AN11610

LPC5410x I2C SPI Secondary Bootloader

Rev. 1.0 — 8 April 2015 Application note

Document information

Info	Content
Keywords	LPC5410x, I2C, SPI, Firmware update, Secondary Bootloader
Abstract	This application note describes and implements a secondary bootloader via the I2C/SPI bus of the LPC5410x MCU. This secondary bootloader allows easy firmware update in an application environment, similar to a sensor hub, where the application processor can update the LPC5410x's firmware via the I2C/SPI secondary bootloader.

Revision history

Rev	Date	Description
1.0	20150408	Initial Version

Contact information

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

1. Introduction

The LPC5410x is an ARM Cortex-M4 based microcontroller for embedded applications. It includes an optional ARM Cortex-M0+ coprocessor, 104 kB of on-chip SRAM, 512 kB on-chip flash, numerous timers, four USARTs, two SPIs and three Fast-mode plus I2C-bus interfaces with high-speed slave mode and one 12-bit 4.8 Msamples/sec ADC.

The LPC5410x supports ARM Serial Wire Debug (SWD) mode for Coretx-M4 and, if present, the Cortex-M0+ core. Device programming can be achieved through the SWD port. In addition, the LPC5410x contains an on-chip boot ROM that supports In-System Programming (ISP) when the part resides in the end-user board and In-Application Programming (IAP) as directed by the end-user application code. The ISP mode is supported via USART0. The state of the pin P0_31 determines whether to enter ISP mode at boot time.

Table 1. ISP modes

Boot mode	P0_31	Description
No ISP	HIGH	ISP bypassed. Part attempts to boot from flash
USART0	LOW	Part enters ISP via USART0

For some applications, where the LPC5410x is a slave processor to the host processor, it is often necessary to program the LPC5410x through the host processor because the programming interface through the SWD and ISP via UART are not provided in the system. There are a broad range of applications that use the LPC5410x as a slave processor, for example, the unmanned vehicles, gaming, and Robot to name a few. The sensor hub application for smart phone products is another example, where the LPC5410x is used as a sensor hub. In this use case, the flash device must be programmed through a host interface, which is an interface between the application processor (AP) and the sensor hub.

The Secondary Bootloader (SBL) described and implemented in this application note provides a solution for the host processor to program the slave processor. It utilizes the boot ROM's IAP functionalities and allows programming the LPC5410x flash through I2C slave interface or SPI slave interface which are the common interfaces used between the host processor (referred to as AP in a sensor hub application) and the sensor hub.

Fig 1 shows an example of a system setup where the AP can program the LPC5410x via I2C/SPI interface assisted by the SBL code.

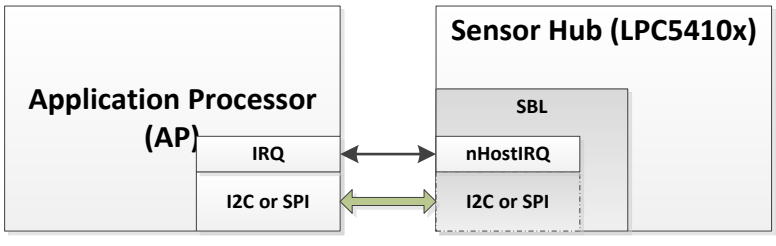


Fig 1. Secondary boot loader for sensor hub application

2. SBL Functionalities and Boot Process with SBL

The SBL is located at the first sector of user flash and contains routines to perform the functionalities described in Table 2.

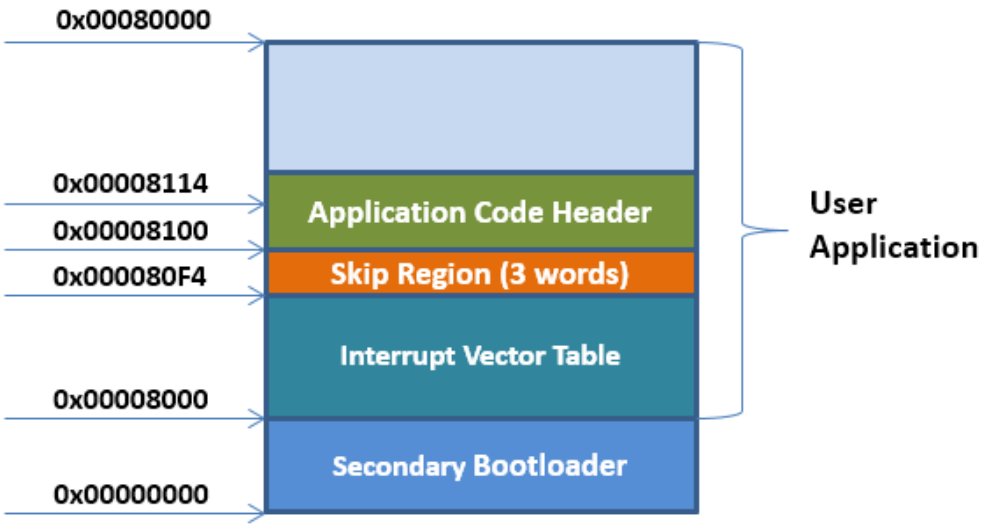
Table 2. SBL functionalities

Functionalities	Description
Host Interface Detection	Described in section 2.3
I2C and SPI communication	Interface with the host processor
Flash IAP programming	Described in section 2.4
Application image CRC checking	Verify CRC before booting

For applications using a secondary boot loader, the SBL is the boot strap mechanism that initializes the user application. With a role this important, it may be necessary to protect the SBL from accidental or malicious data tampering. The LPC5410x series features a flash sector locking ability, allowing the user to pick a flash sector to “lock down” and make the current flash contents permanent. Locking sector 0 after the SBL has been programmed will guarantee that the SBL is tamper proof while still allowing other flash sectors to be modified. Note that the CRP level of the chip is determined in sector 0, so when locking sector 0, the current CRP level will also become permanent. For more information on flash sector locking, please reference NXP AN 11653.

2.1 Memory map with applications boot with SBL

The SBL assumes that the user application starts from 0x00008000 in the flash region. Therefore, applications that boot from the SBL must be initially set up with the vector table at address 0x00008000. The SBL will set the Cortex-M4’s vector table offset register VTOR to 0x00008000 before passing the control to the application. The SBL uses the stack pointer at address 0x00008000 and the application entry point at address 0x00008004 to start the application. See Fig 2 for flash memory allocation of the SBL and the user application. See section 4.11 for details on the application image header requirement.



Below is the LPC5410x Flash Sector/Page information (all sector size is 32kB):

Sector Number	Page Numbers	Address Range
0	0-127	0x0000 0000 – 0x0000 7FFF
1	128-255	0x0000 8000 – 0x0000 FFFF
2	256-383	0x0001 0000 – 0x0001 7FFF
3	384-511	0x0001 8000 – 0x0001 FFFF
4	512-639	0x0002 0000 – 0x0002 7FFF
5	640-767	0x0002 8000 – 0x0002 FFFF
6	768-895	0x0003 0000 – 0x0003 7FFF
7	896-1023	0x0003 8000 – 0x0003 FFFF
8	1024-1151	0x0004 0000 – 0x0004 7FFF
9	1152-1279	0x0004 8000 – 0x0004 FFFF
10	1280-1407	0x0005 0000 – 0x0005 7FFF
11	1408-1535	0x0005 8000 – 0x0005 FFFF
12	1536-1663	0x0006 0000 – 0x0006 7FFF
13	1664-1791	0x0006 8000 – 0x0006 FFFF
14	1792-1919	0x0007 0000 – 0x0007 7FFF
15	1920-2047	0x0007 8000 – 0x0007 FFFF

Fig 2. Flash memory map for applications boot with SBL

2.2 Boot process with SBL

All LPC5410x parts with a secondary bootloader flashed will go through the following boot sequence. See Fig 3.

