Class:	**CPE301L Digital Systems Architecture and Design 1001**		Semester:	**Spring 2025**
Points		Document author:	**Narek Kalikian**	
		Author's email:	**kalikn1@unlv.nevada.edu**	
		Document topic:	**Postlab 3**	
Instructor's comments:				

1. Introduction / Theory of Operation

In this lab, we learned about and implemented interrupts. We learned how to create time delays using internal libraries to access the util/delay.h header file. Furthermore, we learned how to enable and disable interrupts, use both external and Pin Change interrupts, and differentiate between interrupts and polling.

2. Prelab Content

A.) Explaining Each Bit of Listed Registers:

EIMSK (External Interrupt Mask) Register:

- Bit 0 (INT0) - External interrupt request 0 Enable
  - 1 = Enable, 0 = Disable
- Bit 1 (INT1) - External interrupt request 1 Enable
  - 1 = Enable, 0 = Disable
- Bit 2 (INT2) - External interrupt request 2 Enable
  - 1 = Enable, 0 = Disable
- Bit 3 (INT3) - External interrupt request 3 Enable
  - 1 = Enable, 0 = Disable
- Bit 4 (INT4) - External interrupt request 4 Enable
  - 1 = Enable, 0 = Disable
- Bits 5 - 7 - Not used

EIFR (External Interrupt Flag) Register:

- Bit 0 (INTF0) - External interrupt 0 flag
  - Set when interrupt occurs
- Bit 1 (INTF1) - External interrupt 1 flag
  - Set when interrupt occurs
- Bit 2 (INTF2) - External interrupt 2 flag

    - Set when interrupt occurs
- Bit 3 (INTF3) - External interrupt 3 flag
    - Set when interrupt occurs
- Bit 4 (INTF4) - External interrupt 4 flag
    - Set when interrupt occurs
- Bits 5 - 7 - Not used

EIRCA (External Interrupt Control A) Register:

- Bit 0 (ISC00) and Bit 1 (ISC01) - INT0 interrupt sense controls
- Bit 2 (ISC10) and Bit 3 (ISC11) - INT1 interrupt sense controls
- Bit 4 (ISC20) and Bit 5 (ISC21) - INT2 interrupt sense controls
- Bit 6 (ISC30) and Bit 7 (ISC31) - INT3 interrupt sense controls

SEI (Set Global Interrupt Enable) Register:

- The I-bit is used. It is set in the status register to enable global interrupts. This register is used after configuring interrupt sources (which will be discussed below)

CLI (clear Global Interrupt Enable) Register:

- Similar to the SEI register, the I-bit is once again used. Clearing this bit in the status register disables global interrupts. The interrupts are disabled temporarily.

TIMSK1 (Timer 1 Interrupt Mask) Register:

- Bit 0 (OCIE1A) - Output compare A match interrupt enable
- Bit 1 (OCIE1B) - Output compare B match interrupt enable
- Bit 2 (OCIE1C) - Output compare C match interrupt enable
- Bit 3 - Not used
- Bit 4 (ICIE1) - Input capture interrupt enable for timer 1
- Bits 5 - 7: Not used

B.)

- Interrupt Service Routine: A callback subroutine that is triggered when an interrupt is received. ISRs are used for tasks that need to be handled immediately. They need to happen quickly to prevent the loss of data.
- AVR Interrupt Sources:
    - External Interrupts/ Port Pins (INT0, INT1, INT2)
    - Timers/Counters
    - EEPROM (EE_RDY)
    - ADC Conversion Complete
    - Analog Comparator (ANA_COMP)
    - Serial Transfer Complete (SPI, STC)

C.)

- Switch Debouncing: The process of removing unwanted noise from input elements such as buttons, switches, etc. This is used to prevent extra activations or slower functions from triggering more often than necessary.
- Avoid Using Hardware: Use a resistor-capacitor in conjunction with a Schmitt trigger diode to smooth out the transition.
- Software: Switch debouncing can be achieved by creating a delay in software. The software program then checks the appropriate pin level and performs necessary functions after the delay has occurred.

