ECE-412

Spring 2022
Lab # 4

Drew Branum, David Antosh, Calvin Lam, Ankit Kanotra

PWM, Timer/Counter, Interrupt Vectors & Handlers, Reset

ECE-412

Abstract

The ATMega328P(B) is connected to an external circuit constructed on a breadboard with a LCD display and a buzzer. Pulse Width Modulation (PWM) and the 16-bit Timer-Counter peripheral are researched using the Internet and ATMega328P(B) data-sheet. Software is provided and used to change waveforms from 10HZ to 120HZ to analyze the effects on the LCD. The time periods between logic high and low are also analyzed. The code is executed, the LCD "breathes," and the buzzer buzzes. The RESET and TIM1_COMPA within the given code are then modified to maintain an output frequency of 120 HZ. The oscilloscope's measurements and the sound of the buzzer are analyzed. More given code, LCDPWM, is executed and evaluated within Atmel Studio in conjunction with a provided video. The previous code, not LCDPMW, is then modified to output 120HZ using interrupts to change duty cycles when the interrupt handler is invoked. The LCDPWN C code is then modified to analyze the code changes. Conclusions are made about the given C and Assembly code, the LCD, the PWM, and the buzzer.

Body

The system clock of the ATMega is set to 16MHZ by default with the pre-scaler being 256. The time clock is then calculated by dividing 16MHZ by 256 to yield 31.25KHZ. To find the 16-bit value needed to maintain 10HZ, the 31.25KHZ is divided by 10HZ yielding 3125 or 0x0C35. This value is then set within the program using r17 and r18, setting r17 to 0x0C and r18 to 0x35. To find the 16-bit value needed to maintain 120HZ, the 31.25KHZ is divided by 120HZ yielding 260 or 0x0104. This value is then set within the program using r17 (0x04) and r18 (0x01). The duty cycles on both are 50% because the circuit is high (on) and low (off) the same amount of time. This means that the circuit is on 50% of the time. In the 10HZ waveform, the high and low are maintained for about 0.1 seconds each. In the 120HZ waveform, the high and low and maintained for 0.00832 seconds each making it indistinguishable. The oscilloscope's measurements regarding the 10HZ waveform …. The buzzer sounds as if it is matching the waveform when it is high (on). It buzzes when the light turns on and seems to match the waveform frequency.

The oscilloscope measurement for 120HZ is generated due to modifications made to the RESET function. Since 120 HZ is a higher frequency and a fixed TOP value is sufficient, mode 14 is used, so the mode of operation is set to fast PWM. In the RESET function, the clock prescaler is changed to 64, causing the timer to be incremented every 64 clock cycles. The ICR1 register is used to store the TOP value. These modifications cause the generation of square waveforms with a frequency of 120 HZ. The buzzer sound does not remain the same, it sounds as if it is fluctuating. It does sound higher than the 10HZ frequency.

The functions of Part 2 and Part 3 differ from Part 4. The code for parts 2 and 3 cause the brightness of the LCD to remain the same since the TOP value (value stored in ICR1), which determines the frequency, is treated as a static variable in both parts. For the code in LCDPWM main.c however, the TOP value (value stored in OCR1A) is continuously changing based on the "dir" value (increments or decrements when the beginning or ending of the TOP value is reached), causing the brightness of the LCD to fluctuate.

In the modified TIM1_COMPA handler, the value of global variable "dir" is stored in R16. The value of dir (0 or 1) determines if the LCD will increase or decrease in brightness. R16 is then compared to 1, and if the value of R16 is not equal to 0, the program branches to SETDIR0, which sets the value of DIR by loading R16 with 0 and then loading dir with R16. After SETDIR0, the CHECK2 subroutine is executed, which begins by comparing R16 and dir. If R16 does not equal 0, the SETDIR1 subroutine is executed which sets the value of dir by loading R16 with 1 and then loading dir with R16. After the comparison or execution of SETDIR1, OCR1AH is stored in R25 and OCR1AL is stored in R24. The high byte of the OCR1A register is then compared to 0x00, and if it is less than 0x00, the program branches to the subroutine SETDIR1, which returns back to CHECK2 upon completion. If the high byte of the OCR1A register does not equal 0x00, the program branches to the DECR subroutine which returns back to CHECK2 upon completion. The DECR subroutine decrements the low byte of the OCR1A register by the value stored in the global variable "const". If the high byte of

OCR1A is equal to 0 or DECR has been executed, the low byte of OCR1A is compared to 0x00 and the program branches to SETDIR1 if OCR1AL is equal to 0x00. Otherwise, the DECR subroutine is executed again. After DECR has been executed, the program returns back to the TIM1_COMPA handler and R25 is updated with the new value of OCR1AH while R24 is updated with the new value of OCR1AL. R25 is then compared to 0x08. If R25 is less than 0x08, the INCR subroutine is executed. Otherwise, if R25 is not equal to 0x08, the SETDIR0 subroutine is executed. If R25 is equal to 0x08, R24 is compared to 0x1B. If R24 is greater than 0x1B, SETDIR0 is executed. Otherwise, the INCR subroutine is executed.

