CG1112 Engineering Principles and Practices II for CEG

Week 4 Studio 1 – Interrupts

GRADED STUDIO

INTRODUCTION

In this studio we will be looking at a very important topic in Computer Engineering: Interrupts! Briefly summarizing the lecture, an interrupt is a special hardware signal that CPUs (and MCUs) provide so that hardware systems can ask for attention from the CPU.

One common example of where interrupts are used is in your keyboard and mouse. Your CPU goes about minding its own business rendering stuff like Goat Simulator for your enjoyment, but when you press a key or move the mouse, the hardware for the keyboard or mouse (specifically, the USB port the mouse or keyboard are connected to), triggers an interrupt signal and the CPU immediately stops rendering Goat Simulator to see which key you pressed, or how you have moved your mouse.

This studio consists of four activities:

Activity 1: In the first activity you will write a program that flashes one LED rapidly, until you press a button, and then it flashes the other LED rapidly, until you press a button again.

Activity 2: In the second activity, you will repeat what you did in Activity 1, but this time using interrupts.

Activity 3: In this activity you will learn to deal with switch bounces.

Activity 4: You will work on a bare-metal version of Activity 2, and in the process learn how to do interrupt programming on the AVR without using the Arduino library.

This is a graded lab, and you should type in your answers in a WORD file called <student-id>.docx, where <student-id> is your student ID on your student card. You may work on the Studio with other team members, but the thoughts and insights in the writeup and submission should be completely your own. Do not copy work from your team-mates.

So if your student ID is A1495832X, then your word file should be called A1495832X.docx. Please upload to your respective group's workbin by the end of the studio.

GROUP SETUP AND EQUIPMENT/COMPONENTS

You should work in groups of 2 or 3. The following will be provided to you:

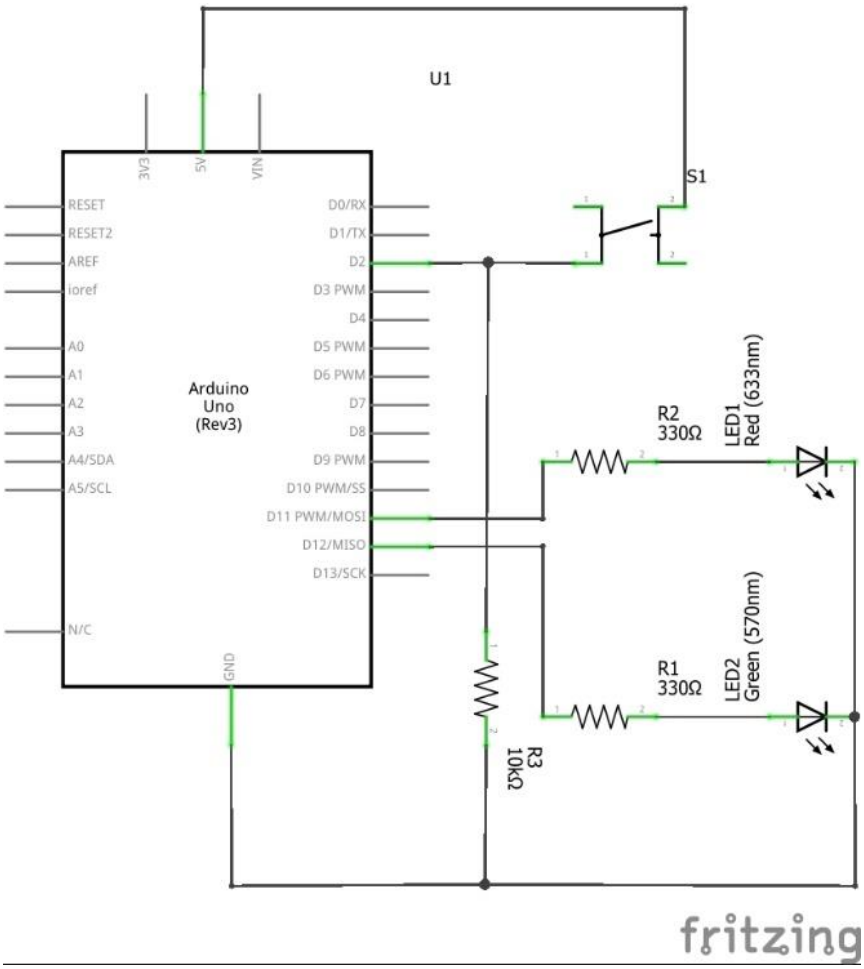
Arduino UNO	x1
LED, Red	x1
LED, Green	x1
Resistor, 330 ohm, 0.25W	x2
Resistor, 10K, 0.25W	x1
Pushbutton Switch	x1
Breadboard	x1
Jumper wires	

