AN11285

Writing a flash programming algorithm for unsupported devices

Rev. 1 — 6 November 2012 | Application note

Document information

Info	Content
Keywords	LPC43xx, LPC18xx, MCB4300, MCB1800, MCB1857, MCB4357, FLM, external flash, LPCXpresso flash programming
Abstract	The Application Note describes the steps involved in writing a flash programming algorithm for unsupported Cortex -M devices. The algorithm will be used in conjunction with the LPCXpresso IDE to flash the binary files onto the Flash memory on the board. The accompanied project implements a Flash Programming algorithm for the MCB1800/4300.

Revision history

Rev	Date	Description
1	20121106	Initial version.

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The LPCXpresso IDE uses either built-in, or user loadable flash drivers. The existing built-in support is typically for parts with internal flash, and no external bus. If the need to support an unsupported device arises, it is possible to write your own driver file or adapt an existing Code Red example. This application note explains the procedure to adapt an existing flash programming algorithm project for the Cortex-M processors (.CFX file). The flash programming algorithm is written using LPCXpresso IDE. The LPCLink debugger is used to program the external flash.

The MCB4300/1800 development board is used as the target board. The LED blinky code is programmed using the .CFX file. The MCB4300/1800 development board contains the NXP LPC4357/1857 ARM Cortex-M3 processor.

The features of this board are:

- 180 MHz ARM Cortex-M3 processor-based MCU in LBGA256
- On-chip SRAM: 136 kB (LPC1857), 200 kB (LPC1850)
- On-chip flash: 1 MB dual bank (LPC1857), no on-chip flash (LPC1850)
- On-board memory: 16 MB NOR flash, 4 MB Quad-SPI flash, 16 MB SDRAM, and 16 kB EEPROM (I2C)
- Color QVGA TFT LCD with touch screen.
- 10/100 Ethernet port
- High-speed USB 2.0 host/device/OTG interface (USB host + micro USB device/OTG connectors)
- Full-speed USB 2.0 host/device interface (USB host + micro USB device connectors)
- CAN interfaces
- Serial/UART port
- MicroSD card Interface
- Four user push-buttons and one reset button
- Digital temperature sensor (I2C)
- Analog voltage control for ACD input
- Audio CODEC with line-in/out and microphone/headphone connector + speaker
- Debug interface connectors
  - 20-pin JTAG (0.1 inch)
  - 10-pin Cortex debug (0.05 inch)
  - 20-pin Cortex debug + ETM Trace (0.05 inch)

This application note describes the following:

- Finding the required hardware lines.
- Creating the CFX project.
- Writing the plash programming algorithm.
- Flashing a binary file using the CFX file and verifying it.

2. Finding the required hardware lines

The external flash memory has 32-bit wide data lines (two 16-bit memory modules cascaded to form a 32-bit wide memory) and 24-bit wide address lines. The memory chip has 22 physical address lines, but address lines, A[23:2], of the MCU are connected to address lines A[21:0] of the memory. The address lines A[1:0] of the MCU also need to be multiplexed because the MCU accesses the memory at word boundaries, although they are never used. The LPC1857 supports up to 32-bit address and data lines which need to be configured.