Source Code (Software)

```
.set    dir = 1                 //Direction of LCD breathing
.set    const = 5               // Constant for changing OCR1A

.org    0                       //Start of program is at 0
jmp     RESET                   //Initial setup of everything to default values
jmp     INT0_H                  //Interrupt 0 handler
jmp     INT1_H                  //Interrupt 1 handler
jmp     PCINT0_H                //Pin change interrupt 0
jmp     PCINT1_H                //Pin change interrupt 1
jmp     PCINT2_H                //Pin change interrupt 2
jmp     WDT                     //Watchdog timeout handler
jmp     TIM2_COMPA              //TC2 Compare Match A handler
jmp     TIM2_COMPB              //TC2 Compare Match B handler
jmp     TIM2_OVF                //TC 2 overflow handler
jmp     TIM1_CAPT               //TC 1 capture event handler
jmp     TIM1_COMPA              //TC 1 compare match A handler
jmp     TIM1_COMPB              //TC 1 compare match B handler
jmp     TIM1_OVF                //TC 2 overflow handler
jmp     TIM0_COMPA              //TC 1 compare match A handler
jmp     TIM0_COMPB              //TC 1 compare match B handler
jmp     TIM0_OVF                //TC 0 overflow handler
jmp     SPI_TC                  //SPI Transfer Complete
jmp     USART_RXC               //USART receive complete
jmp     USART_UDRE              //USART data register empty
jmp     USART_TXC               //USART transmit complete
jmp     ADCC                    //ADC conversion complete
jmp     EE_READY                //EEPROM ready
jmp     ANA_COMP                //Analog Comparison complete
jmp     TWI                     //I2C interrupt handler
jmp     SPM_READY               //store program memory ready handler

RESET:                          //Initialize the ATMega328P
cli                             //Disable global interrupts
ldi     r16,0xFF                //Set PB1 or OC1A as output
out     DDRB,r16
ldi     r16,0x1B                //Set clock prescaler
sts     TCCR1B,r16
ldi     r16,0x82                //Set waveform generator to mode 14, with Fast PWM and
                                //ICR1 as TOP value, and set compare output mode to
                                //clear OC1A on compare match and set OC1A at
                                //BOTTOM
sts     TCCR1A,r16
ldi     r16,0x08                //Set TOP value to 2075 to set clock frequency to ~120 Hz
ldi     r17,0x1B
sts     ICR1H,r16
sts     ICR1L,r17
ldi     r16,0x00                //Set timer counter to 0
sts     TCNT1H,r16
```