Please use the cutters/strippers provided to you.

ACTIVITY 1 – POLLING USING ARDUINO WIRING LANGUAGE

Step 1

To begin this activity, assemble the following circuit using the components given to you:



Step 2

Open the Arduino IDE and load in the sketch called w4s1.ino **AND SAVE IT AS w4s1p1.ino**. Please make sure you do not modify the original w4s1.ino sketch as you will need it again in Activity 2.

The code for this sketch is shown below (It is provided to you, you do not need to type it in):

```
#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 setup() {
  // put your setup code here, to run once:

  pinMode(REDPIN, OUTPUT);
  pinMode(GREENPIN, OUTPUT);
  pinMode(SWITCHPIN, INPUT);
}

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() {
  // put your main code here, to run repeatedly:

  if(turn == 0)
    flashGreen();

  if(turn == 1)
    flashRed();
}
```

Compile and run the program, uploading it to your Arduino, and confirm that the green LED is flashing once, pausing for a second, then twice, pausing for a second, then three times, etc.

Question 1. (3 marks)

Now modify your program so that flashGreen flashes the green LED as described above, but when you press the button, flashRed starts flashing the red LED, again as described above. When you press the button, flashGreen will start flashing the green LED, etc. Demo your program to your TA, and cut and paste your code to your report and explain your code.

NOTE: You may find that your button doesn't work very reliably; e.g. it may take a few presses before the other LED starts flashing. This is normal and we will learn how to fix this in Activity 3.

ACTIVITY 2 – INTERRUPTS USING THE ARDUINO WIRING LANGUAGE

Step 1

Using the same circuit as in Activity 1, create a new Arduino sketch called w4s1int.ino, and key in the following code.

```
#define buttonPin     2
#define LEDPIN        12

static volatile int onOff = 0;

void switchISR()
{
  onOff = 1 - onOff;
}
void setup() {
  // put your setup code here, to run once:
  attachInterrupt(digitalPinToInterrupt(buttonPin), switchISR, RISING);
  pinMode(LEDPIN, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  if(onOff == 0)
    digitalWrite(LEDPIN, LOW);
  else
    digitalWrite(LEDPIN, HIGH);
}
```

Compile and run the program. You will see that none of the LEDs light up. Press the button and the green LED will light up (again you may need to press the button more than once). Press the button again and the green LED will go off.

Question 2a. Look up and explain what the attachInterrupt function does and what arguments it takes. In this example we set the interrupt mode to RISING. What does RISING mean? List down the other possible modes and explain how they are different from RISING. (3 marks)

Question 2b. Your sketch does not actually read pin 2 (i.e. there is no "digitalRead" call), where the switch is connected to. Explain, based on your understanding of interrupts, how your sketch is able to switch the LED on and off when you press the button. (4 marks)