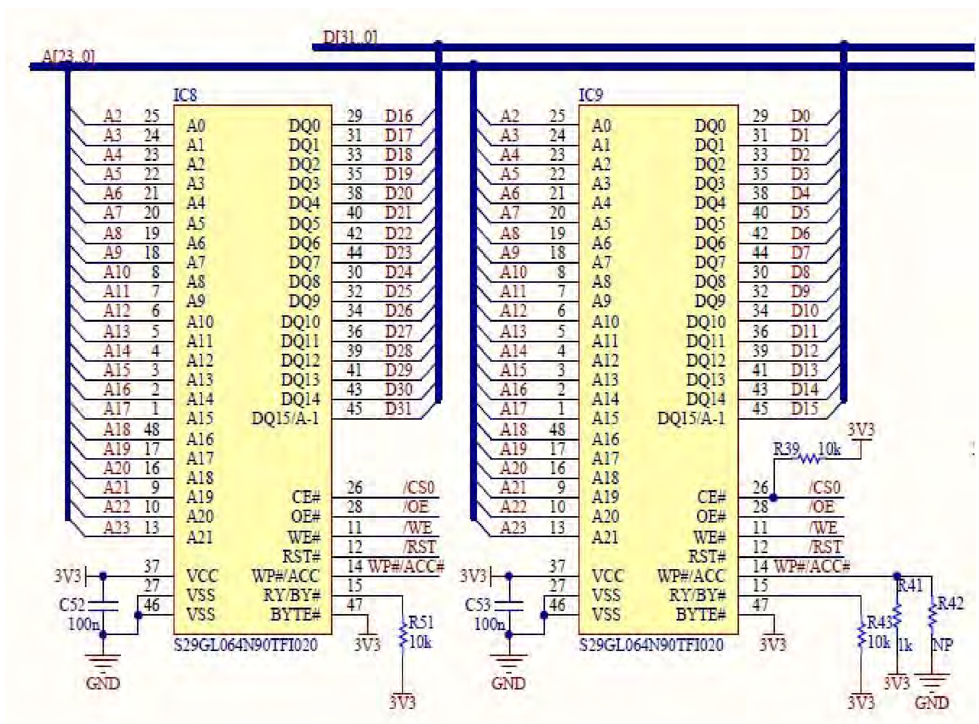

Fig 1. Schematic showing the connections required for the flash memory used

The control lines CS#, OE# and WE# are also to be configured. Each of the two cascaded memory chip data lines are 16-bit wide. The lower two bytes of the data word are connected to one of the chips and the upper two bytes are connected to the other. This allows the MCU to see the data bus width as 32 bits.

3. Creating the flash programming algorithm

The LPCXpresso IDE allows creation of flash programming algorithms for unsupported devices. The algorithm source code is implemented as a project with special compiler and linker settings. The LPCXpresso IDE comes with the projects for programming the external flash on the Hitex board. This project can be used as a template to create projects for different boards.

Your flash driver project as a minimum should include six source/header files. These are:

1. crt_flash_if.c – MemoryDevice_t data definition (do not edit)
2. crt_flash_if.h – struct FlashDevice, MemoryDevice_t, and Flash API declarations (do not edit)
3. FlashPrg.c – Flash API implementation
4. FlashDev.c – struct FlashDevice data definition(s).
5. main.c – RAM resident test harness used to exercise the Flash API. A test harness can typically use the linker scripts created by the Code Red IDE Wizard.
6. cr_startup_xx.c – Part-specific startup code included for use with the test harness (e.g. cr_startup_lpc18xx.c, cr_startup_lpc178x.c, etc.). In the standalone flash driver build, the Init Flash API may need to duplicate part of the standard startup sequence.

The flash driver project may include multiple configurations. As a minimum, you'll need a debug configuration for the test harness, and separate configurations for each flash device. If multiple flash configurations are supported, you can use conditional compilation directives where appropriate in FlashDev.c and FlashPrg.c for each project configuration.

The following steps show how to create a new flash programming algorithm:

- Navigate to the directory C:\nxp\LPCXpresso_version\lpcxpresso\Examples\FlashDrivers\NXP\LPC18xx_43xx
- Unzip the folder LPC1850A_4350A_Hitex_SST39VF3201B. The project folder created is the external flash programming algorithm for the Hitex board. (This step is optional)
- Rename this if desired. (This step is optional)
- Open the project in the LPCXpresso IDE. (Either import the unzipped folder or import the zip file)
- Rename the project if desired
- Go to project properties->C/C++ Build->Settings and select the Build artifact tab (Shown in Fig 2).
- In the Settings, select the "FlashDriver_XXMB" configuration
- Change the artifact name
- Change the artifact extension to "cfx". Once these steps are complete press the OK button in the dialog box and close the properties window.

