AN11333

Interrupt handling during IAP calls for LPC177x_8x and LPC407x_8x

Rev. 1 — 13 January 2014 Application note

Document information

Info	Content
Keywords	IAP Interrupt IAP LPC177x_8x LPC407x_8x Cortex-M3 Cortex-M4
Abstract	This application note explains how interrupts can be handled when an IAP call is in process with NXP's Cortex 32-bit LPC407x_8x and LPC177x_8x MCU families

Revision history

Rev	Date	Description
1	20140113	Initial version.

Contact information

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

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

1. Introduction

NXP's LPC Cortex-M0, Cortex-M3 and Cortex-M4 flash parts have a ROM boot loader that controls initial operation after reset as well as provides the tools for programming the on-chip flash memory.

There are two programming features built in the ROM boot loader for reprogramming the on-chip flash. One feature is In-System Programming (ISP), which programs on-chip flash through a UART serial port using the boot loader software. The other feature is In-Application Programming (IAP), where the end-user's application code can erase and program the on-chip flash memory in a running system. This application note is focused on how to handle interrupts during IAP calls on NXP's LPC407x_8x and LPC177x_8x MCU families.

The LPC177x_8x is part of NXP's Cortex-M3 MCU family. The LPC407x_8x is part of the Cortex-M4 MCU family based on an LPC177x_8x with Floating Point Unit (FPU) enhancement and some additional peripherals like the NXP unique SPI Flash Interface (SPIFI) and dual analog comparator unit. From a memory map and IAP application point of view, these two devices are identical. The sample project provided with this application note has configurations for the LPC1788 and LPC4088.

2. Interrupt during IAP

Immediately after the CPU is reset, the interrupt vectors are located at address 0. When an IAP call is initiated, the boot loader will temporarily disable access to the user flash data. The user flash address space is mapped to some configuration data that is needed by IAP calls. Hence the original interrupt vector location does not contain the correct interrupt vector. Therefore if an interrupt happens while an IAP call is in process, the interrupt cannot be handled properly and the MCU will behave unpredictably. In some cases, the part resets itself when the interrupt is handled incorrectly as demonstrated in this application note.

The solution for the above issue is to relocate the interrupt vector table as well as the interrupt service routine to the SRAM area. This applies to all current production NXP flash parts that support IAP functionality. This application note provides examples on how to implement this relocation on the LPC407x_8x and LPC177x_8x MCUs.

2.1 IAP call review

The IAP routine should be called with a word pointer in register r0 pointing to memory (RAM) containing command code and parameters. The result from the IAP command is returned in the table pointed to by register r1.

The IAP function could be called in the following way using C.

Define the IAP location entry point. For NXP Cortex M devices, bit 0 of the IAP location is set.

#define IAP_LOCATION 0x1FFF1FF1

Define data structure or pointers to pass IAP command table and result table to the IAP function:

unsigned int command_param[5];

unsigned int status_result[5];

Or

*unsigned int * command_param;*

unsigned int status_result;*

*command_param = (unsigned long *) 0x…*

*status_result = (unsigned long *) 0x…*

Define a pointer to function type, which takes two parameters and returns void. The IAP returns the result with the base address of the table residing in r1.

*Typedef void (*IAP) (unsigned int [], unsigned int []);*

IAP iap_entry;

Setting the function pointer:

Iap_entry = (IAP) IAP_LOCATION;

Whenever you wish to call IAP you could use the following statement.

Iap_entry (command_param, status_result).

Fig 1 lists the available IAP commands:

IAP Command	Command Code
Prepare sector(s) for write operation	50 decimal
Copy RAM to Flash	51 decimal
Erase sector(s)	52 decimal
Blank check sector(s)	53 decimal
Read part ID	54 decimal
Read Boot Code version	55 decimal
Read device serial number	58 decimal
Compare	56 decimal
Reinvoke ISP	57 decimal

Fig 1. IAP commands

2.2 Interrupt vector table review

The interrupt vector table associates an interrupt handler with an interrupt request. When the CPU is interrupted, it looks up the interrupt handler in the interrupt vector table and transfers control to it.

After initial boot, the Cortex-M microcontroller's interrupt vector table is located at 0x00. For Cortex-M3 and Cortex-M4, ARM incorporates a Vector Table Offset Register (VTOR) that allows remapping the interrupt vector table to alternative locations in the memory map. This mechanism, as well as its usage in the interrupt handling, is introduced in section 2.3. This application note provides sample code that implements interrupt handling utilizing a VTOR register.

