Abstract
This application note describes the design and implementation of a secondary bootloader which can update the user application code in on-chip flash via UART with 1K XMODEM protocol, SD/MMC with file system, EEPROM with I2C interface and CAN interface using IAP (In-Application Programming).
Revision history

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>20090526</td>
<td>Initial version</td>
</tr>
</tbody>
</table>

Contact information

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

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

In most LPC2000 devices, the primary bootloader is the firmware which resides in the Boot Block and is executed every time the part is powered on or reset. The secondary bootloader in this document refers to a user-defined application that provides the user with an option to update the code or execute the previously programmed code.

In Application Programming (IAP executes erase and write operations on the on-chip flash memory, as directed by the end-user application code.

Code update is a typical application of IAP. This document describes four secondary bootloaders using different interfaces (UART using the 1K XMODEM protocol, SD/MMC using a FAT file system, I2C interfaced to EEPROM and CAN).

2. LPC2000 flash programming

2.1 Sector description

The IAP commands operate on a sector-by-sector basis. This means that in order to make any modifications (even if it is just one byte) in a particular sector, the entire sector must be erased.

The user application and the secondary bootloader share the same on-chip flash space. This means that the user’s application code should not be placed inside any of the sectors on which the bootloader resides. Therefore, the application should be erased and programmed in sectors – separate from the bootloader.

IAP, ISP, and RealMonitor routines are located in the primary bootloader (Boot block). The boot block is present at addresses 0x0007 E000 to 0x0007 FFFF (8 kB) in all devices. Depending on the device, not all of the flash is available to the user. This is the case if the part contains 512 kB of flash. Rather than having 512 kB available, the user will only have 504 kB available for the application. In devices that have less than 512 kB flash available, please refer to the respective user manuals.

**Fig 1** indicates the correspondence between sector numbers and memory addresses for LPC23xx devices containing 128, 256 and 512 kB of flash respectively. For other LPC2000 devices please refer to the user manual.
2.2 In Application Programming (IAP)

2.2.1 IAP introduction

A boot code 8 kB in size is programmed into the on-chip flash after factory. The boot code controls initial operation after reset and also provides the means to accomplish programming of the flash memory. This could be initial programming of a blank device, erasure and re-programming of a previously programmed device, or programming of the Flash memory by the application program in a running system.

In Application Programming (IAP) executes erase and write operations on the on-chip flash memory, as directed by the end-user application code.

2.2.2 IAP application

Using IAP, users can update the application code by various communication interfaces such as UART, USB or Ethernet. Flash sectors that aren’t used for the secondary bootloader or the user application may be used as non-volatile data storage.

While the application is running, the user can update some portion of the code using IAP commands which we call “online code updates”. It is not necessary to power off or even to remove the chip from the board to have it serviced by some commercial programming tools.

Having the device work as data storage, PCB costs and sizes can be reduced. Caution is however advised in handling data storage sectors. Since sectors functioning as data storage can be erased, no application code should be contained in these sectors.

Frequent erasing and programming will reduce the flash’s lifecycle. The LPC2000 provides a minimum of 10,000 write/erase cycles and 10 years of data retention.

2.2.3 IAP commands

For in application programming, the IAP routine should be called using a word sized pointer that has been loaded into register r0, which is pointing to memory (on-chip RAM).
containing the command code and its parameters. Register r1 contains the results of the IAP command returned by a pointer (pointing to a table). The user can reuse the command table for the results by passing the same pointers into registers r0 and r1. Ensure that the results table is large enough to store all the results coming from the IAP command issued. Refer to Fig 2.

![Fig 2. IAP parameter passing](image)

The IAP commands and codes are listed in Fig 3.

<table>
<thead>
<tr>
<th>IAP Command</th>
<th>Command Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare sector(s) for write operation</td>
<td>5010</td>
</tr>
<tr>
<td>Copy RAM to flash</td>
<td>5110</td>
</tr>
<tr>
<td>Erase sector(s)</td>
<td>5210</td>
</tr>
<tr>
<td>Blank check sector(s)</td>
<td>5310</td>
</tr>
<tr>
<td>Read Part ID</td>
<td>5410</td>
</tr>
<tr>
<td>Read Boot code version</td>
<td>5510</td>
</tr>
<tr>
<td>Compare</td>
<td>5610</td>
</tr>
<tr>
<td>Reinvoke ISP</td>
<td>5710</td>
</tr>
</tbody>
</table>

![Fig 3. IAP command summary](image)

For detailed information, please refer to the LPC2000 user manual.