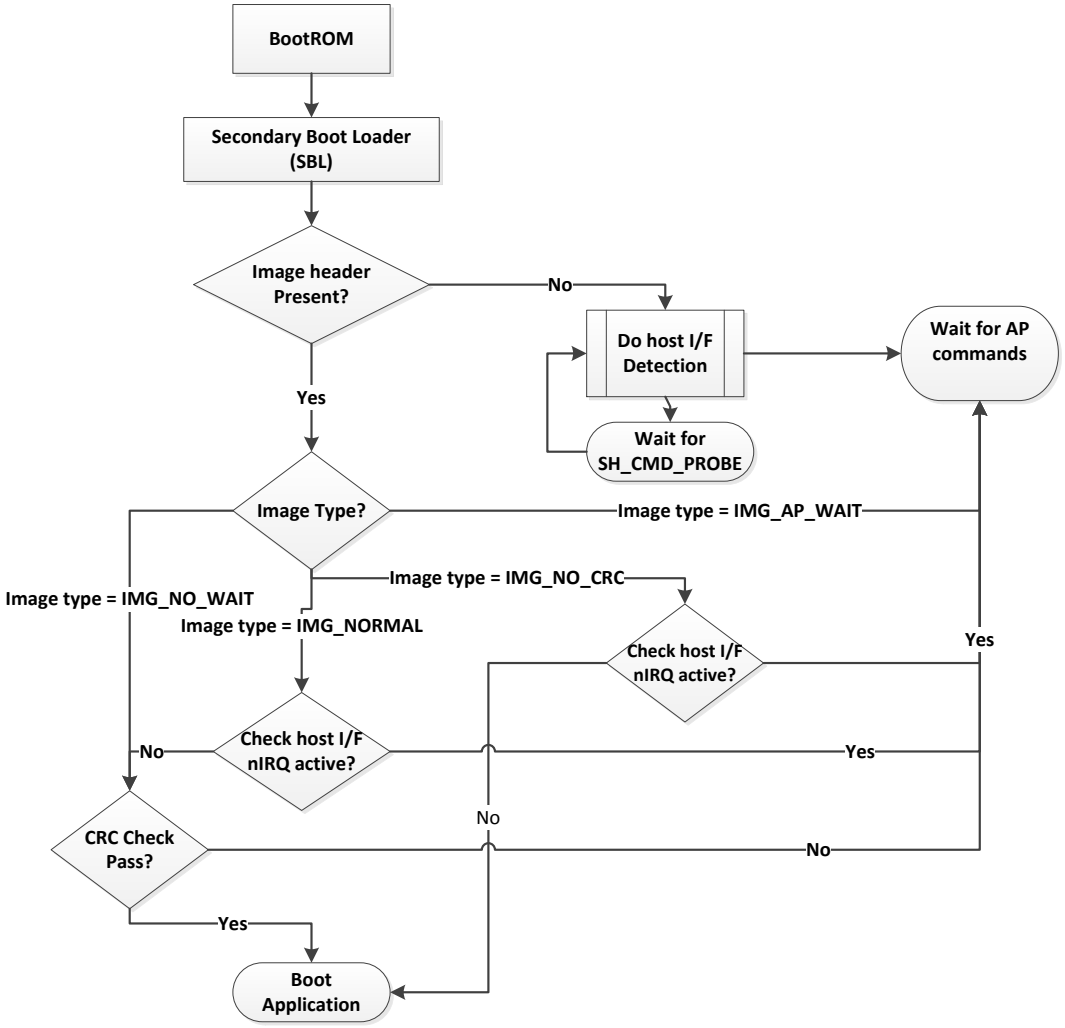


Fig 3. Boot flow of LPC5410x with secondary boot loader

- After reset (Power on reset, Watch Dog Reset, external reset, BOD reset or software system reset), the Boot ROM will run and pass the control to the SBL.
- To allow proper handshaking between the SBL and the application, an image header is required in the application image at offset 0x100 (0x00008100 absolute flash address). Before booting the application, the SBL checks for the presence of the image header. For more details, see section 4.1, Image header construction.
- If the image header is absent, the SBL will start host I/F auto detection. For more details, see section 2.3, Host interface detection.
- If the image header is present then the SBL checks the image type. For more details, see section 4.1.1.
- Depending on the image type, the SBL either checks the image integrity and boots the image automatically or enters an AP command processing loop (where the AP controls when to boot the application).

2.3 Host interface detection

The LPC5410x has three I2C interface ports (I2C0, I2C1, and I2C2) and two SPI interface ports (SPI0 and SPI1). To talk to the Host Processor, the SBL needs to know the host interface (I/F) being used to connect the AP to LPC5410x. In this design, the SBL performs an auto-detection of the host interface by monitoring the interface pins.

Depending on the interface type, the host I/F has the following pins:

- I2C
  - SCL: I2C clock pin. Host as an I2C master will drive this pin.
  - SDA: I2C data pin.
  - nHostIRQ: Active low Host nIRQ pin.

  Note that the SBL only supports three I2C addresses and the host application can choose one of the following addresses: 0x18, 0x1C, and 0x30.
- SPI
  - SCK: SPI clock pin. Host as the SPI master drives clock on this pin during communication. The SBL assumes SPI interface to operate in **Mode 0** defined in LPC54xxx user manual.
  - MOSI: SPI Master Output Slave Input pin. Data from master (host) is driven on this pin.
  - MISO: SPI Master Input Slave Output pin. Data to master (host) is provided on this pin.
  - SSEL: SPI select pin. SPI master (host) asserts this pin during communication with slave. Only SSEL0 pins are supported. The SBL assumes this pin to be active low (the host should pull this pin low during communication with LPC5410x).
- For auto detection to work, the host should continuously send SH_CMD_PROBE (0xA5) command until the nHostIRQ line is asserted by LPC5410x. After detecting the host I/F, the SBL drives the nHostIRQ line active. For the SBL to know which GPIO pin is used for nHostIRQ line, the SH_CMD_PROBE packet should be constructed properly. For SPI, the MISO pin should be set properly to receive data from LPC5410x.

2.4 SBL flash IAP programming support

The commands described in sections 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13, 6.14 utilize the IAP commands described in UM10850, Chapter 31, LPC5410x Flash API. When working with the SBL, it is not necessary for the user to check the detailed implementation of these commands. However, users can review this chapter for a background of the SBL implementation.

2.5 Download SBL to LPC5410x

Downloading of the SBL to LPC5410x can be achieved by many ways depending on the situation. In production, the SBL can be pre- programmed to the LPC5410x via a debugger or UART ISP mode whichever is available. In prototyping, the part can most likely be programmed after fitted on the board assume the SWD or ISP UART port is accessible.

The sample applications are built based on NXP's LPCXpresso54102 development board as shown in Fig 4. The LPCXpresso54102 development board comes with built in LPC-Link2 based debug probe, based on NXP LPC43xx MCU. This LPC-Link2 based debug probe is compatible with LPCXpresso IDE out-of-the-box, and with other toolchains via optional ARM CMSIS-DAP firmware.

When JP5 is not fitted, connecting a USB cable connected from J6 provides power to the system. With this configuration, the preprogrammed CMSIS-DAP Link2 and a virtual communication port (VCOM) UART bridge functionality are active and allows down loading the firmware with Flash Magic (http://www.flashmagictool.com/). You can verify the proper enumeration by making sure a CMSIS-DAP Human Interface Device as well as an LPC-LinkII VCom Port show up under the Device Manager. For more information regarding using LPCXpresso54102, please visit below web page:

http://www.nxp.com/demoboard/OM13077.html

Fig 4. LPCXpresso54102 development board

When using FlashMagic, following below steps to download the SBL to LPC5410x:

- Put the LPC5410x into ISP mode by pressing down the ISP button and then toggling the Reset button (Low -> High).
- Choose the SBL hex file from the application note package and allow Flash Magic to successfully program the SBL.
- After downloading the SBL hex file, press the Reset button on LPCXpresso54102 board to allow the boot ROM and SBL boot.

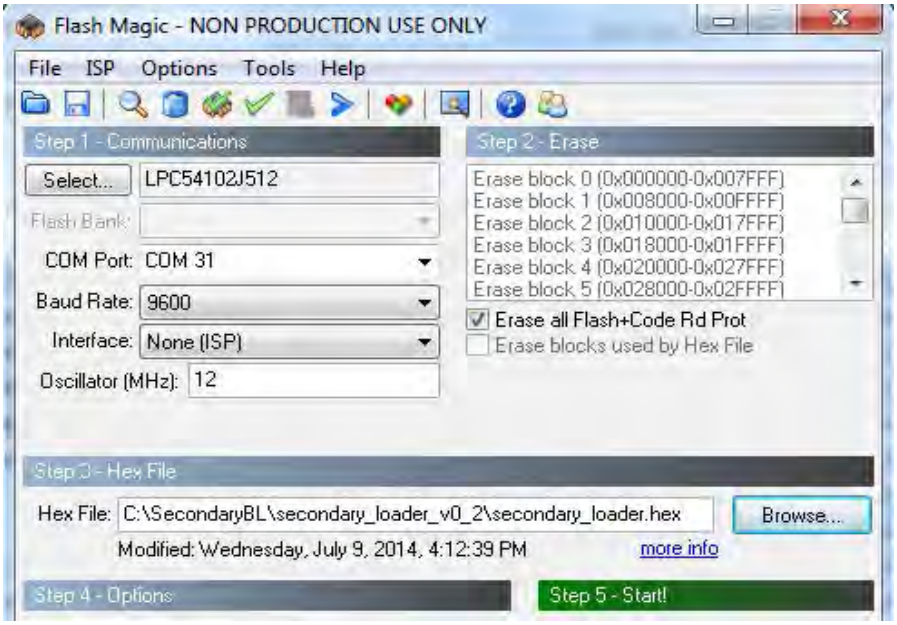


Fig 5. Download SBL to LPC54102 with Flash Magic

3. Emulated host processor/slave processor communication

A Windows PC application (PC_Android_Emulator.zip) to emulate an Android application processor is also provided with this application note. This emulator application has implemented test code for all of the major functionalities of the secondary bootloader.

3.1 System introduction

The windows PC application talks to the SBL through the USB to I2C/SPI Bridge (implemented with LPC43xx) via NXP's USBSerialIO library. See Fig 6 for the high level block diagram of the system. For information on the LPCUSBSIO library, go to:

http://www.lpcware.com/search/gss/libusbsio

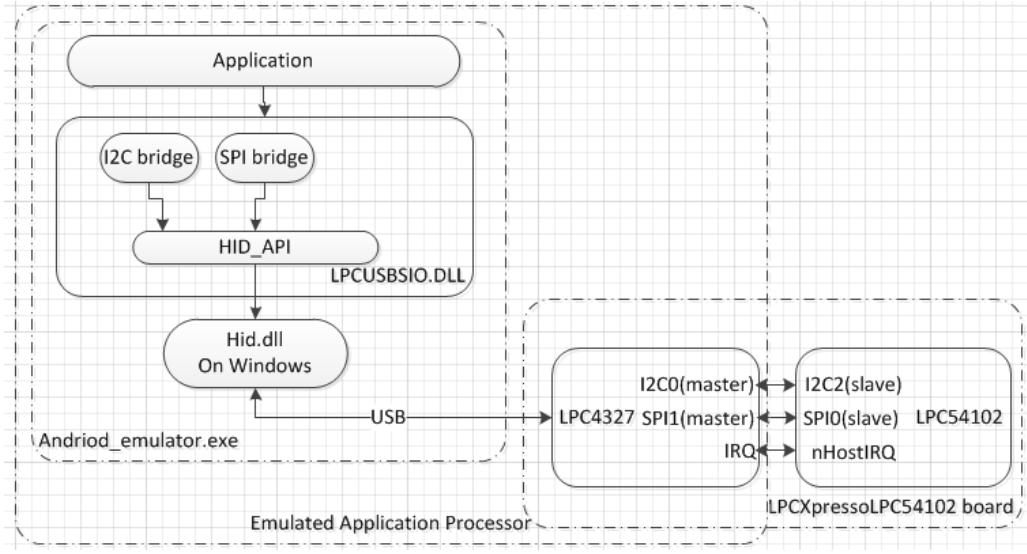


Fig 6. Emulated Host Processor communication with Sensor Hub with LPCXpresso54102 board

3.2 Host commands

As seen from Fig 6, running the Android Emulator from the PC, allows the user to communicate with the LPC43xx and they together work as the host processor.

After successfully downloading the SBL following instructions in section 2.5 and pressing the reset button, the user can double click on the Andriod_Emulator.exe to get the options to communicate with the LPC5410x via I2C or SPI. In this example, the I2C interface is chosen to communicate with LPC54102 via the I2C interface.

