AN12563

SPI Flash Programming Algorithms for LP546xx

Rev. 0 — 26 August, 2019 Application Note

Contents

1 Introduction

LPC546xx has external spi flash interface(SPIFI) which can R/W external flash with single/dual/quad mode. It also supports XIP mode which means code can be executed directly on spi flash. SPIFI greatly expands application's code size, and makes it possible to store large data(image or even videos). However, downloading data into spi flash in development phase is always a troublesome problem, since different spi flash vendors have different command sets. Customers often meet the situation that original flash programming algorithm in MDK cannot fit their hardware, which causes downloading data incorrectly.

This short AN give a solution to this issue. It provides step-by-step guide on how to create a flash programming file for MDK, and provide a pre-compiled flash programming algorithm which can be used directly.

This AN is helpful to:

1. Who uses MDK with LPC546xx/LPC540xx and wants to create a flash programming algorithm for their own spi flash part.
2. Who wants to get knowledge for LPC546xx SPIFI peripheral
3. Who uses MDK with LPC546xx/LPC540xx and wants a common programming algorithm file to support most SPI flash parts on markets.

1.1 Glossary

Table 1. Abbreviation

Items	Description
SPIFI	SPIFI module on LPC546xx series, which is used for access external spi flash.
MDK(Keil)	Arm's Integrated development environment
FLM	MDK's flash programming algorithm file(ELF format file)

2 Implementation

2.1 Overview

Flash Programming Algorithms(for MDK, use FLM for short) are a piece of software to erase or download applications to Flash devices. A Pack with Device Support usually contains predefined FLM file that is supported by the DFP. A template for creating algorithms is available in the CMSIS Pack. In MDK, the Flash Programing Algorithms is an FLM file. In fact, FLM file is in ELF format.

SPIFI in LPC546xx is a peripheral used to access external spi flash. Currently, if you want to download data into external spi flash via MDK. Only LPC540xx_MT25QL128. FLM is available on DFP package. It means that only MT25QL128 is supported. Although this algorithm can support most MICRON spi flash parts, it usually cannot support another vendor, such as Winbond or ISSI. The reason is obvious: LPC540xx_MT25QL128 FLM algorithm uses quad program and quad read command set, in which those command operands are different from vendor to vendor.

To address this issue. We must create a common FLM file which is compatible with most spi flash on markets. Although, there may have some tradeoff. To achieve this goal, we must first implement SPIFI driver on LPC546xx, which is used to create flash programming algorithms.

2.2 SPIFI

MCUXpresso SDK provides SPIFI driver for accessing external spi flash, and example code for SPIFI. SPIFI example is under:

\boards\lpcxpresso54608\driver_examples\spifi\polling_transfer

Before creating FLM file, you need SPIFI driver code to let you R/W external spi flash. This driver code and example is in *polling_transer* demo. Make sure you can run polling_transfer demo successfully on your hardware platform, the SPIFI driver code used by this demo is reused in flash programming algorithm. If *polling_transer* demo cannot run successfully on your hardware, see section 2.3 about how to modify this example to fit your hardware.

2.3 How to modify SPIFI driver to support new spi flash device

There are some reasons that cause this demo fail to run:

- Incompatible pin configuration
- Spi flash command sets incompatible

2.3.1 Incompatible pin configuration

This issue is easy to find and easy to fix. Make sure that your hardware connection for spi flash is same with LPC546xxXpresso board, otherwise, you must modify function *BOARD_InitPins* in *pin_mux.c* to align with your hardware.

2.3.2 Configurate spi flash command list

Different spi flash products have different command operands, especially for quad erase and quad programming function. Nearly all spi flash products use the same command operand for single erase, single read and single programming. One simple way to do with incompatible command operand is to use single read and single program function. This method might cause slower programming speed. Figure 1 shows SPIFI command table which use single wire read and programming command, this command list is compatible for most spi flash in market.

```
#define COMMAND_NUM (6)
#define READ (0)
#define PROGRAM_PAGE (1)
#define GET_STATUS (2)
#define ERASE_SECTOR (3)
#define WRITE_ENABLE (4)
#define WRITE_REGISTER (5)

spifi_command_t command[COMMAND_NUM] = {
    {PAGE_SIZE,     false, kSPIFI_DataInput,    0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeAddrThreeBytes, 0x03}, /*   single wire read */
    {PAGE_SIZE,     false, kSPIFI_DataOutput,   0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeAddrThreeBytes, 0x02}, /*   program page */
    {1,             false, kSPIFI_DataInput,    0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeOnly,           0x05}, /*   get status */
    {0,             false, kSPIFI_DataOutput,   0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeAddrThreeBytes, 0x20}, /*   erase 4K sector */
    {0,             false, kSPIFI_DataOutput,   0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeOnly,           0x06}, /*   write enable */
    {1,             false, kSPIFI_DataOutput,   0, kSPIFI_CommandAllSerial,     kSPIFI_CommandOpcodeOnly,           0x01}};/*   write registor */
```