The VTOR register is not available in the Cortex-M0 architecture. Interrupt handling during IAP calls for these MCUs is described in other dedicated application notes. For NXP's LPC4300 and LPC1800 MCU families, NXP introduced a set of MEMMAP registers that can define the shadow address when accessing memory at address 0x00. Interrupt handling during IAP calls using these registers is described in other dedicated application notes.

2.3 Vector table offset register for Cortex-M3 and Cortex-M4

The Cortex-M3 and Cortex-M4 architecture incorporates a mechanism that allows remapping the interrupt vector table to alternative locations in the memory map. This is controlled via the VTOR located at 0xE00ED08 in the system control block.

The vector table may be relocated anywhere within the bottom 1 GB of address space. The VTOR indicates the offset of the vector table base address from memory address 0x00000000.

Bits	Name	Function
[31:30]	-	Reserved.
[29:8]	TBLOFF	Vector table base offset field. It contains bits[29:8] of the offset of the table base from the bottom of the memory map. **Remark:** Bit[29] determines whether the vector table is in the code or SRAM memory region: Bit[29] is sometimes called the TBLBASE bit. • 0 = code • 1 = SRAM.
[7:0]	-	Reserved.

Fig 2. VTOR bit assignments

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. As there are less than 64 vectors for LPC177x_8x and LPC407x_8x, an alignment of 256 bytes is sufficient.

2.4 Sample project implementation

The sample projects are provided based on Keil and LPCXpresso tool chain. The Keil version is located at folder \SampleProject_AN_InterruptDuringIAPCalls\Keil\Examples\IAP\.

See Fig 3 for the subfolder contents.

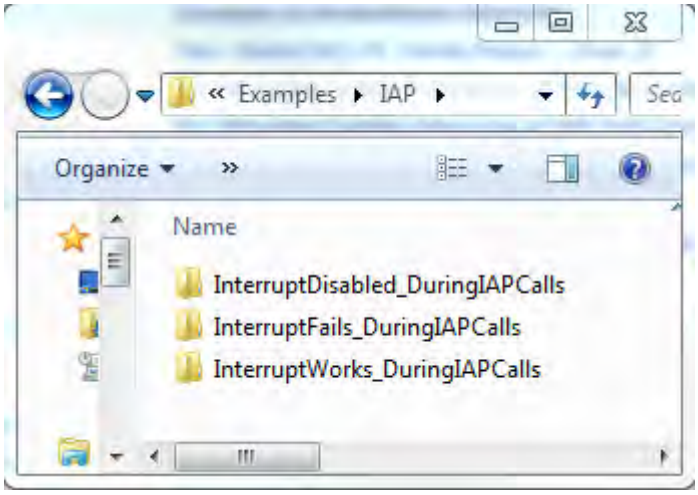

Fig 3. Keil example projects

- InterruptDisabled_DuringIAPCalls is the sample project where no interrupt is enabled during IAP calls. This serves as a baseline of discussion.
- InterruptFails_DuringIAPCalls is the sample project where the System tick timer interrupt is enabled during IAP Calls. This serves as a statement of the problem to be solved.
- InterruptWorks_DuringIAPCalls is the sample project where the System tick timer interrupt vector as well as its service routine is relocated to internal SRAM.

All three Keil sample projects are tested on Embedded Artists' developer's kit for LPC1788 and LPC4088.

The LPCXpresso based sample projects are packed in \SampleProject_AN_InterruptDuringIAPCalls\LPCXpresso.zip. You can import the project to a folder of your choice. After import, the folder contains the contents shown in Fig 4.

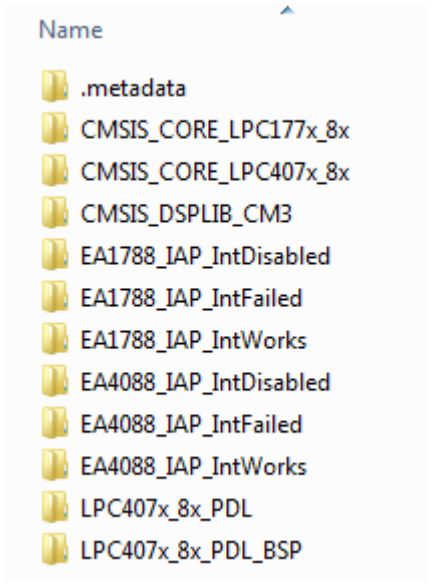

Fig 4. LPCXpresso example projects

Some of the projects are device and board support packages. The rest of the projects starting with “EAxxxx” are sample projects that perform the same tasks as the three Keil projects. These sample projects are our focus in this application note. The names of the projects clearly indicate whether they are for LPC1788 or LPC4088.