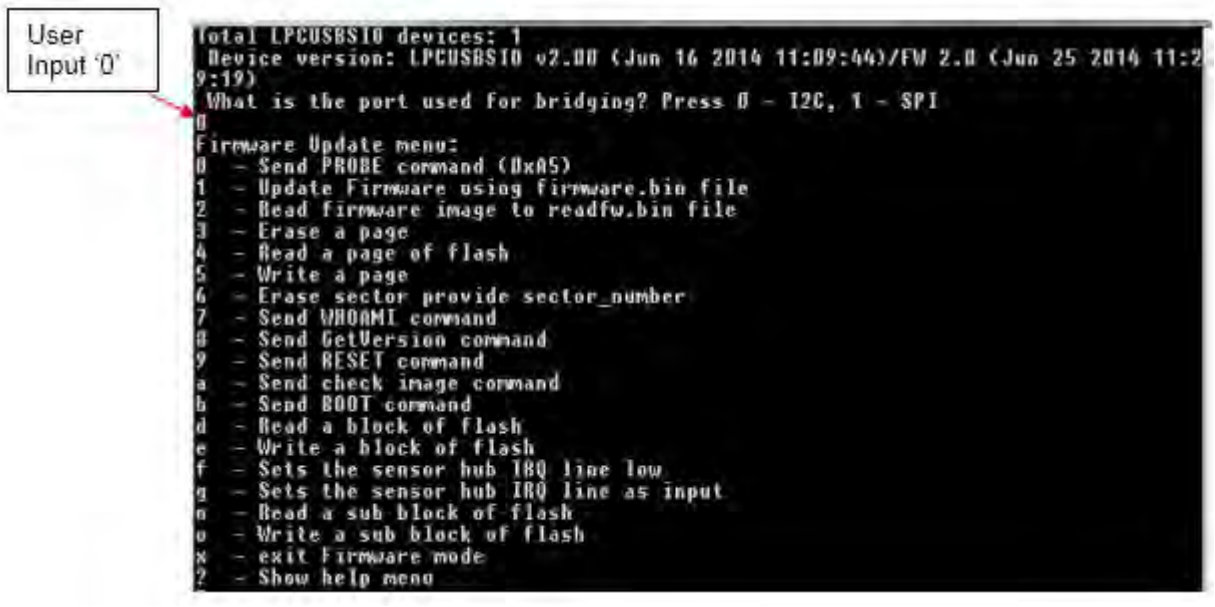


Fig 7. Android Emulator Host Commands

On receiving the selected communication channel, the Android Emulator presents a menu of commands supported. The commands are self-explanatory. The Android Emulator translates these user commands to messages expected by the SBL. See Chapter 6 for the message communication between the LPC43xx and LPC54102 on the I2C and SPI port. Note that the LPC43xx implements the USB to I2C/SPI Bridge protocol as well as LPCLink2 debugging protocol.

The PC might be missing the msvcr110d.dll used in this application. This dll is included with this application note as msvcr110d.zip. For 64 bit Windows, place this dll in folder C:\Windows\SysWOW64. For 32 bit Windows, place this dll in folder C:\Windows\System32.

4. Application implementation

An application that boots from the SBL must meet the following requirements (see section 2.1):

- Uses no Flash memory in the region of 0x00000000 to 0x00007FFF.
- Requires an image header at address 0x00008100 in Flash.

Within this application note, the LPCXpresso, Keil, and IAR IDE sample projects that boot from SBL are included. File LPCXpresesso_BlinkySolution.zip contains the LPCXpresso sample project. File keil_iar_BlinkySolution.zip contains the Keil/IAR sample projects. Note that all sample projects use I2C as the host interface, but the valid SPI host interface setting is provided in the image header. The user can easily change the host interface choice from I2C to SPI to test the SPI host interface functionality.

4.1 Image header construction

The SBL expects all application images to have an image header at offset 0x100 (absolution address 0x00008100 in Flash).

Table 3. Image Header

Field	Offset	Size (bytes)	Value	Description
Header Marker	0x0	4	0xFEEDA5A5	
imageType	0x4	1		Image type. 0 - IMG_NORMAL 1 - IMG_AP_WAIT 2 - IMG_NO_WAIT 3 - IMG_NO_CRC
ifType	0x5	1		Host interface type and port. 1 – I2C0 port 2 – I2C1 port 3 – I2C2 port 4 – SPI0 port 5 – SPI1 port
irqPortPin	0x6	1		GPIO pin used for Host interface IRQ function. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number
misoPortPin	0x7	1		MISO pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
sselPortPin	0x9	1		SSEL pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
sckPortPin	0xA	1		SCK pin used for SPI host

Field	Offset	Size (bytes)	Value	Description
				interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
checksum	0xB	1		XOR of all the 7 bytes above.
CRCLen	0xC	4		Length of the image. Applications requiring fast boot time can have partial CRC check by altering this length field. Note, the CRCValue field should be calculated for the partial image only.
CRCValue	0x10	4		CRC32 of the image excluding this field.

4.1.1 Image type

The “Image Type” field influences the boot flow of the secondary boot loader.

4.1.1.1 IMG_NORMAL

IMG_NORMAL (0): For this type of image, the SBL checks the nHostIRQ line before checking the CRC32 of the image. The SBL gets the information about nHostIRQ pin from the host I/F pin configuration information embedded in the image header. This is the recommended image type. The nHostIRQ check provides AP the chance to re-program the sensor hub if the application image gets corrupted or crashes before reaching a state to respond to AP commands.

4.1.1.2 IMG_AP_WAIT (1)

For this type of image, the SBL goes straight to the command processing loop. The host/AP must send SH_CMD_BOOT (0xA3) command to boot the application. Before issuing boot command the AP can check the integrity of the application image by sending SH_CMD_CHECK_IMAGE (0xA4) command.

4.1.1.3 IMG_NO_WAIT (2)

For this type of image, the SBL only does CRC32 check on the application image. It the check passes the SBL passes control to the application image.

4.1.1.4 IMG_NO_CRC (3)

For this type of image, the SBL checks the nHostIRQ line although it does not check the CRC32 of the image. The SBL gets the information about nHostIRQ pin from the host I/F pin configuration information embedded in the image header. This mode should be used only during development and when the user has SWD program access to the part.

4.1.1.5 Special notes on booting IMG_NORMAL and IMG_NO_CRC images

For IMG_NORMAL and IMG_NO_CRC image boot, the host can use the nHostIRQ line to stop booting the image and perform reprogram of the part. In this case, the host I/O line that is connected to the LPC5410x first works as an output and pulls low. The nHostIRQ line on the LPC5410x first works as an input to sense that the host has pulled this line low. When the SBL senses this line being pulled low, it stops proceeding to check the CRC32 of the image. Then the host needs to reconfigure the nHostIRQ line to be switched to being an input to allow the nHostIRQ line on the LPC5410x to drive it.

With the emulated Android AP/Sensor Hub environment as described in section 3, the usage of nHostIRQ in IMG_NORMAL image booting can be described as shown in Fig 8.

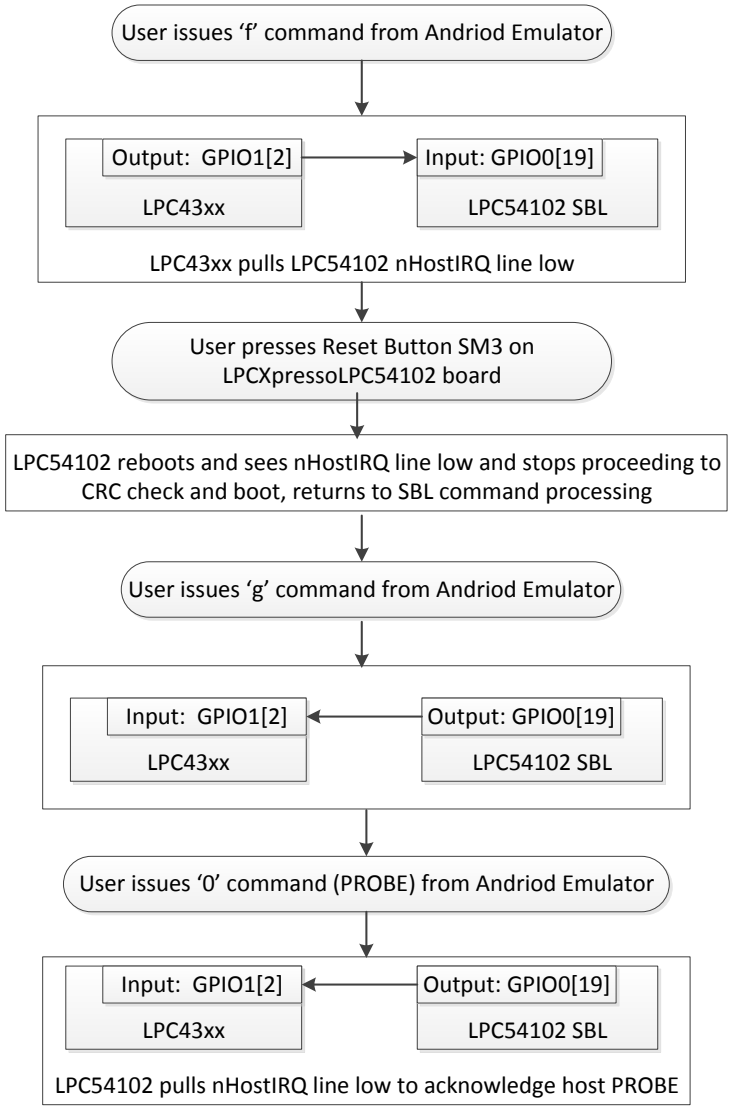


Fig 8. nHostIRQ line status when host wants to stop IMG_NORMAL image boot

With the Android emulator sample project attached with this application note, users can understand this handshaking process by following the procedure shown in Fig 9:

1. Program the sample application image (any of the Keil/IAR or LPCXpresso version).
2. Press the Reset button to boot the application image.
3. Issue ‘f’ command to pull nHostIRQ low.
4. Press the Reset button to reset the LPCXpresso54102 board.
5. Issue ‘g’ command to program nHostIRQ as input.
6. Now the user can issue ‘0’ command to re-establish the communication with the SBL and can optionally program a new image as shown in Fig 9.

