EECE259 Quiz 4: Interrupts
Common Mistakes

Generally, many students did not demonstrate a good understanding of interrupts. This is not surprising, because it is a hard thing to master, and this was your first time trying.

The most common mistake was **assuming** that each loop iteration takes a fixed amount of time, such as 1ms or 1s. This is incorrect – loops will always run as fast as the CPU can run them! The only way to slow down a loop is to insert a deliberate time delay, such as a call to `timedDelay()`.

A few solutions tried to pause for a specific amount of time by "iteration counting" (eg, `i=50000; while(i>0) i--;`) assuming that each iteration would take only 1 or 2 instructions. This is a very unreliable way to wait for a period of time that should be precisely measured!

Many solutions lost track of the actual speed, the speed required to drive the motor, and the amount of time that had elapsed.

A few students still don't know how to read from the switches or write to the LEDs in C language (you cannot write to the switches, or read from the LEDs!).

Another common mistake is writing a loop like this:

```
int SW = *pSWITCH ;
while( SW ) {
     // loop body
}
```

Please remember that pointers (* in C) are not the same as references (& in C++). In this code, the value of `SW` in the loop will never change. This is because the assignment `SW=*pSWITCH` will **copy the value** from `*pSWITCH` to the variable `SW` **only once**, at the line where this assignment statement occurs. Each time through the loop, the value of `SW` is read, but `SW` itself never changes (it does not read `*pSWITCH` again).

NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **QUIZ 4 – 201** **Apr 1, 2011**

Write a **C program** to regulate the speed of a mobile robot using pulse width modulation (PWM) to travel at 75% of maximum speed. The motor is controlled by LEDG0: 1 turns the motor on, and 0 turns it off. The motor speed can be controlled by sending a pulse to the motor every 100ms: the longer the width of the pulse, the faster the motor will go. For example, if the motor is on for 75ms, and off for 25ms, it would *ideally* achieve 75% speed. However, due to friction and other forces, the actual robot speed is often lower.

A speedometer signal on SWITCH 0 is a pulse signal that is high for a certain percentage of the 100ms period; this is the true speed of the robot. For example, if SW0 is high for 70ms, then low for 30ms, then the robot is only travelling at 70% and not 75%. At the end of every 100ms, if the robot is travelling too slowly (below 75% speed), increase the next pulse width by 1ms; if it is travelling too fast, decrease it by 1ms.

```
#include "259macros.h"
volatile unsigned int *pCOUNTER_STATUS;   //write clrs irq, write 1 to enable irqs

/* global variables */
int time = 0___________________________________________________________________________;

int motor = 75_________________________________________________________________________;

int speed = 0_________________________________________________________________________;
```

```
int main(...)
{
        initInterrupts();

        counterEnableIRQ(ONE_MS,cntrISR);

        while(1) {

          // no code necessary in main
          // loop, everything done in ISR


          // REMEMBER: there is no concept
          // of time in this infinite loop!
          // Code here would be executed
          // repeatedly whenever the ISR
          // is not running!

        }

}
```

```
/* ISR should be called every __1ms___ */
// 1 mark for <= 1 ms ISR resolution
void cntrISR()
{
    /* remember: no waiting in here */

    // increment speed if SW0 is on
    // 1 mark for masking *pSWITCH,
    // 1 mark for tracking speed
    // speed
    speed += *pSWITCH & 0x1 ;

    // keep track of how many ms have
    // elapsed; 1 mark for tracking time
    time++ ;

    // clear the IRQ; 1 mark for clearing
    *pCOUNTER_STATUS = 1 ;

    // drive motor every 1s before limit
    // 2 marks total:
    // 1 mark for driving motor for time < motor
    // 1 mark for using *pLED
    if ( time < motor )
            *pLEDG = 1 ;
    else
            *pLEDG = 0 ;

    // every 100 ms, update limit by
    // +/- 1 if either wetness or dryness
    // was over 75ms (i.e. 75%)
    // 3 marks total:
    // 1 mark for checking speed,
    // 1 mark for modifying motor,
    // 1 mark for resetting variables
    if ( time == 100 ) {
            if ( speed < 75 )
                    motor++ ;
            else if ( speed > 75 )
                    motor-- ;

            // reset variables
            time = 0 ;
            speed = 0 ;

    }

}
```

NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **QUIZ 4 – 202** **Apr 1, 2011**

Write a **C program** to control an automated houseplant watering machine that pours water once every 60 seconds. The amount of water needed depends upon wetness and dryness sensors, but initially assume you must pour water for 30 seconds out of 60 seconds. The machine is controlled by LEDG0: when 1, the machine will continuously pour water at a steady but very slow rate; when 0, the machine does not pour water. The input on SWITCH0 is a WET sensor that is 1 while the plant is too wet, and SWITCH1 is a DRY sensor that is 1 while the plant is too dry. The goal is to keep both sensors at 00. The machine must adjust the next watering by +/-1 second, depending whether the last 60s was too wet or too dry. You should read the sensors throughout the 60 seconds, and only add more water if the DRY sensor was 1 for more than 50% of the time; likewise add less water if the WET sensor was 1 for more than 50% of the time (they will never both be 1). During the 60s, the sensors can only go from 0-to-1 and from 1-to-0 once (they will never toggle back and forth quickly).

```
#include "259macros.h"
volatile unsigned int *pCOUNTER_STATUS;   //write clrs irq, write 1 to enable irqs

/* global variables */
int time = 0_____________________________________________________________________;

int pour = 30____________________________________________________________________;

int wetness = 0 , dryness = 0____________________________________________________;
```

```
int main(...)
{
        initInterrupts();

        counterEnableIRQ(1000*ONE_MS, cntrISR);

        while(1) {

          // no code necessary in main
          // loop, everything done in ISR

          // REMEMBER: there is no concept
          // of time in this infinite loop!
          // Code here would be executed
          // repeatedly whenever the ISR
          // is not running!
          // Code here would be executed
          // repeatedly whenever the ISR

        }
}
```

```
/* ISR should be called every _1000ms_ */
// 1 mark for <= 1000 ms ISR resolution
void cntrISR()
{
    /* remember: no waiting in here */

    // every second, check switch state
    // and update variable
    // 1 mark for masking *pSWITCH,
    // 1 mark for tracking
    if ( *pSWITCH & 0x1 ) // SW0
            wetness++ ;
    else if ( *pSWITCH & 0x2 ) // SW1
            dryness++ ;

    // keep track of how many seconds
    time++ ;

    // clear the IRQ;    1 mark for clearing
    *pCOUNTER_STATUS = 1 ;

    // pour water every 1s before limit
    // 2 marks total:
    // 1 mark for pouring water when time < pour,
    // 1 mark for using *pLED
    if ( time < pour )
            *pLEDG = 1 ;
    else
            *pLEDG = 0 ;

    // every 60 seconds, update limit by
    // +/- 1 if either wetness or dryness
    // was over 30s (i.e. 50%)
    // 3 marks total:
    // 1 mark for checking speed,
    // 1 mark for modifying motor,
    // 1 mark for resetting variables
    if ( time == 60 ) {
            if ( wetness > 30 )
                    pour-- ;
                    pour++ ;

            // reset variables
            time = 0 ;
            wetness = 0 ;
            dryness = 0 ;
    }
}
```