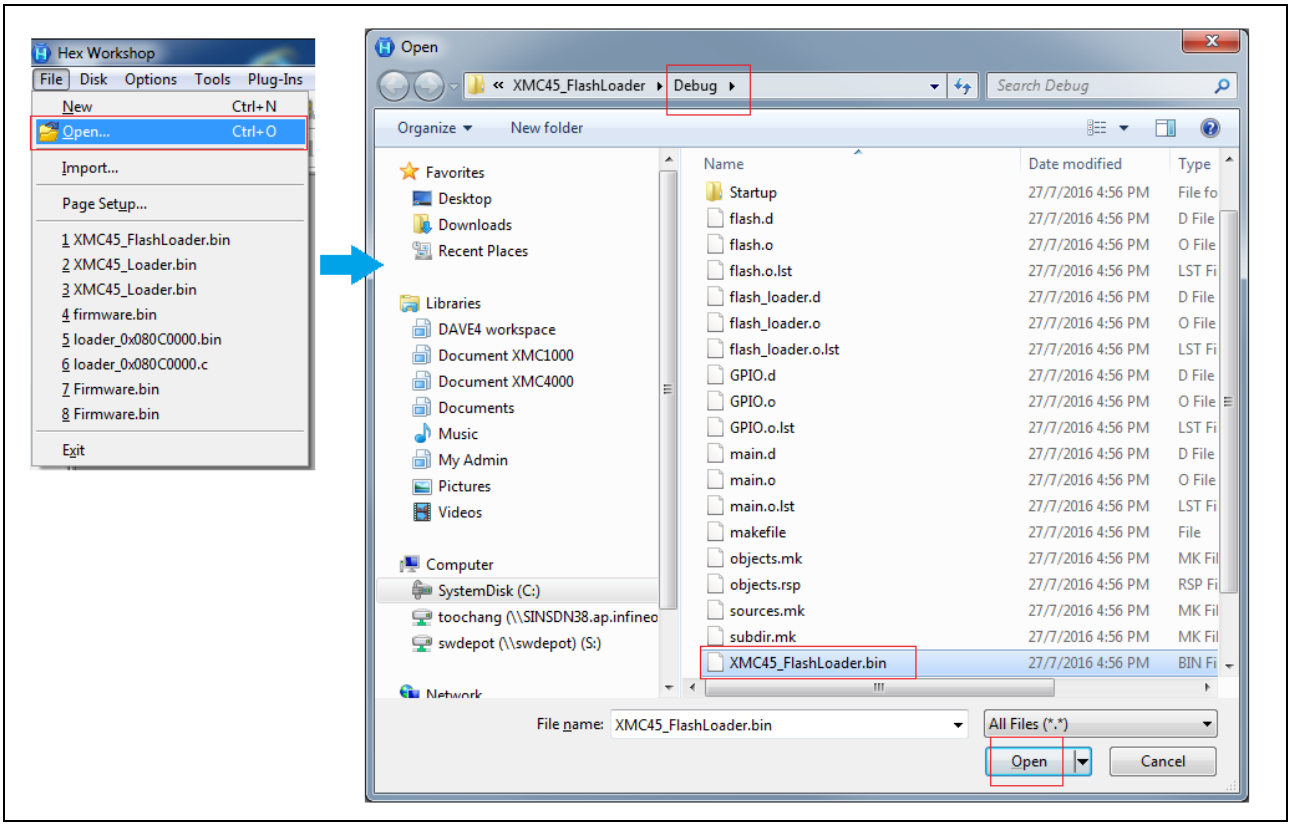


**Figure 14 Open binary file**

6. To export the binary file to a C source file click on "**File**" >> "**Export**".
7. Choose"**Desktop**" as a temporary saving location.
8. Provide a name for the C source file as "**XMC45_FlashLoader.c**".
9. Click "**Save**" to complete the process.