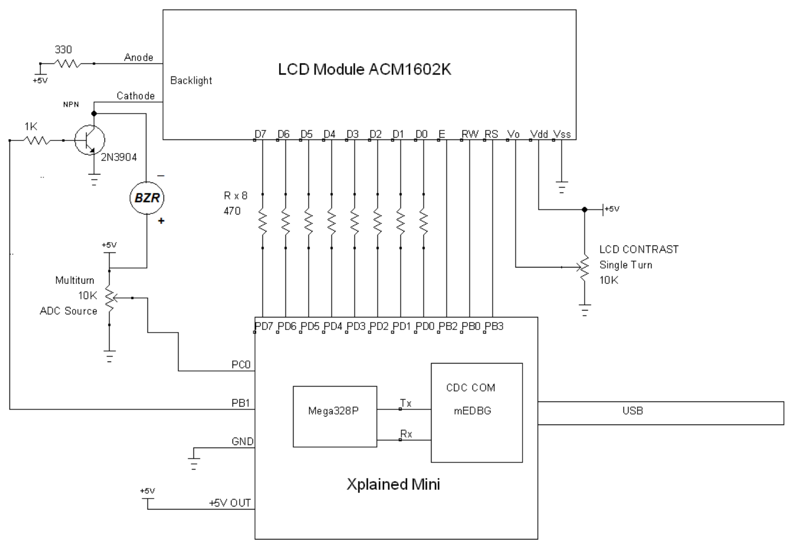
```
            sts     TCNT1L,r16
            ldi     r16,0x02                //Enable timer to interrupt
            sts     TIMSK1,r16
            sei                             //Enable global interrupts

here:
            rjmp    here                    //Create an infinite loop while LCD dims/brightens via
                                            //interrupts

INT0_H:
            nop                             ;external interrupt 0 handler
            reti
INT1_H:
            nop                             ;external interrupt 1 handler
            reti
PCINT0_H:
            nop                             ;pin change interrupt 0 handler
            reti

PCINT1_H:
            nop                             ;pin change interrupt 1 handler
            reti

PCINT2_H:
            nop                             ;pin change interrupt 2 handler
            reti

WDT:
            nop                             ;watchdog timeout handler
            reti

TIM2_COMPA:
            nop                             ;TC 2 compare match A handler
            reti

TIM2_COMPB:
            nop                             ;TC 2 compare match B handler
            reti

TIM2_OVF:
            nop                             ;TC 2 overflow handler
            reti

TIM1_CAPT:
            nop                             ;TC 1 capture event handler
            reti

TIM1_COMPA:                                 ;TC 1 compare match A handler
            lds     r16,dir                 //Get current LCD direction
            cpi     r16,1                   //Checks what the LCD brightness should be doing
            brne    SETDIR0                 //Reset direction variable to 0
```

```
            lds     r25,OCR1AH          //Checking OCR1A < 2075
            lds     r24,OCR1AL
            cpi     r25,0x08
            brlt    INCR
            brne    SETDIR0
            cpi     r24,0x1B
            brge    SETDIR0             //If OCR1A is >2075 make adjustments to variables, so
                                        //the brightness of the LCD can go in the reverse direction

INCR:
            adiw    r24,const           //Add the const variable to the register pair r24:r25
            sts     OCR1AH,r25          //Load OCR1A with the new incremented value
            sts     OCR1AL,r24
            rjmp    CHECK2              //Start the second check

SETDIR0:
            ldi     r16,0               //LCD direction variable is set to 0
            sts     dir,r16             //Stores value

CHECK2:
            lds     r16,dir             //Second comparison: dir == 0 && OCR1A > 2075
            cpi     r16,0               //Checks if current direction variable is 0
            brne    SETDIR1             //Check direction, if not 0 then set to 1
            lds     r25,OCR1AH
            lds     r24,OCR1AL
            cpi     r25,0x00            //Compare the high byte of OCR1A to 0x00
            brlt    SETDIR1             //If OCR1AH is less than 0x00, then the check fails, so set
                                        //dir to 1
            brne    DECR                //If OCR1AH is not equal to 0x00, then the check passes,
                                        //so decrement OCR1A
            cpi     r24,0x00            //Checks low bit of OCR1A to see if it equals 0
            breq    SETDIR1             //Set direction to 1 if the compare above was true
DECR:
            sbiw    r24,const           //Subtract the const variable to the register pair r24:r25
            sts     OCR1AH,r25          //Load OCR1A with new decremented value
            sts     OCR1AL,r24
            reti                        //Return from interrupt

SETDIR1:
            ldi     r16,1               //LCD direction variable is set to 1
            sts     dir,r16             //Stores direction variable
            reti                        //Return from interrupt

TIM1_COMPB:
            nop                         ;TC 1 compare match B handler
            reti

TIM1_OVF:
            nop                         ;TC 1 overflow handler
            reti
```

```
TIM0_COMPA:
            nop                     ;TC 0 compare match A handler
            reti

TIM0_COMPB:
            nop                     ;TC 1 compare match B handler
            reti

TIM0_OVF:
            nop                     ;TC 0 overflow handler
            reti

SPI_TC:
            nop                     ;SPI Transfer Complete
            reti

USART_RXC:
            nop                     ;USART receive complete
            reti

USART_UDRE:
            nop                     ;USART data register empty
            reti

USART_TXC:
            nop                     ;USART transmit complete
            reti

ADCC:
            nop                     ;ADC conversion complete
            reti

EE_READY:
            nop                     ;EEPROM ready
            reti

ANA_COMP:
            nop                     ;Analog Comparison complete
            reti

TWI:
            nop                     ;I2C interrupt handler
            reti

SPM_READY:
            nop                     ;store program memory ready handler
            reti
```

ECE-412

Schematics (Hardware)



Title	ECE412 Lab 4 Circuit	
Author	Eugene Rockey, CopyRight 2018, All Rights Reserved	
File	C:\Users\Donkey\Desktop\Lab4.dsn	Document
Revision 1.0	Date 2/16/2018	Sheets 1 of 1

Analysis

By using and analyzing interrupt handlers along with the reset vector and Timer/Counter registers, experience and knowledge using pulse width modulation was gained, which consequently allowed PWM to be successfully executed in the project. Since PWM is used to efficiently control the amount of power supplied to a device, it is a valuable technique when dealing with embedded systems that require power modulation. For example, PWM might be used to manage the speed of a motor or generate an audio signal with varying frequency.

Conclusion

Experiencing the use cases of timers, counters, and interrupt driven assembly code were the goals of this exercise. Timers and counters were used to constantly make the application move to a new state, and when certain parameters are met, interrupts will trigger in the program and will subsequently adjust the intensity of the LCD’s backlights with the Pulse Width Modulation power controlling method.

References

ATMega328PB Datasheet (n.d.). Retrieved April 15, 2022 from http://ww1.microchip.com/downloads/en/DeviceDoc/40001906A.pdf