D.)

RET:

- Standard return instruction used in regular function calls
- Used to exit a function or a subroutine
- Interrupts are not re-enabled
- Interrupts remain in their current state(s)

RETI:

- Return from interrupt instruction used interrupt service routines (ISRs)
- Used specifically to exit an ISR
- I-bit is set in the status register to re-enable interrupts
- Interrupts are enabled after execution

3. Description of Experiments

Experiment 1 - Code:

Lab3_Exp1.c | C:\Users\Narek\Documents\Atmel Studio\7.0\GccApplication4\GccApplication4\Lab3_Exp1.c

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

ISR(INT0_vect)
{
    cli(); // Disable global interrupts
    while (!(PIND & (1 << PIND2))) {  // Check if PD2 (INT0 pin) is low
        PORTC = 0xff;  // Turn on PORTC (all pins high)
        _delay_ms(300);
        PORTC = 0x00;  // Turn off PORTC
        _delay_ms(300);
    }
    sei(); // Enable global interrupts before breaking
}

int main(void)
{
    DDRC = 0xff;   // Set PORTC as output
    DDRD &= ~(1 << PIND2);  // Set PD2 (INT0 pin) as input
    PORTD |= (1 << PIND2);  // Enable pull-up resistor on PD2

    EIMSK = 0x01;  // Enable INT0 (external interrupt 0)
    EICRA = 0x03;  // Set INT0 to trigger on rising edge (for example)

    sei();  // Enable global interrupts
    while (1)
    {
        PORTC = 0xff;   // Set PORTC high
        _delay_ms(1000);
        PORTC = 0x00;   // Set PORTC low
        _delay_ms(1000);
    }
}
```

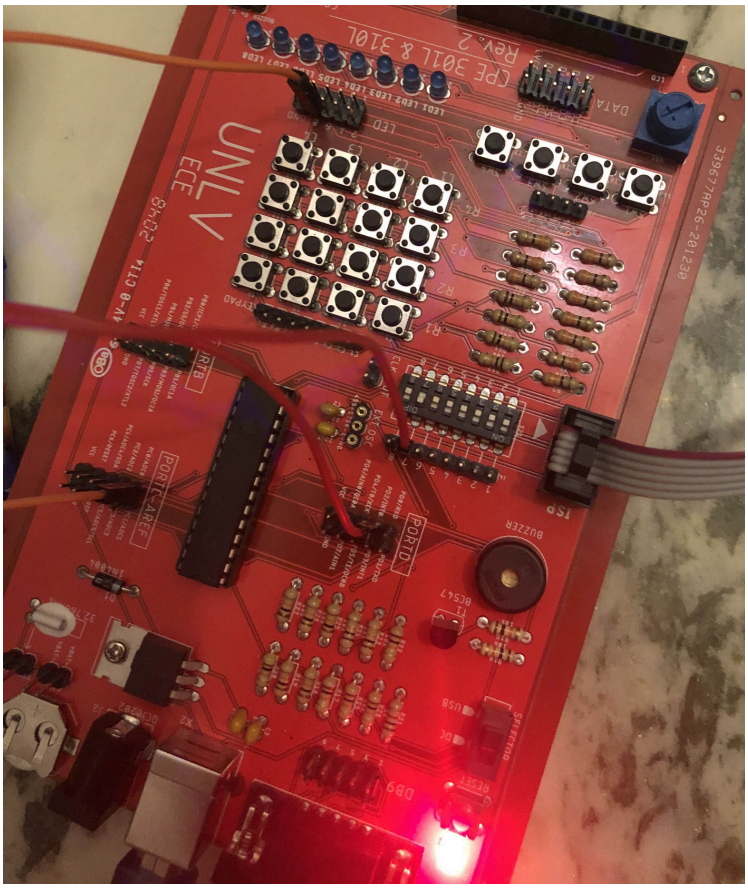
Output

Show output from: Build