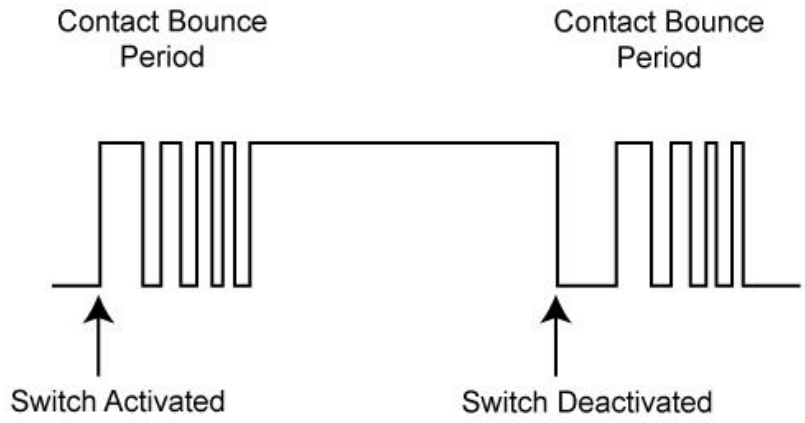
Open up w4s1.ino again, **AND SAVE IT AS w4s1p2.ino**. Ensure that your unmodified sketch looks exactly like the unmodified w4s1.ino shown on pages 3 to 4 in Activity 1.

Question 3a. Modify your sketch in w4s1p2.ino to use interrupts to switch between flashing the green LED and flashing the red LED. Cut and paste the code into your report, explaining what you have done, and show your TA what you have done. (5 marks)

Question 3b. Does the polling version or interrupt version work better? Examine the code and explain why. (3 marks)

ACTIVITY 3 – FIXING THAT ANNOYING SWITCH PROBLEM

In Activities 1 and 2 we see that the switch is not operating reliably. To understand why, we have to remember that the push button switch (and just about all switches) are mechanical devices with metal springs. When you push the switch, the metal contacts within the switch vibrates, causing the switch contacts to open and close repeatedly during the "contact bounce period". This is shown in the diagram below:



Effectively this is like as though you are pressing the switch repeatedly, causing the LED to switch on and off too quickly to be seen, making it appear as though the switch press wasn't registered at all.

To fix this problem, we note from the diagram above that the "keypresses" made by the vibrations are very rapid. If we ignore key presses with very short intervals, we are effectively ignoring any contacts made due to switch bouncing.

This is called "debouncing".

How do we do this on the Arduino? The Arduino library has a millis() function that returns the number of milliseconds since the Arduino was powered up.

We can then use the following algorithm to debounce the switch:

```
currTime = millis();
if(currTime – lastTime > THRESHOLD)
{
   lastTime = currTime;
   /* DO WHATEVER WE NEED TO DO WHEN WE PRESS THE SWITCH */
}
```

This algorithm checks the current "time" as returned by millis, and subtracts away the last time we recognized a switch press (tracked by "lastTime"). If the difference is more than a certain number of milliseconds given by THRESHOLD, we recognize this as a switch press, and save the time in lastTime.

Question 4a. (3 marks)

Modify the appropriate part of w4s1p2.ino to debounce the switch.

Note that you must declare lastTime as an unsigned long (since millis() returns an unsigned long) and initialize it to 0. You must also declare currTime as an unsigned long. Both must be declared as global variables.

You must also #define THRESHOLD to an appropriate value in milliseconds , found by trial and error.

```
#define THRESHOLD 10 // Example value. Find your own.
unsigned long lastTime=0, currTime;
```

Cut, paste and explain the changes you made to your report. (3 marks)

Question 4b. (2 marks)

Are you able to find an optimum value for THRESHOLD that gives more reliable button operation than the non-debounced version of your program? If yes, what is the value you found? If no, why

Question 4c. (2 marks)

What do you think would happen if THRESHOLD was too small? Too big?