2.2.4 Using IAP

Fig 4 shows the necessary steps to perform the flash programming using IAP.
2.2.4.1 Define system parameters

Some constants such as system clock, IAP entry point, input and output buffers should be defined before an IAP call.

```c
#define IAP_CLK Fcclk
#define IAP_LOCATION 0x7FFFFFF1
#define iap_entry(a, b) ((void (*)())(IAP_LOCATION))(a, b)

unsigned long command[5] = {0, 0, 0, 0, 0};
unsigned long result[3] = {0, 0, 0};
```

2.2.4.2 Select sector

The sectors have to be selected before any erase or programming operation. More than one sector can be selected.
2.2.4.3 Erase sector

Like other Flash implementations, the LPC2000 on-chip should be erased before programming. However, if the target sector is already erased, it is not necessary to erase the sector again. More than one sector can be erased at a time.

2.2.4.4 Program sector

During this stage, the data will be programmed from on-chip RAM to Flash.

Note:
1. Data can only be programmed from on-chip SRAM to on-chip Flash.
2. The address in on-chip Flash should be on a 256 byte boundary.
3. On-chip RAM should be located on the local bus which means neither USB SRAM nor Ethernet SRAM can be used.

4. Number of programmed bytes each time should be 256, 512, 1024 or 4096.

```c
unsigned long IAP_CopyRAMToFlash(unsigned long dst, unsigned long src, unsigned long number)
{
    command[0] = IAP_CMD_CopyRAMToFlash;
    command[1] = dst;
    command[2] = src;
    command[3] = number;
    iap_entry(command, result);
    return result[0];
}
```

### 2.2.4.5 Verify data

With this function, user does not have to write their own code to compare the data in RAM and Flash.

**Note:** Source address, destination address and number of bytes should be a word boundary or a multiple of 4.

```c
unsigned long IAP_Compare(unsigned long dst, unsigned long src, unsigned long number, unsigned long *offset)
{
    command[0] = IAP_CMD_Compare;
    command[1] = dst;
    command[2] = src;
    command[3] = number;
    iap_entry(command, result);
    if (result[0] == IAP_STA_COMPARE_ERROR)
        *offset = result[1];
    return result[0];
}
```

### 2.2.4.6 Interrupts during IAP

The on-chip flash memory is not accessible during erase/write operations. When the user application code starts executing, the interrupt vectors from the user flash area are active.
The user should either disable interrupts, or ensure that the user’s interrupt vectors reside and are active in RAM, before making a flash erase/write IAP call. The IAP code does not use or disable interrupts.

```c
__swi(0x00) void SwiHandle1(int Handle);
#define IRQDisable() SwiHandle1(0)
#define IRQEnable() SwiHandle1(1)

/*
 * prepare and erase the sectors with index from "start" to "end"
 * if successful, return TRUE, otherwise FALSE
 */
BYTE IAP_PrepareErase_Sector(DWORD start, DWORD end)
{
    DWORD IAP_status;
    BYTE result = FALSE;
    IRQDisable();
    IAP_status = IAP_PrepareSec(start, end);
    if (IAP_status == IAP_STA_CMD_SUCCESS)
    {
        IAP_status = IAP_EraseSec(start, end);
        if (IAP_status == IAP_STA_CMD_SUCCESS)
        {
            result = TRUE;
        }
    }
    IRQEnable();
    return result;
}
```

Fig 10. Interrupt handle during IAP

### 2.2.4.7 RAM used by IAP command handler

Flash programming commands use the top 32 bytes of on-chip RAM. The maximum stack usage in the user allocated stack space is 128 bytes (for LPC2300) and grows downwards.