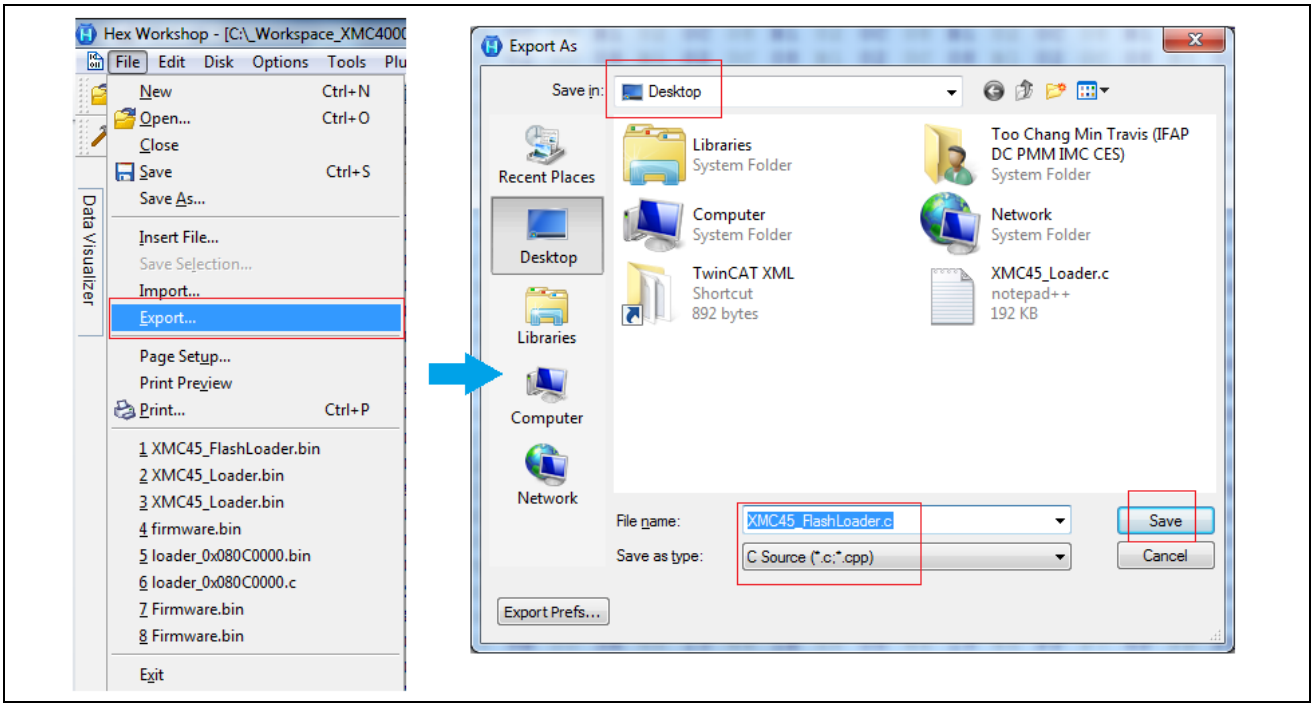


Figure 15 Export binary file as C source file to desktop

10. Close the **HEX Workshop** software after exporting.

11. Preview the generated "**XMC45_FlashLoader.c**" that can be found on the desktop. This file will be reused in the **XMC45_Production_Firmware** project.

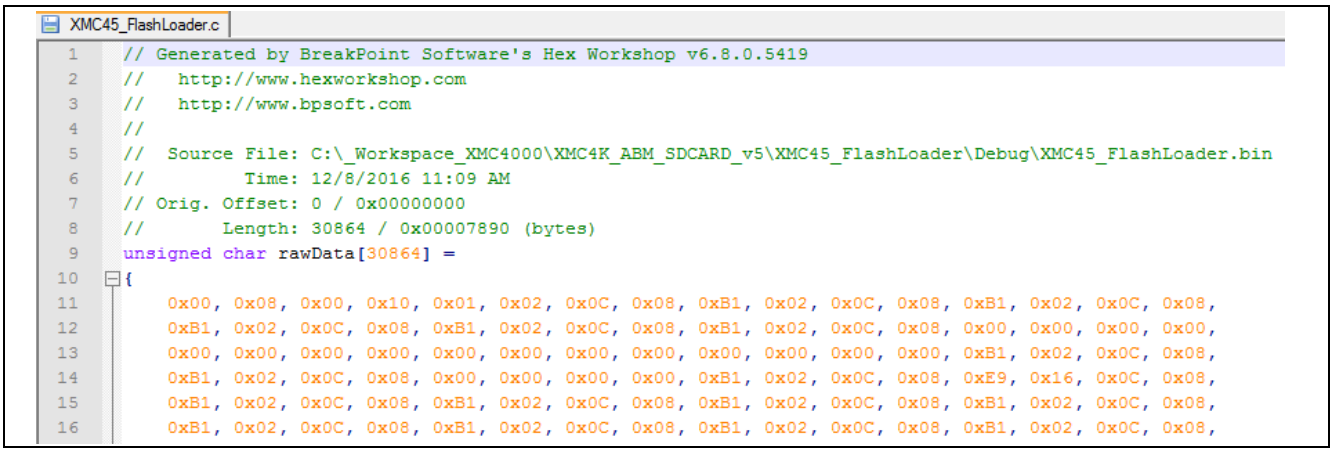


Figure 16 Preview of XMC45_FlashLoader.c

12. The next step is to create the project **XMC45_Production_Firmware.**

(see creation of XMC45_Production_Firmware)

## 3.3 Details on flash loader script file handling

As the flash loader needs to be mapped to flash memory location 0x080C0000, we need to ensure that the origin of the flash is as below

- FLASH_1 cached ORIGIN : 0x080C0000
- FLASH_1 uncached ORIGIN : 0x0C0C0000

*Note: A copy of the script file for these purposes can be found in the folder "**..\DAVE4 linker_script\XMC45_FlashLoader linker_script for PRODUCTION**". This file can be copied and used to overwrite the existing linker script file.*

linker_script.ld

```
 *      - Product splitting
 *      - Copyright notice update
 *
 * 2015-11-24:
 *      - Compatibility with GCC 4.9 2015q2
 *
 * 2016-03-08:
 *      - Fix size of BSS and DATA sections to be multiple of 4
 *      - Add assertion to check that region DSRAM_1_system does not overflowed no_init section
 *
 * @endcond
 *
 */

OUTPUT_FORMAT("elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(Reset_Handler)

MEMORY
{
    FLASH_1_cached(RX) : ORIGIN = 0x080C0000, LENGTH = 0x100000
    FLASH_1_uncached(RX) : ORIGIN = 0x0C0C0000, LENGTH = 0x100000
    PSRAM_1(!RX) : ORIGIN = 0x10000000, LENGTH = 0x10000
    DSRAM_1_system(!RX) : ORIGIN = 0x20000000, LENGTH = 0x10000
    DSRAM_2_comm(!RX) : ORIGIN = 0x30000000, LENGTH = 0x8000
}
```

**Figure 17 Edit script file for flash loader**

However for debugging purposes we can map the flash loader to the beginning of the flash (0x08000000) as some debuggers are not capable of debugging at location 0x080C0000.

(see flash loader TEST MODE)

# 4 Project for production firmware

The production firmware is a baseline firmware which provides a convenient single software package that includes both the flash loader and the user application software.

This has the benefit of removing the extra process of flash programming the Flash loader software followed by the application software.

While the application software is running, it needs a triggering mechanism to invoke the firmware update. For this example we will make use of Button 1. So, when Button 1 is pressed the application software will enter firmware update mode and eventually reset to Alternate Boot Mode. Hence, after the RESET, the application software will execute the XMC45_FlashLoader.