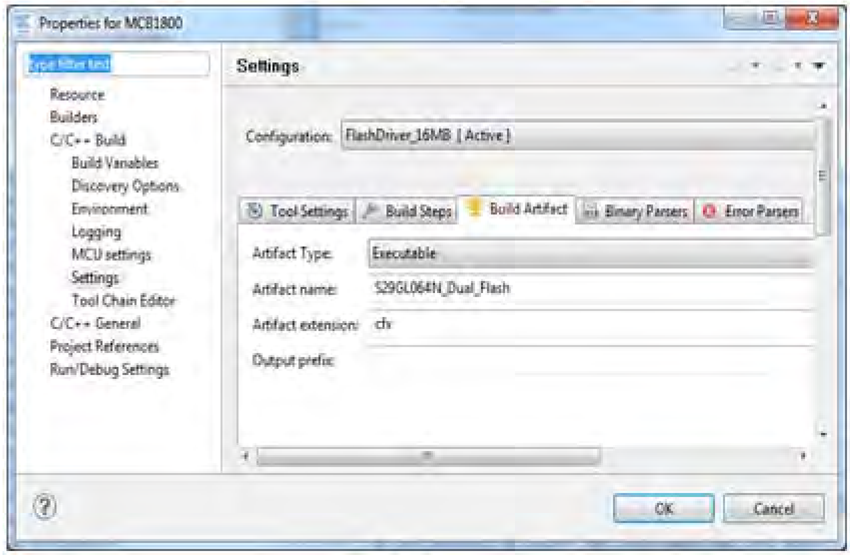

Fig 2. Build artifact settings window

- Adapt the programming algorithms in the file FlashPrg.c.
- Adapt the device parameters in the file FlashDev.c.
- Build the project. To do this select the project and select the Project menu and select the Build project option. This builds the corresponding cfx file in the project folder.

4. Writing the flash programming algorithm

The flash programming algorithm consists of the flash driver API (FlashPrg.c) and FlashDevice data (FlashDev.c) declared in the FlashDevice Structure (crt_flash_if.h). The flash API is a standard interface known to select Code Red IDE debug utilities. If a driver file is provided to the debug session then the utility will attempt to use it for in-application flash programming.

The FlashPrg.c has various functions.

The mandatory functions are:

- Init()
- UnInit()
- EraseSector()
- ProgramPage()

The optional functions are:

- EraseChip()
- BlankCheck()
- Verify()

The init() function is responsible for the main initializations required. The first three calls (for every operation) from the host-side are made to this function. It's called three times successively, one each for erase/program/verify. The separate calls support unique setup for each operation, but it's unlikely a driver needs to be written this way unless under rare circumstances. These calls add significant overhead (slower operation), so they are not called before each erase/program/verify call. Initialization may include setting up an external memory controller, configuring address/data lines and/or chip selects, and configuring the core clock/PLL frequency for flash operations. The Code Red IDE host-side is not guaranteed to provide a valid clock frequency to an Init call, so it's safer to complete all clock/PLL setup in this code.

The ProgramPage function is used to program the flash with the user provided binary file. The function is as shown in Fig 3. The ProgramPage function uses a sequence of instructions before the actual data is written to the flash memory. This sequence is determined from the user manual of the flash memory used. After every word is written to the data lines the memory is polled to determine if the write has been successfully completed. If it was successful then the next word is written into the memory, otherwise an error is returned.

```
int ProgramPage (unsigned long adr, unsigned long sz, unsigned char *buf)
{
      int i;

      for (i = 0; i < ((sz+3)/4); i++)  {
        // Start Program Command
        M32(base_adr + (0x0555 << 2)) = 0x00AA00AA;
        M32(base_adr + (0x02AA << 2)) = 0x00550055;
        M32(base_adr + (0x0555 << 2)) = 0x00A000A0;
        M32(adr) = *((unsigned long *) buf);
        if (Polling(adr) != 0) return (1);
        buf += 4;
        adr += 4;
      }

      return (0);                                   // Finished without Errors
}
```

Fig 3. Program function

The EraseSector function is used to erase a sector and the EraseChip function is used to erase the whole chip.

The sequence of instructions that initiate the two erase functions differ by only a single instruction. The set of instructions that are to be used in order to initiate an erase is provided in the product user manual. Fig 4 and Fig 5 show the EraseChip and EraseSector functions, respectively.


```
int EraseChip (void)
{

    // Start Chip Erase Command
    M32(base_adr + (0x555 << 2)) = 0x00AA00AA;
    M32(base_adr + (0x2AA << 2)) = 0x00550055;
    M32(base_adr + (0x555 << 2)) = 0x00800080;
    M32(base_adr + (0x555 << 2)) = 0x00AA00AA;
    M32(base_adr + (0x2AA << 2)) = 0x00550055;
    M32(base_adr + (0x555 << 2)) = 0x00100010;

    return (Polling(base_adr));  // Wait until Erase completed
}
```