### 2.2.4.8 Running the user application

After loading the application, the user can execute the application by modifying the PC register by pointing it to the starting address of the application code.

```c
#define AP_ADDR 0x8000 // where the user app located
typedef void (*FP)(void);

/*
 * run the user application from pre-specified address
 */
void Run_Application()
{
    FP fp;
    fp = (FP)AP_ADDR;
    (*fp)();
}
```

Fig 11. run the user application
3. Secondary bootloader and user application

3.1 Secondary bootloader

After power on or reset, the secondary bootloader will always run and display a menu on a PC terminal window (e.g., Teraterm). With this menu, the user can erase or program the internal flash. Depending on the bootloader selected the application code can be programmed using one of the four different interfaces (UART, SD Card, I²C, or CAN). The secondary bootloader resides in sectors starting at 0 and must not overlap the user application code.

![Bootloader memory map](image)

3.1.1 Bootloader Features and Constraints

The UART, SD/MMC, I²C, and CAN bootloaders function as a reference to allow the user to customize it to their particular needs. The bootloaders at their current state have the following design constraints and features:

- The bootloader’s default user application code starting address is 0x10000.
- The bootloader will always wait for user input before continuing on into the application code.
- The bootloader has no knowledge whether user code is present or where it is located at in flash.
- The SD/MMC bootloader requires a SD Card to be inserted for proper operation.
- The SD/MMC bootloader allows the user to select which binary file should be flashed onto the microcontroller.
- If the application code utilizes interrupts, it needs to maintain its own interrupt vector table in SRAM, see section 3.2.2.
- The user application’s starting address can be changed at compile time; however, it must match the address at which the code is to be programmed using the bootloader.

3.2 User application

3.2.1 Memory map

The user application is located in different sectors along with the secondary bootloader and consumes the on-chip RAM starting from 0x40000000.
If the user application utilizes interrupts, the first 64 bytes (0x40) at the bottom of the on-chip RAM should be used as the interrupt vector table. Refer to section 3.2.2. Fig 14 shows the memory configuration for the user application in Keil’s uVision 3 IDE. The on-chip RAM is from 0x40000040 instead of 0x40000000.

3.2.2 Interrupt vector table re-mapping

Because of the location of the interrupt vectors on the ARM7 processor (at addresses 0x0000 0000 through 0x0000 001C, as shown in Fig 15 below), a small portion of the Boot ROM and SRAM spaces need to be re-mapped in order to allow alternative uses of interrupts in the different operating modes described in Fig 16 below.
The portion of memory that is re-mapped allows for interrupt processing in different modes and includes the interrupt vector area (32 bytes) as well as an additional 32 bytes (a total of 64 bytes) to facilitate branching to interrupt handlers at distant physical addresses. The remapped code locations overlay addresses 0x0000 0000 through 0x0000 003F.

The interrupt vectors of the secondary bootloader occupies the physical address from 0x0000 0000 through 0x0000 003F in on-chip flash, thus the interrupt vectors for the user application has to be re-mapped to the bottom of the on-chip SRAM using “User RAM Mode”.

Before entering the main application, the user should copy the 64 bytes (32 bytes interrupt vector and 32 bytes additional bytes) into the bottom of the on-chip RAM and set the mapping mode to “User RAM Mode”.

Fig 17 shows how to copy the 64 bytes of interrupt vectors from on-chip flash to the bottom of on-chip RAM.

---

### Table: ARM exception vector locations

<table>
<thead>
<tr>
<th>Address</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 0000</td>
<td>Reset</td>
</tr>
<tr>
<td>0x0000 0004</td>
<td>Undefined Instruction</td>
</tr>
<tr>
<td>0x0000 0008</td>
<td>Software Interrupt</td>
</tr>
<tr>
<td>0x0000 000C</td>
<td>Prefetch Abort (instruction fetch memory fault)</td>
</tr>
<tr>
<td>0x0000 0010</td>
<td>Data Abort (data access memory fault)</td>
</tr>
<tr>
<td>0x0000 0014</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0000 0018</td>
<td>IRQ</td>
</tr>
<tr>
<td>0x0000 001C</td>
<td>FIQ</td>
</tr>
</tbody>
</table>

