

APPLICATION NOTE

ABSTRACT

Simple interrupt handling using Timer 1 peripheral on the LPC2106 is shown in this application note. Code samples are provided using which the user could run the application from Flash or SRAM classifying the Timer 1 peripheral as FIQ or IRQ.

AN10254

Philips LPC210x microcontroller family

Amitkumar Bhojraj

2003 Dec 12

Philips LPC210x microcontroller family

AN10254

INTRODUCTION

This application note shows how an interrupt could be handled in the ARM architecture. The code provided shows the handling of both a vectored IRQ (Interrupt Request) and FIQ (Fast Interrupt). The interrupt vectors lie from 0x00–0x1C in the Flash address space. If the application is run from SRAM then the interrupt vectors need to be remapped to 0x40000000–0x4000001C. This is done using the Memory Map register (details in System Control Block section in LPC2104/5/6 User Manual). The code is developed in ADS (ARM Development Suite) v1.2 and is written to run from Flash. After each section the changes needed to run the application from SRAM is provided. Interrupt Service Routines (ISR) are written in C for IRQ and in assembly for FIQ.

The application note is arranged in the following manner:

- Timer1 is configured to trigger a IRQ interrupt and the code runs from Flash
- Changes and additions needed to run the above code from SRAM
- Timer1 is configured to trigger a FIQ interrupt and the code runs from Flash
- Changes and additions needed to run the above code from SRAM

TIMER1 IS CONFIGURED TO TRIGGER AN IRQ INTERRUPT AND THE CODE RUNS FROM FLASH

This example application has the following files:

1. Interrupt Vector Table (ivt.s)
2. Startup Assembly code (init.s)
3. Main C file

Only the relevant files are included here in this application note. The C files would remain very much the same if ported to a different cross-compiler but the assembler directives in the assembly files need to be changed.

Interrupt Vector table

This code should be linked to 0x0. Here the interrupt vectors are provided for the ARM core.

```

; -----
;                               Assembler Directives
; -----
        AREA IVT, CODE           ; New Code section
        CODE32                   ; ARM code
        IMPORT start             ; start symbol not
                                ; defined in this
                                ; section
Entry                                       ; Defines entry point
; -----
        LDR    PC, =start
        LDR    PC, Undefined_Addr
        LDR    PC, SWI_Addr
        LDR    PC, Prefetch_Addr
        LDR    PC, Abort_Addr

; At 0x14 the user should insert a signature (checksum).
; This signature enables the bootloader to determine if
; there is valid user code in the Flash. Currently most of
; the Flash programming tools (debuggers and ISP utility)
; have this feature built-in so the end user need not worry
; about it. If the tool does not provide this feature then
; the value has to be computed manually and has to be
; inserted at 0x14. Details on computation of checksum
; could be found in the Flash programming chapter in the
; LPC2104/5/6 User Manual.

        DCD    ... .
        LDR    PC, [PC, #-0xFF0]
        LDR    PC, FIQ_Addr

Undefined_Addr DCD    Undefined_Handler
SWI_Addr      DCD    SWI_Handler
Prefetch_Addr DCD    Prefetch_Handler

```

Philips LPC210x microcontroller family

AN10254

```

Abort_Addr      DCD      Abort_Handler
FIQ_Addr        DCD      FIQ_Handler

; -----
;                               Exception Handlers
; -----

; The following dummy handlers do not do anything useful in
; this example. They are set up here for completeness.

Undefined_Handler
                B        Undefined_Handler

SWI_Handler
                B        SWI_Handler

Prefetch_Handler
                B        Prefetch_Handler

Abort_Handler
                B        Abort_Handler

FIQ_Handler
                B        FIQ_Handler

END

```

On reset the first instruction to be executed in the application will be

```
LDR PC,=start
```

which will branch to the assembly startup code which will enable the interrupts and set the stack pointers for IRQ and Supervisor modes.

The interrupt vector of importance would be the IRQ interrupt.

```
LDR PC, [PC, #-0xFF0]
```

This instruction will load the PC with the address of the ISR from VIC Vector Address Register (0xFFFF F030) and transfer execution there.