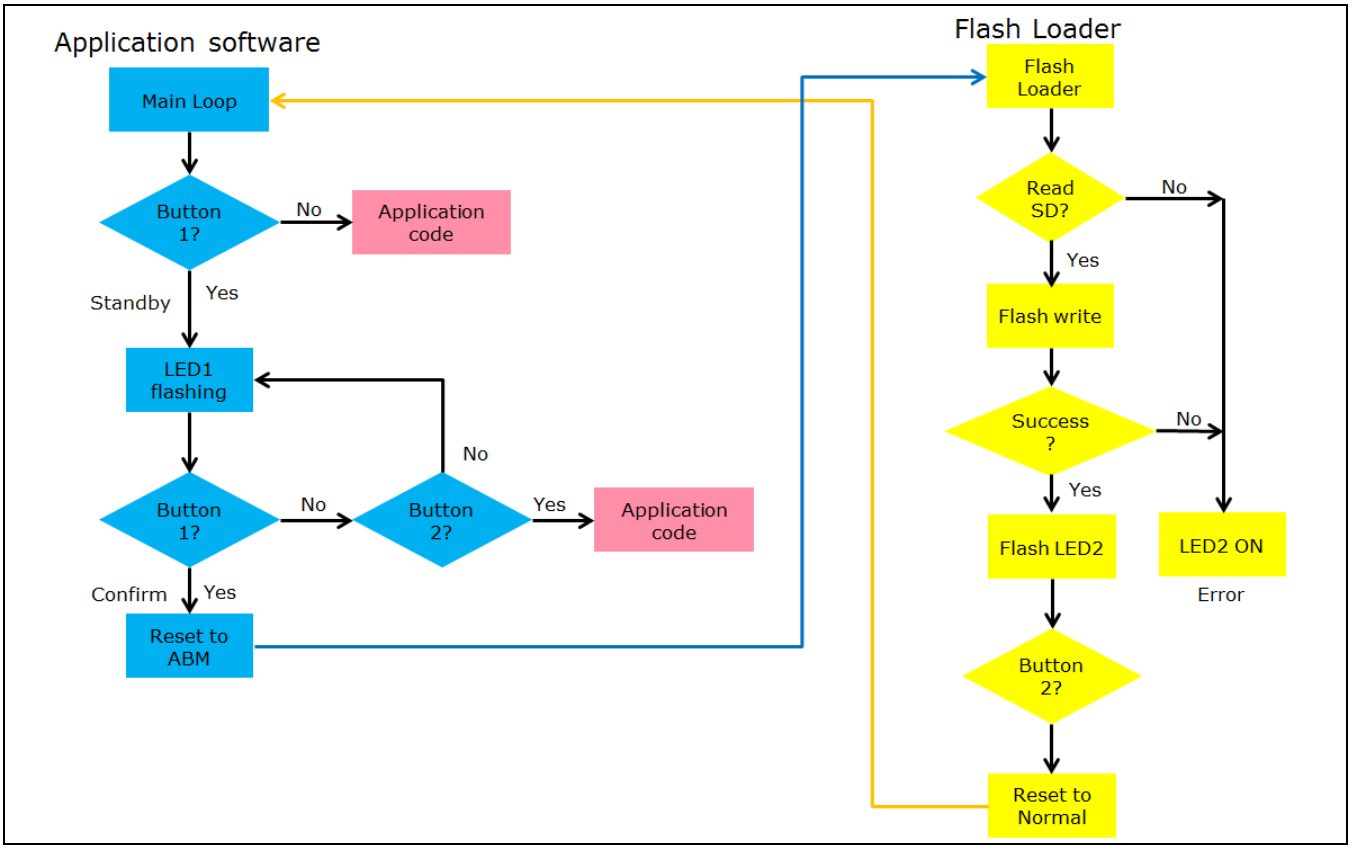


Figure 18 Production firmware flow chart

## 4.1 Creation of XMC45_Production_Firmware project

Follow the procedures below to create the project **XMC45_Production_Firmware**

1. Create a DAVE™ CE XMC4500 project called "**XMC45_Production_Firmware**".
2. Copy all the source files from the folder..**\DAVE4 Project Files\XMC45_Production_Firmware** into the project "**XMC45_Production_Firmware**" (Overwriting files main.c and linker_script.id).