These sample projects share a common system flow. After the part boots up, it initializes UART0 to communicate processes happening in the system. Afterwards, they will enter the while (1) loop to repeatedly perform some flash erase, program and compare IAP calls.

On both LPC4088 and LPC1788 Embedded Artists’ development kit, a USB cable needs to be connected from J25 to a USB port on the PC to enable UART0 debug output on a serial terminal (Tera Term is used for the verification of the sample projects). All steps described in this application note should work on any PCB board with the LPC177x_8x or LPC407x_8x part that has a UART0 port integrated with a RS232 interface.

The following sections explain how these three projects relate to each other and deliver the method of interrupt handling during IAP calls.

2.4.1 Interrupt disabled during IAP calls

Fig 4 is the flow chart of project “InterruptDisabled_DuringIAPCalls”. Here we establish a base line where the IAP calls are properly carried out without any disturbance from interrupt calls.

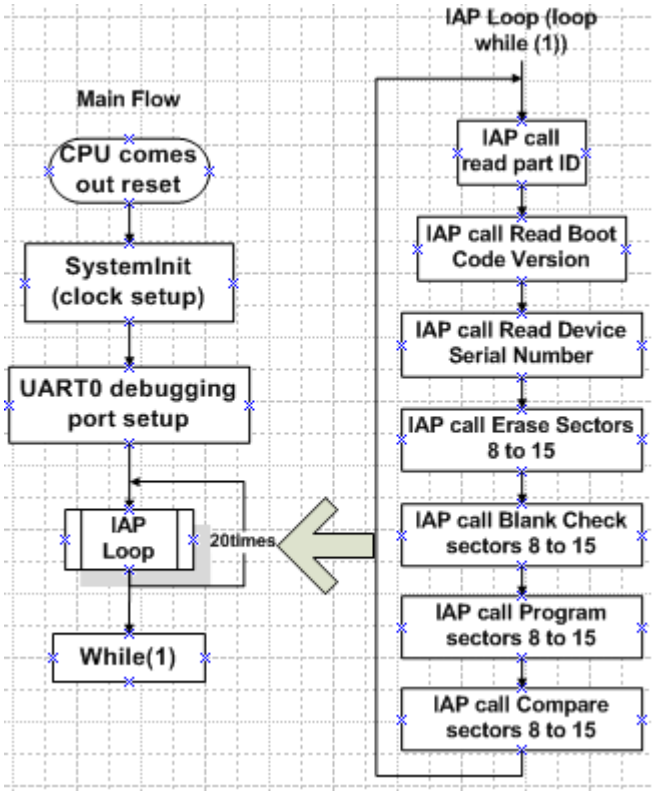

Fig 5. Flow chart: Interrupt disabled during IAP calls

Fig 6 is a snip of the debug output printed on the Tera Term. The read part ID, Boot Code, UID, erase and program IAP calls repeat smoothly.

2.4.2 Interrupt failed during IAP calls

The flow chart for project “InterruptFailed_DuringIAPCalls” is described in Fig 7. The red blocks are added functionalities compared with project “InterruptDisabled_DuringIAPCalls”.