All remaining vectors have dummy interrupt handlers.

Startup Assembly code

```

; -----
;                               Assembler Directives
; -----

                AREA asm_code, CODE    ; New Code section
                CODE32                 ; ARM code
                IMPORT __main          ; main not defined
                                        ; in this section
                EXPORT start           ; global symbol
                                        ; referenced in
                                        ; ivt.s

; -----
start

                ; Enable interrupts

                MSR cpsr_c,#0x13

                ; Set SP for Supervisor mode. Depending upon
                ; the stack the application needs this value
                ; needs to be set.

                LDR SP,=0x4....

                ; Setting up SP for IRQ mode. Change mode to
                ; IRQ before setting SP_irq and then
                ; move back again to Supervisor mode

                MRS R0, CPSR
                BIC R1, R0,#0x1F
                ORR R1, R1,#0x12
                MSR cpsr_c, R1

```

Philips LPC210x microcontroller family

AN10254

```

LDR SP, =0x4....
MSR cpsr_c, R0

; Jump to C code

LDR lr, =__main
MOV pc, lr

```

END

This code is linked from **ivt.s** in the very first instruction. Failure to set the Stack Pointers will lead to a Data Abort and hence the stack initialization should be done before jumping to C main ().

C CODE

The match registers in Timer1 have been configured in such a way that the Timer will interrupt the core and reset on a match. Timer1 runs at full speed at 60 MHz. This code was tested on an evaluation board, which uses a 10 MHz crystal, and the PLL has been set for the same. Please refer to the Timer0 and Timer1 chapter in the LPC2106/5/4 User Manual for detailed description on Timer operation. The ISR's have been kept empty and its up to the end-user to fill it up as required. One could blink a few LED's or toggle some port pins for initial evaluation in the IRQHandler() function. The `_irq` compiler keyword has been used to define the IRQHandler () function as an IRQ ISR.

The C main is reached from **init.s** by executing the following instructions.

```

LDR lr, =__main
MOV pc, lr

```

The C code could be as follows:

```

/*****
                Function declarations
*****/
__irq void IRQHandler(void);
void feed(void);
void Initialize(void);
/*****
                Header files
*****/
#include"LPC210x.h"
/*****
                MAIN
*****/

int main()
{
    /* Initialize the system */
    Initialize();

    /* Start timer */
    T1_TCR=0x1;

while(1)
    {

    }

/*****
                Initialize
*****/
void Initialize()
{
    /*
     * Initialize PLL (Configured for a 10MHz crystal) to
     * boost processor clock to 60MHz
     */

    /* Setting Multiplier and divider values */

```

Philips LPC210x microcontroller family

AN10254

```

    PLLCFG=0x25;
    feed();
    /* Enabling the PLL */
    PLLCON=0x1;
    feed();
    /* Wait for the PLL to lock to set frequency */
    while(!(PLLSTAT & PLOCK)){
    /* Connect the PLL as the clock source */
    PLLCON=0x3;
    feed();

    /*
     * Enabling MAM and setting number of clocks used for
     * Flash memory fetch
     */
    MAMCR=0x2;
    MAMTIM=0x4;

    /*
     * Setting peripheral Clock (pclk) to System
     * Clock (cclk)
     */
    VPBDIV=0x1;

    /* Initialize GPIO */
    IODIR=0xFFFF;
    IOSET=0xFFFF;

    /* Initialize Timer 1 */
    T1_TCR=0x0;
    T1_TC=0x0;
    T1_PR=0x0;
    T1_PC=0x0;

    /* End user has to fill in the match value */
    T1_MR0=0x....;

    /* Reset and interrupt on match */
    T1_MCR=0x3;

    /* Initialize VIC */
    VICINTSEL=0x0;      /* Timer 1 selected as IRQ */
    VICINTEN= 0x20;    /* Timer 1 interrupt enabled */
    VICCNTL0= 0x25;
    /* Address of the ISR */
    VICVADDR0=(unsigned long)IRQHandler;
}
/*****
    Timer 1 ISR
*****/
__irq void IRQHandler()
{
/*
 * The Interrupt Service Routine code will come here. The
 * interrupt needs to be cleared in Timer1 and a write must
 * be performed on the VIC Vector Address Register to
 * update the VIC priority hardware. Here the user could
 * blink a few LED's or toggle some port pins as an
 * indication of being in the ISR
 */
    T1_IR=0x1;
    VICVADDR=0xff;
}

