L37-1

NAME: ____STUDENT #: ______________________

EECE 259: Introduction to Microcomputers # Lecture Quiz **Mar 30, 2011**

An **ABS brake controller** does two things at the same time. First, it monitors an input signal to indicate that the wheel is spinning. Second, if the wheel ever stops spinning, it must quickly release and engage the brakes (pulsing them) several times per second. By pulsing the brakes, the wheel is allowed to spin again and this restores tire traction. For this problem, **write a main program and 2 interrupt service routines** to:

- **Detect wheel spin.** First, your program should use interrupts with KEY3 and count the number of 0-to-1 transitions. For this, KEY3 represents an encoder attached to the wheel: if the wheel is spinning rapidly, you will get many 0-to-1 transitions per 100ms. Due to speed limits, at least 1ms will pass before the next 0-to-1 transition occurs. However, you don't know exactly when it will occur. Hence, use interrupts!
- **If no spin, pulse the brakes.** Second, your program should apply the brakes by sending a '1' to LEDG0. Every 100ms, you should check that at least 5 encoder transitions have been detected, suggesting the wheel is still spinning. . If fewer than 5 transitions have occurred, alternately apply & release the brakes every 100ms. If at least 5 transitions have been detected, the wheel is still spinning so you should apply the brakes again for the next 100ms.

```
#include "259macros.h"

/* global variables */
int counter = 0;
int brake_flag = 0;

int main(...)
{
    initInterrupts();
    enableCounterIRQ(100*ONE_MS,cntrISR);
    enableKeyIRQ( 0x8, keyISR );
    /* write your code below */
    while(1) {



      }

}
```

```
/* this ISR will be called every 100ms */
void cntrISR()
{
    /* remember: no waiting in here */



}

void keyISR()
{
    /* remember: no waiting in here */



}
```

NAME: ____ STUDENT #: ______________________

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Mar 30, 2011**

An **ABS brake controller** does two things at the same time. First, it monitors an input signal to indicate that the wheel is spinning. Second, if the wheel ever stops spinning, it must quickly release and engage the brakes (pulsing them) several times per second. By pulsing the brakes, the wheel is allowed to spin again and this restores tire traction. For this problem, **write a main program and 2 interrupt service routines** to:

- **Detect wheel spin.** First, your program should use interrupts with KEY3 and count the number of 0-to-1 transitions. For this, KEY3 represents an encoder attached to the wheel: if the wheel is spinning rapidly, you will get many 0-to-1 transitions per 100ms. Due to speed limits, at least 1ms will pass before the next 0-to-1 transition occurs. However, you don't know exactly when it will occur. Hence, use interrupts!
- **If no spin, pulse the brakes.** Second, your program should apply the brakes by sending a '1' to LEDG0. Every 100ms, you should check that at least 5 encoder transitions have been detected, suggesting the wheel is still spinning. . If fewer than 5 transitions have occurred, alternately apply & release the brakes every 100ms. If at least 5 transitions have been detected, the wheel is still spinning so you should apply the brakes again for the next 100ms.

```
#include "259macros.h"

/* global variables */
int counter = 0;
int brake_flag = 0;

int main(...)
{
    initInterrupts();
    enableCounterIRQ(100*ONE_MS,cntrISR);
    enableKeyIRQ( 0x8, keyISR );
    /* write your code below */
    while(1) {

        ; // do nothing

    }

}
```

```
/* this ISR will be called every 100ms */
void cntrISR()
{
    /* remember: no waiting in here */
    if( counter >= 5 )
            brake_flag = 1;
    else
            brake_flag = !brake_flag;

    counter = 0;
    *pLEDG = brake_flag;

    *pCOUNTER_STATUS = 1;  // clears irq

}

void keyISR()
{
    /* remember: no waiting in here */

    int keypress = *pKEY_EDGECAPTURE;
    *pKEY_EDGECAPTURE = 0; // clears irq

    if( keypress & 8 )
            counter++;

}
```