Fig 4. EraseChip function

```
int EraseSector (unsigned long adr)
{

    // Start Erase Sector Command
  M32(base_adr + (0x555 << 2)) = 0x00AA00AA;
  M32(base_adr + (0x2AA << 2)) = 0x00550055;
  M32(base_adr + (0x555 << 2)) = 0x00800080;
  M32(base_adr + (0x555 << 2)) = 0x00AA00AA;
  M32(base_adr + (0x2AA << 2)) = 0x00550055;
  M32(adr) = 0x00300030;

  //extra addition - start
  do {                          // Wait for Sector Erase Timeout
    fsr.v = M32(adr);
  } while ((fsr.b.q3 == 0) || (fsr.b.q3h == 0));
  //extra addition - end

  return (Polling(adr));       // Wait until Erase completed    //Mod
}
```

Fig 5. EraseSector function

A host-side flash program sequence is typically BlankCheck -> EraseSector (erase by sector) -> ProgramPage -> Verify. There may be one or more pages within a sector. For those pages not on a sector boundary, the sequence is ProgramPage -> Verify. If the Verify API call is not implemented, the debug utility will verify flash contents against its image buffer. This is recommended, since it typically results in faster driver operation. Otherwise, the debug utility uses a subsequent Verify call to the target-resident driver after each ProgramPage.

At the end of the erase/program sequences, the last three calls are made to the flash driver UnInit code. It's called three times successively to handle erase/program/verify setup restore, if required. It’s rarely necessary to implement this call. These calls can add significant overhead (slow operation), so are not called after each erase/program/verify call. Finally, the flash driver block is unloaded by restoring the RAM contents overwritten by the flash driver block, and the processor is typically reset.

The main parameter definition is the mandatory FlashDevice structure. This structure defines various parameters of the flash device. The structure used for the MCB1800 is shown in Fig 6.

The main definitions in the structure are as follows:

- Name of the device. This name appears in the “add flash programming algorithm” window.
- The data bus width. This can be internal, EXT16BIT, EXT32BIT, etc.
- The start address of the flash memory. This is required since the flash memory is mapped and the start location of this mapped memory is to be indicated to the user. This can be found by locating which bank the external flash is connected to. For this example, the external flash is controlled by the CS0 line.
- The size of the flash memory.
- Page size is used by the host-side as the maximum size image buffer the host can safely send per call to the target-resident ProgramPage API. This is set to 0xFF.

  It is important to note that the device page size and the page size specified in the FlashDevice structure are different. The page specified in the structure is used by the host-side as the maximum size image buffer the host can safely send per call to the target-resident ProgramPage function (defined in FlashPrg.c) call. The ProgramPage implementation must accommodate any restrictions imposed by a fixed size device page, and be prepared to handle partial buffers. The exported page size is directly related to flash program time.
- Content of erased memory (typically 0xFF).
- Timeout (in milliseconds) of ProgramPage function.
- Timeout (in milliseconds) of the EraseSector function.
- Sector size and the relative start address of the first sector (offset from start of flash).

```
struct FlashDevice const FlashDevice  =  {
   FLASH_DRV_VERS,              // Driver Version, do not modify!

   "S29GL064N Dual Flash (16MB) Keil_MCB1850_4350 " __DATE__ " "__TIME__, // De

   EXT32BIT,                    // Device Type
   0x1C000000,                  // Device Start Address
   0x01000000,                  // Device Size in Bytes (16MB)
   1024,                        // Programming Page Size
   0,                           // Reserved, must be 0
   0xFF,                        // Initial Content of Erased Memory

   100,                         // Program Page Timeout 100 mSec
   3000,                        // Erase Sector Timeout 1000 mSec

// Specify the sector sizes and the relative start address of the first sector

   {{0x002000, 0x000000},          // Sector Size 4kB (1024 Sectors)
   {SECTOR_END }}
};
```

Fig 6. The flash device structure

For the external flash memory used on the MCB1800, the following values are used:

- Name of device – S29GL064N dual flash (16 MB) Keil_MCB1850_4350.
- Device Type – EXT32BIT. This indicates an external memory of 32-bit wide is used.
- Start Address – 0x1C000000. This can be found out by determining the bank to which the external flash is connected. For the MCB1800, the bank is CS0. The memory map diagram is shown in Fig 7).
- Device size – 0x1000000. The external memory is 16 MB in size.
- Page size – 1024. Each page is 1 kB.
- Initial flash content – 0xFF. This sets the initial state of the flash, erased state, to 0xFF.
- Program page timeout and erase sector timeout are 100 mSec and 1000 mSec respectively.
- Sector size and relative start address of the sector – 0x2000, 0x0. The start address is 0x00 and the sector size is 0x2000.