```

Philips LPC210x microcontroller family

AN10254

```

/*****
Feed Sequence for PLL
*****/
void feed()
{
    PLLFEED=0xAA;
    PLLFEED=0x55;
}

```

CHANGES AND ADDITIONS NEEDED TO RUN THE ABOVE CODE FROM SRAM

The linker should be configured in such a way that the interrupt vector table (ivt.s) should be linked to the bottom of the SRAM (0x4000000). Be sure that the relevant interrupt vectors lie between 0x40000000–0x4000003F. The other files are linked within the code itself and can lie in SRAM.

Also the interrupt vectors have to be remapped to SRAM. Using the MEMAP register and configuring it for User RAM mode could achieve this.

```

int main()
{
    ... .
        /* Initialize MEMAP */
        MEMAP=0x2;
    ... .
}

```

TIMER1 IS CONFIGURED TO TRIGGER A FIQ INTERRUPT AND THE CODE RUNS FROM FLASH

The example application has the following files:

1. Interrupt Vector Table (ivt.s) (FIQ ISR placed here itself)
2. Startup Assembly code (init.s)
3. Main C file

Only the relevant files are mentioned here in this application note. The C files would remain very much the same if ported to the different cross-compiler but for the assembly files the assembler directives need to be changed.

Interrupt Vector table

This code should be linked to 0x0. Here the interrupt vectors and the FIQ ISR have been provided for the ARM core.

```

; -----
;                               Assembler Directives
; -----
        AREA IVT, CODE           ; New Code section
        CODE32                   ; ARM code
        IMPORT start             ; start symbol not
                                ; defined in this
                                ; section
T1_IR   EQU      0xE0008000
Entry   ; Defines entry point
; -----
        LDR      PC, =start
        LDR      PC, Undefined_Addr
        LDR      PC, SWI_Addr
        LDR      PC, Prefetch_Addr
        LDR      PC, Abort_Addr
        DCD      ; Signature
        LDR      PC, IRQ_Addr
; -----
; Instead of placing the LDR instruction here, the FIQ ISR
; itself is placed at 0x1C
; -----
        ; Clear the Timer 1 interrupt
        MOV     R8, #0x1
        LDR     R9, =T1_IR
        STR     R8, [R9]

```

Philips LPC210x microcontroller family

AN10254

```

; The end user could add more code here, which could
; be blinking a few LED's or toggling a few port
; pins on the evaluation board as an indication to
; the outside world

; Return to C main
SUBS PC,R14,#0x04

; -----
Undefined_Addr DCD      Undefined_Handler
SWI_Addr       DCD      SWI_Handler
Prefetch_Addr DCD      Prefetch_Handler
Abort_Addr     DCD      Abort_Handler
IRQ_Addr       DCD      IRQ_Handler
; -----
;                               Exception Handlers
; -----
; The following dummy handlers do not do anything useful in
; this example. They are set up here for completeness.

Undefined_Handler
        B          Undefined_Handler
SWI_Handler
        B          SWI_Handler
Prefetch_Handler
        B          Prefetch_Handler
Abort_Handler
        B          Abort_Handler
IRQ_Handler
        B          IRQ_Handler

```

END

The user should take note of the following points:

1. FIQ has the last interrupt vector assigned to it (0x1C). Hence the user need not jump from this location to an ISR but could place the ISR from this location itself.
2. No stack operations are carried out since banked FIQ registers (R8, R9) are used.
3. A write need not be performed on the VICVectorAddress register as in the case of an IRQ.

Startup Assembly code

```

; -----
;                               Assembler Directives
; -----
        AREA asm_code, CODE      ; New Code section
        CODE32                   ; ARM code
        IMPORT __main             ; main not defined
                                   ; in this section
        EXPORT start              ; global symbol
                                   ; referenced in
                                   ; ivt.s
; -----
start

        ; Enable interrupts

        MSR cpsr_c, #0x13

        ; Set SP for Supervisor mode. Depending upon
        ; the stack the application needs this value
        ; needs to be set.