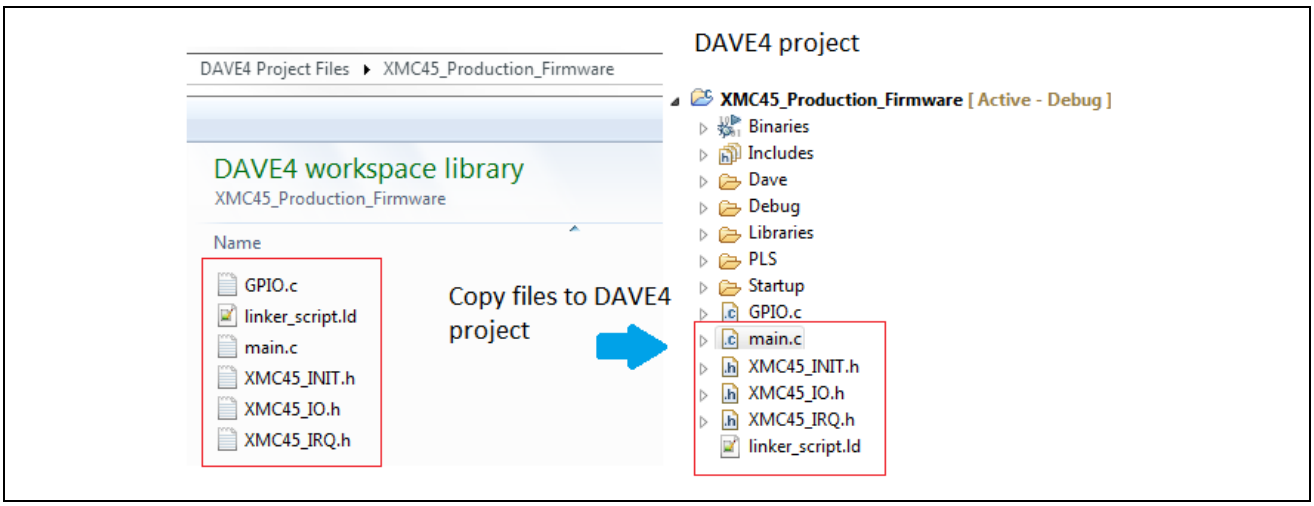


**Figure 19 Copy source file to XMC45_Production_Firmware project**

3. Copy the previously generated file "**XMC45_FlashLoader.c**" from the desktop into the project folder.

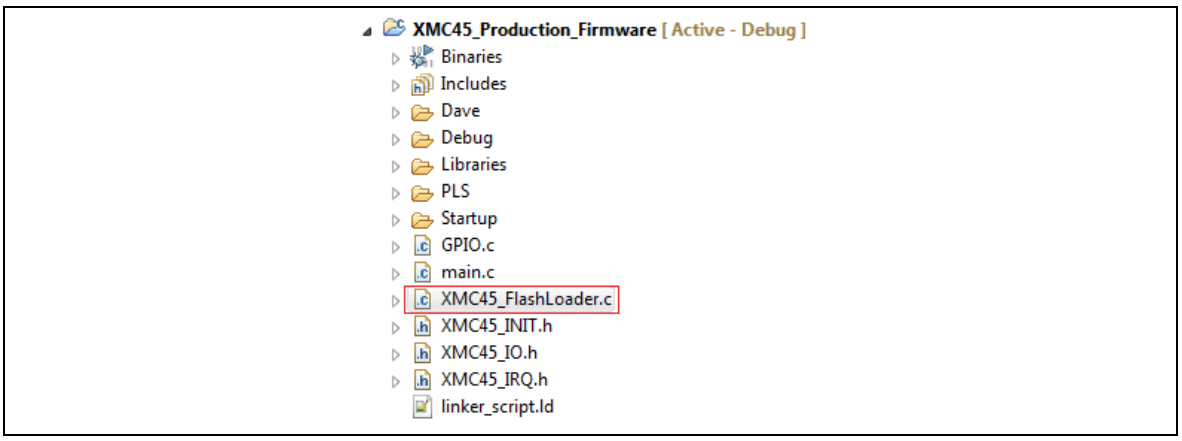


**Figure 20 Copy XMC45_FlashLoader.c to XMC45_Production_Firmware project**

4. Open and edit the file "**XMC45_FlashLoader.c**" with the highlighted code.

```
const unsigned char __attribute__((section(".abm_rawData"))) rawData[30864] =
```

5. Save the project by clicking on the save button 
6. The ABM header has been hard coded into the file **main.c**. (see ABM header Installation)
7. The script file has been modified to map the Flash Loader software and ABM header to the required flash memory address. (see production firmware script file handling)
8. Click the compile button  to compile the software

9. After compilation, click the debug button  to download the software.
10. Exit the debugger mode as flash programming is not recommended in debugger mode.
11. Here you should see both **LED1** and **LED2** turn **ON**. (Without flashing)
12. At this point you should be ready to test this project. (see how to test?)

## 4.2 Details on ABM header Installation

The ABM header is required to be installed in the projects **XMC45_Production_Firmware** and **XMC45_Update_Firmware_A/B.**

However, should there be any changes to the ABM header parameters (eg. Start address etc) it is necessary to recalculate the HeaderCRC32. This can be done in the **XMC45_FlashLoader** project in TEST_MODE (see ABM header generation). The ABM header needs to be installed at location **0x0801FFE0** using the script file (see figure 21).

```
/*******************************************************************************************
 * Notes:
 *
 * To make a ABM header
 * - Go to Project XMC45_FlashLoader and enable Test_Mode
 * - ABM1 Header is located at 0x0801FFE0
 * http://mcuoneclipse.com/2012/11/01/defining-variables-at-absolute-addresses-with-gcc/
 *******************************************************************************************/
static const ABM_Header_t  __attribute__((section(".flash_abm")))       // Update_ABM: Check the script file
ABM1_Header = {
        .MagicKey           = MAGIC_KEY,
        .StartAddress       = 0x080C0000,                               // Update_ABM: Take care of the startup address
        .Length             = 0xFFFFFFFF,
        .ApplicationCRC32   = 0xFFFFFFFF,
        .HeaderCRC32        =  0x2008751C                               // Update_ABM: Take care of the CRC value
};
```

**Figure 21 ABM header**

## 4.3 Details on production firmware script file handling

The script file of the project **XMC45_Production_Firmware** needs to be modified as shown below, to map the ABM header and the XMC45_FlashLoader to the follow flash location.

- ABM header mapping location 0x0801FFE0
- XMC45_FlashLoader mapping location 0x080C0000

```
    /* Exception handling, exidx needs a dedicated section */
    .ARM.extab :
    {
        *(.ARM.extab* .gnu.linkonce.armextab.*)
    } > FLASH_1_cached AT > FLASH_1_uncached

    __exidx_start = .;
    .ARM.exidx :
    {
        *(.ARM.exidx* .gnu.linkonce.armexidx.*)
    } > FLASH_1_cached AT > FLASH_1_uncached
    __exidx_end = .;


/* Update_ABM: Take care of the ABM1 address (0x0801FFE0) */
/* http://mcuoneclipse.com/2012/11/01/defining-variables-at-absolute-addresses-with-gcc/ */
    .abm ABSOLUTE(0x0801FFE0): AT(0x0801FFE0 | 0x04000000)
    {
       KEEP(*(.flash_abm))
    } > FLASH_1_cached

/* Update_ABM: Take care of the loader startup address */
    .abm_rawData (0x080C0000):
    {
       KEEP(*(.abm_rawData))
    } > FLASH_1_cached
```

For ABM header

For XMC45_FlashLoader

**Figure 22 Edited linker script file**

*Note: A copy of the script file for these purposes can be found i then folder **"..\DAVE4 linker_script\XMC45_Production_Firmware**". This file can be copied and used to overwrite the existing linker script file.*

# 5 Project for XMC45_Update_Firmware project

The objective of this project is to create a software project for firmware updating purposes which will eventually generate a **firmware.bin** file that can be copied into the SD Card for flash programming.

## 5.1 Creation of XMC45_Update_Firmware

Follow the procedures below to create project **XMC45_Update_Firmware_A**

1. Create a DAVE™ CE XMC4500 project called "**XMC45_Update_Firmware_A**".
2. Copy all of the files from folder..**\DAVE4 Project Files\ XMC45_Update_Firmware_A** into the "**XMC45_Updata_Firmware_A**" project.

