Team Members: Louis Solovy & Hongyu Zeng

Professor Beichel

ECE:3360 Embedded Systems

Post-Lab Report 4

**1. Introduction**

The goal of this lab was to gain some experience with advanced timer/counter functionality, pulse width modulation, interrupts, and LCDs. The lab’s objective was to construct a brightness adjustable LED light source that utilizes a LCD, rotary pulse generator (RPG), and a pushbutton switch (PBS). When it came to creating the debounce for the RPG and PBS we had the choice of either creating a hardware debounce or a software debounce. For the actual display of the light source we had to use two white light-emitting LEDs that can be turned on and off with a npn transistor. To control the brightness of the LED was needed to create a PWM signal (variable duty cycle DC). The functionality of the circuit would need to toggle the LEDs with the PBS, the default DC value would need to be set to 50% when turning the device on, the DC must be adjustable in 5% increments from 5%-100% using the RPG.If the RPG is rotated CW it will increase the DC and if it is rotated CCW it will decrease the DC. Another functionality of the circuit needed to utilize creating a PWM signal generated by the microcontroller with one 8-bit Timers/Counters that must have a frequency of 1.25kHz. When it came to adjusting the DC value, it should only be possible if the LEDs are on. The functionality of the LCD will display the DC percentage and also tell if the LEDs are on or off.

**2.Schematic**

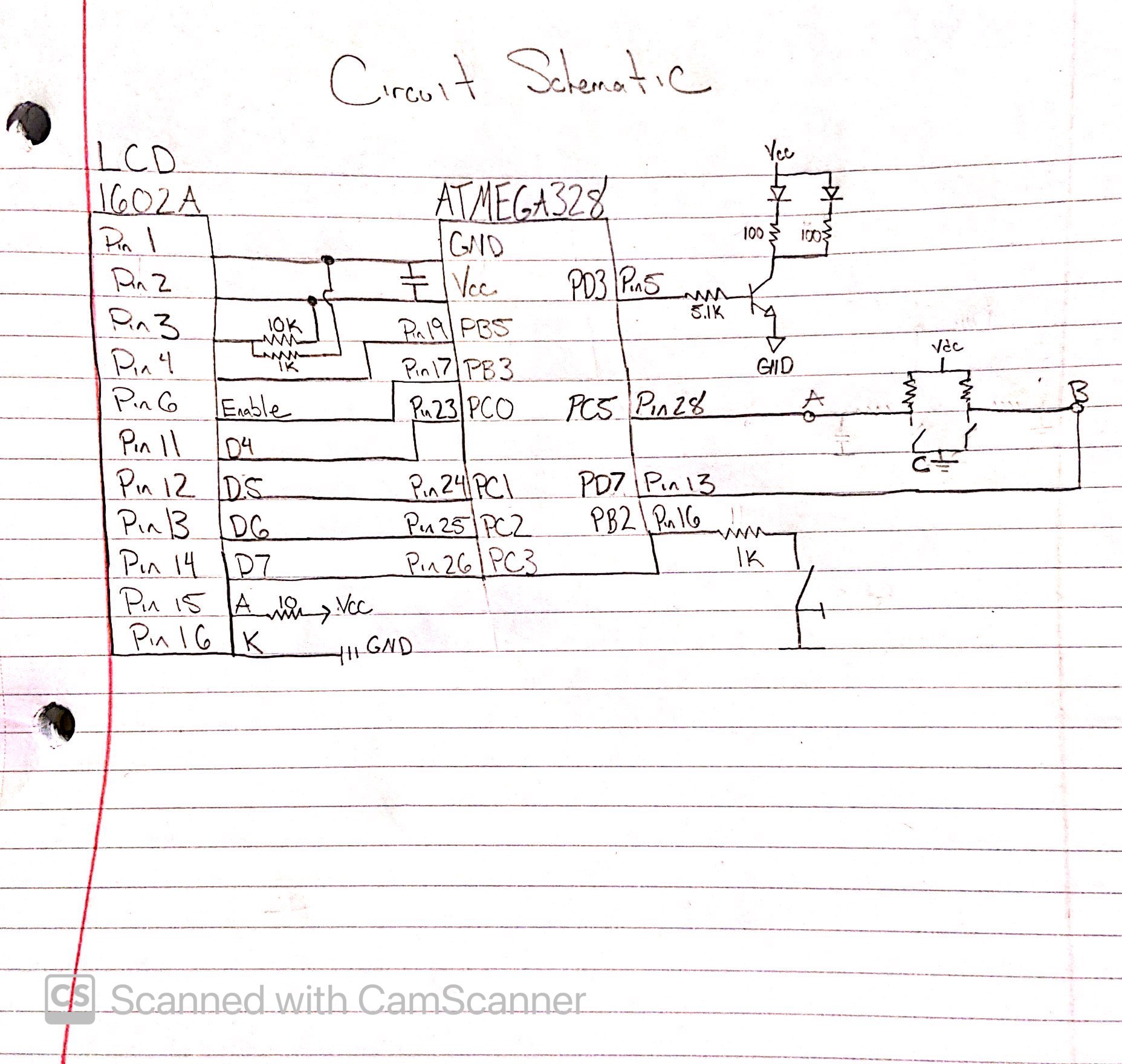
Figure 1: Drawn Circuit Diagram as implemented.