```

Philips LPC210x microcontroller family

AN10254

```

LDR SP, =0x4....

; Setting up SP for FIQ mode. Change mode to
; FIQ before setting SP_fiq and then
; move back again to Supervisor mode

MRS R0, CPSR
BIC R1, R0, #0x1F
ORR R1, R1, #0x11
MSR cpsr_c, R1
LDR SP, =0x4....
MSR cpsr_c, R0

; Jump to C code

LDR lr, =__main
MOV pc, lr

```

END

This code is linked from **ivt.s** in the very first instruction. Failure to set the Stack Pointers will lead to a Data Abort and hence the stack initialization should be done before jumping to C main (). The Startup is not much different than the one used for IRQ except for the following instruction

```
ORR R1, R1, #0x11
```

C code

The match registers in Timer1 have been configured such that the Timer will interrupt the core and reset on a match. Timer1 runs at full speed at 60 MHz. This code was tested on an evaluation board, which uses a 10 MHz crystal, and the PLL has been set for the same. Please refer to the Timer0 and Timer1 chapter in the LPC2106/5/4 User Manual for detailed description on Timer operation.

The C main is reached from **init.s** by executing the following instructions.

```

LDR lr, =__main
MOV pc, lr

```

The C code could be as follows:

```

/*****
                Function declarations
*****/
void Initialize(void);
void feed(void);
/*****
                Header files
*****/
#include "LPC210x.h"
/*****
                MAIN
*****/
int main()
{
    Initialize();

    /* Start timer */
    T1_TCR=0x1;

while(1)
    {
        }
}
/*****
                Initialize
*****/
void Initialize()

```

Philips LPC210x microcontroller family

AN10254

```

{
/* Initialize PLL */

/* Setting Multiplier and divider values */
PLLCFG=0x25;
feed();
/* Enabling the PLL */
PLLCON=0x1;      feed();
/* Wait for the PLL to lock to set frequency */
while(!(PLLSTAT & PLOCK)){}
/* Connect the PLL as the clock source */
PLLCON=0x3;
feed();

/*
 * Enabling MAM and setting number of clocks used for
 * Flash memory fetch
 */
MAMCR=0x2;
MAMTIM=0x4;

/*
 * Setting to peripheral Clock (pclk) to System
 * Clock (Cclk)
 */
VPBDIV=0x1;

/* Initialize GPIO */
IODIR=0xFFFF;
IOSET=0xFFFF;

/* Initialize Timer 1 */
T1_TCR=0x0;
T1_TC=0x0;
T1_PR=0x0;
T1_PC=0x0;

/* Set the match value in the match register */
T1_MR0=0x....

/* Reset and interrupt on match */
T1_MCR=0x3;

/* Initialize VIC */
VICINTSEL=0x20;      /* Timer 1 selected as FIQ */
VICINTEN= 0x20;     /*Timer 1 interrupt enabled*/
}
/*****
Feed Sequence for PLL
*****/
void feed()
{
    PLLFEED=0xAA;
    PLLFEED=0x55;
}

```

The VIC settings are slightly different for the FIQ. All that needs to be done for the FIQ is that the peripheral should be selected as an FIQ and enabled.

Philips LPC210x microcontroller family

AN10254

CHANGES AND ADDITIONS NEEDED TO RUN THE ABOVE CODE FROM SRAM

The linker should be configured in such a way that the interrupt vector table (ivt.s) should be linked to the bottom of the SRAM (0x40000000). Be sure that the relevant interrupt vectors lie between 0x40000000–0x4000003F. The other files are linked within the code itself and can lie in SRAM.