### Table: LPC2300 Memory mapping modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Activation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Loader mode</td>
<td>Hardware activation by any Reset</td>
<td>The Boot Loader always executes after any reset. The Boot ROM interrupt vectors are mapped to the bottom of memory to allow handling exceptions and using interrupts during the Boot Loading process. A sector of the Flash memory (the Boot Flash) is available to hold part of the Boot Code.</td>
</tr>
<tr>
<td>User Flash mode</td>
<td>Software activation by boot code</td>
<td>Activated by the Boot Loader when a valid User Program Signature is recognized in memory and Boot Loader operation is not forced. Interrupt vectors are not re-mapped and are found in the bottom of the Flash memory.</td>
</tr>
<tr>
<td>User RAM mode</td>
<td>Software activation by user program</td>
<td>Activated by a User Program as desired. Interrupt vectors are re-mapped to the bottom of the Static RAM.</td>
</tr>
<tr>
<td>User External Memory mode</td>
<td>Software activation by user code</td>
<td>Activated by a User Program as desired. Interrupt vectors are re-mapped to external memory bank 0.</td>
</tr>
</tbody>
</table>

Fig 15. ARM exception vector locations

Fig 16. LPC2300 Memory mapping modes
3.2.3 User application size

One key parameter needed to update the user application is the size of the application itself. When loading the user application from the SD/MMC or UART interface, the end and the size of the binary file can be easily determined.

CAN and I²C only handles the physical data transfers from the interface onto on-chip memory. Without an additional protocol, loading the application via CAN or I²C requires prior knowledge of the application size before it can be programmed into flash. The following figures show how to record the image size.

---

```assembly
AREA   RESET, CODE, READONLY

Vectors
LDR PC, Reset_Handler
LDR PC, Under_ADR
LDR PC, SWI_Handler
LDR PC, FIQ_Handler
LDR PC, Iirq_ADR

; Vector table re-mapping in user application (file lpc2300.s)

Fig 17. Vector table re-mapping in user application (file lpc2300.s)
Fig 18. Record user application image size in lpc2300.s

```c
#include "ram_intvec.h"