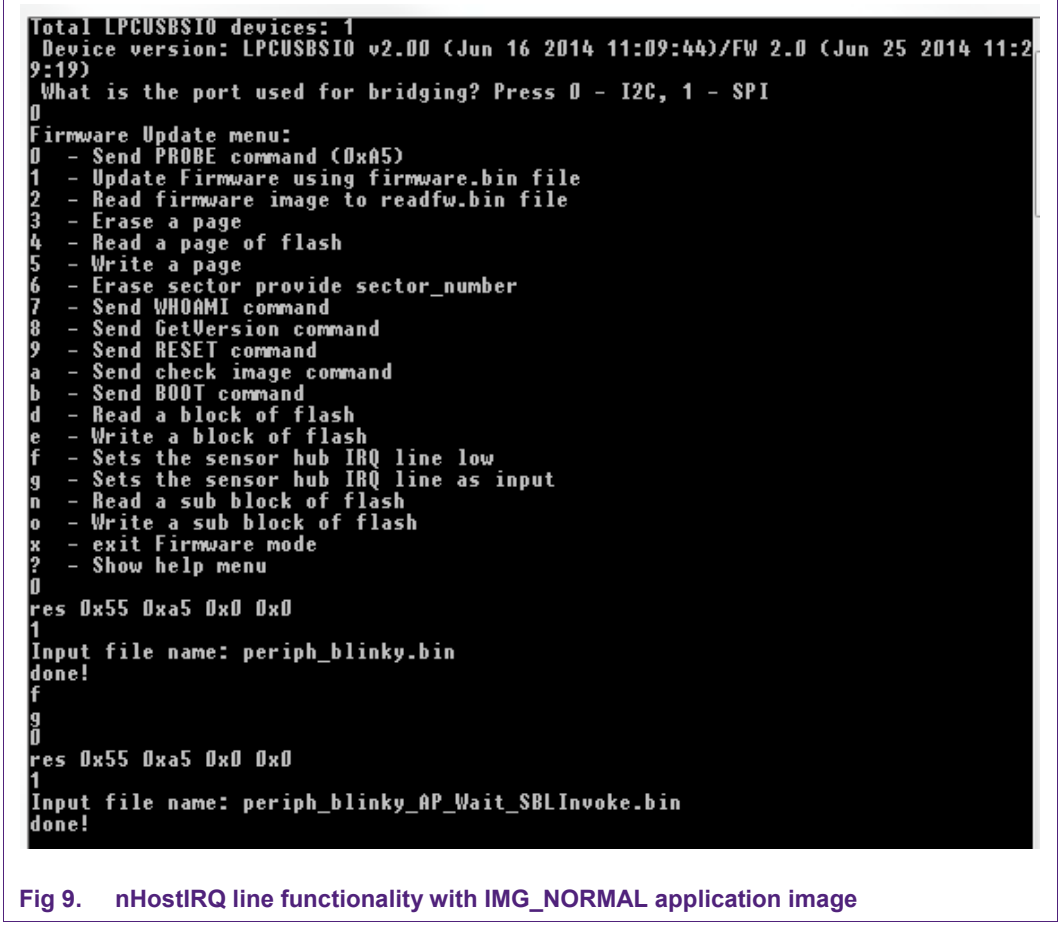


Fig 9. nHostIRQ line functionality with IMG_NORMAL application image

During the LPC5410x boot time, the SBL responds transparently to Android emulator commands ‘f’ and ‘g’. No data is sent to the SBL and no response data is received from the SBL. As shown in Fig 3, pulling the IRQ line from the AP must be done with IMG_NORMAL application image at LPC5410x boot time, right after the SBL has identified an IMG_NORMAL image.

4.1.2 Handling the CRC field in the image header

Fig 3 flow chart illustrates how the SBL supports CRC checking on the application image before passing control to the application. A CRC generating tool loaderCrcGen.exe utility (loaderCrcGen.zip) is included with this application note that can generate the CRClen field and the CRCValue field in the image header.

After generating the normal boot image, the user can run this tool to generate the binary which automatically inserts the CRClen and CRCValue fields to the new binary.

This application generates the CRC32 image in the application booted by the loader. The CRC32 length field in bytes is generated at image offset 0x010C. The CRC32 value field in bytes is generated at image offset 0x0110. This tool will add zero filled padding bytes to the end of the image to get the alignment to 32-bits, if needed. The generated CRC32 value consists of the start of the image up to the specified CRC32 length. The CRC32 length can be the entire length of the file or the first 0x110 bytes, depending on the selected option. Generating a CRC32 on only the first 0x110 byte may help with multi-part binary images.

To run the tool, use the following syntax:

loaderCrcGen.exe <-n1 or -n2> input_file_binary_file output_file_binary_file

- Use option -n1 to generate check CRC32 for the entire file.
- Use option -n2 to generate check CRC32 for just the first 0x110 bytes of the file.

See Fig 10 for the process to generate the binary file containing the CRC field, xpresso_blinky_crc.bin from periph_blinky.bin.

Fig 10. Using the loaderCrCGen tool

Fig 11 shows that the resulting xpresso_blinky_crc.bin file has the CRClen field and the CRCValue file inserted at the correct location. Notice that if CRClen and CRCValue field generation is skipped, the default sample project has 0x00000000 for CRClen field and 0xFFFFFFFF for CRCValue field, which effectively bypasses the CRC checking steps.

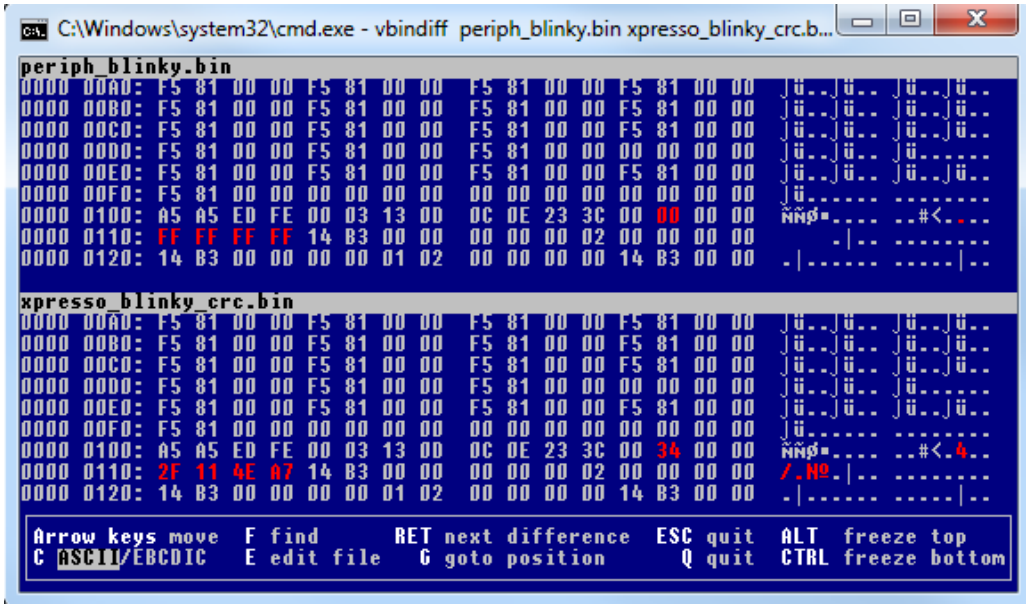


Fig 11. CRClen filed and CRCValue field inserted

4.2 Embedding SBL image header in application image

4.2.1 Embedding image header in application image with LPCXpresso IDE

To embed the header in an application image startup file based on LPCXpresso IDE, Fig 12 can be used as a reference.

```
#define IMAGE_HEADER_MARKER 0xFEEDA5A5
// imageType
#define IMG_NORMAL 0
#define IMG_AP_WAIT 1
#define IMG_NO_WAIT 2
#define IMG_NO_CRC 3
// ifType
#define I2C0_PORT 1
#define I2C1_PORT 2
#define I2C2_PORT 3
#define SPI0_PORT 4
#define SPI1_PORT 5
// irqPortPin
// IRQ port (bits 7:5) and pins (bits 4:0)
#define IRQ_PORT_PIN ((0 << 5) + 19) // P0_19
// misoPortPin
// SPI MISO port (bits 7:5) and pins (bits 4:0)
#define MISO_PORT_PIN ((0 << 5) + 13) // P0_13
// mosiPortPin
// SPI MOSI port (bits 7:5) and pins (bits 4:0)
#define MOSI_PORT_PIN ((0 << 5) + 12) //P0_12
// sselPortPin
// SPI SEL port (bits 7:5) and pins (bits 4:0)
#define SSEL_PORT_PIN ((0 << 5) + 14) //P0_14
// sckPortPin
// SPI SCK port (bits 7:5) and pins (bits 4:0)
#define SCK_PORT_PIN ((1 << 5) + 3) //P1_3
#define CHECKSUM_VAL (IMG_NORMAL ^ I2C2_PORT ^ IRQ_PORT_PIN ^
MISO_PORT_PIN ^ MOSI_PORT_PIN ^ SSEL_PORT_PIN ^ SCK_PORT_PIN)
#define PINCONFIG_WORD0         ((MISO_PORT_PIN << 24) | (IRQ_PORT_PIN <<
16) | (I2C2_PORT << 8) | IMG_NORMAL)
#define PINCONFIG_WORD1         ((CHECKSUM_VAL << 24) | (SCK_PORT_PIN
<< 16) | (SSEL_PORT_PIN << 8) | MOSI_PORT_PIN)
//*********************************************************************
```

```
// The vector table.
//*****************************************************************
extern void (* const g_pfnVectors[])(void);
__attribute__ ((section(".isr_vector")))
void (* const g_pfnVectors[])(void) = {
   // Core Level – CM4
      &_vStackTop,          // The initial stack pointer
      ResetISR,             // The reset handler
      NMI_Handler,          // The NMI handler
      HardFault_Handler,    // The hard fault handler
      MemManage_Handler,    // The MPU fault handler
      BusFault_Handler,     // The bus fault handler
      UsageFault_Handler,   // The usage fault handler
      0,                    // Reserved
      0,                    // Reserved
      0,                    // Reserved
      0,                    // Reserved
…
      RIT_IRQHandler,       // RIT Timer
      Reserved41_IRQHandler,  // Reserved
      Reserved42_IRQHandler,  // Reserved
      Reserved43_IRQHandler,  // Reserved
      Reserved44_IRQHandler,  // Reserved
     0,
      0,
      0,
      /* SBL Image header */
      (void*)IMAGE_HEADER_MARKER,       //Image header marker
      (void*)PINCONFIG_WORD0,     //PINCONFIG word0
      (void*)PINCONFIG_WORD1, //PINCONFIG word1
      (void*)0x00000000,          // Image length
      (void*)0xFFFFFFFF,          // CRC32 of the image
}; /* End of g_pfnVectors */
```

Fig 12. Image Header construction with LPCxpresso IDE (cr_startup_lpc5410x.c)