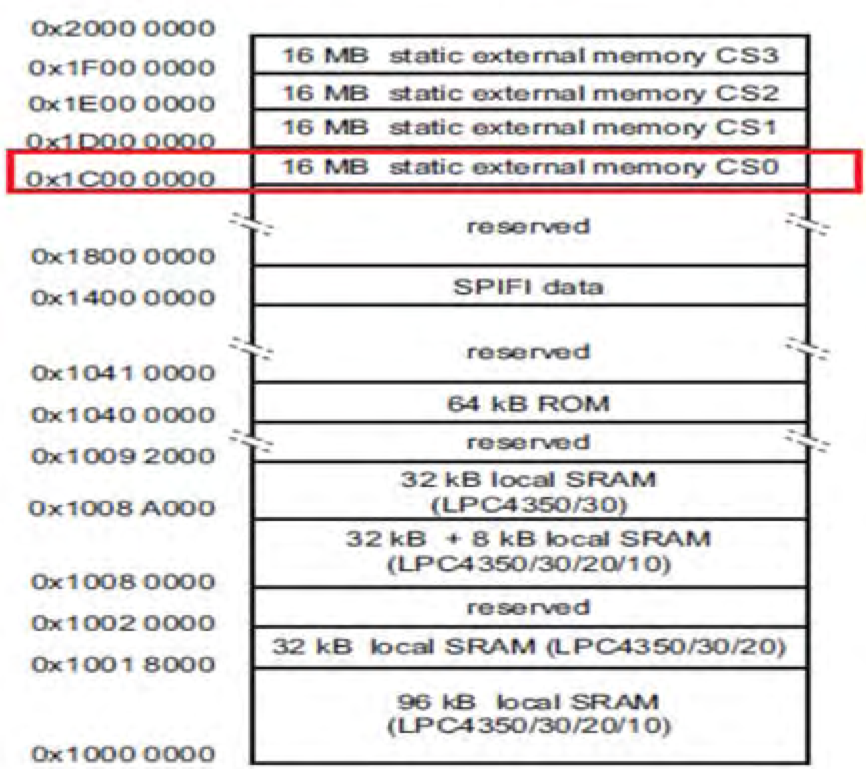

Fig 7. Static external memory mapping

Note: For the SPIFI memory the existing flash programming algorithm (written for the Hitex board) can be used (LPC1850A_4350A_SPIFI).

5. Hardware connections

Jumpers P2_9, P2_8, P1_2 and P1_1 are to be changed. P2_8 should be in the HIGH position and the other three are to be in the LOW position; this configuration is for booting from external 32-bit memory. Connect the JTAG debugger to the any of the JTAG headers and power up the board using a USB cable or a power source.

6. Flashing a binary file using the cfx file and verifying it

An example test harness (main.c) is supplied in the SPIFI example project, and should be modified for your use. The purpose of this file is to validate the operation of each method exported by the flash API.

The Init function as described earlier configures your hardware to erase/program your flash device. Be certain the setup configures your hardware as intended (memory controller, address/data lines, clock/PLL, etc.).

The ProgramPage function is usually tested by writing a repeating pattern to flash and verifying the flash memory against the pattern.

The EraseSector method receives an absolute address of a sector boundary. Be certain it translates the sector address to the represented sector, and completely erases the sector. The contents of the sector should show valEmpty content for each byte in the sector.

Note the test harness example does not make use of the FlashDevice toProg and toErase timeout values. Use the vendor documentation, and your test harness observations to make adjustments to the exported timeout values.

The UnInit code is often unimplemented. The host-side will at least reset the core after flash program, so it’s often not needed.

Once the project is built and the .cfx file has been generated it needs to be placed in a specific folder of LPCXpresso installed directory. The folder C:\nxp\LPCXpresso_version\lpcxpresso\bin\Flash contains all the .cfx files. The newly generated .cfx file needs to placed here (the .cfx file provided along with this app note can be used here). Once the .cfx file has been added, the project that is to be flashed needs to be changed in order to incorporate these changes. First, the new memory (external flash) needs to be added to the selected MCU. This can be done by selecting the C/C++ Build->MCUsettings (in the Project Properties window) and clicking Edit and adding the memory as shown in Fig 8. Note that the values of location (0x1C000000) and Size(0x100000) are in sync with the value defined in the flash device structure.