Figure 1 shows the circuit diagram as it was implemented for this lab. The I/O configuration for the LCD consisted of Pin 1 to GND, Pin 2 to Vcc, Pin 3 to Vee (contrast voltage) that consisted of 1K and 10K resistor connected to Vcc, Pin 4 (R/S) to PB5, Pin 6 (Enable) to PB3, Pin 11 (D4) to PC0, Pin 12 (D5) to PC2, Pin 13 (D6) to PC2, and Pin 14 (D7) to PC3. We decided to add the back light on so this was just setting Pin15 (A) to Vcc with a 10 ohm resistor and Pin 16 (K) to GND. To get the value of the 10 ohm resistor we used the datasheet of the LCD to find that the voltage was 4.2 with a 110mA. The following calculation was done, 5V-4.2V = 0.8V -> 0.8V/110mA which told us to use 10 ohms. Note that we needed to add a capacitor of .1uF that is connected to the Vcc and GND of the circuit. Next, the inputs of the circuit were the RPG, PWM signal, and the PBS. The PWM signal schematic was given to us and can be seen in Figure 2. Since we were using a software debounce for the RPG, we didn’t need to add any resistors to the circuit, instead we activated the pull up resistors within the code. For the PBS, even though we were doing a software debounce we still needed to add a 1k resistor because we needed to select an R so that I <= 5mA. This was a simple calculation of R =V/I -> R = 5V/5mA = 1k ohms. Figure 3 shows a more indepth diagram of the RPG. Figure 4 shows a diagram on how we knew how to configure the LCD.