4.2.2 Embedding image header in application image with Keil and IAR IDE

To embed the header in the application image, the startup file based on Keil/IAR IDE can be used as a reference. See Fig 13.

```
; Select number of reserved 0 fields to force PINCFGTABLEFLASH to align
; at offset 0x100
                DCD     0, 0, 0
; Start of in-FLASH pin configuration table (offset 0x100)
                EXPORT  PINCFGTABLEFLASH
PINCFGTABLEFLASH
                DCD     0xFEEDA5A5    ; Image header marker
                  ; img_type: 0 = Normal image check IRQ line to halt boot
                  ; img_type: 1 = Wait for AP to send SH_CMD_BOOT command
                  ; img_type: 2 = Boot image with no AP checks
                  ; img_type: 3 = No CRC or AP checks needed. Used during development
                EXPORT PINONLYCFGTABLEFLASH
PINONLYCFGTABLEFLASH
                DCB     0       ; img_type: See img_type values above
                DCB     3       ; ifSel: Interface selection for host (0=AUTODETECT, 1=I2C0,
2=I2C1, 3=I2C2, 4=SPI0, 5=SP1) choice of 0 does not seem to exist in the Xpresso IDE
code???
                DCB     ((0 << 5) + 19); hostIrqPortPin: Host IRQ port (bits 7:5) and pins (bits 4:0)
                DCB     ((0 << 5) + 13); hostMisoPortPin: SPI MISO port (bits 7:5) and pins (bits
4:0)
                DCB     ((0 << 5) + 12); hostMosiPortPin: SPI MOSI port (bits 7:5) and pins (bits
4:0)
                DCB     ((0 << 5) + 14); hostSselPortPin: SPI SEL port (bits 7:5) and pins (bits
4:0)
                DCB     ((1 << 5) + 3); hostSckPortPin: SPI SCK port (bits 7:5) and pins (bits 4:0)
                DCB     0^3^((0 << 5) + 19)^((0 << 5) + 13)^((0 << 5) + 12)^((0 << 5) + 14)^((1 <<
5) + 3)
                EXPORT CRC32_LEN
                EXPORT CRC32_VAL
CRC32_LEN     DCD       0x00000000
                          CRC32_VAL             DCD       0xFFFFFFFF
```

Fig 13. Image Header Construction with Keil/IAR IDE (reference keil_startup_lpc5410x.s/iar_startup_lpc5410x.s)

4.3 Host interface pins

The SBL checks the pin configuration block (offset 0x4 to 0xB of the image header) using XOR files. Apart from XOR checking SBL also checks if valid port-pin is set for the corresponding pins.

The port-pin value is an 8 bit field with the lower 5 bits representing the pin number and the upper 3 bits representing the port number.

4.4 IDE Handling

This application note includes a blinky project which is implemented with Keil, IAR and LPCXpresso IDE.

4.4.1 Locate the application to 0x00008000 in flash

The toolchain links the image to run at address 0x00000000 by default. This must be changed to address 0x00008000.

- **LPCXpresso IDE**

Adjust the LPCXpresso project MCU settings to locate the application's RO base at 0x00008000 as shown in Fig 14.

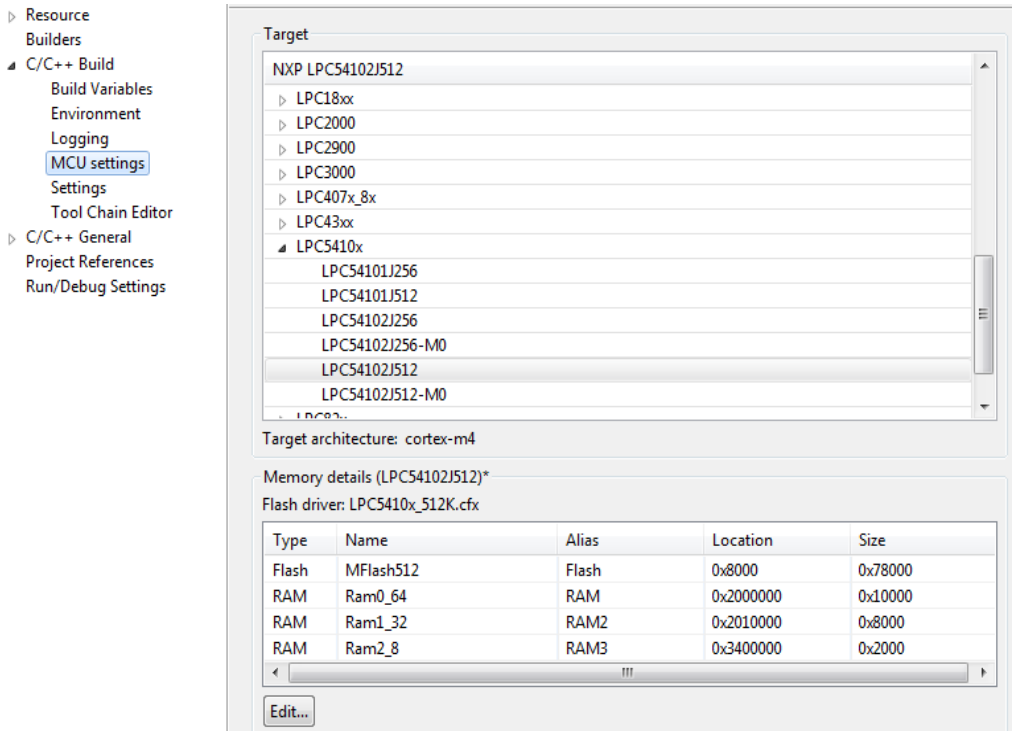


Fig 14. Locate the application to 0x00008000 in flash with LPCXpresso IDE

- **Keil uVision IDE**

Adjust the Keil project MCU settings to locate the application's RO base at 0x00008000 as shown in Fig 15. This step can alternatively be done with the Keil scatter file.

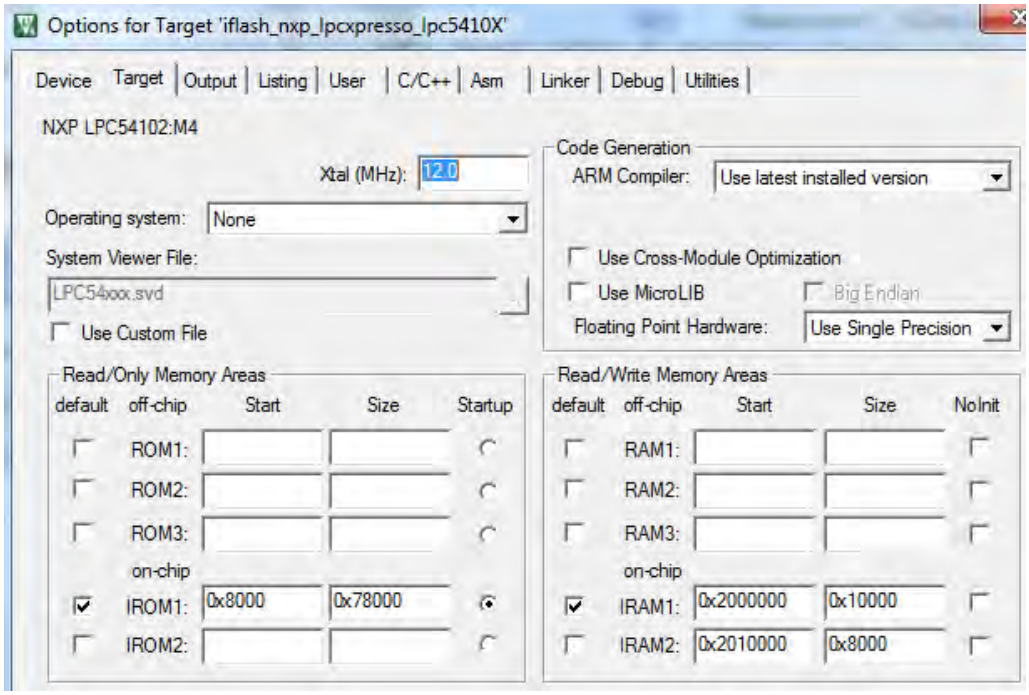


Fig 15. Locate the application to 0x00008000 in flash with Keil IDE

- **IAR EWARM IDE**

Included with the sample IAR project is a linker configuration file (ICF file) for correctly setting up the application address to boot from the SBL. This file is called "LPC5410x_SBL.icf".

Open the project options for the IAR project. These can be opened by right clicking on the project in the project browser and then selecting "Options…"

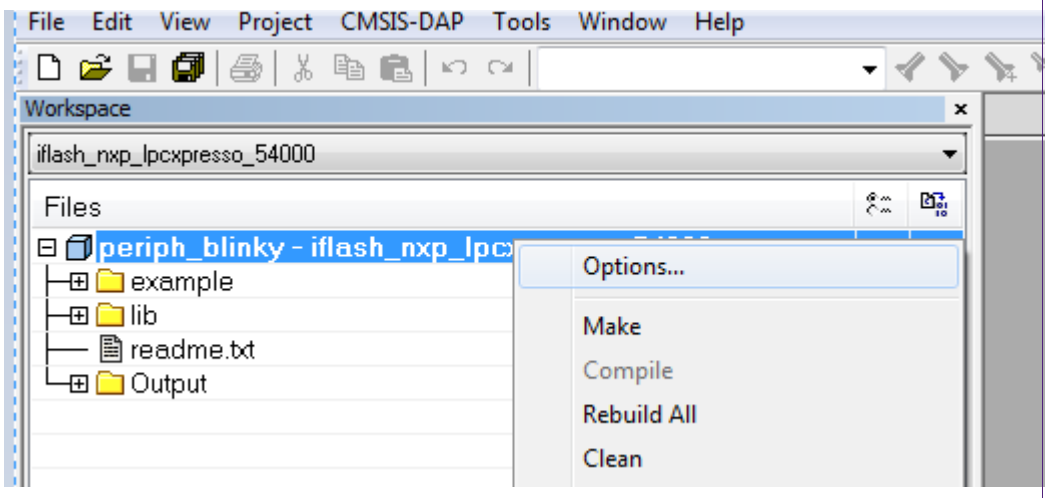


Fig 16. IAR project Option dialog invocation

Select the Linker category and the Config tab. Check the “Override default” box for the Linker Configuration File. In the file path box below it, click the ‘…’ box and then locate the copied “**LPC5410x_SBL.icf**” file. Select this file for the Linker Configuration File.