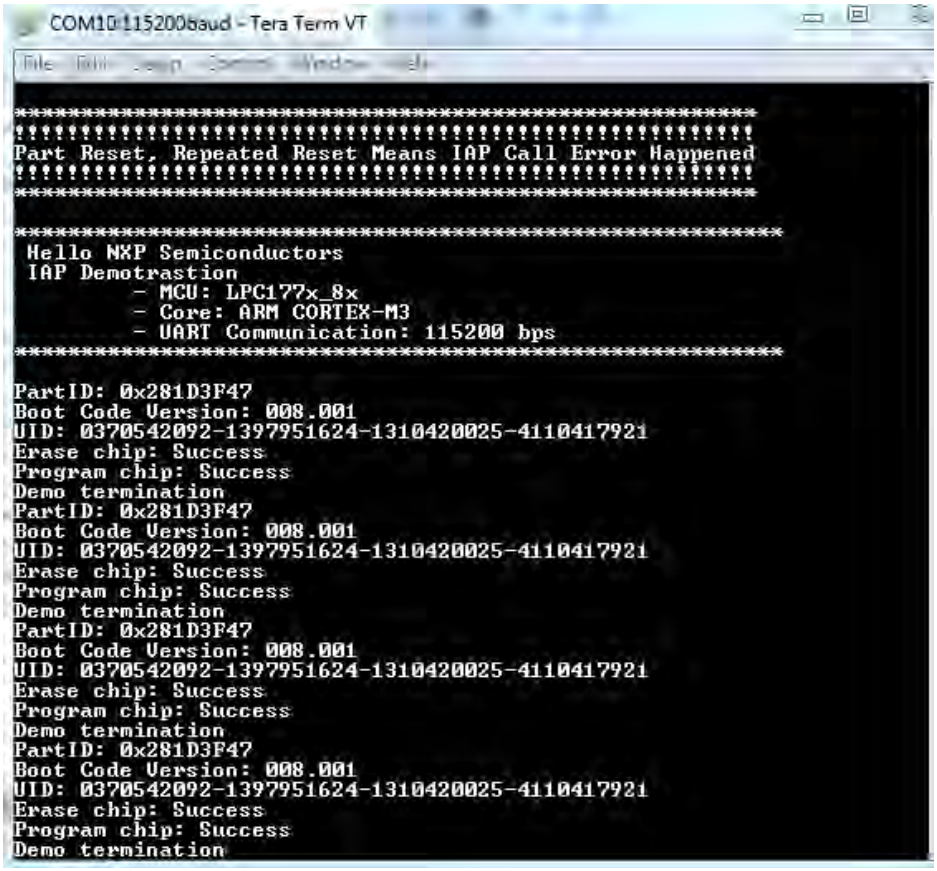

Fig 6. Tera Term output: IAP with no interrupt enabled

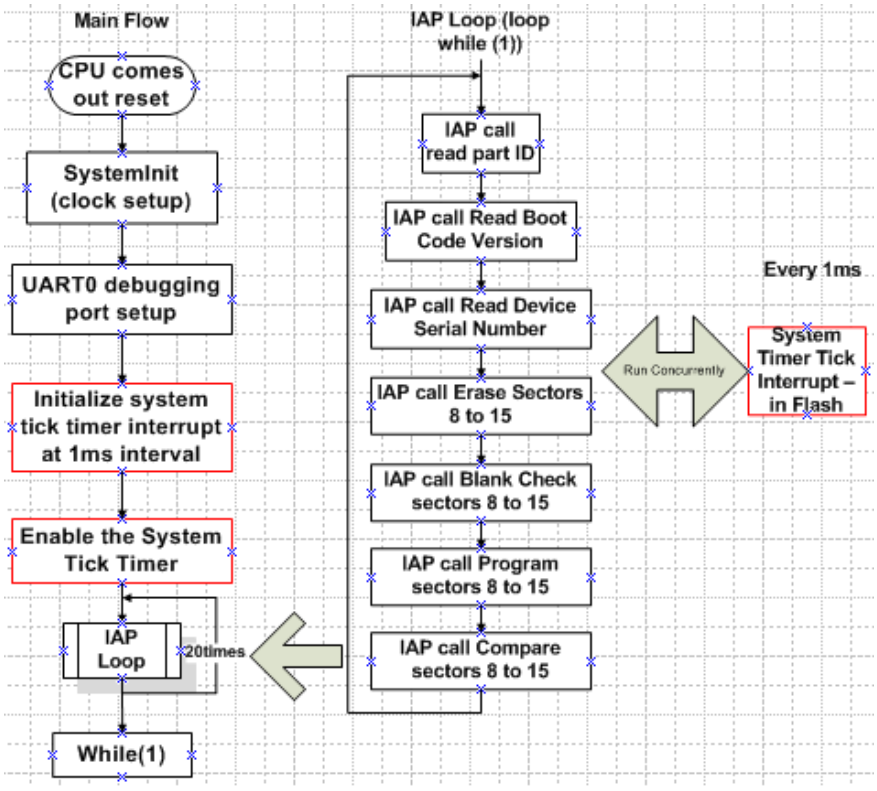

Fig 7. Flow chart: Interrupt failed during IAP calls

This project enables a 1 ms system tick timer interrupt that runs concurrently with all the IAP calls that are happening in the main while(1) loop. When the interrupt vector is fetched in the middle of an IAP call, the interrupt is not properly handled by the microcontroller. In this case, the microcontroller will reset as indicated by the debug output “!! Part Reset, Repeated Reset Means IAP Call Error Happened!!” as shown in Fig 8.