```
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "C:\Users\Narek\Documents\Atmel Studio\7.0\GccApplication4\Gcc
Done building target "Build" in project "GccApplication4.cproj".
Done building project "GccApplication4.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Experiment 1 - Board Photo:



Experiment 2 - Code:

Lab3_Exp2.c

main | int main(void)

```
// ISR for INT0
ISR(INT0_vect)
{
    cli(); // Disable global interrupts
    while (!(PIND & (1 << PIND2))) {  // Check if PD2 (INT0 pin) is low
        PORTC = 0xff;  // Turn on PORTC (all pins high)
        _delay_ms(300);
        PORTC = 0x00;  // Turn off PORTC
        _delay_ms(300);
    }
    sei(); // Enable global interrupts before breaking
}

// ISR for INT1
ISR(INT1_vect)
{
    cli(); // Disable global interrupts
    while (1) {
        PORTC = 0x00; // Turn off PORTC (all pins low)
        PORTB = 0xff; // Turn on PORTB (all pins high)
        _delay_ms(300);
        PORTB = 0x00; // Turn off PORTB (all pins low)
        _delay_ms(300);
    }
    sei(); // Enable global interrupts
}

int main(void)
{
    DDRC = 0xff;   // Set PORTC as output
    DDRD &= ~(1 << PIND2) & ~(1 << PIND3);  // Set PD2 (INT0 pin) and PD3 (INT1 pin) as input
    PORTD |= (1 << PIND2) | (1 << PIND3);  // Enable pull-up resistors on PD2 and PD3
    DDRB = 0xff;   // Set PORTB as output

    EIMSK = 0x03;  // Enable INT0 (external interrupt 0) and INT1 (external interrupt 1)
    EICRA = 0x0F;  // Set INT0 and INT1 to trigger on rising edge

    sei();  // Enable global interrupts
    while (1)
    {
        PORTC = 0xff;   // Set PORTC high
        _delay_ms(1000);
        PORTC = 0x00;   // Set PORTC low
        _delay_ms(1000);
    }
```