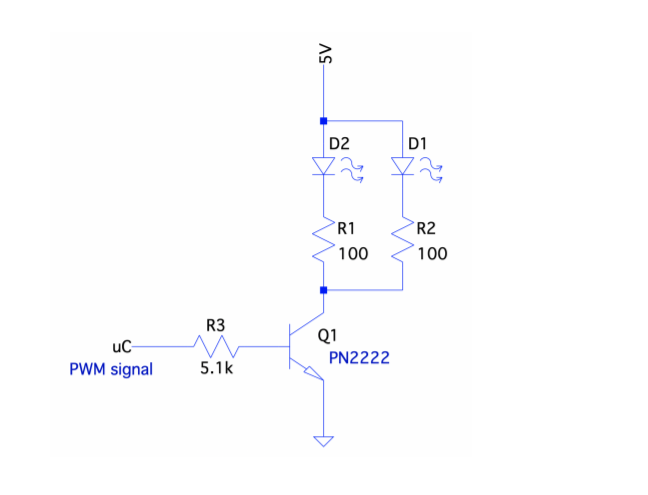
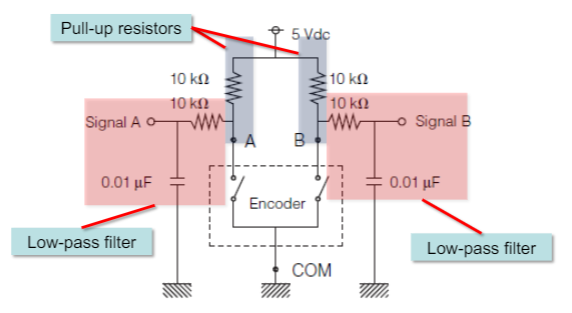
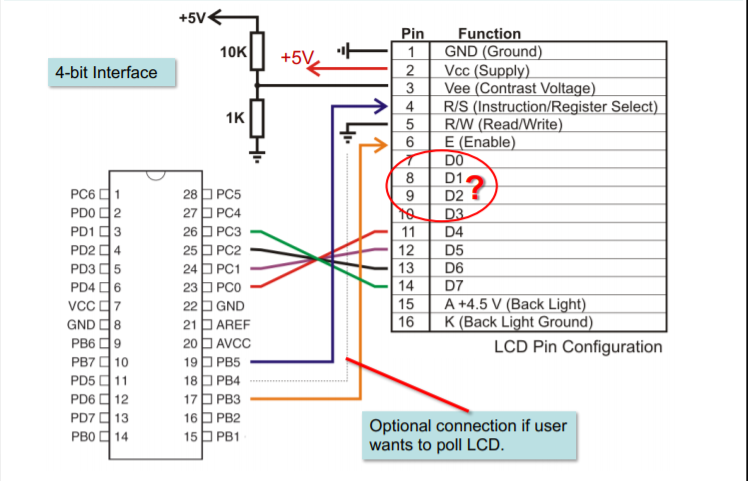
Figure 2: Circuit for operating the 2 LEDs Figure 3: Detailed diagram of RPG

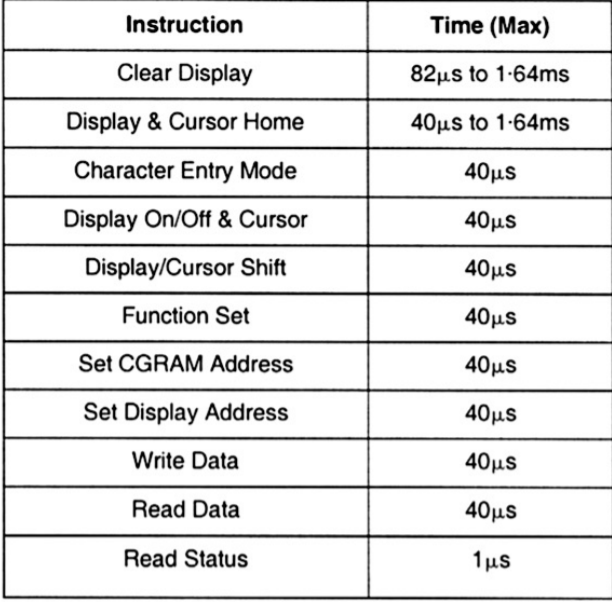
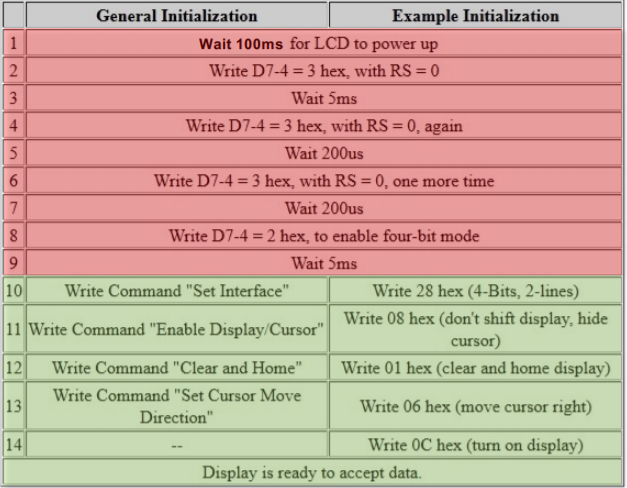
Figure 4: LCD configuration



