ECSE 421 – Embedded Systems
Laboratory #3

Interrupts & I/O

Zichen Gao (261055991)
Presented to Prof. Jeremy Cooperstock
Due March 14th 2025

Contents

Exercise 1: A Simple Doorbell

Task: Write a program that rings the buzzer every time the button is pressed, using an interrupt and ISR to control the buzzer. You should not have anything running in the void loop() method, other than a delay as a placeholder. Implement a two-tone buzz (or feel free to get more creative with your chime!).

*To generate blocking delays inside the ISR, I made use of the fact that the Serial monitor requires time to transmit which is dependent on the baud rate. Hence, we can estimate a delay by sending a certain number of characters over to the serial monitor. Assuming that the delay in ms produced by sending one character is about 50/BAUD_RATE * 1000, we can calculate that in order to generate a delay of X ms, we need to send out X*BAUD_RATE/(50000) characters.*

See code and video for the rest of the exercise.

Question: Will the delay() function work inside the ISR (interrupt service routine)? What about micros()? It may be helpful to try this out. Give an explanation for your answers.

The delay() function in Arduino works by having a low priority interrupt count a timer. However, since our ISR will (likely be) of higher priority than the timer ISR, it will never count during an ISR and hence delay() does not work inside ISRs. Although the micros() function reads directly from a timer register, it unreliable behavior inside of ISRs.

According to this Stack Overflow post (https://arduino.stackexchange.com/questions/22212/using-millis-and-micros-inside-an-interrupt-routine), the implementation of micros() looks something like the code below:

```
unsigned long micros() {
	unsigned long m;
	uint8_t oldSREG = SREG, t;
	cli();
	m = timer0_overflow_count;
	t = TCNT0;
	if ((TIFR0 & _BV(TOV0)) && (t < 255))
```

```
        m++;
        SREG = oldSREG;
        return ((m << 8) + t) * (64 / clockCyclesPerMicrosecond());
}
```

Note that the `timer0_overflow_count` *variable, which is included in return value of the function, is updated by an ISR. This means that the micros() function will work fine as long as the delay is not too long, as we will miss the ISR that updates the* `timer0_overflow_count` *variable, which makes the micros() function reset to 0.*

Exercise 2: Your Rotary Encoder, Enhanced!

Task: Write a program that outputs a counter on the serial monitor. The counter is initialized at 0 and is incremented by 1 for each clockwise increment of the rotary encoder. Similarly, it is decremented by 1 for each counterclockwise increment. The counter should be reset to 0 when the encoder is pressed down like a button. Use an interrupt for the rotary encoder and do NOT use pinmode().

For this task, I created my own setPinMode() functions which should work for all Arduino digital/analog pins and all standard modes (INPUT, OUTPUT, INPUT_PULLUP). I largely depended on [1] *and* [2] *as reference. The rest of this task can be seen in the video/code.*

Question: Let's say you want an interrupt to occur every 500 ms. Assuming you're using Timer1, what valid combination of OCRx value and prescaler value would allow the interrupt to occur at the desired frequency? Show your calculations.

If the Arduino is operating at its default CPU frequency of 16 MHz, then the rate at which the timer increments is given by:

$$f_{\text{timer}} = \frac{16\ \text{MHz}}{\text{prescaler}}$$

And hence the time it will take for an interrupt to occur is the timer period (inverse of timer rate) multiplied by the OCRx value:

$$T_{\text{interrupt}} = 1/f_{\text{timer}} * \text{OCRx} = \frac{\text{prescaler}}{16\ \text{MHz}} * \text{OCRx}$$

Keep in mind that the OCRx register for Timer1 is a 16-bit timer, which means that the maximum value for the OCRx is 65535. Therefore, we need to choose a large enough value of prescaler. If we choose the prescaler to be 256, then we can solve for the OCRx value, which will be:

$$\text{OCRx} = 500\ \text{ms} * \frac{16\ \text{MHz}}{256} = 31250$$

Task: Using one of the timers for the PWM and the rotary encoder to control its duty cycle, implement a program letting you control the brightness of an LED by rotating the rotary encoder. Do not use pinMode() or analogWrite().

In general, the PWM is implemented by using the Timer1 output compare ISR. The length of the 'HIGH' state of the pulse is determined by a fixed value of the OCRx register. When the ISR is entered, (i.e. when the pulse is transitioning to the 'LOW' state) we change the OCRx register's value to one that is stored in a global volatile variable ocrval*. With this setup, we can control the PWM duty cycle simply by changing this variable.*

$$PWM\ (\%) = \frac{\text{LED\_OCR\_VAL\_DEFAULT}}{\text{LED\_OCR\_VAL\_DEFAULT} + \text{ocrval}} \times 100\%$$

For the rotary encoder, I used two ISRs to handle the SW pin and the CLK pin respectively. The handleButton() ISR is entered when the button (SW) is pressed, upon which it sets the variable ocrval *to* LED_OCR_VAL_DEFAULT, *which makes the PWM duty cycle to be 50%. The handleRotation() ISR is entered when a rotation event is detected, upon which it reads the DT pin to determine whether or not it was a CW or CCW rotation. It increments or decrements the* ocrval *variable to change the PWM duty cycle.*

The main challenge was to figure out how exactly the PWM was to be implemented, as well as figuring out how to setup the timer using the registers.