*Note: To prevent the application software from overwriting the ABM header, the ABM header is included again in the XMC45_Update_Firmware project.*

3. Next, to generate a binary file "**firmware.bin**" from this project, **right** click on the "**XMC45_Update_Firmware_A**" project and select "**Properties**".
4. Select **C/C++ Build >> Settings >> ARM-GCC-Create Flash Image**
5. Replace "**${OUTPUT_PREFIX}${OUTPUT}** " with "**firmware.bin**" in the command line as shown below and click the **Apply** button

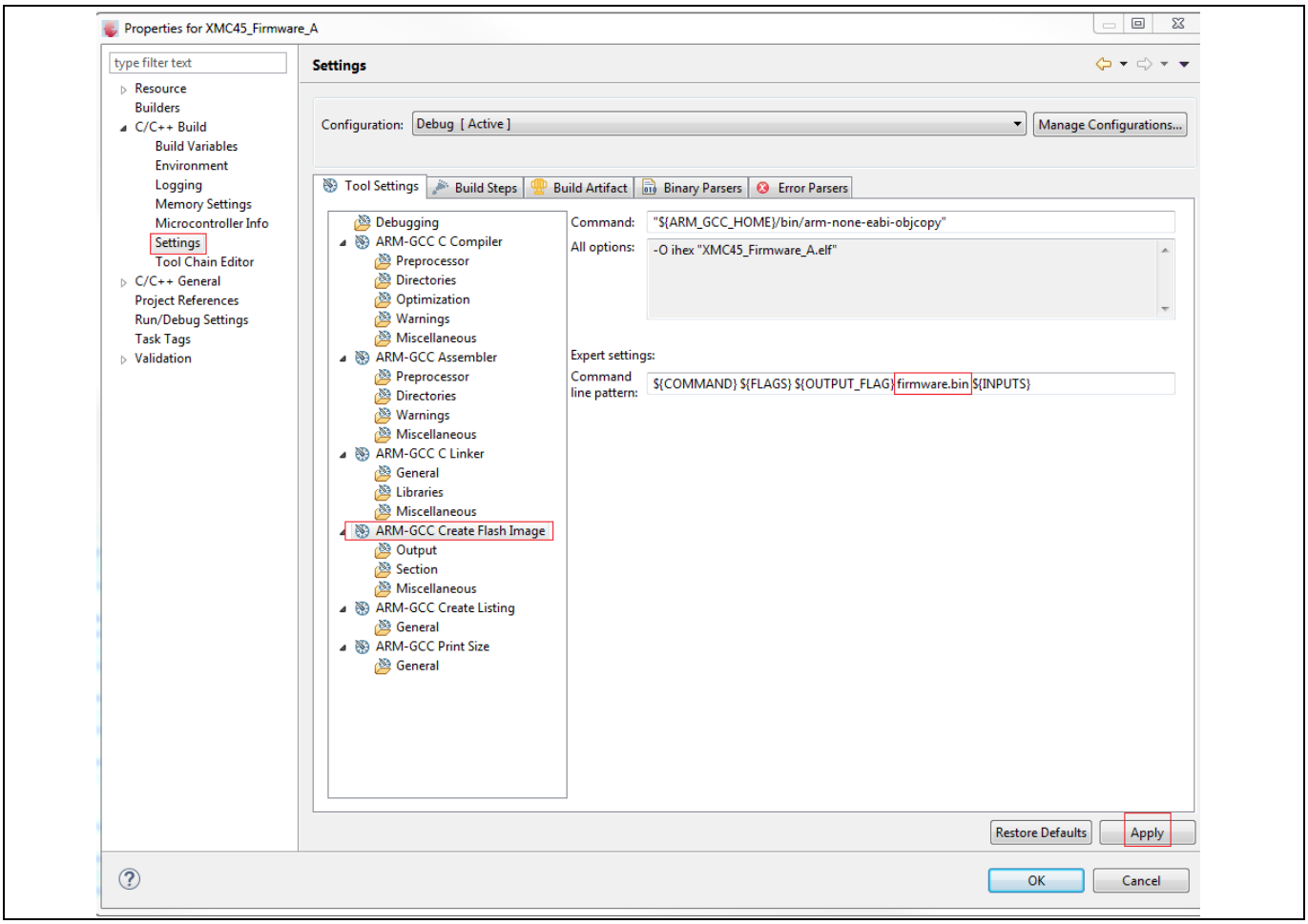


**Figure 23 Setting to generate "firmware.bin"**

6. Then, select **Output** and for output file format, select **binary**.
7. Click on **Apply** and then **OK**

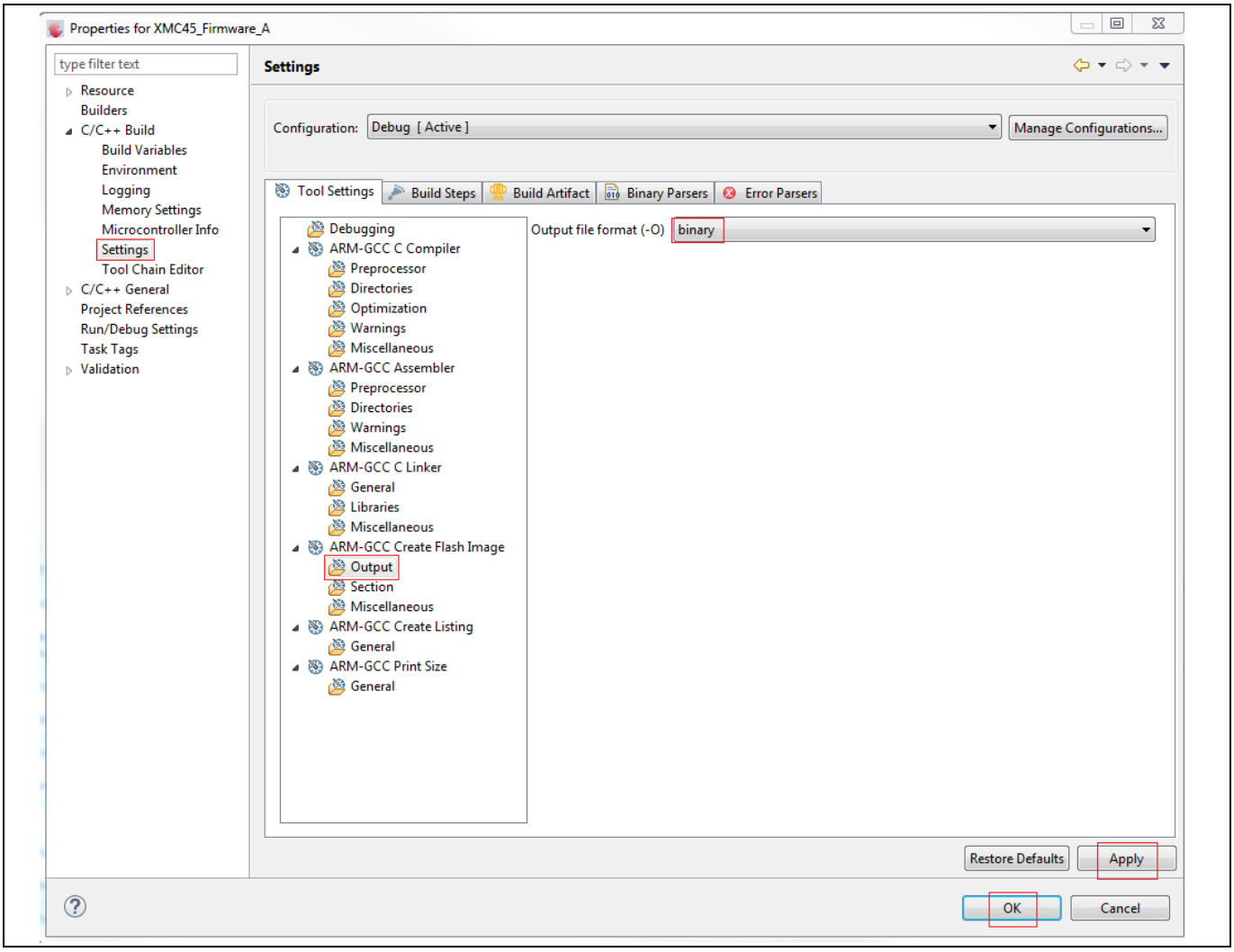


**Figure 24 Setting to generate “firmware.bin”**

8. Click on the compile button  to compile the software
9. Ensure that the **firmware.bin** file is generated.

# 6 Flash loader test mode

The Flash loader test mode allows the user to generate a preview of the Alternate Boot Mode(ABM) header and also to perform general system testing.

As some debuggers do not allow the Flash loader software to be executed in flash memory location 0x080C0000, the script file has to be changed to map the Flash loader software to flash memory location 0x08000000 (default flash starting address).

### Difference in linker_script files

There are 2 linker script files provided for the Flash loader for PRODUCTION and TEST MODE purposes.

