Experiment 4: Interrupts — Part I

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
141	Oct. 12, 2014		H

Contents

1 Objectives **1**

2 Parts List **1**

3 Background **1**

3.1 Interrupts 1

3.2 External Interrupts Using GPIO Pins 2

3.2.1 Enabling External Interrupts on GPIO Pins 2

3.2.2 Enabling `EINT3` in NVIC 2

3.2.3 `EINT3` ISR 3

3.2.4 Interrupt Status 3

4 Tasks **4**

4.1 One External Interrupt 4

4.2 Two External Interrupts 4

5 Resources **4**

1 Objectives

- Using external hardware interrupts

2 Parts List

- LPC1769 LPCXpresso board
- USB A-Type to Mini-B cable
- Breadboard
- LEDs
- Push-buttons or switches
- 330-Ohm Resistors
- Jumper wires

3 Background

3.1 Interrupts

Interrupts allow for suspending the currently executing code, and having the CPU switch to execute a routine associated with the received interrupt request.

There are two basic concepts in our discussion of interrupts:

ISR (interrupt service routine)
The routine that needs to be executed when an interrupt request is received.

IVT (Interrupt Vectors Table)
An array of addresses pointing to the start of all available ISRs.

In LPC1769, there are 35 hardware interrupts. Each interrupt is identified by a number called `IRQn`, or *Interrupt ID* as called in the LPC1769 manual. Below are some examples of hardware interrupts and their IDs:

Table 1: Example Interrupt IDs

Interrupt ID	Interrupt Source
1	Timer 0
2	Timer 1
5	UART 0
6	UART 1
13	SPI
18	EINT0
19	EINT1
20	EINT2
21	EINT3
33	USB Activity Interrupt

Note
Since other hardware interrupts require understanding the functions with which they are associated, which we did not cover yet, we will concentrate on external interrupts.

Processing external interrupts involves:

- Detecting the interrupt signal on one of the pins (should be enabled by software).
- Jumping to the interrupt service routine associated with that request.

External interrupts can be triggered simply by a push-button or a switch. However, the difference between such implementation and what you did in Experiment 3 should be clear. In Experiment 3, we used *polling*, where the CPU is always busy reading the pin in order to detect a change that would trigger some action. When using interrupts, however, the CPU is available to execute other code. When the push-button is pressed, the CPU stops whatever it is doing and jumps to the routine associated with that interrupt request.

3.2 External Interrupts Using GPIO Pins

There are 4 external interrupts available to the developer, called `EINT0`, `EINT1`, `EINT2` and `EINT3`. In the previous ARM versions, a pin's function must be set for the pin to act as an external interrupt. This is done using the `PINSEL` register (to be discussed in the next experiment). However, one of the new features of the newer Cortex family is accepting external interrupts from some GPIO pins! Any GPIO pin used for external interrupt will be using external interrupt channel 3 (`EINT3`). This is important to know because, in CMSIS, the ISR name or handler for `EINT3` is `EINT3_IRQHandler()`.

Note
You can use GPIO pins only from ports 0 and 2 for external interrupts. You have about 40 different pins to choose from. Compare that, for example, to ARM7 where only 7 pins are available for external interrupts.

In `LPC17xx.h`, the structure that deals with GPIO external interrupts is `LPC_GPIOINT`, which includes a few fields that control the GPIO pins when acting as an external interrupt.

3.2.1 Enabling External Interrupts on GPIO Pins

You can set external interrupts to be generated on the *rising edge* on a GPIO pin by setting the `IO0IntEnR` and `IO2IntEnR` registers, depending on the port to which the pin belongs. These names refer to 32-bit registers. Setting a bit to `1` enables rising-edge interrupts at the corresponding pin.

If you prefer to generate interrupts on the *falling edge*, you can use the `IO0IntEnF` and `IO2IntEnF` registers instead.

For example, to enable rising-edge interrupts on pin 0 of port 2 only:

```
LPC_GPIOINT->IO2IntEnR = 1;
```

3.2.2 Enabling `EINT3` in NVIC

The Nested Vectored Interrupt Controller (NVIC) offers very fast interrupt handling and provides the vector table [keil-nvic].

In addition, the NVIC:

- Saves and restores automatically a set of the CPU registers (R0-R3, R12, PC, PSR, and LR).
- Does a quick entry to the next pending interrupt without a complete pop/push sequence.
- Provides many other advanced features.

Interrupt Numbers In CMSIS, interrupts are enumerated from negative to positive numbers:

- Core interrupt numbers rank from `-15` to `-1`.
- External interrupts rank from `0` to `n`.

Exercise

Find the interrupt number definitions of the LPC1769 in the `lpc17xx.h` header file.

NVIC Functions The CMSIS core module defines a set of interrupt helper functions. For example, to enable interrupts for a given interrupt ID, you can use the function:

```
NVIC_EnableIRQ(IRQn);  // IRQn is the interrupt ID discussed above
```

For `EINT3`, `IRQn` is 21 (see Example Interrupt IDs Table). You can use this number directly or use the given name in `lpc17xx.h`: `EINT3_IRQn`.

Tip

You need two statements to enable `EINT3` on a GPIO pin:

1. Set the corresponding bit in one of the four interrupt enable registers.
2. Enable `EINT3` in NVIC.

Also, you need to write the interrupt handler: `EINT3_IRQHandler()`.

3.2.3 `EINT3` ISR

The ISR for `EINT3` is a C-function that has the following format:

```
void EINT3_IRQHandler() {
    // your code
    // clear the interrupt request
}
```

Clearing Interrupts At the end of any ISR, the corresponding interrupt should be cleared to allow future requests of the same interrupt.

To clear the interrupts of a port pin, set the corresponding bit to `1` in register `IO0IntClr` or `IO2IntClr`, depending on the port. Both registers are fields of the `LPC_GPIOINT` structure.

3.2.4 Interrupt Status

You can check for pending GPIO interrupts by reading the appropriate status register. There are 4 status registers for ports 0 and 2 that indicate whether an interrupt is pending, and whether it is triggered by a rising edge or a falling edge. They are `IO0IntStatR`, `IO2IntStatR`, `IO0IntStatF`, and ` IO2IntStatF`.

For Example, if bit 9 of `IO2IntStatR` is `1`, then P2.09 has a pending rising-edge interrupt request.

You do not need to worry about the status when you have only one external interrupt for `EINT3`. If you have multiple external interrupts, however, you can check the status registers in your ISR to determine the source of the request and, in turn, how to handle it.

All status registers are fields of the `LPC_GPIOINT` structure.

4 Tasks

4.1 One External Interrupt

Use a push-button to initiate an external interrupt. Do something interesting in the ISR!

4.2 Two External Interrupts

Use two external interrupts, where each interrupt triggers a different task. For example, each interrupt could blink an LED at a different rate.

5 Resources

[keil-nvic]
ARM Ltd. *Nested Vectored Interrupt Controller*. 2013.
http://www.keil.com/support/man/docs/gsac/gsac_nvic.htm