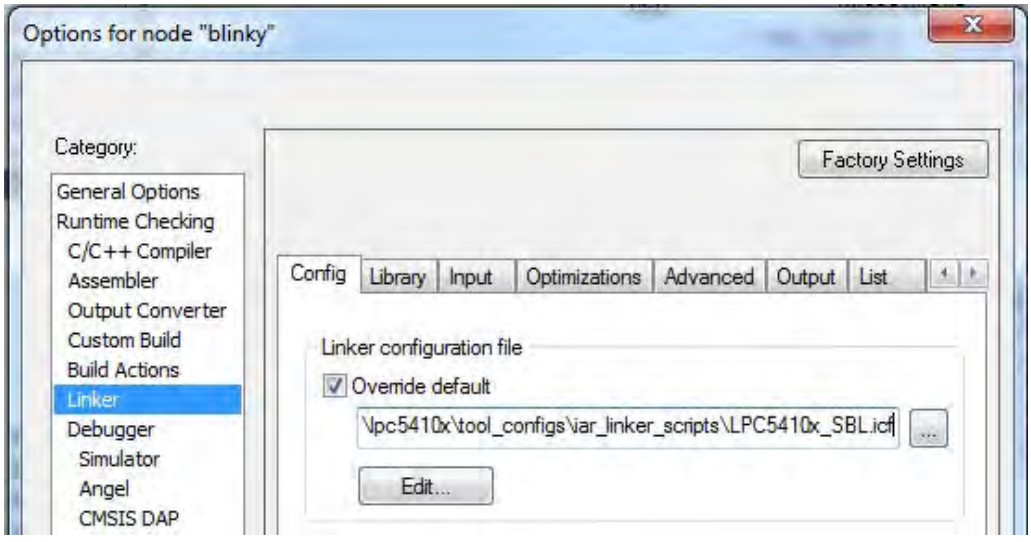


Fig 17. Select to use LPC5410x_SBL.icf with IAR IDE

Fig 18 is the content of the LPC5410x_SBL.icf file.

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00008000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x00008000;
define symbol __ICFEDIT_region_ROM_end__   = 0x00078000;
define symbol __ICFEDIT_region_RAM_start__ = 0x02000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x0200FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x800;
/**** End of ICF editor section. ###ICF###*/
define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to
__ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to
__ICFEDIT_region_RAM_end__];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
//initialize by copy with packing = none { section __DLIB_PERTHREAD }; // Required in a multi-
threaded application
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

Fig 18. LPC5410x_SBL.icf for IAR IDE

4.5 Generating binary application image to download via SBL

The SBL expects an application binary image to download.

4.5.1 LPCXpresso IDE

Open the Properties dialog of the blinky project, select the Settings option and click the Build steps tab. Enable the following post build command as shown in Fig 19:

arm-none-eabi-objcopy -v -O binary "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin"

 # checksum -p ${TargetChip} -d "${BuildArtifactFileBaseName}.bin"

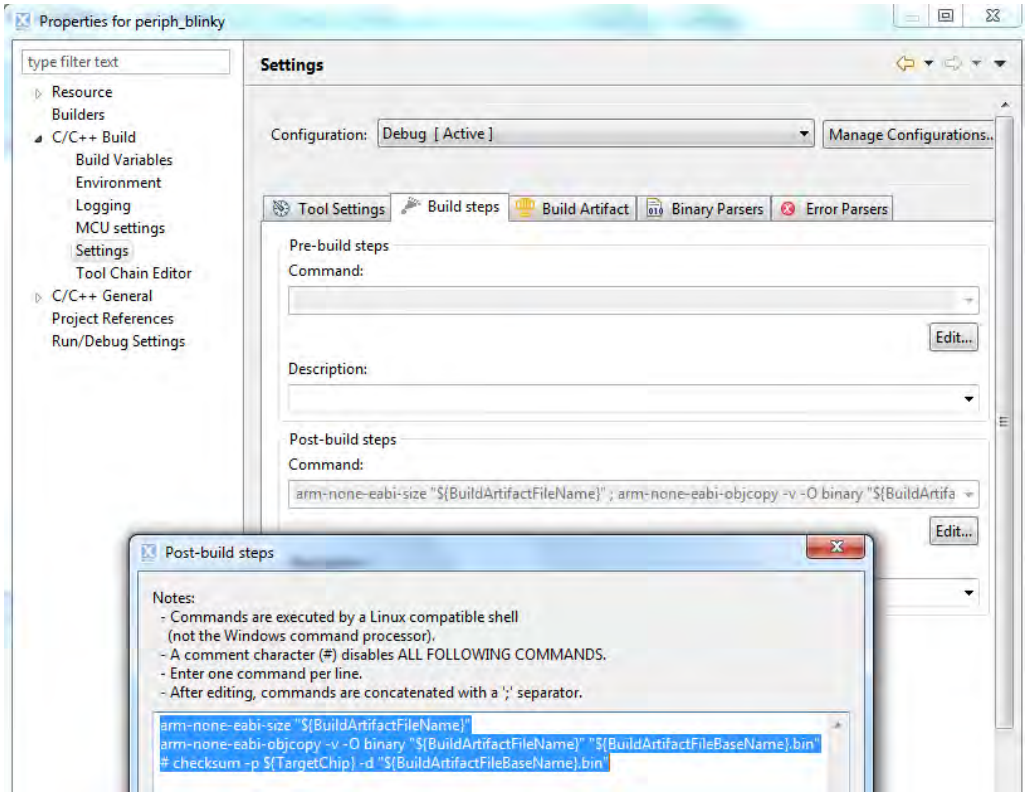


Fig 19. Generating .bin output from LPCXpresso IDE

4.5.2 Keil uVision IDE

Add this user action to instruct the Keil IDE to generate a binary application image as shown in Fig 20 Run #2:

$K\ARM\ARMCC\bin\fromelf.exe --bin --output=@L.bin !L

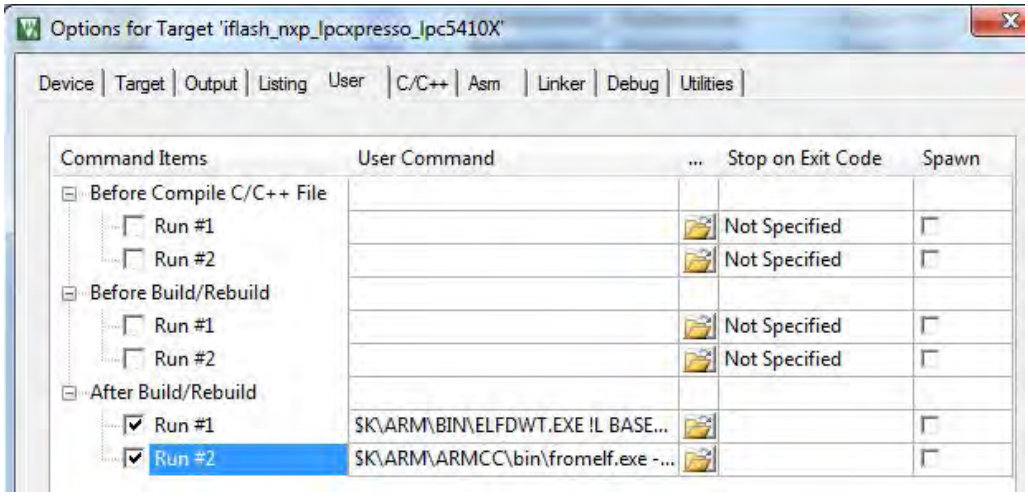


Fig 20. Generating .bin output from Keil IDE

4.5.3 IAR EWARM IDE

From the IAR project Options dialog, select the Output Converter category. Then check the “Generate additional output” box and select “binary” for the Output format.

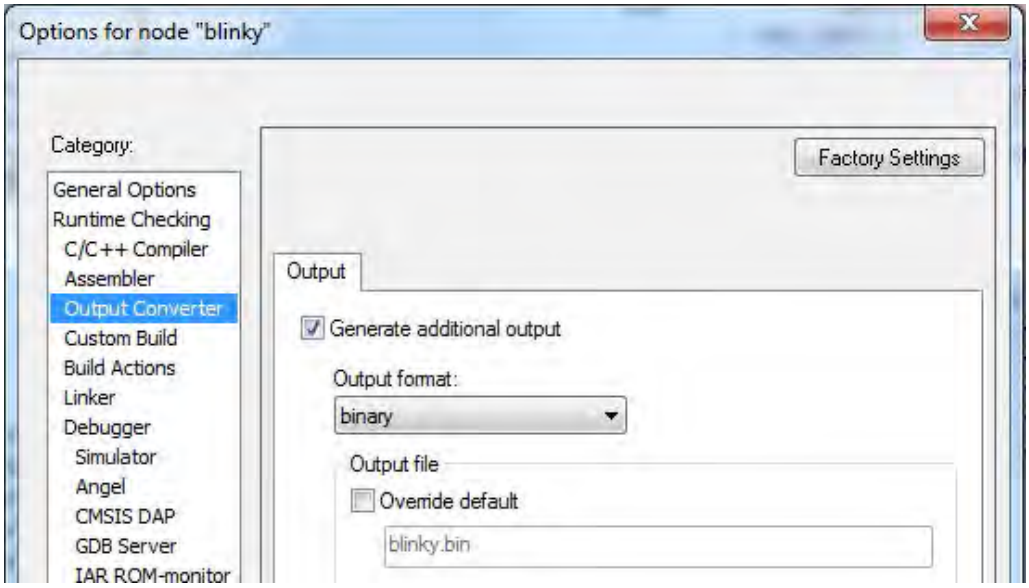


Fig 21. Generating .bin output from IAR IDE

When these settings are adjusted, compile the project to generate periph_blinky.bin that allows booting from the SBL.

5. Sample projects for firmware update

The host processor should send flash programming commands to update the firmware. This sequence of commands must be sent to the LPC5410x to program a new application image at location 0x00008000 onwards in flash:

1. Send **SH_CMD_PROBE (0xA5)** command with appropriate parameters. Note that the host I/F pin configuration information passed should match the board.
2. Wait for acknowledgement from LPC5410x. The acknowledgement is indicated by pulling nHostIRQ line low.
3. Once nHostIRQ line is low, AP should read the response data which is 4 bytes long.
4. Read firmware binary file and start programming to flash at address 0x8000 onwards by sending **SH_CMD_WRITE_BLOCK (0xA6)** command.
5. Wait for positive response.
6. Repeat steps 4 and 5 until the whole image is written to flash. During each iteration, the block number parameter field is incremented. Note that the very first SH_CMD_WRITE_BLOCK command should have block number parameter set to 64.