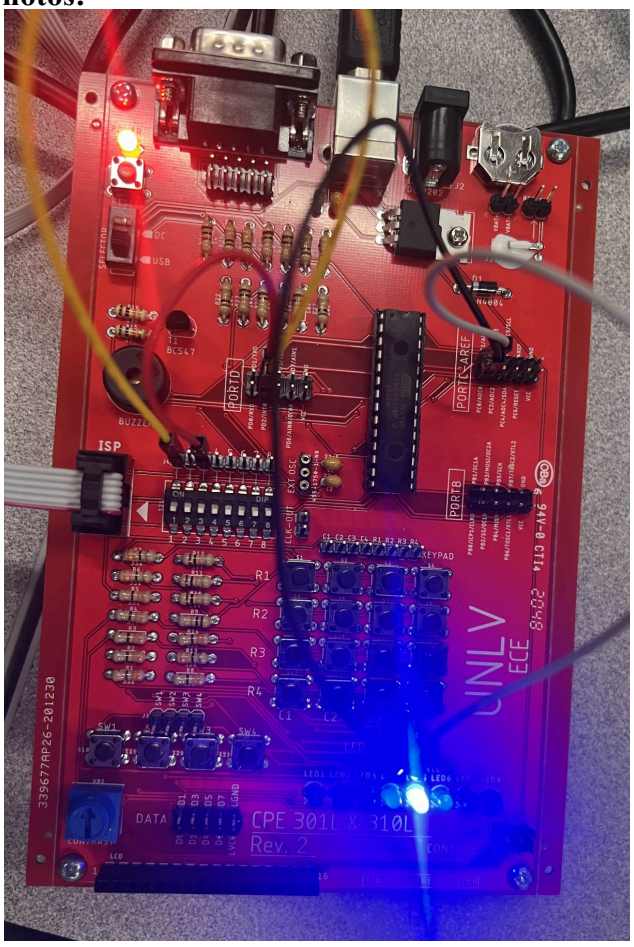
100 %

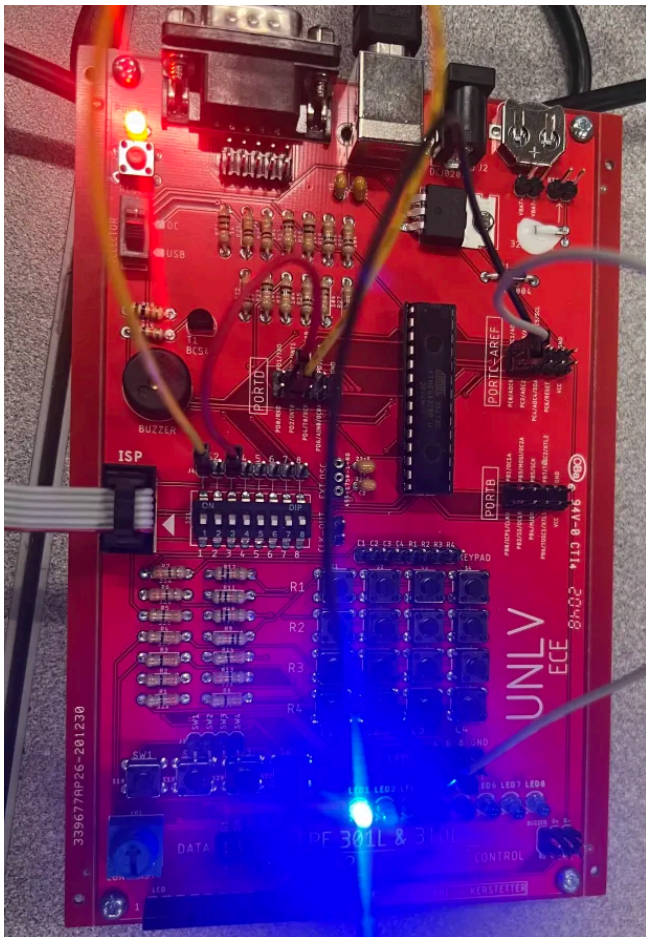
Output

Show output from: Build

========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========

Experiment 2 - Board Photos:





Experiment 3 - Code:

Lab3_Exp3.c

main | int main(void)

```
#define F_CPU 16000000;

ISR(PCINT0_vect) // calls int0
{
    sei(); // set the interrupt
    while (1) {
        PORTB = 0b11110000; // set upper bits high
        _delay_ms(500); // add delay
        PORTB = 0b01110000; // set 3 of the upper bits high
        _delay_ms(500); // add delay
        PORTB = 0b00110000; // set 2 of the upper bits high
        _delay_ms(500); // add delay
        PORTB = 0b00010000; // set only 1 of the upper bits high
        _delay_ms(500); // add delay
        PORTB = 0b00000000; // set all the bits low
        _delay_ms(500); // add delay
    }
}

ISR(PCINT1_vect) // calls int1
{
    sei(); // set the interrupt
    while (1) {
        PORTB = 0b11000000; // set 2 of the upper bits high
        _delay_ms(500); // add delay
        PORTB = 0b01100000; // set 2 of the upper bits high in a different pattern
        _delay_ms(500); // add delay
        PORTB = 0b00110000; // set 2 of the upper bits high in a different pattern
        _delay_ms(500); // add delay
        PORTB = 0b10010000; // set 2 of the upper bits high in a different pattern
        _delay_ms(500); // add delay
    }
}

int main(void)
{
    DDRC = 0xFE; // set all of the bits high except one to reserve for push button
    DDRB = 0xFE; // set all of the bits high except one to reserve for push button
    PCMSK0 |= (1<<PCINT0); // enable the pin change interrupt
    PCMSK1 |= (1<<PCINT8); // enable the pin change interrupt
    PICR |= (1<<PCIE1) | (1<<PCIE0); // set the interrupts to be control with PCIEs
    sei(); // set the global interrupt
    while(1) // infinite loop
    {
    }
```