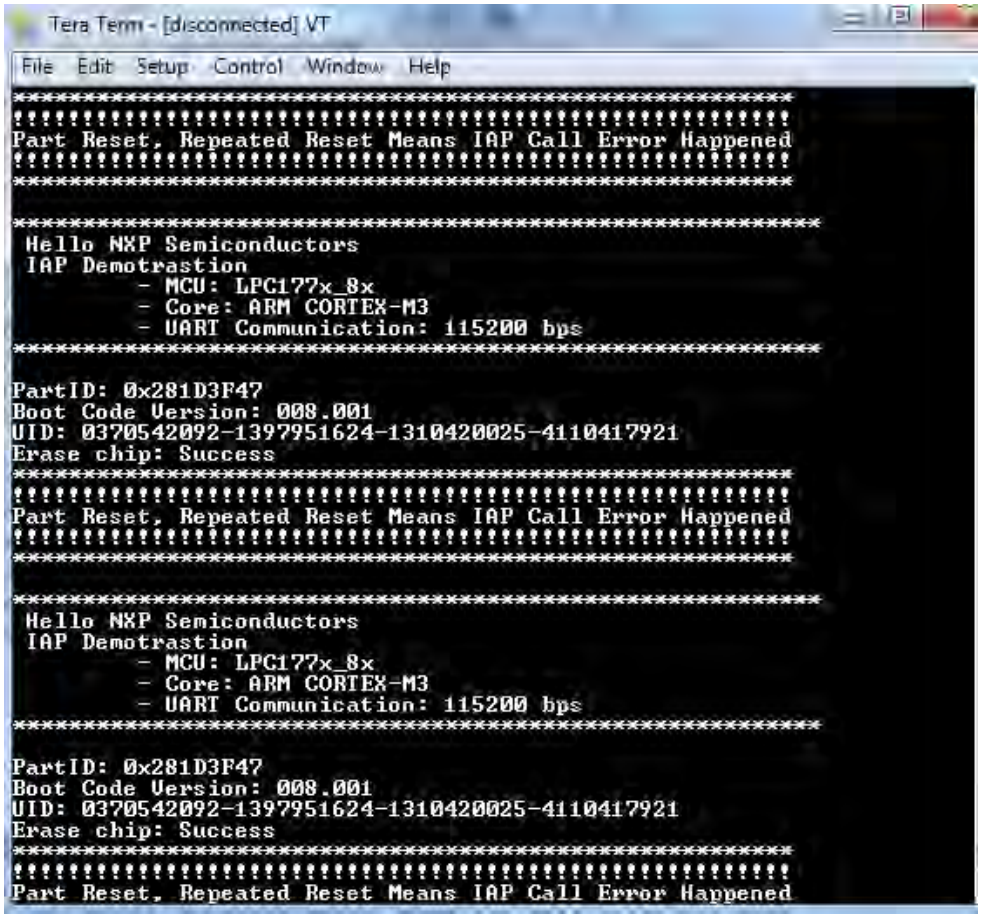

Fig 8. System timer tick interrupt is not properly handled during IAP calls

2.4.3 Interrupt works during IAP calls

Project “InterruptWorks_DuringIAPCalls” relocates the system tick timer interrupt vector table and the corresponding interrupt service routine to internal SRAM location. Then it temporarily maps VTOR to the new SRAM location before the IAP calls starts. After this, when the system tick timer interrupt happens, the CPU can still fetch the correct address of the interrupt service routine and take care of the interrupt. Fig 9 is the flow chart that details the execution sequence of the major events. The green blocks are added or modified functionalities compared with project “InterruptFailed_DuringICAPCalls”.