NAME: ____ STUDENT #: ______________________ L37-2

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Mar 30, 2011**

An **Alarm System** does several things at the same time. First, it blinks a **warning light** on LEDG0 (500ms on, 500ms off) to indicate whether the system is armed. Second, every 10ms it checks to make sure that no alarm has been triggered. An alarm is triggered if the system is armed and the value of **any sensor** on SW[9:4] changes from its previous state; these keys must be polled every 10ms. Third, an alarm is always triggered immediately via interrupt if the **panic buttons** on KEY3 or KEY2 are pressed. Fourth, a triggered alarm causes the alarm lights on LEDR to blink (250ms on, 250ms off). Fifth, once triggered, an alarm is cleared by setting the alarm password on SW[3..0] and then pressing the clear button on KEY1. Sixth, as the alarm is cleared, if the password remains on SW[3:0], then the system is NOT armed so no alarm can be triggered and LEDG0 remains off; if the value of SW[3:0] ever changes becomes armed; if set back to the password it only unarms by KEY1.

```
#include "259macros.h"

/* global variables */



int main(...)
{
    initInterrupts();
    enableCounterIRQ(__________,cntrISR);
    enableKeyIRQ(_____________, keyISR );
    /* write your code below */
    while(1) {




       }

}
```

```
/* ISR should be called every _____ */
void cntrISR()
{
    /* remember: no waiting in here */




}

void keyISR()
{
    /* remember: no waiting in here */




}
```

NAME: ____STUDENT #: ______________________

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Mar 30, 2011**

An **Alarm System** does several things at the same time. First, it blinks a **warning light** on LEDG0 (500ms on, 500ms off) if the system is armed but there is no alarm. Second, every 10ms it checks to make sure that no alarm has been triggered. An alarm is triggered if the system is armed and the value of **any sensor** on SW[9:4] changes from its previous state; these keys must be polled every 10ms. Third, an alarm is always triggered immediately via interrupt if the **panic buttons** on KEY3 or KEY2 are pressed. Fourth, a triggered alarm causes the alarm lights on LEDR to blink (250ms on, 250ms off). Fifth, once triggered, an alarm is cleared by setting the alarm password on SW[3..0] and then pressing the clear button on KEY1. Sixth, as the alarm is cleared, if the password remains on SW[3:0], then the system is NOT armed so no alarm can be triggered and LEDG0 remains off; if the value of SW[3:0] ever changes becomes armed; if set back to the password it only unarms by KEY1.

```
#include "259macros.h"

/* global variables */
enum Modes { IDLE, ARMED, ALARM };        int count = 0;
enum Modes mode = IDLE;                   int red   = 0;
int sensors;                              int green = 0;
int password = 0xB;                       int oldpw = 0xB;
```

```
int main(...)
{
    initInterrupts();
    enableCounterIRQ( 10*ONE_MS ,cntrISR);
    enableKeyIRQ(  8 | 4 | 2  , keyISR );

    /* write your code below */
    while(1) {

    }
}
```

```
/* ISR should be called every 10ms  */
void cntrISR()
{
    /* remember: no waiting in here */
    *pCOUNTER_STATUS = 1; // clear irq

    count++;
    if( mode==ARMED && (count%50==0) ) {
            green = !green;
            count = 0;
    }

    if( mode==ALARM && (count%25==0) ) {
            red   = !red;
            count = 0;
    }

    *pLEDR = red;
    *pLEDG = green;
}

void keyISR()
{
    /* remember: no waiting in here */
    int key = *pKEY_EDGECAPTURE;
    *pKEY_EDGECAPTURE = 0;
    if( key & (8|4) ) {
            mode = ALARM;
            red  = 1;
    }
}
```

NOTE: This is only a PARTIAL SOLUTION. You must complete it yourself. Hint: the **main program** and **both ISRs** are all **incomplete**.