**CG1112 Engineering Principles and Practices**

**Week 4 Studio 1 – Interrupts**

**Answer Book**

**Student Number: A0214561M**

**Name: Zhuang Jianning**

**Pre-start Check (Check Fail: -5 mark penalty)**

**Did you name your file AxxxxxxY.docx, where AxxxxxxY is your student number? -2 marks if NO.**

**Did you fill in your student number and name above? -3 marks if NO.**

Question 1 (3 MARKS)

#define REDPIN 11

#define GREENPIN 12

#define SWITCHPIN 2

#define LED\_DELAY 100

static volatile int turn = 0; // 0 = green, 1 = red

void setup() {

pinMode(REDPIN, OUTPUT);

pinMode(GREENPIN, OUTPUT);

pinMode(SWITCHPIN, INPUT);

}

void testButton(){

int buttonState = digitalRead(SWITCHPIN);

if (buttonState == HIGH){

turn = 1 - turn;

}

}

void flashGreen(){

int counter = 1;

while(turn==0){

for(int i=0; i<counter; i++){

digitalWrite(GREENPIN, HIGH);

delay(LED\_DELAY);

digitalWrite(GREENPIN, LOW);

delay(LED\_DELAY);

}

counter++;

delay(1000);

testButton();

}

}

void flashRed(){

int counter = 1;

while(turn==1){

for(int i=0; i<counter; i++){

digitalWrite(REDPIN, HIGH);

delay(LED\_DELAY);

digitalWrite(REDPIN, LOW);

delay(LED\_DELAY);

}

counter++;

delay(1000);

testButton();

}

}

void loop() {

if(turn == 0)

flashGreen();

if(turn == 1)

flashRed();

}

When the button is pressed, the other LED should start flashing in the given pattern. This can be done by checking if the button has been pressed, such that the switch is closed and input D2 reads HIGH, then flipping the value of turn from 0 to 1 or 1 to 0. We can do this check within the while loop of flashGreen() and flashRed() by inserting the following function so a button press can break the loop.

void testButton(){

int buttonState = digitalRead(SWITCHPIN);

if (buttonState == HIGH){

turn = 1 - turn;

}

}

Question 2a. (3 MARKS)

The attachInterrupt function is used to specify an Interrupt Service Routine (ISR) to process interrupts. The recommended syntax of attachInterrupt is:

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)

The first parameter to attachInterrupt() is an interrupt number. It is recommended to use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number as the specific pins with interrupts and their mappings to interrupt numbers may vary for different boards. pin is the actual Arduino pin number.

The second parameter is the ISR to call when the interrupt occurs. This function must take no parameters and return nothing. The code within is executed when the interrupt occurs, usually modifying a global variable which acts as data passed between ISR and main.

The third parameter is the mode which defines when the interrupt should be triggered. The interrupt mode RISING means the interrupt is triggered when the pin goes from LOW to HIGH.

There are 4 modes predefined as valid values:

LOW: triggers the interrupt whenever the pin is LOW

CHANGE: triggers the interrupt whenever the pin changes value (HIGH to LOW or LOW to HIGH)

RISING: triggers the interrupt when the pin goes from LOW to HIGH

FALLING: triggers the interrupt when the pin goes from HIGH to LOW

Question 2b. (4 MARKS)

The attachInterrupt function specifies an ISR (switchISR) to trigger an interrupt whenever pin 2 goes from LOW to HIGH (RISING).

At each instruction cycle, the processor samples the interrupt trigger signal. In our case, whenever pin 2 goes from LOW to HIGH due to our external stimuli of pressing the button, it triggers an interrupt which stops whatever the processor is doing by asserting one of the Interrupt Request (IRQ) lines, and starts the execution of an Interrupt Service Routine(ISR). The ISR tells the processor what to do when the interrupt occurs. In our case, the switchISR flips the value of the global variable onOff. Once that ISR code has executed, the processor goes back to whatever it was originally doing. The LED will switch ON or OFF depending on the value of onOff.

Question 3a. (5 MARKS)

#define REDPIN 11

#define GREENPIN 12

#define SWITCHPIN 2

#define LED\_DELAY 100

// This variable decides which LED's turn it is to flash.