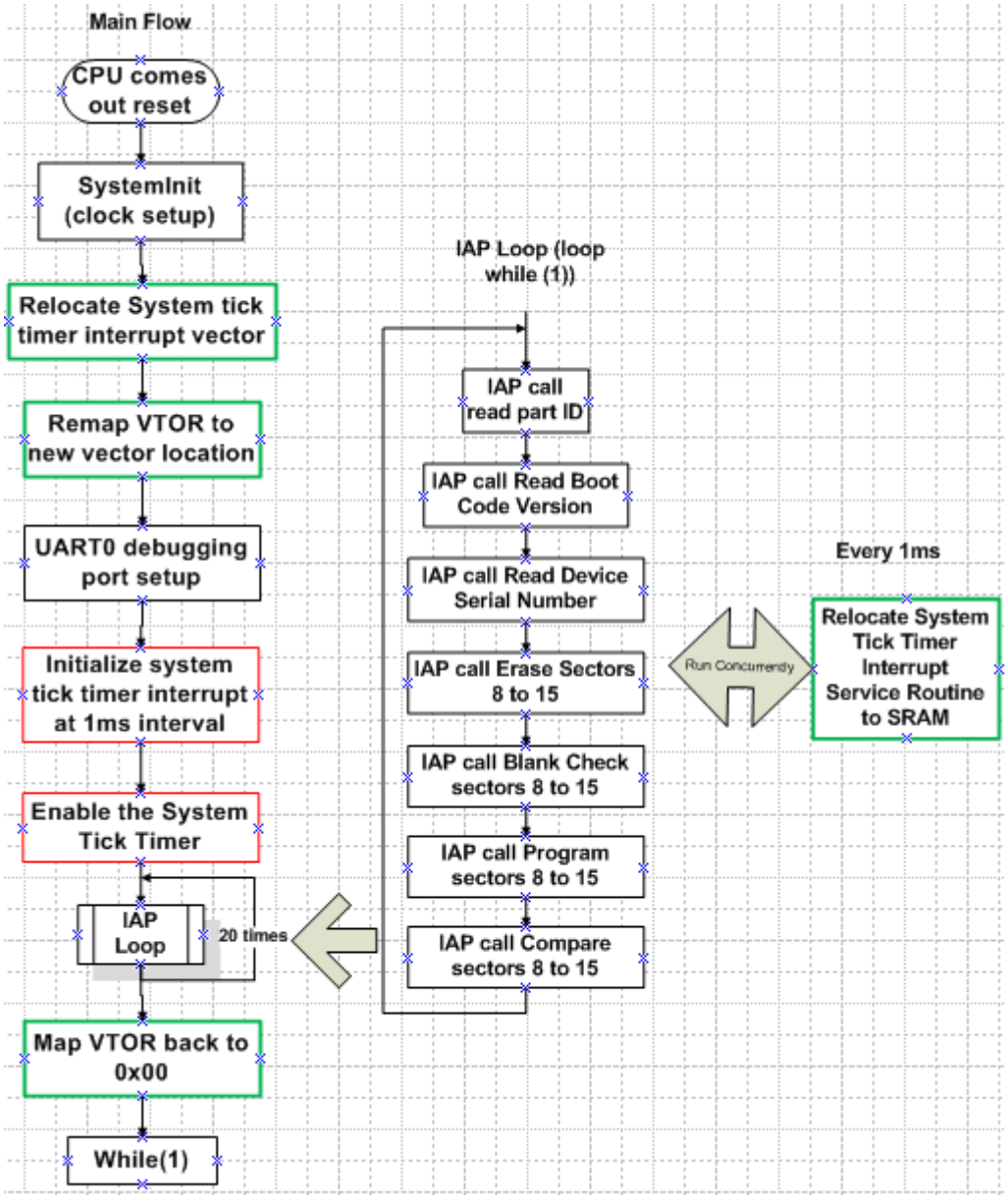

Fig 9. Flow Chart: Interrupt Works during IAP Calls

The methods of relocating the interrupt vector table and the corresponding interrupt service are detailed in section 2.4.3.1 and 2.4.3.2.

2.4.3.1 Relocate the interrupt vector table

The new vector table location is picked to be at 0x10008000. The following routine can relocate a particular interrupt vector. Notice that the interrupt number for the system tick timer is number 15.

```
#define SYSTICK_INT                15
volatile uint32_t int_vector_table[64] __attribute__((at(0x10008000)));
void init_interrupt_controller(int IntToBeReAllocated)
```

```
{
  volatile const uint32_t *org_table = 0x00000000;
     int_vector_table[IntToBeAllocated]=org_table[IntToBeAllocated];
  }
```

With the above definition, the following function call will relocate the system tick timer interrupt to 0x10008000+15*4. Notice that the SYSTICK interrupt number is 15 and only this SYSTICK interrupt is relocated the SRAM with other interrupt stays at 0x0.

init_interrupt_controller(SYSTICK_INT);

Next we modify the Vector Table Offset Register to point to the new location.