This firmware update process can be emulated using the attached Android Emulator program. As shown in Fig 22, after initiating the Android emulator, the user selects

whether to communicate with the SBL through I2C or SPI. In this case the SPI interface has been chosen.

The following IDEs were used to implement and test the sample projects shipped with this application note:

- Keil 5.12/5.13
- IAR 7.30
- LPCXpresso 7.62

In the example in Fig 22, command “1” updates the application firmware blinky_IAR.bin with an image type of IMG_NORMAL. Subsequently, command ‘b’ boots this application.

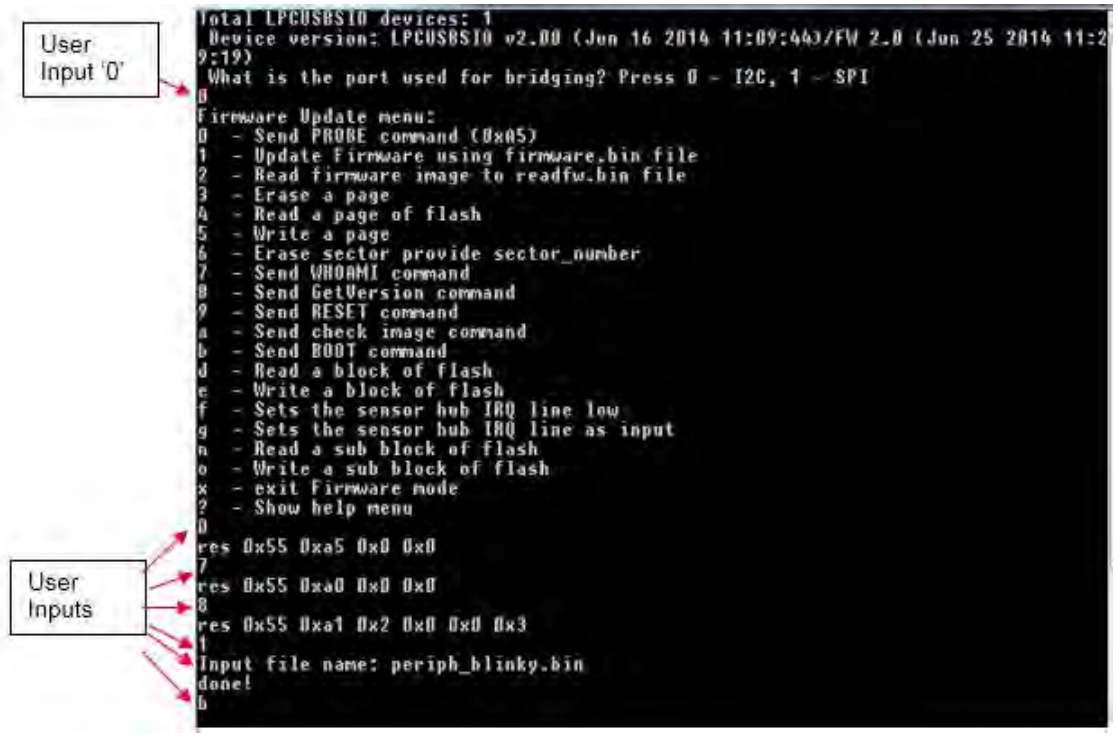


Fig 22. Running the Android Emulator

5.1 Factory programming

During factory assembly, the LPC5410x part has no application image and therefore remains in host interface detection mode. Once the part receives the SH_CMD_PROBE (0xA5), SBL enters command processing loop to receive flash programming commands.

5.2 Invoking SBL from user application

To perform a firmware update the user application can invoke the SBL by calling the function pointer present at address 0x00007F00. The pin configuration information is same as the one present in image header. Fig 23 shows a sample code of the invocation method. Notice that by default the code to invoke SBL from application is commented out. The user needs to enable these lines to test the invoking functionality.

```
/** Structure used to setup host interface for the secondary loader. The SPI port/pin selections
for SSEL, SCK, MOSI, and MISO only need to be configured if ifSel is SL_SPI0 or SL_SPI1. */
typedef struct {
//   uint32_t header_marker;
     uint8_t img_type;        /*!< Image type (SL_IMAGE_T)  */
     uint8_t ifSel;          /*!< Interface selection for host (SL_IFSEL_T) */
     uint8_t hostIrqPortPin; /*!< Host IRQ port (bits 7:5) and pins (bits 4:0) */
     uint8_t hostMisoPortPin;   /*!< SPI MISO port (bits 7:5) and pins (bits 4:0) */
     uint8_t hostMosiPortPin;   /*!< SPI MOSI port (bits 7:5) and pins (bits 4:0) */
     uint8_t hostSselPortPin;   /*!< SPI SEL port (bits 7:5) and pins (bits 4:0) */
     uint8_t hostSckPortPin;         /*!< SPI SCK port (bits 7:5) and pins (bits 4:0) */
     uint8_t checksum;       /*!< Checksum. XOR of the remaining 7 bytes of this structure. */
} SL_PINSETUP_T;

/** Image header structure. All application images should define this structure at of offset 0x100.
 *   Adding it to the start-up file is recommended.
 */
typedef struct _IMG_HEADER_T {
     uint32_t header_marker;                    /*!< Image header marker should always be set to
0xFEEDA5A5 */
     SL_PINSETUP_T hostifPinCfg;                  /*!< Host interface pin configuration */
     const uint32_t crc_len;                   /*!< Image length or the length of image CRC check
should be done. For faster boot application could set a smaller length than actual image */
     const uint32_t crc_value;                 /*!< CRC vale  */
} IMG_HEADER_T;

#ifdef WIN32
typedef unsigned char bool;
#define STATIC static
#define INLINE
#endif

typedef bool (*InBootSecondaryLoader)(const SL_PINSETUP_T *pSetup);

/* Address of indtrect boot table */
#define SL_INDIRECT_FUNC_TABLE      (0x00007F00)

/* Placement addresses for app call flag and app supplied config daa
   for host interface pins. Note these addresses may be used in the
   startup code source and may need values changed there also. */
#define SL_ADDRESS_APPCALLEDFL      (0x02000000)
#define SL_ADDRESS_APPPINDATA       (0x02000004)

/* Function for booting the secondary loader from an application. Returns with false if the pSetup
strructure is not valid, or doesn't return if the loader was started successfully. */
STATIC INLINE bool bootSecondaryLoader(const SL_PINSETUP_T *pSetup)
{
     InBootSecondaryLoader SL, *pSL = (InBootSecondaryLoader *)
SL_INDIRECT_FUNC_TABLE;
     SL_PINSETUP_T *pAppPinSetup = (SL_PINSETUP_T *) SL_ADDRESS_APPPINDATA;

     *pAppPinSetup = *pSetup;

     SL = *pSL;
     return SL(pSetup);

}
```

Fig 23. Sl_protocol.h

sl_protocol.h file is included as part of this application note. The function is called via an indirect function pointer provided in the SBL code. Invocation of the function from the application is performed using the function below:

*bool bootSecondaryLoader(const SL_PINSETUP_T *pSetup);*

When called from the application, the loader uses the pin configuration for the host interface as defined by the application. The bootSecondaryLoader() function does not return response once the SBL has started. However, if the pin configuration passed to the function is invalid, the function returns false (0) status.

An example for using I2C2 as the host interface with P0.19 as the host interrupt line are provided with the sample projects attached with this application note.

```
#include "board.h"

SL_PINSETUP_T pinSetup;

    pinSetup.img_type = IMG_AP_WAIT;
    pinSetup.ifSel = SL_I2C2;                                   /* Uses I2C2 as the host interface */
    pinSetup.hostIrqPortPin =  ((0 << 5) + 19);/* Host IRQ on port0 */
    /* hostMisoPortPin, hostMosiPortPin, hostSselPortPin, hostSckPortPin pins do not matter
when using I2C */
    pinSetup.hostMisoPortPin = ((0 << 5) + 13);
    pinSetup.hostMosiPortPin = ((0 << 5) + 12);
    pinSetup.hostSselPortPin = ((0 << 5) + 14);
    pinSetup.hostSckPortPin = ((1 << 5) + 3);
    pinSetup.checksum = pinSetup.img_type ^ pinSetup.ifSel ^          pinSetup.hostIrqPortPin
^ pinSetup.hostMisoPortPin ^ pinSetup.hostMosiPortPin ^ pinSetup.hostSselPortPin ^

    pinSetup.hostSckPortPin;

/*
    * Notice that by default the code to invoke SBL from application is                commented
out. User needs to enable these lines to test invoking SBL from application functionality.
    */
#if 1
    for (i=0; i<30000000; i++);

   __disable_irq();
    /* invoke secondary loader from app. Will not return if successful */
        if (bootSecondaryLoader(&pinSetup) == false) {
                /* Handle error, pin setup table was invalid */
        }
        __enable_irq();

#endif
```

Fig 24. Invoke SBL from LPCXpresso application

```
#include "sl_protocol.h"
extern SL_PINSETUP_T PINONLYCFGTABLEFLASH;
 /*
    * Notice that by default the code to invoke SBL from application is commented out.   User needs to enable these lines to test invoking SBL from application functionality.
    */
   #if 1
   for (i=0; i<30000000; i++);
      __disable_irq();
     /* invoke secondary loader from app. Will not return if successful */
     if (bootSecondaryLoader(&PINONLYCFGTABLEFLASH) == false) {
           /* Handle error, pin setup table was invalid */
     }
     __enable_irq();
#endif
```

Fig 25. Invoke SBL from Keil/IAR application

6. Secondary bootloader message

6.1 SH_CMD_WHO_AM_I (0xA0)

This command is used to identify the presence of SBL. It is recommended that applications should also handle this command but return response with different Start of Packet (SoP) value.

6.1.1 Command packet

Table 4. WHO_AM_I_Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA0	Command Identifier

6.1.2 Response packet

Table 5. WHO_AM_I Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of packet identifier
Command	0x1	1	0xA0	Command Identifier
length	0x2	2	0x0004	Length of the response packet

6.2 SH_CMD_GET_VERSION (0xA1)

This command is used to get the version information of SBL.

6.2.1 Command packet

Table 6. GET_VERSION Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA1	Command Identifier

6.2.2 Response packet

Table 7. GET_VERSION Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
length	0x2	2	0x0002	Length of the response packet.
Major	0x4	1	version	Major version number.
Minor	0x5	1	version	Minor version number.
Command	0x1	1	0xA1	Command Identifier.