The PRODUCTION linker script file is used to map the flash loader software to flash location 0x080C0000 (see details on Flash loader script file handling) and the TEST MODE linker script file is bascially the default linker script file generated by DAVE™.

For convenience, to switch between TEST MODE and PRODUCTION mode, both the linker script files are replicated in the following folders.

- ..\DAVE4 linker_script\XMC45_FlashLoader linker_script for PRODUCTION
- ..\DAVE4 linker_script\XMC45_FlashLoader linker_script for TEST MODE

Hence, by replacing the linker script file of the XMC45_FlashLoader project with the one from the folder above, we can use the Flash loader software for Production or TEST MODE.

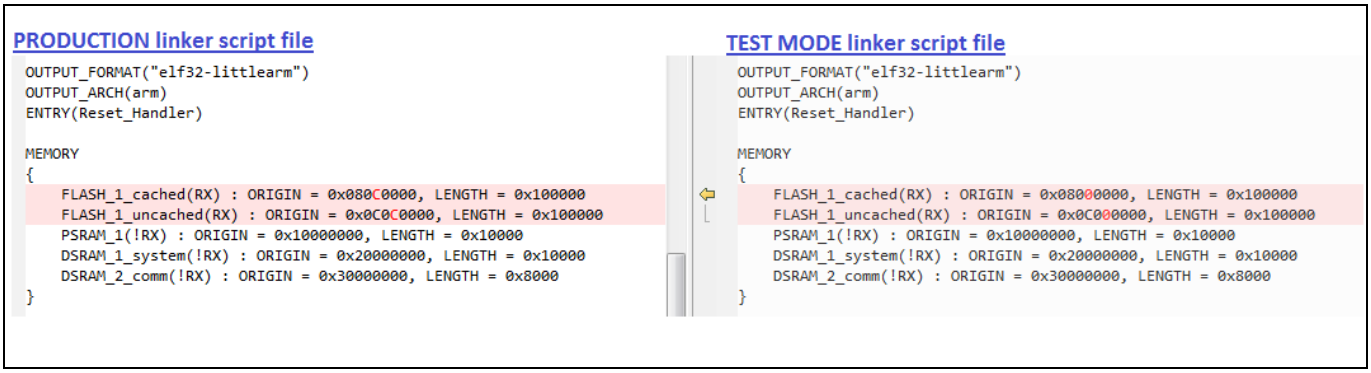


**Figure 25 Differences between linker script file of Production and TEST MODE.**

*Note: Please note that the linker script file might be different depending on the DAVE™4 version. Therefore, it is important to know what needs to be modified for the linker script file.*

## 6.1 Setup for test mode

The test mode can be enabled by the following procedure.

1. Copy the linker script file from the folder "**DAVE4 linker_script\XMC45_FlashLoader linker_script for TEST MODE**".
2. Replace the linker script file in the project folder "**XMC45_FlashLoader**" with the copied script file; this is to map the Flash loader software to flash memory location 0x08000000 for testing purposes.