/* Record user application size in secondary bootloader */

area RESET, CODE, READONLY

ifdef SIZE_INT_INFO

import | Image$SR_ROM$| Image$SR_ROM$| Image$SR_ROM$| Image$SR_ROM$

endif

fig 19. Control symbols definition for lpc2300.s in user application

ifdef SIZE_INT_INFO

dc | Image$SR_ROM$| Image$SR_ROM$| Image$SR_ROM$

endif

ifdef SIZE_INT_INFO

dc | Image$SR_ROM$| Image$SR_ROM$

endif

Fig 20. Retrieve user application size in secondary bootloader

```
4. Demo description

4.1 Demo setup

4.1.1 Hardware setup

This demo is tested on the KEIL MCB2300 evaluation board (version 4.7). For more information about MCB2300, please refer to: http://www.nxp.com/redirect/keil.com/mcb2300/.

No other special hardware is needed except a RS-232 cable, Standard USB (A-B) cable (for power), and an EEPROM with I²C interface (if using the I²C bootloader). The RS-232 cable is for the connection between the PC and MCB2300 on COM1. The ULINK-ME JTAG module can be used to program the bootloader onto the board. Optionally, the FlashMagic tool can be used to program the bootloader using the RS-232 cable connected onto UART0 (COM0).

UART1 (COM1) Setting: 115200 baud, 8N1

The FlashMagic tool is available for free at: http://www.nxp.com/redirect/flashmagictool.com/

4.1.2 Software setup

Keil uVision3 and Flashmagic are used to edit, compile, link, debug and download the code. Tera term (or other tools) is used for serial communication between PC terminal and MCB2300 and configured at 115200 baud, 8-bits, no parity, 1 stop bit, XON/XOFF.

Because the IAP routines are run in thumb code, the bootloaders need to have ARM/Thumb interworking enabled. Fig 21 shows the configuration in Keil uVision3 IDE.

![Fig 21. Enable ARM/Thumb interworking in KEIL uVision3 IDE](image)

4.2 Start the demo

The reason for having four different bootloaders instead of just one bootloader is to minimize the size of the secondary bootloader itself. The smaller the bootloader, the more flash is available for the user application.

At power on or reset, a menu will be displayed in the Tera Term if properly configured. Most of the functions listed in menu are same. With these functions, the user can erase or program the flash with the application code using a specific interface.
Throughout this section "<starting address>" refers to the actual address which should be entered into the serial terminal.

For example, if you want to use address 0x10000 and the command is shown as “prog <starting address>”, then the user should type: “prog 0x10000”.

4.2.1 Code update via UART

Type command “prog <starting address>” and then send a binary file using 1K XMODEM protocol. See Fig 23.

Then input “run <starting address>”, which should jump to the application.

Note: The “address” should be the same with the entry point of the binary file.
To simplify the test of the CAN interface and protocol, CAN1 will act as transmitter and CAN2 as receiver on the MCB2300 board. Since CAN transceivers are already integrated in MCB2300, the user only needs to connect pins 2 and 7 directly together to both CANs as shown in Fig 24.

The user application is linked at address of 0x70000 but is programmed at address 0x60000. Afterwards, type “prog 0x60000 0x70000”. This causes the user application code to be copied from address 0x60000 onto address 0x70000 via the CAN interface.

An additional EEPROM (24LC64) can be used on the MCB2300 board to test the I2C interface and protocol.

As shown in Fig 26, the 24LC64 works as an I2C slave with address of 0XA0.
Note: The SDA and SCL bus requires a pull-up resistor to $V_{cc}$ (typical 10 kΩ for 100 kHz, 2 kΩ for 400 kHz).

The application code needs to be already programmed into EEPROM.

By typing command “prog <starting address>“, the user application code will be copied from EEPROM via I2C interface onto internal flash at the address specified in the “prog” command.

Then input “run <starting address>“, which should jump to the application.

![24LC64 circuit diagram]

**Fig 26. 24LC64 circuit**

### 4.2.4 Code update via SD/MMC

The SD/MMC bootloader utilizes a FAT file system so that the user can just copy the binary application code onto a SD Card from a workstation and then flash the application from the SD/MMC interface.

Using this bootloader, the user can list separate files located on the SD card and flash the microcontroller with the name specified. See **Fig 27**.
Fig 27. Code update via SD/MMC
5. Legal information

5.1 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.

5.2 Disclaimers

General — 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.

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 medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for 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.

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 national authorities.

5.3 Trademarks

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

1. Introduction ......................................................... 3
2. LPC2000 Flash Programming.............................3
   2.1 Sector description.................................................. 3
   2.2 In Application Programming (IAP) ......................... 4
   2.2.1 IAP introduction ..................................................... 4
   2.2.2 IAP application ...................................................... 4
   2.2.3 IAP commands ...................................................... 4
   2.2.4 Using IAP .............................................................. 5
   2.2.4.1 Define system parameters ......................................... 6
   2.2.4.2 Select sector ...................................................... 6
   2.2.4.3 Erase sector ....................................................... 7
   2.2.4.4 Program sector................................................... 7
   2.2.4.5 Verify data .......................................................... 8
   2.2.4.6 Interrupts during IAP .......................................... 8
   2.2.4.7 RAM used by IAP command handler ......................... 9
   2.2.4.8 Running the user application.............................. 9
3. Secondary bootloader and user application...10
   3.1 Secondary bootloader ......................................... 10
   3.1.1 Bootloader Features and Constraints..................10
   3.2 User application................................................... 10
   3.2.1 Memory map ....................................................... 10
   3.2.2 Interrupt vector table re-mapping ...................... 11
   3.2.3 User application size ........................................... 13
4. Demo description .............................................. 15
   4.1 Demo Setup ........................................................ 15
   4.1.1 Hardware setup................................................. 15
   4.1.2 Software setup ................................................... 15
   4.2 Start the demo.................................................... 15
   4.2.1 Code update via UART ....................................... 16
   4.2.2 Code update via CAN ........................................ 17
   4.2.3 Code update via I2C ........................................... 17
   4.2.4 Code update via SD/MMC ................................... 18
5. Legal information .............................................. 20
   5.1 Definitions............................................................ 20
   5.2 Disclaimers.......................................................... 20
   5.3 Trademarks ......................................................... 20
6. Contents............................................................. 21