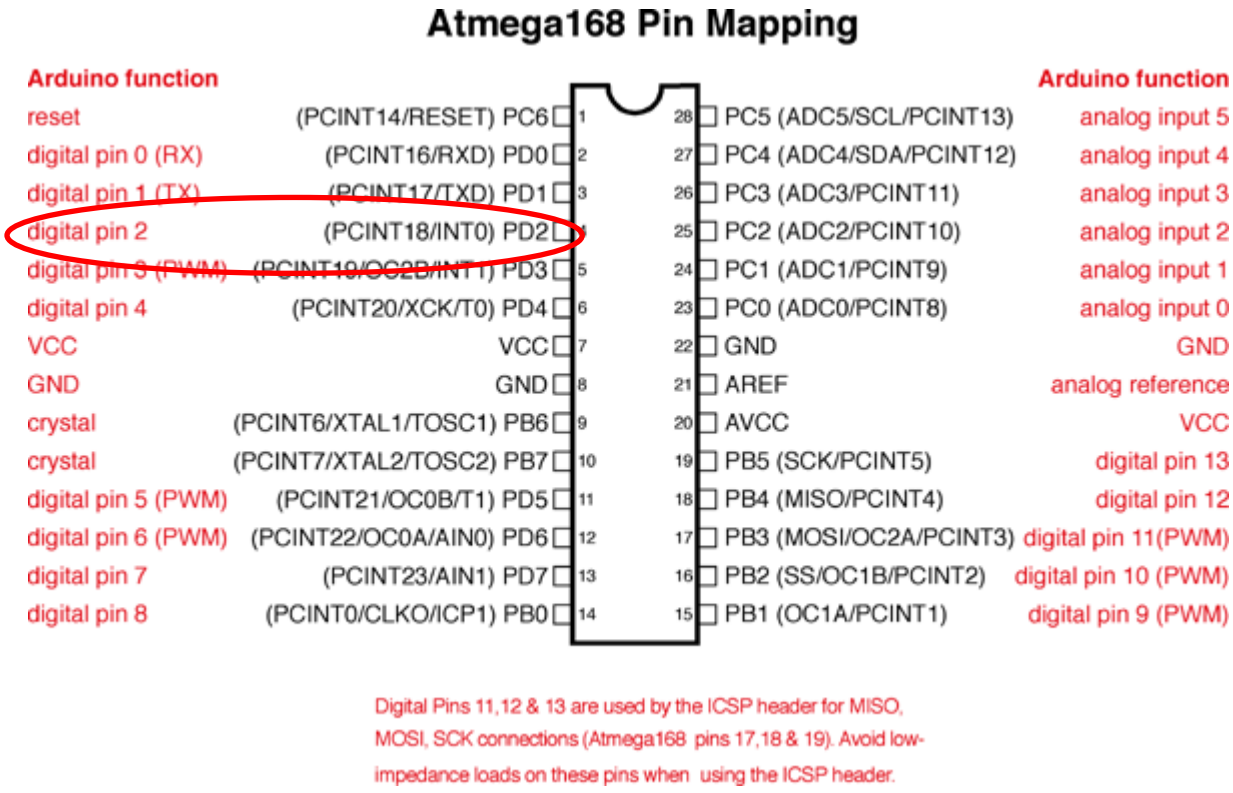
ACTIVITY 4 – INTERRUPTS USING BARE-METAL AVR PROGRAMMING

We have used the Arduino Wiring Language to do polling and interrupt programming. Now we will see how to do interrupt programming using the bare-metal AVR libraries.

The AVR MCU has 26 possible interrupts, some of which are shown in the table below:

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A

We know that our push button is connected to digital pin 2 on the Arduino. Looking at the following diagram that maps Arduino pins to the pins on the Atmega328P (diagram actually says Atmega168. This is ok, the Atmega168 is just a 16 KB version of the Atmega328P, and the P just means that our MCU supports low power operations):



From the diagram above we can see that digital pin 2 (circled) is INT0, and looking at the table of interrupts shown above, we see that INT0 is External Interrupt Request 0. We will now begin looking at how to program interrupts in AVR.