// 0 = green, 1 = red

static volatile int turn = 0;

void switchISR(){

turn = 1 - turn;

}

void setup(){

pinMode(REDPIN, OUTPUT);

pinMode(GREENPIN, OUTPUT);

attachInterrupt(digitalPinToInterrupt(SWITCHPIN), switchISR, RISING);

}

void flashGreen(){

int counter=1;

while(turn==0){

for(int i=0; i<counter; i++){

digitalWrite(GREENPIN, HIGH);

delay(LED\_DELAY);

digitalWrite(GREENPIN, LOW);

delay(LED\_DELAY);

}

counter++;

delay(1000);

}

}

void flashRed(){

int counter=1;

while(turn==1){

for(int i=0; i<counter; i++){

digitalWrite(REDPIN, HIGH);

delay(LED\_DELAY);

digitalWrite(REDPIN, LOW);

delay(LED\_DELAY);

}

counter++;

delay(1000);

}

}

void loop() {

if(turn == 0)

flashGreen();

if(turn == 1)

flashRed();

}

Again, the attachInterrupt function specifies an ISR (switchISR) to trigger an interrupt whenever pin 2 goes from LOW to HIGH (RISING).

Whenever we press the button, the switch is closed and SWITCHPIN (pin 2) changes from LOW to HIGH. The ISR is invoked and the value of turn is flipped. This will cause a break in the while loop within flashGreen() or flashRed() and return control to the loop() function which then causes the other LED to flash based on the value of turn.

Question 3b. (3 MARKS)

The interrupt version works better. The processor samples the interrupt signal at each instruction cycle and the value of turn is flipped whenever pin 2 changes from LOW to HIGH.

However, for the polling version, we can only digitalRead the value of the SWITCHPIN after the for loop used to flash the LED. There is also a 1 second delay before the next iteration of the while loop so the value of digitalRead(SWITCHPIN) might not be HIGH at that exact moment and we do not flip the value of turn. Hence, the button does not appear to work as reliably.

Question 4a. (3 MARKS)

#define REDPIN 11

#define GREENPIN 12

#define SWITCHPIN 2

#define LED\_DELAY 100

#define THRESHOLD 200

// This variable decides which LED's turn it is to flash.

// 0 = green, 1 = red

static volatile int turn = 0;

static unsigned long lastTime = 0;

static unsigned long currTime;

void switchISR(){

currTime = millis();

if (currTime - lastTime > THRESHOLD){

lastTime = currTime;

turn = 1 - turn;

}

}

To determine if a switch press is valid, we check if the interval after the previous recorded keypress is long enough so that we ignore any contacts made due to switch bouncing. We declare the global variables currTime and lastTime = 0 and add a check for the interval currTime – lastTime within the ISR. We only flip the value of turn if it is a valid switch press and update lastTime with that value of currTime. After some trial and error, a THRESHOLD of 200 milliseconds produces reliable results after each intentional key press.

Question 4b. (2 MARKS)

We were able to find a good value for THRESHOLD that returned more reliable results compared to the non-debounced version. When THRESHOLD was set to 200, the LED changes reliably when the button is pressed. It is also responsive enough that successive presses longer than 200 milliseconds apart are considered valid keypresses. When THRESHOLD was set to larger values, the button is less responsive to intentional successive presses. When THRESHOLD was set to smaller values, the LED will not switch sometimes possibly due to bouncing.

Question 4c. (2 MARKS)

If THRESHOLD was too small, it might recognize a later contact bounce to be a valid key press because lastTime is still set as the last valid RISING and is not updated after each RISING.

If THRESHOLD was too big, it would ignore successive valid presses of the button that are shorter than the time interval.

Question 5 (5 MARKS)

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#define LED\_DELAY 100

static volatile int turn = 0;

void flashGreen(){

int count=1;

int i;

while(turn==0){

for(i=0; i<count && turn==0; i++){

// Switch green LED at pin 12 on. Pin 12 is PB4

PORTB |= 0b00010000;

// Delay 250ms

\_delay\_ms(LED\_DELAY);

PORTB &= 0b11101111;

\_delay\_ms(LED\_DELAY);

}

\_delay\_ms(1000);

count++;

}

}

void flashRed(){

int count=1;

int i;

while(turn==1){

for(i=0; i<count && turn==1; i++){

// Switch green LED at pin 12 on. Pin 12 is PB4

PORTB |= 0b00001000;

// Delay 250ms

\_delay\_ms(LED\_DELAY);

PORTB &= 0b11110111;

\_delay\_ms(LED\_DELAY);

}

\_delay\_ms(1000);

count++;

}

}

void setup()

{

// Set pins 11 and 12 to output. Pin 11 is PB3, pin 12 is PB4

DDRB |= 0b00011000;

EICRA |= 0b00000010;

EIMSK |= 0b00000001;

sei();

}

ISR(INT0\_vect){

turn = 1 - turn;

}

void loop()

{

if(turn==0)

flashGreen();

else

flashRed();

}

We switch to bare-metal programming

**For TA Use:**

**Studio Marks: \_\_\_\_\_\_\_\_\_\_\_ / 30**

**PENALTIES: \_\_\_\_\_\_\_\_\_\_\_\_/-5**

**Total: \_\_\_\_\_\_\_\_\_\_\_\_/ 30**