6.3 SH_CMD_RESET (0xA2)

This command is used to have the AP do software reset on LPC5410x.

6.3.1 Command packet

Table 8. RESET Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA2	Command Identifier

6.3.2 Response packet

None

6.4 SH_CMD_BOOT (0xA3)

This command is used to boot the application image. On receiving this command SBL cleans resource (disables clocks and IRQs) enabled by SBL and loads the value at location 0x0008000

6.4.1 Command packet

Table 9. BOOT Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA3	Command Identifier.

6.4.2 Response packet

None

6.5 SH_CMD_CHECK_IMAGE (0xA4)

This command is used to check if the flash has a valid application image. SBL checks the presence of “Image header” at offset 0x100 of the application image flashed from location 0x00008000. This command computes CRC32 checksum of the image (excluding Checksum field) based on the length field. If checksum matches, the command returns 0 as response data or else returns computed CRC32.

6.5.1 Command packet

Table 10. CHECK_IMAGE Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA3	Command Identifier.

6.5.2 Response packet

Table 11. CHECK_IMAGE Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA4	Command Identifier.
length	0x2	2	0x0008	Length of the response packet.
CRC32	0x4	4	CRC32	0 – If checksum value present in image header matches the computed value.

6.6 SH_CMD_PROBE (0xA5)

This command sends the Host interface type and port, the HOSTint_nirq line configuration, the SPI pin configuration as well as an XOR checksum byte of this information to the SBL. The host needs to repeatedly send this command until a proper response is received from the SBL.

6.6.1 Command packet

Table 12. PROBE Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA5	Command Identifier.
ifType	0x1	1		Host interface type and port. 1 – I2C0 port 2 – I2C1 port 3 – I2C2 port

Field	Offset	Size (bytes)	Value	Description
				4 – SPI0 port 5 – SPI1 port
irqPortPin	0x2	1		GPIO pin used for Host interface IRQ function. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number
misoPortPin	0x3	1		MISO pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
mosiPortPin	0x4	1		MOSI pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
sselPortPin	0x5	1		SSEL pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
sckPortPin	0x6	1		SCK pin used for SPI host interface. Bits 7-5 : GPIO port number Bits 4-0 : GPIO pin number For applications using I2C host interface set this field to 0.
checksum	0x7	1		XOR of all the 7 bytes above.

6.6.2 Response packet

Table 13. PROBE Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA5	Command Identifier.
length	0x2	2	0x0000	Length of the response packet.

6.7 SH_CMD_WRITE_BLOCK (0xA6)

This command allows writing a block of flash. In current version of SBL the flash block size is 512 bytes.

6.7.1 Command packet

Table 14. WRITE_CLOCK Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA6	Command Identifier.
crcCheck	0x1	1		0 – Do CRC check for this packet. 1 - Ignore CRC field for this packet.
blockNum	0x2	2		Flash block number in which the appended data to be programmed. For example to program flash block 0x8000 this parameter should be set to 64.
data	0x4	512		Data to be programmed in flash.
checksum	0x204	4		CRC32 of the packet excluding this field. Set this field to 0 if *crcCheck* is set to 1.

6.7.2 Response packet

Table 15. WRITE_CLOCK Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA6	Command Identifier.
length	0x2	2	0x0000 0x0004	On Success this field is set to 0. On error this field is set to 4.
errorCode	0x4	4		Error code. 0x0001001: Invalid parameters

6.8 SH_CMD_READ_BLOCK (0xA7)

This command allows reading a block of flash. In current version of SBL the flash block size is 512 bytes.

6.8.1 Command packet

Table 16. READ_BLOCK Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA7	Command Identifier.
reserved	0x1	1	0x00	Should be zero.

Field	Offset	Size (bytes)	Value	Description
blockNum	0x2	2		Flash block number in which the appended data to be programed.

6.8.2 Response packet

Table 17. READ_BLOCK Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA7	Command Identifier.
length	0x2	2	0x0208	Length of the response packet.
data	0x4	512	version	Flash block content.
checksum	0x204	4	Crc32	CRC32 of the packet excluding this field.

6.9 SH_CMD_SECTOR_ERASE (0xA8)

This command allows erasing sector of flash. On LPC5410x the flash sector size is 32Kbytes.

6.9.1 Command packet

Table 18. SECTOR_ERASE Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA8	Command Identifier.
reserved	0x1	1	0x00	Should be zero.
blockNum	0x2	2		Flash sector number to be erased.

6.9.2 Response packet

Table 19. SECTOR_ERASE Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA8	Command Identifier.
length	0x2	2	0x0000 0x0004	On Success this field is set to 0. On error this field is set to 4.
errorCode	0x4	4		Error code. 0x0001001: Invalid parameters

6.10 SH_CMD_PAGE_ERASE (0xA9)

This command allows erasing a flash page. On LPC5410x the flash page size is 256 bytes.

6.10.1 Command packet

Table 20. PAGE_ERASE Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xA9	Command Identifier.
reserved	0x1	1	0x00	Should be zero.
pageNum	0x2	2		Flash page number to be erased.

6.10.2 Response packet

Table 21. PAGE_ERASE Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xA9	Command Identifier.
length	0x2	2	0x0006	Length of the response packet.
errorCode	0x4	4		Error code. 0x0001001: Invalid parameters

6.11 SH_CMD_PAGE_WRITE (0xAA)

This command allows writing a block of flash. In current version of SBL the flash block size is 256 bytes.

6.11.1 Command packet

Table 22. PAGE_WRITE Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xAA	Command Identifier.
crcCheck	0x1	1		0 – Do CRC check for this packet. 1 - Ignore CRC field for this packet
pageNum	0x2	2		Flash page number in which the appended data to be programed. For example to program flash page at 0x8000 this parameter should be set to 128.
data	0x4	256		Data to be programed in flash.
checksum	0x204	4		CRC32 of the packet excluding this field. Set this field to 0 if *crcCheck* is set to 1.

6.11.2 Response packet

Table 23. PAGE_WRITE Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xAA	Command Identifier.
length	0x2	2	0x0000 0x0004	On Success this field is set to 0. On Error this this field is set to 4.
errorCode	0x4	4		Error code. 0x00010001: Invalid parameters

6.12 SH_CMD_PAGE_READ (0xAB)

This command allows reading a page of flash. On LPC5410x the flash page size is 256 bytes.

6.12.1 Command packet

Table 24. PAGE_READ Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xAB	Command Identifier.
reserved	0x1	1	0x00	Should be zero.
pageNum	0x2	2		Flash page number from which the data to be read. For example to program flash page at 0x8000 this parameter should be set to 128.

6.12.2 Response packet

Table 25. PAGE_READ Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xAB	Command Identifier.
length	0x2	2	0x0108 0x0004	On Success this field is set to 256. On error this field is set to 4.
data	0x4	256	*version*	Flash page content.
checksum	0x104	4	*Crc32*	CRC32 of the packet excluding this field.

6.13 SH_CMD_WRITE_SUBBLOCK (0xAC)

This command allows writing a block of flash but by transferring small chunks of data. In current version of SBL the flash block size is 512 bytes. But some host processors cannot send/receive more than 256 bytes in single I2C transaction. Hence this command

support is added so that the host can split the block into multiple sub-block packets and send to LPC5410x.

The LPC5410x collects all sub-blocks before writing to the flash. Hence, the host should send the sub-blocks in sequential order only. If any other command is sent in between the sub-block commands the offset in the collection buffer is reset to 0.

If block number offset falls on a sector boundary the SBL will erase the sector (32KB) first before programming the block.

6.13.1 Command packet

Table 26. WRITE_SUBBLOCK Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xAC	Command Identifier.
subBlock	0x1	1		Bit 0: If set CRC check is not done for this packet. The checksum field should be set to Bits [5:1]: Specifies the sub-block number. Bits [7:6]: Specifies the sub-block size. 00 – 32 bytes 01 – 64 bytes 10 – 128 bytes 11 – 256 bytes
blockNum	0x2	2		Flash block number in which the appended data to be programed. For example to program flash page at 0x8000 this parameter should be set to 64.
data	0x4	Sub-block size		Data to be programed in flash.
checksum	Sub-block size + 4	4		CRC32 of the packet excluding this field. Set this field to 0 if crcCheck is set to 1.

6.13.2 Response packet

Table 27. WRITE_SUBBLOCK Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xAC	Command Identifier.
length	0x2	2	0x0000 0x0004	On Success this field is set to 0. On error this field is set to 4.
ErrorCode	0x4	4		Error code.

Field	Offset	Size (bytes)	Value	Description
				0x0001001: Invalid parameters

6.14 SH_CMD_READ_SUBBLOCK (0xAD)

This command allows reading of a sub-block of flash. In current version of SBL the flash block size is 512 bytes. But some host processors cannot send/receive more than 256 bytes in single I2C transaction. To support those host processors these sub-block commands are added to SBL.

6.14.1 Command packet

Table 28. READ_SUBBLOCK Command packet

Field	Offset	Size (bytes)	Value	Description
Command	0x0	1	0xAD	Command Identifier.
subBlock	0x1	1		Bit 0: If set CRC check is not done for this packet. The *checksum* field should be set to Bits [5:1]: Specifies the sub-block number. Bits [7:6]: Specifies the sub-block size. 00 – 32 bytes 01 – 64 bytes 10 – 128 bytes 11 – 256 bytes
blockNum	0x2	2		Flash block number in which the appended data to be programed. For example to program flash page at 0x8000 this parameter should be set to 64.

6.14.2 Response packet

Table 29. READ_SUBBLOCK Response packet

Field	Offset	Size (bytes)	Value	Description
SoP	0x0	1	0x55	Start of Packet identifier.
Command	0x1	1	0xAD	Command Identifier.
length	0x2	2	(*Sub-block size + 4)* 0x0004	On Success this field is set to (*Sub-block size + 4)*. On error this field is set to 4.
data	0x4	*Sub-block size*	*version*	Flash content.

Field	Offset	Size (bytes)	Value	Description
checksum	*Sub-block size + 4*	4	*Crc32*	CRC32 of the packet excluding this field.

7. Conclusion

This application note provides a reference design for firmware updating of the LPC5410x in a host/slave processor environment. The user can reference this application note to easily customize their own host system and the application.

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.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

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 Licenses

Purchase of NXP <xxx> components
<License statement text>

8.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

8.5 Trademarks

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

<Name> — is a trademark of NXP Semiconductors N.V.

9. List of figures

10. Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

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

Date of release: 8 April 2015
Document identifier: AN11610