Embedded Systems
LA3 (C4)

Name: Ahmed ElShaarany

Date: 6/9/2015

Lab 5 Worksheet

General Interrupt Questions

1. When your program is not in an interrupt service routine, what code is it usually executing? And, what 'name' do we give this code?

   The main() thread and a while(1) loop in most cases. We call this the background thread.

2. Why keep ISR's short? That is, why shouldn't you do a lot of processing in them)?

   ISR's should be short so that they won't overload the CPU and hence the CPU can handle them in a well-timed manner and return to executing the main code.

3. What causes the MSP430 to exit a Low Power Mode (LPMx)?

   An interrupt.

4. Why are interrupts generally preferred over polling?

   Because, in case of polling, the CPU is constantly running to check for the occurrence of an event. This in turn consumes a lot of the CPU's power. On the other hand, interrupts don't require the processor to be running all the time and hence, the CPU load is very low.

Interrupt Flow

5. Name 4 sources of interrupts? (Well, we gave you one, so name 3 more.)
   Hint: Look at the chapter discussion, datasheet or User's Guide for this answer.

   a. Timer_A
   b. Timer_B
   c. GPIO (Port 1)
   d. GPIO (Port 2)
   e. A/D Converter
   f. WDT Interval Timer

6. What signifies that an interrupt has occurred?

   An interrupt flag bit is set.

   What's the acronym for these types of 'bits"?

   IFG

Setting up GPIO Port Interrupts

7. Write the code to enable a GPIO interrupt for the listed Port.Pin?

```
// GPIO pin to use: F5529 = P1.1, FR4133 = P1.2, FR5969 = P1.1
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,GPIO_PIN1); // setup pin as input
GPIO_interruptEdgeSelect(GPIO_PORT_P1,GPIO_PIN1,GPIO_LOW_TO_HIGH_TRANSITION);   //   set
edge select
GPIO_clearInterruptFlag(GPIO_PORT_P1,GPIO_PIN1); // clear individual flag
GPIO_enableInterrupt(GPIO_PORT_P1,GPIO_PIN1); // enable individual interrupt
```

8. Write the line of code required to turn on interrupts globally:

```
__bis_SR_register(GIE); // enable global interrupts (GIE)
```

   Where, in our programs, is the most common place we see GIE enabled?

   Right before the while{} loop

9. Check the interrupt that has higher priority

   WDT Interval Timer

10. Where do you find the name of an “interrupt vector” (e.g. PORT1_VECTOR)?

    msp430.h

11. Write the code to set the interrupt vector?

```
// Put’s the ISR function’s address into the Port 1 vector location

#pragma vector= PORT1_VECTOR

__interrupt void pushbutton_ISR (void){

// Toggle the LED on/off

GPIO_toggleOutputOnPin( GPIO_PORT_P1, GPIO_PIN0 );

}
```

What is wrong with this GPIO port ISR?

From the documentation, it is mentioned that all of the input pins for GPIO port 1 (P1) share a single, grouped interrupt. This means your GPIO ISR must always verify which pin actually caused an interrupt whenever the ISR is executed, and this code does not verify which pin actually caused the interrupt.

12. How do you pass a value into (or out from) and interrupt service routine (ISR)?

    Use Global Variables

13. For dedicated interrupts (such as WDT interval timer) the CPU clears the IFG flag when responding to the interrupt. How does an IFG bit get cleared for group interrupts?

    By reading the Interrupt Vector (IV) register to determine the highest-priority, pending interrupt source. We do so by reading the IV register within the context of a switch statement.

14. Creating ISR's for grouped interrupts is as easy as following a 'template'. The following code represents a grouped ISR template.
    Fill in the appropriate blank line to respond to the Port 1 pin used for the pushbutton on your Launchpad. (F5529/FR5969 = P1.1; FR4133 = P1.2)
    Add the code needed to toggle the LED (on P1.0) in response to the button interrupt.

```
#pragma vector=PORT1_VECTOR

__interrupt void pushbutton_ISR (void) {
switch(__even_in_range( P1IV, 0x10 )){
   case 0x00: break; // None
   case 0x02: break; // Pin 0
   case 0x04: GPIO_toggleOutputOnPin( GPIO_PORT_P1, GPIO_PIN0 );// Pin 1
              break;
   case 0x06: break;// Pin 2
   case 0x08: break;// Pin 3
   case 0x0A: break;// Pin 4
   case 0x0C: break;// Pin 5
   case 0x0E: break;// Pin 6
   case 0x10: break;// Pin 7
   default:   _never_executed();
}}
```

Once you implement the lab properly, answer the following questions:

1. Why do you think sometimes the button may appear as if it is not working? (try pressing the button many times in a row)

   Because toggling is probably happening multiple times faster than we can see it.

2. How can you check what is going on?

   By adding a global variable that counts how many times the ISR is run when we push the button once.

3. What did you find?

   I found that the counter actually counts that the ISR is executed twice in one push button meaning that it does toggle but then it toggles back again to the original value.

4. Why do you think it happens?

   This is probably because push-buttons are known to have a bouncing problem, meaning that they sometimes count as one push and sometimes as multiple pushes. Using a debouncer module or code for the push button would be a good idea.

Extra credit: (40 points)

When you halt execution verify you are still within the while-loop in the main program! If not, what did you miss? Fix it!

We must force the exit from lpm3 after handling the interrupt using

```
__bic_SR_register_on_exit(LPM3_bits);
```