Figure 1. Common LUT for spi flash operation

2.4 Create Flash Programming Algorithm File

2.4.1 Implement Flash Programming Code

Flash programming algorithms are defined with functions to erase and program the Flash device. Special compiler and linker settings are required. A step-by-step guide for how to create a flash programming algorithm is available, refer to CMSIS documentation:

http://www.keil.com/pack/doc/CMSIS_Dev/Pack/html/flashAlgorithm.html

Here are some important key points and tips:

1. Do not create a new uVision project, Copy the content from the ARM:CMSIS Pack folder (usually *C:\Keil\ARM\Pack\ARM\CMSIS\version\Device_Template_Flash*) to a new folder.
2. There is no main entry for flash programming algorithm. It is a position independent code.
3. The file **FlashPrg.c** contains the mandatory Flash programming functions **Init**, **UnInit**, **EraseSector**, and **ProgramPage**. Optionally, depending on the device features (or to speed up execution), the functions **EraseChip**, **BlankCheck**, and **Verify** can be implemented. As Table 2 shows:

Table 2. Flash programming algorithm API need to implement

Function Name	Indication	Description
BlankCheck	optional	Check and compare patterns
EraseChip	optional	Delete entire Flash memory content
EraseSector	mandatory	Delete Flash memory content of a specific secto
Init	mandatory	Initialize and prepare device for Flash programming
ProgramPage	mandatory	Write the application into the Flash memory
UnInit	mandatory	De-initialize the microcontroller after one of the Flash programming steps.
Verify	optional	Compare Flash memory content with the program code.

The functions which are considered to keep maximum compatibility with different vendors, have already been implemented in attached software. The **Read** and **ProgramPage** API uses spi one wire mode to keep compatibility with different flash vendors.

1. The file **FlashDev.c** contains parameter definitions for **FlashDevice** structure which is recognized by MDK. This structure contains all flash description including name, flash size, sector size, start address and so on, as Figure 2 shows.

```
struct FlashDevice const FlashDevice  =  {
   FLASH_DRV_VERS,            // Driver Version, do not modify!
   "LPC5460x SPIFI ALL", // Device Name
   EXTSPI,                    // Device Type
   0x10000000,                // Device Start Address
   0x01000000,                // Device Size (16MB)
   4096,                      // Programming Page Size (16 pages a 256)
   0,                         // Reserved, must be 0
   0xFF,                      // Initial Content of Erased Memory
   300,                       // Program Page Timeout 300 mSec
   3000,                      // Erase Sector Timeout 3000 mSec

// Specify Size and Address of Sectors
   0x001000, 0x000000,        // Sector Size 4kB (4096 Sectors)
   SECTOR_END
};
```

Figure 2. Flash programming algorithm API need to implement

2.4.2 Compile and create FLM file

After finish implementing **FlashDev.c** and **FlashPrg.c**. and compile the whole project. output file will be generated. In attached software, the output name is: **LPC5460x_SPIFI_ALL.FLM**

3 How to use FLM file

3.1 Add FLM in to Flash Programming Algorithm List

Perform following steps to Add FLM in to Flash Programming Algorithm List:

1. Copy the FLM file into ARM *\Keil\ARM\Flash* folder.
2. Open your application MDK project. Open Flash Download dialog and add LPC5460x SPIFI ALL into Programming Algorithm.
3. Remove previous spi flash FLM file if exist. This will let MDK use our FLM when downloading data into spi flash. As Figure 3 shows.