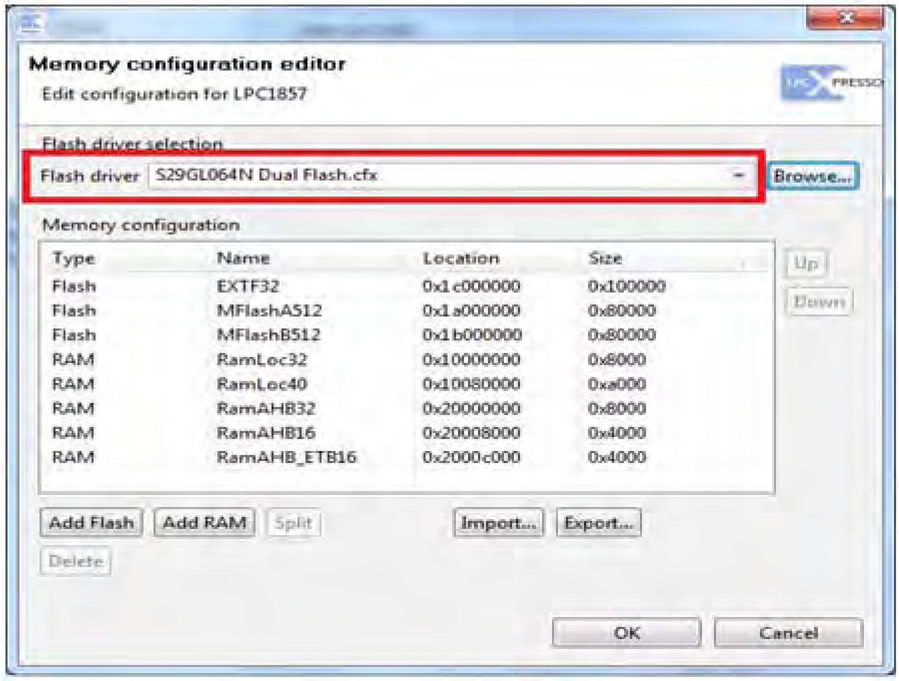

Fig 8. Adding the new Flash memory map to the chip select

The same dialog box (Edit, shown in Fig 8) can be used to add the flash driver (S29GL064N_Dual_Flash.cfx provided as part of the App note). The window appears as shown above.