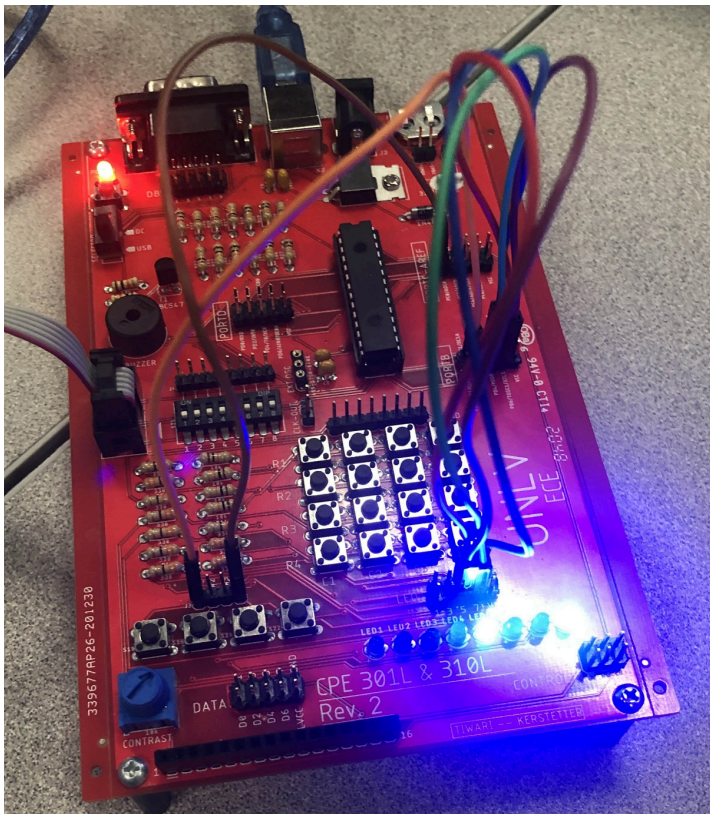
100 %

Output

Show output from: Build

========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========

Experiment 3 - Board Photo:



4. Questions and Answers

1. Context Saving and Importance for Interrupts: Context saving is the process of saving the current state before an interrupt occurs. It allows the current state to be returned to and executed after the interrupt. Context switching is important for interrupts because it prevents data loss and also allows for multitasking.

2. Two Interrupts Activated at the Same Time: When two interrupts activate simultaneously, the processor must decide which interrupt will occur first based on a predetermined hierarchy of interrupt priority. These priorities can be determined and set by software or a processor's dedicated interrupt controller. All of this is variable and dependent on the processor and system architecture. For example, in AVR, RESET has the highest priority followed by INT0, and so on.

3. An Interrupt is Activated While a Current Interrupt is Being Served: If an interrupt is currently being served and another interrupt is activated, the latter will be in a pending state and will only be served once the current ISR has finished executing and the interrupts are re-enabled. So, the program will not immediately jump to a new interrupt. AVR handles this by using an interrupt queue for those interrupts that are activated in the middle of another interrupt so that it can place them in the pending state and follow up on executing them when it's their turn.

5. Conclusions

Using the knowledge we gained in the previous lab and the processes we followed in this lab, we were successfully able to write high-level code to implement both external and Pin Change interrupts on our microcontrollers. We were able to test the successful implementation of our interrupt code by using various onboard LEDs to see when certain interrupts were occurring. We did not run into any real issues during this lab as we're slowly becoming more familiar with the software and hardware and how they must interact with one another.