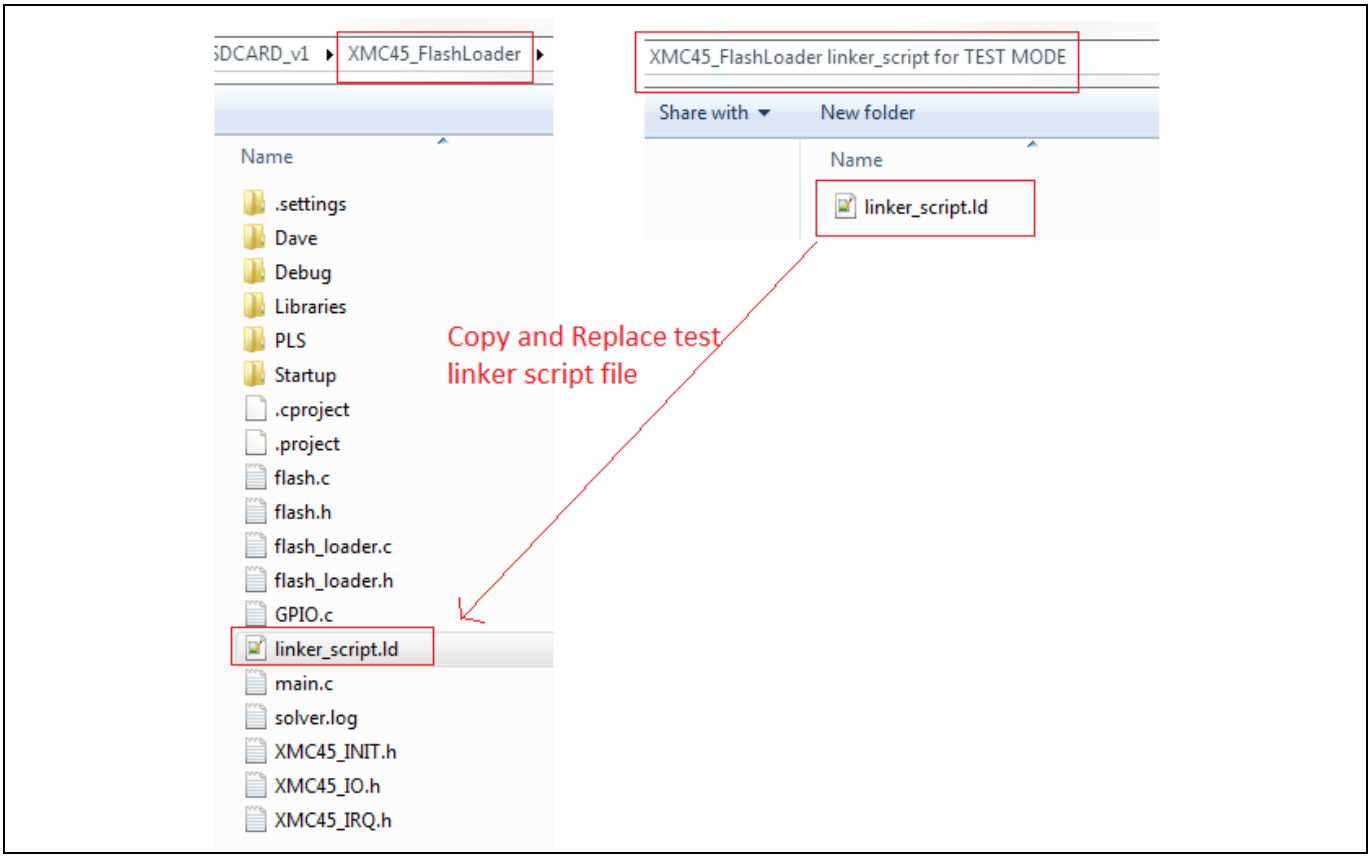


**Figure 26 Copy and replace test linker script**

3. Enable test mode by "**#define TEST_MODE 1**" in file "**XMC45_INIT.h**".
4. Copy any **firmware.bin** file(s) to the SD card
5. Insert the SD card into the SD slot of the XMC4500 relaxkit
6. Compile and Run the software
7. Monitor the variable **PSRAM_Header** for ABM header generation.

   (see ABM header generation)
8. Monitor the variable **SYS_TEST** for system testing status.

   (see system test)

## 6.2 ABM header generation

Ensure that the function "**ABM_Make_ABM_header()**" had been executed, then monitor the generated values of the variable structure "**PSRAM_HEADER**".

Perform a screen shot of this ABM header, as these values will be reused and hard coded into both the "**XMC45_Production_Firmware**" and "**XMC45_Firmware_A/B**" projects.

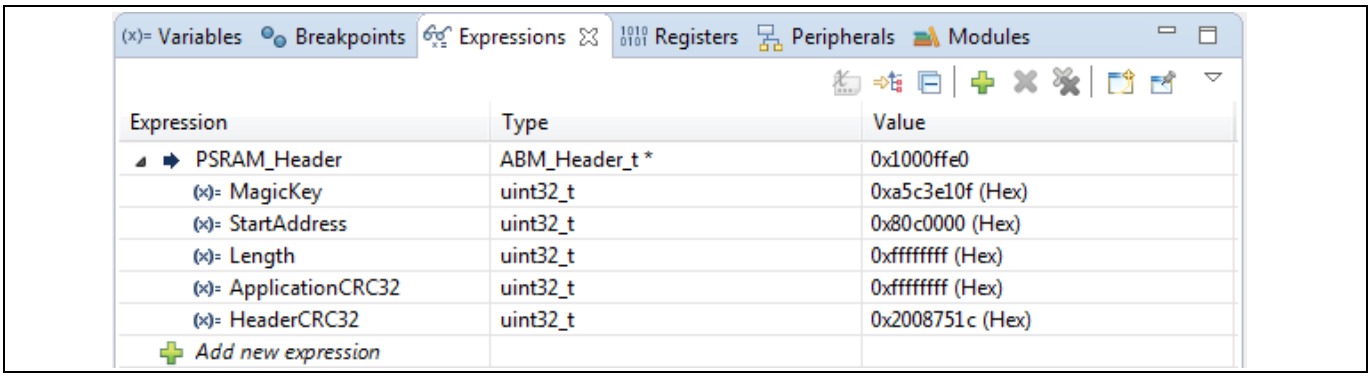


**Figure 27 ABM header generation in TEST_MODE**

## 6.3 System test

The system testing provides general testing of the software flow sequence and also reading the SD card for a binary file "**firmware.bin**".