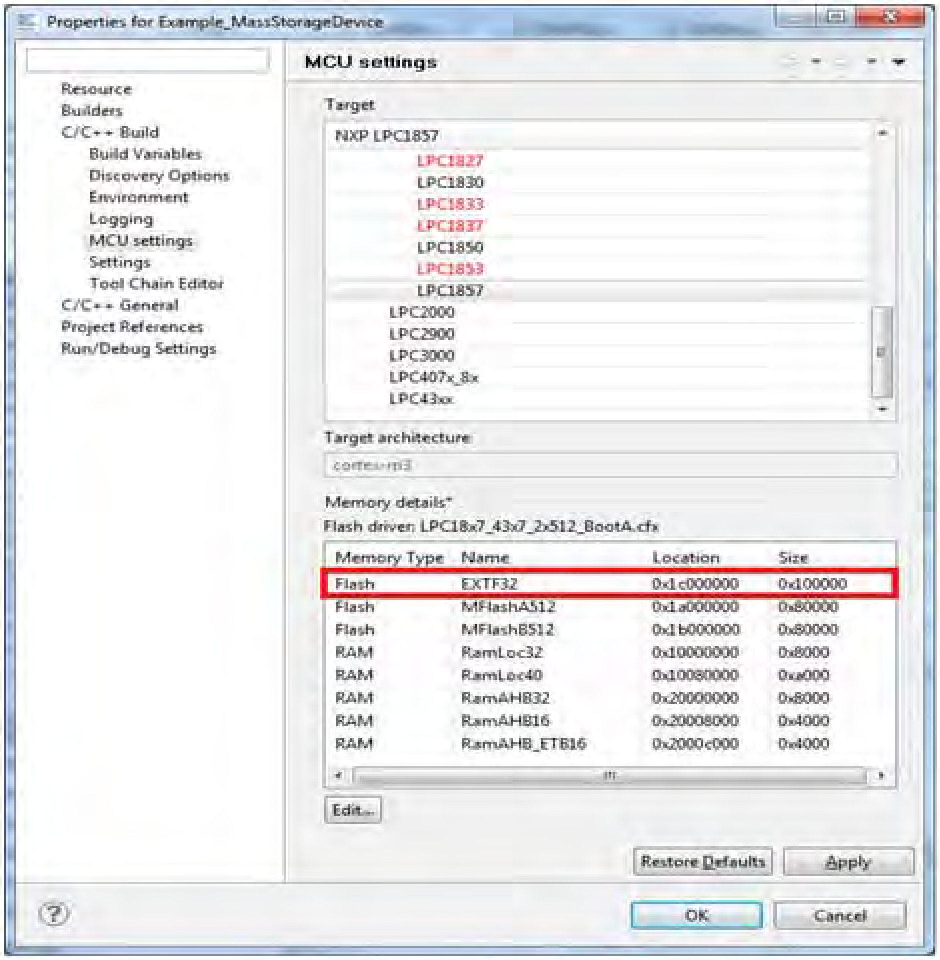

Fig 9. Adding the flash driver

The value 0x1C000000 is specified as the location because this is what is specified in the FlashDevice structure. For flashing the SPIFI memory select the appropriate memory address (0x14000000 is one of the SPIFI memory banks on the LPC1857/4357). Note that the corresponding flash driver also needs to be selected.

Once these changes are done and the sample application is built, the binary file is programmed into the flash memory by clicking on the program flash button (Fig 10), and selecting the corresponding .axf file (Fig 11). This will program the flash memory using the newly written flash programming algorithm. The board needs to be reset after this.

Fig 10. Program flash button

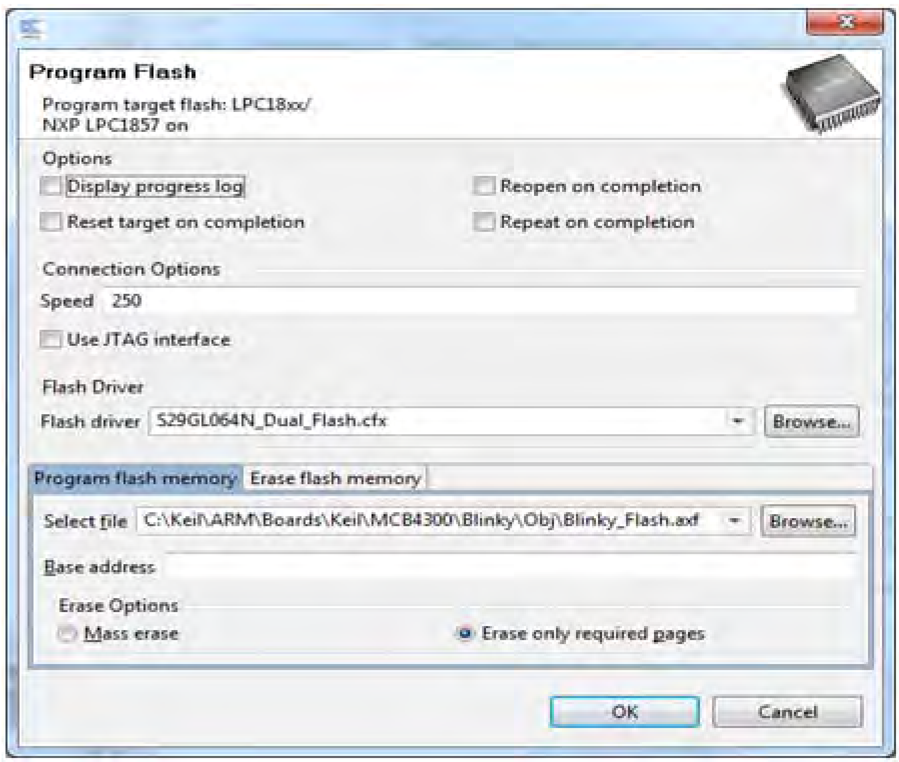

Fig 11. Program flash window (LPCXpresso version used is v4.2.4_beta [Build 299])

7. References

[1] MCB1800v1-1keil – Keil MCB1800/4300 schematic.

[2] UM10503 – LPC18xx user manual

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. List of figures

10. Contents

Date of release: 6 November 2012
Document identifier: AN11285