Also the interrupt vectors have to be remapped to SRAM. Using the MEMAP register and configuring it for User RAM mode could achieve this. Please refer to the Memory Mapping Control Section in the System Control Block Chapter of the LPC2106/5/4 User Manual for detailed description.

```
int main()
{
  ....
      /* Initialize MEMAP */
      MEMAP=0x2;
  ....
}
```

Since only the first 64 bytes could be remapped the Interrupt vector Table needs to have the following modifications:

1. The FIQ ISR cannot reside at 0x4000001C. Instead a jump will be placed at this address to the FIQ Handler (written in assembly in this case).
2. The Interrupt Vector Table is condensed as compared to the Flash version to accommodate the symbol definition of the FIQ Handler within the first 64 bytes. If the user uses the same Interrupt Vector Table (without the FIQ ISR) as the Flash version then the code will land up in the dummy Prefetch Handler

Interrupt Vector Table

```
; -----
;                               Assembler Directives
; -----
      AREA IVT, CODE           ; New Code section
      CODE32                   ; ARM code
      IMPORT start             ; start symbol not
                               ; defined in this
                               ; section
      IMPORT FIQhandler

entry
; -----
      LDR    PC, =start
      LDR    PC, Undefined_Handler
      LDR    PC, SWI_Handler
      LDR    PC, Prefetch_Handler
      LDR    PC, Abort_Handler
      NOP
      LDR    PC, IRQ_Handler
      LDR    PC, =FIQhandler    ; jump to ISR
; -----
;                               Exception Handlers
; -----
; The following dummy handlers do not do anything useful in
; this example. They are set up here for completeness.
Undefined_Handler
      B      Undefined_Handler

SWI_Handler
      B      SWI_Handler

Prefetch_Handler
      B      Prefetch_Handler

Abort_Handler
      B      Abort_Handler

IRQ_Handler
      B      IRQ_Handler

END
```

Philips LPC210x microcontroller family

AN10254

On an FIQ exception, the instruction to be executed would be:

```
LDR    PC, =FIQhandler
```

which would then jump the ISR shown below.

FIQ Handler

For this configuration, the ISR has been written as an assembly routine. Hence there is no handler in C code. If the user wishes then a branch and link could be made from this routine to a C subroutine.

```
; -----
;                               Assembler Directives
; -----
                AREA FIQH, CODE      ; New Code section
                CODE32                ; ARM code
T1_IR           EQU      0xE0008000
; -----

FIQHandler

                ; Clear the Timer 1 interrupt

                MOV R8, #0x1
                LDR R9, =T1_IR
                STR R8, [R9]

                ; Here the main ISR could be inserted
                ; or even a jump to a C routine

                ; Return to C main

                SUBS PC, R14, #0x04

END
```

CONCLUDING STATEMENTS

- The code execution begins either at 0x0(for Flash) or 0x40000000(for SRAM). Hence it is of prime importance that the interrupt vector table (ivt.s) is linked to the bottom of the Flash or SRAM.
- Other files are linked within the code itself.
- If the application is run from SRAM the interrupt vectors need to be remapped using the MEMMAP register. Be sure that all the interrupt vectors lie in the first 64 bytes. May result in Prefetch abort if the symbol definitions lie outside this range.
- Setting of the stack pointers is important. May result in Data abort if not set correctly.
- The code samples are well commented and tested.

If the user is interested in the scenario of how the ARM core responds if the interrupt occurs while the interrupts are disabled then please refer to the FAQ (IP FAQ's by Feature Type) on the ARM website (link mentioned in References)

REFERENCES

ARM Architecture Reference Manual: Second Edition, edited by David Seal: Addison-Wesley: ISBN 0-201-73719-1 (Known as the "ARM ARM". ARM Doc No.: DDI-0100). Also available in PDF form on the ARM Technical Publications CD.

ARM Developer Suite – Version1.2– Developer Guide: http://www.arm.com/documentation/Software_Development_Tools/index.html

ARM FAQ: <http://www.arm.com/support/ARM7TDMI-S.html>

Philips LPC210x microcontroller family

AN10254

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 60134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

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

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information

For additional information please visit
<http://www.semiconductors.philips.com>. Fax: +31 40 27 24825

© Koninklijke Philips Electronics N.V. 2002
All rights reserved. Printed in U.S.A.

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com

Date of release: 12-03

Document order number:

9397 750 12478

Let's make things better.