If I was in a lab with access to an oscilloscope, it would be easier to determine if my code is working since, I can visually see and calculate exactly the PWM cycle using an advanced oscilloscope (instead of just observing the LED go dimmer and dimmer).

Question: Because Timer2 has only 8 bits, it overflows much more frequently than Timer1. If you were to implement the blink example from Lab 1 using Timer2, what is the least frequently that the LED could blink? What would the OCRx and prescaler values be? Show your calculations.

If the CPU operates at 16 MHz *as is typical on the Arduino Uno R3, we use the following equation:*

$$T_{\text{interrupt}} = 1/f_{\text{timer}} * \text{OCRx} = \frac{\text{prescaler}}{16\ \text{MHz}} * \text{OCRx}$$

Where we can solve for the highest possible $T_{\text{interrupt}}$ *value since that would correspond with the lowest possible interrupt frequency. In this case, we need the prescaler and the OCRx values to be maximized. For an 8-bit timer, the maximum value of the OCRx register is 255. In addition, according to Table 15-6 of the ATmega328P datasheet, the maximum value of the prescaler is 1024, obtained when the bits CS12 and CS10 of the TCCR1B register are both set to '1' with CS11 being set to 0. Therefore,*

$$f_{\text{interrupt,min}} = \frac{1}{T_{\text{interrupt,max}}} = \frac{16\ \text{MHz}}{1024 \cdot 255} = 61.2745\ \text{Hz}$$

Which means that an LED using this timer's output compare interrupt could at minimum blink about 61 times per second, with OCRx=255 and prescaler=1024.

The rest of this task is shown in the video/code.

Exercise 3: Test your Reaction Time

Task: Implement the reaction time game according to the specifications given above. Use an interrupt for the button and Arduino timer interrupt for the time limit. (Edit March 4) Read from the joystick in a non-blocking manner.

I configured two timers for this task to fire an interrupt every 1 millisecond. TIM0 (8-bit) is used to time the individual trials (i.e., how long it took the user to press the correct input) while TIM1 (16-bit) is used to time the overall duration of the game.

Once again, to initialize the timer registers correctly, I referred to [1].

The joystick switch button pin (SW) is hooked to a pin change interrupt. However, due to the other joystick pins (Vx, Vy) being analog, pin change interrupts did not work adequately, and hence regular polling had to be used to correctly read the inputs.

The rest of this task can be seen in the video/code.

Exercise 4: Door Safety System, Revisited

Task: Starting with your Lab 2 code, refine your design so that it is more efficient. Make use of timers and interrupts.

Timers:

Only one (hardware) timer is used for this task, namely TIM1. This timer is configured to fire once every millisecond.

For this task, I defined a custom struct type (timer_t) to help me manage the various timing elements involved in the door safety system. Each timer struct instance can be configured by setting the reset threshold (i.e., the timer at which the timer will 'reset') and by setting the timer to either auto reset (once it reaches the threshold) or not.

All timers are updated inside the TIM1 output compared to ISR (i.e., they update every 1ms).

Interrupts (and why I couldn't use it for anything other than magnet sensor):

The only sensor for which I used an interrupt (again, pin change interrupt) is for the magnetic sensor. This is because it has a digital output pin which signals when a magnetic field is detected and PCINT works best with such digital pins.

Also, due to the nature of how keypad input is processed, interrupts cannot be used. To read from a specific row on the keypad, all pins corresponding to other rows must be set to HIGH, then the columns pins can be read to determine which key was pressed on that row [3].

The ultrasonic sensor also requires an input first (set the TRIG pin for a certain amount of time and then read the pulse from the ECHO pin), which removes the possibility of using an interrupt.

The rest of this task can be seen in the video/code.

Appendix

Use of LLMs:

Model:

ChatGPT4o

Prompt:

"https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf https://docs.arduino.cc/retired/hacking/software/PortManipulation/ Can you cite these two sources in IEEE format?"

Result:

Here are the citations in IEEE format:

1. Microchip Technology Inc., *ATmega328P Automotive Microcontroller Datasheet*, 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed: 09-Mar-2025].
2. Arduino, *Port Manipulation*, [Online]. Available: https://docs.arduino.cc/retired/hacking/software/PortManipulation/. [Accessed: 09-Mar-2025].

Let me know if you need any modifications!

References

[1] Microchip Tecnology Inc., "ATmega328P Automotive Microcontroller Datasheet," 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[2] Arduino, "Port Manipulation," [Online]. Available: https://docs.arduino.cc/retired/hacking/software/PortManipulation/.

[3] Arduino Get Started, “Arduino – Keypad” [Online]. Available: https://arduinogetstarted.com/tutorials/arduino-keypad.