Step 1

We will now write the bare-metal version of our w5s1int.ino sketch in Step 1 of Activity 2. Our program will do two things:

i) Set the appropriate bits in the External Interrupt Control Register A (EICRA) so that the interrupt is triggered on the rising edge of the signal on digital pin 2:

For your convenience we have reproduced the diagram on Page 89 on how to configure INT0 (and INT1 – which is on digital pin 3 of the Arduino):

17.2.1. External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Name: EICRA
Offset: 0x69
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:2 – ISC1n: Interrupt Sense Control 1 [n = 1:0]

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in the table below. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Value	Description
00	The low level of INT1 generates an interrupt request.
01	Any logical change on INT1 generates an interrupt request.
10	The falling edge of INT1 generates an interrupt request.
11	The rising edge of INT1 generates an interrupt request.

Bits 1:0 – ISC0n: Interrupt Sense Control 0 [n = 1:0]

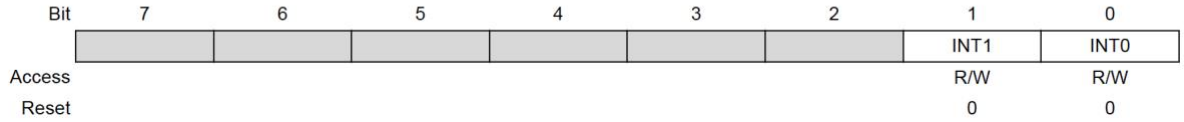
The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in table below. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Value	Description
00	The low level of INT0 generates an interrupt request.
01	Any logical change on INT0 generates an interrupt request.
10	The falling edge of INT0 generates an interrupt request.
11	The rising edge of INT0 generates an interrupt request.

Based on this table, we see we must set ISC01 and ISC00 to 11.

We must also set the appropriate bits in the EIMSK register:

Name: EIMSK
Offset: 0x3D
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x1D



Bit	7	6	5	4	3	2	1	0
							INT1	INT0
Access							R/W	R/W
Reset							0	0

We also need to call sei() to ensure that interrupts are turned on.

ii) Our code will capture the interrupt vector for INT0, called INT0_vec, using the provided ISR macro.

Now create a new sketch called w4s1int2, and key in the following code:

```
#include <avr/io.h>
#include <avr/interrupt.h>

static volatile int onOff=0;

/* Our setup routine */
void setup()
{
	// Set the INT0 interrupt to respond on the RISING edge.
	// We need to set bits 0 and 1
	// in EICRA to 1

	EICRA |= 0b00000011;

	// The green LED is on Pin 12, which is on PB4. Set it to OUTPUT
	DDRB |= 0b00010000;

	// Activate INT0
	EIMSK |= 0b00000001;

	// Ensure that interrupts are turned on.
	sei();
}

/* Here we declare the Interrupt Service Routine (ISR) for INT0_vect */
ISR(INT0_vect)
{
	onOff=1-onOff;
}

/* Our setup routine */
void loop()
{
	// Switch off the LED if onOff is 0, otherwise switch it on.
	if(onOff==0)
		PORTB &=0b11101111;
	else
		PORTB |= 0b00010000;
}
```

Upload the sketch, then press the switch to verify that it works.

Step 2.

The w4s1p3.ino file provided to you is the bare metal version of the w4s1p1.ino Arduino sketch in Activity 1. The code is reproduced here for your convenience (**you do not need to type it in**):

```
#include <avr/io.h>

// Remember to #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;
}

void loop()
{
	if(turn==0)
		flashGreen();
	else
		flashRed();
}
```

Question 5. (5 marks)

Modify the code in the w5s1p3 project to use interrupts to change between flashing the green LED and the red LED.

Instead of configuring your interrupt to be triggered on the rising edge, configure it to be triggered on the FALLING edge.

Cut and paste the code into your report, and show what you have done to your TA.

Notice that we are not doing a debounced version in bare-metal. We will revisit this topic at a later studio when we look at programming the AVR timer.