*Note:Please note that flash write is disabled during TEST_MODE*

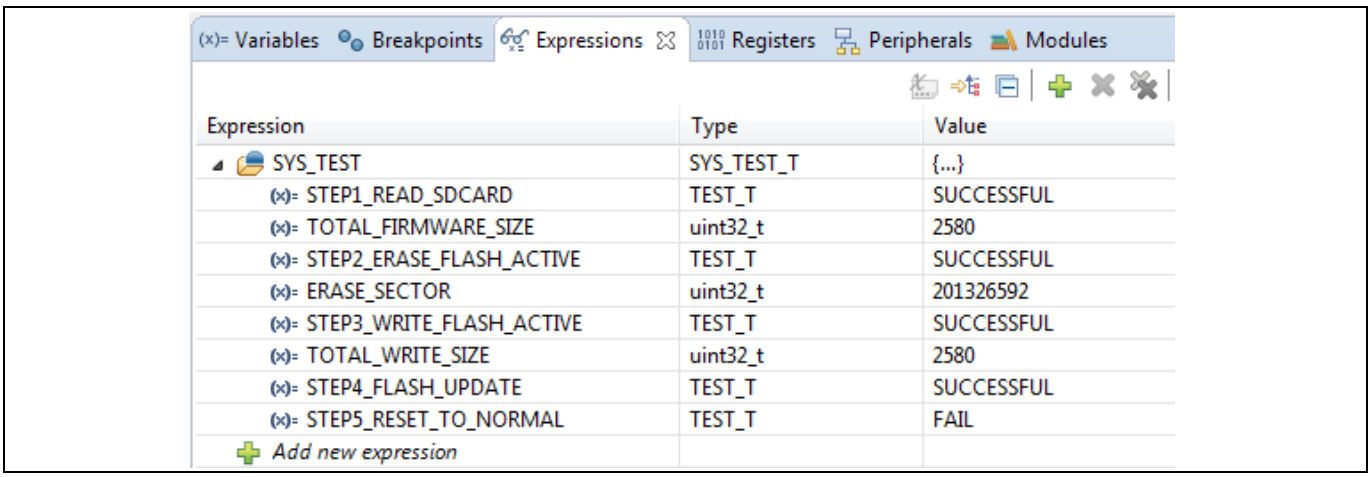


**Figure 28 System self test in TEST_MODE**

## 6.4 Exit TEST_MODE

After completion of the ABM header generation and system testing, the user can exit the TEST_MODE by following this procedure.

1. Copy the linker script file from the folder "**DAVE4 linker_script\XMC45_FlashLoader linker_script for PRODUCTION**".
2. Disable test mode by commenting "**// #define TEST_MODE 1**" in file "**XMC45_INIT.h**".
3. Compile the project.

# 7 How to test?

This chapter describes how to perform a firmware update via SD card after completing the both Flash loader and production firmware projects.

Follow the procedure below to perform a firmware update via SD card.

1. Copy "**firmware.bin**" from the folder**..\Test firmware_bin Files\firmware A\B** to the SD card and insert the SD card into SD slot of the XMC4500 relaxkit.

   (See Creation of XMC45_Update_Firmware)

2. Press the **RESET** button.
3. Press **Button1** to enter flash update mode.

   Observe: **LED1** will start flashing.

4. Press **Button1** again to begin flash update (or **Button2** to exit flash update)

   Observe: **LED2** will turn on which indicates Flash programming is in progress

   Observe: **LED2** will then start flashing which indicates that flash programming has been successful.

5. Press **Button2** to execute the latest firmware

*Note:If LED2 is always ON, this indicates an error such as no SD card or flash programming has not been successful. In this case, press the RESET button to recover.*

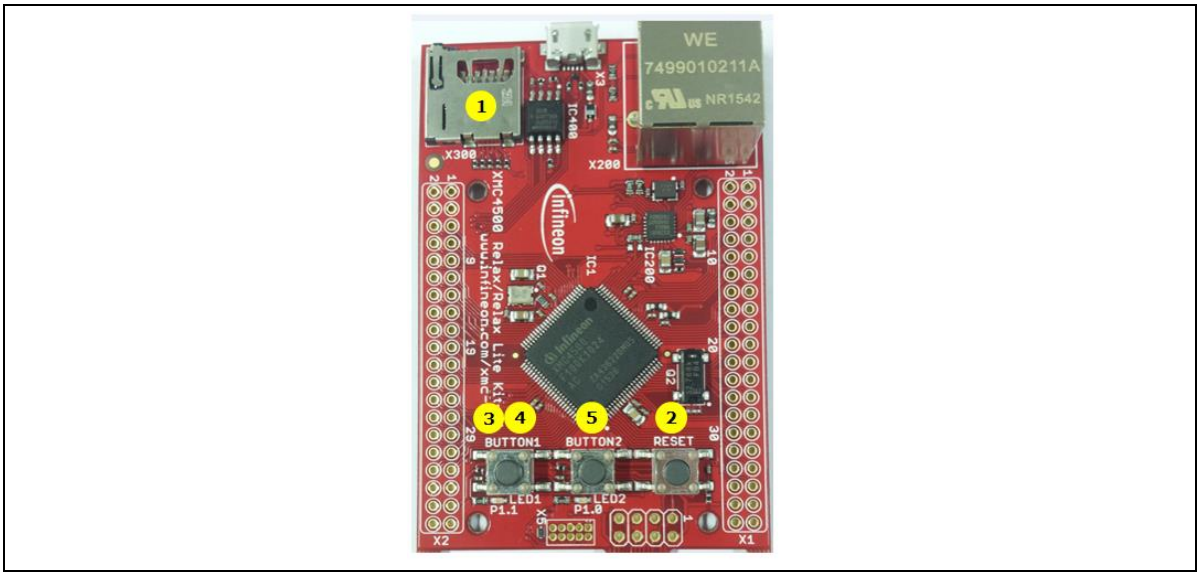


**Figure 29 Firmware update via SD card**

[1] A Reference. See the code examples at **www.infineon.com**

# Revision history

## Major changes since the last revision

| Page or reference | Description of change |
|---|---|
| | |
| | |
| | |

**Edition 10.09.2016**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about this document?**

**Email:** erratum@infineon.com

**Document reference**
**AN_201609_PL30_027**

**IMPORTANT NOTICE**

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

**WARNINGS**

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.