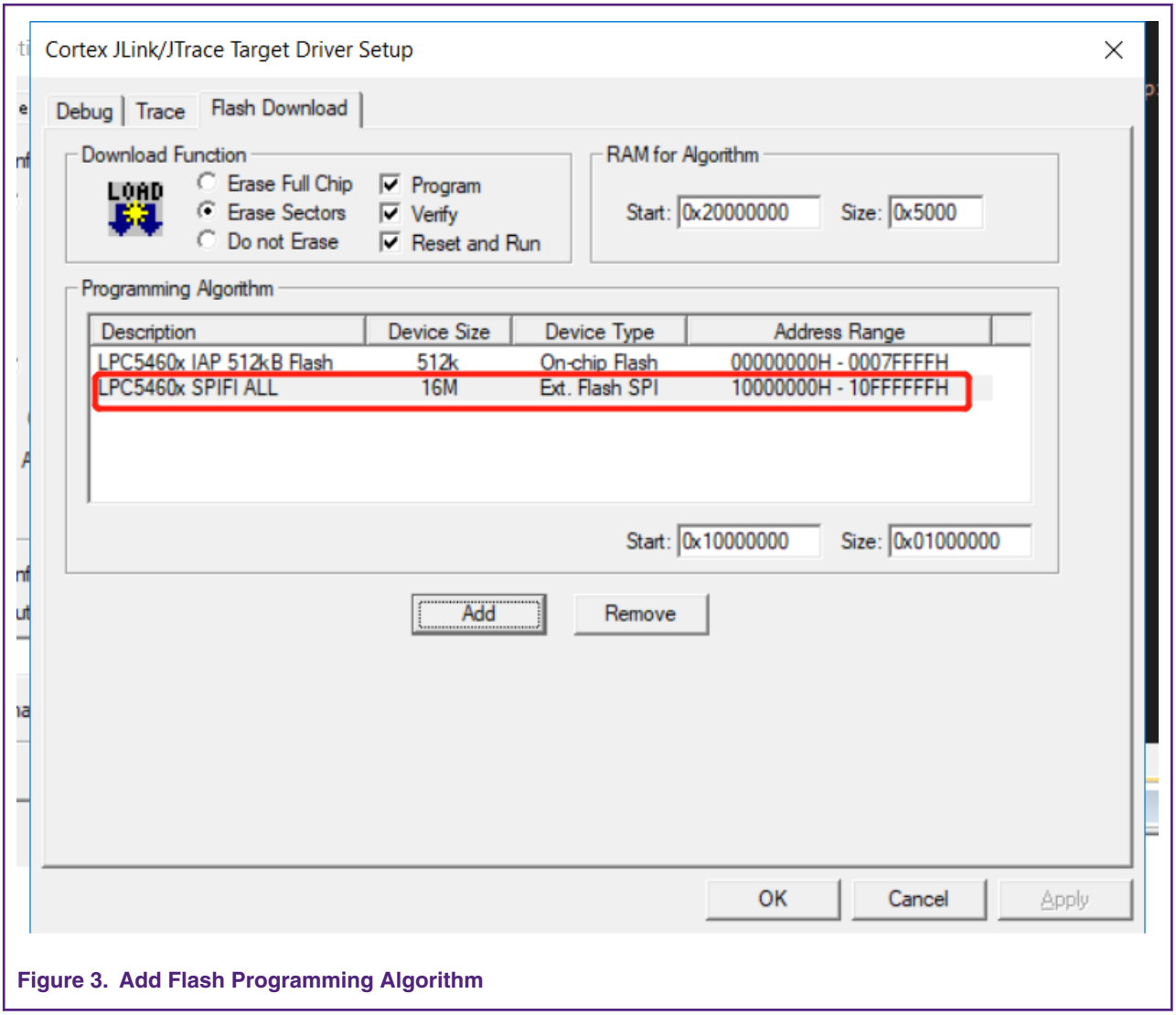

Figure 3. Add Flash Programming Algorithm

3.2 Verify result

Click load button to start downloading your application, as Figure 4 shows.

Figure 4. Download application to target

MDK shows the output log in the Build Output window. The log “Verify OK.” indicates your application downloaded in to MCU successfully. As Figure 5 shows.

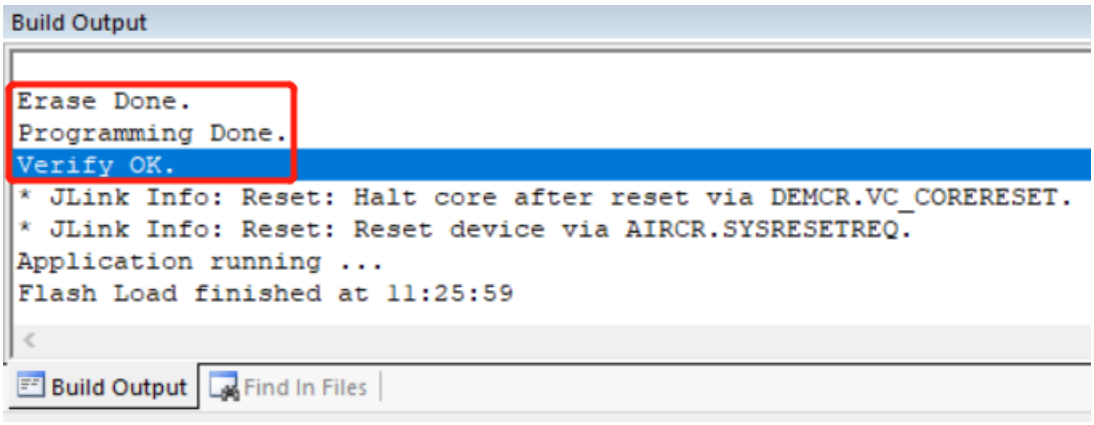

Figure 5. MDK output log "Verify OK"

4 Conclusion and Limitation

This AN is a guide on how to create a spi flash programming algorithm for MDK. It also provides a pre-compiled, ready to use FLM file which can support most spi flash on market.

To keep compatibility, the FLM project in attached software uses default core clock, single wire read and single wire programming command operand. The download speed is slower than using quad mode ones.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 26 August, 2019

Document identifier: AN12563

arm