SCB->VTOR = 0x10008000;

Notice that after the IAP routines are finished, we reset the VTOR to its original location and allow the system to enable other interrupts when needed.

SCB->VTOR = 0x00;

2.4.3.2 Relocate the interrupt vector routine

There are two .c files that are related with the interrupt service routine: lpc_systick.c and IntRoutine.c. Using the Keil compiler, the interrupt routines are easily relocated.

```
.
RW_IRAM1 0x10000000 0x00010000  {  ; RW data
  lpc_systick.o (+RO +ZI +RW)
  IntRoutine.o (+RO +ZI +RW)
 .ANY (+RW +ZI)
}
```

Fig 10. Relocate the interrupt service routines (.sct file)

Fig 11 shows the memory mapping of the I-Code and D-code memory space of the 512kB LPC177x_8x and LPC407x_8x flash part. Notice that the interrupt vector table and interrupt service handler are relocated from flash to RAM.

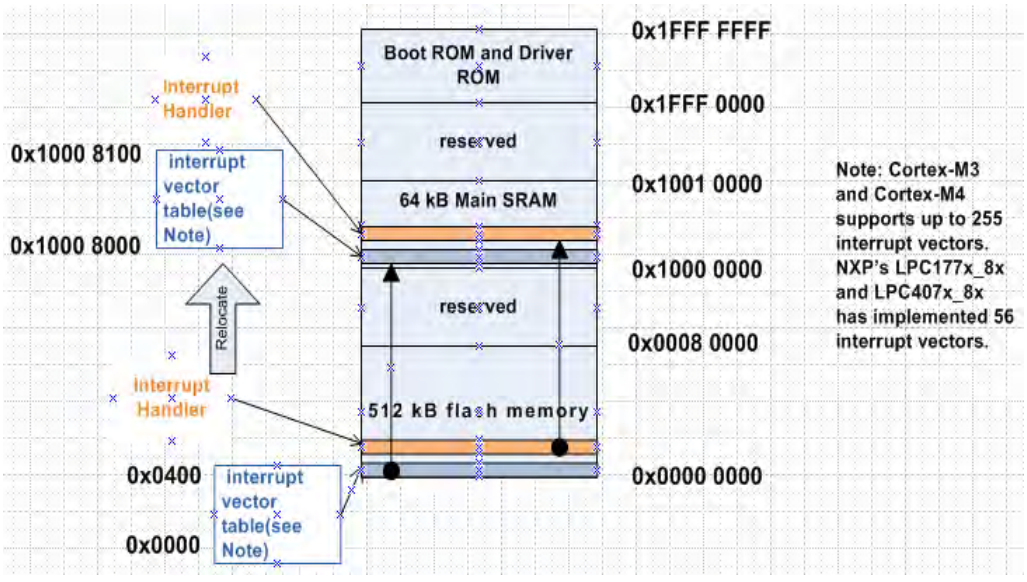

Fig 11. I Code and D Code memory space (LPC407x_8x and LPC177x_8x 512 kB flash part)

2.4.3.3 Demonstration of successful interrupt handling during IAP calls

After relocating the interrupt vector table as well as the interrupt service routine, the system timer tick interrupts are handled properly with the same outputs on Tera Term as shown in Fig 6. One thing to note is that the group of IAP calls proceeds more slowly compared to the system tick timer interrupts being disabled. This is due to the 1 ms interval interrupt for the CPU to handle besides the IAP calls.

3. Conclusion

This application note provides example implementation for interrupt handling when IAP calls could happen concurrently. Following the steps explained in this application note, critical interrupts do not need to be disabled when IAP calls are carried out. This improves embedded system design flexibility with NXP's microcontrollers. Although the sample projects are based on LPC407x_8x and LPC177x_8x, the same steps are applicable to other Cortex-M3 and Cortex-M4 NXP MCU families.

4. Legal information

4.1 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.

4.2 Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

4.3 Trademarks

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

5. List of figures

6. Contents

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 13 January 2014
Document identifier: AN11333