**3. Discussion**

The planning for this project consisted of two basic phases: Hardware and Software. The hardware design was to create a circuit that would display a message about the DC and LEDs using an LCD, PBS, RPG, and a PWM signal. Since this lab had similarities to the previous labs hardware, the set up was exactly the same for the RPG and PBS. The only hardware that was different in this lab was the LCD and the PWM signal. But since the diagrams to configure these were given to us, it wasn’t difficult to set them up. Figures 2 and 4 were the schematics that we had to look at when it came to configuring the circuit. To test to make sure that the LCD was set up correctly the professor gave us a test file that if it was working, then the LCD would display “Hello World :-)”. After we made sure that all of our hardware was set up correctly it was time to move to the Software phase of this lab.

The first software concept that we decided to tackle was initializing the LCD to display “DC = “ on the first line and “LEDs = “ on the second line. This was done by following a diagram that we got from the professor. The initialization would first write the upper nibble only in 8-bit mode, and then write the upper and lower nibble in 4-bit mode. This consisted of making a timer with an 8 prescaler that would be 100us and from there would be multiplied to be used for the delay that we would need. For the 8-bit mode we needed to wait many times but have different delays for each. After writing D7-4 = 3 hex, with RS = 0 four times we would have to wait 5ms, 200us, 200us, and 5ms in that order. But we initially needed to wait 100ms to wait for the LCD to power up. After we finished 8-bit mode we had to configure the 4-bit mode which was writing the commands: set interface, enable display/cursor, clear and home, and set cursor move direction. For the delay we used for 4-bit mode we used 1.7ms because we needed to make sure that it was greater than 1.64ms because the clear display and cursor home commands sometimes took up that much time. Figure 5 shows the diagram we followed to get the correct initialization of the LCD, red represents 8-bit mode and green represents 4-bit mode. Figure 6 shows how we figure out the timing that would be needed for each instruction.

Figure 5: LCD initialization Figure 6: LCD command execution times

After we got our LCD initialization correct, we tested it by printing out the messages that would be required for the lab which were “DC = “ and “LEDs = “. Since this worked by using, for example “msg1: .db "DC = ", 0x00” and “ldi r30,LOW(2\*msg1)” with “ldi r31,HIGH(2\*msg1)” would load the Z register low and high and then call the display function. The next thing to tackle was the PWM signal. We started by using an 8-bit Timer/Counter TC2 that would have a prescaler of 8. We first needed to set the signal to non-inverted with the OCR2B to mode 3, which was the “Fast PWM” as OCR2B was the value to compare to. Since we would be in a comparing output mode, we needed to get the halfway mark of the signal, between top and bottom. Since the bottom was equal to 0 and the top was 255, the halfway mark was going to be 127. This will be useful when it comes to talking about the RPG. For the PWM signal to work adn light up the LEDs, we needed to make sure that OC2B was configured as an output. After we finished the PWM signal we needed to test it to make sure that the LEDs would light up to 50%. We also tested it with 255 instead of 127 to make sure that the LEDs would like up 100%.

The next software part we decided to tackle was to get the LCD to display the DC percentage of the lights using a bit unsigned technique that was shown in the slides. This technique would help us display the variable sting from the value of the DC. We needed to figure out how to get the correct character formatting for the LCD. We followed the instructions on how to isolate digits of the duty cycle. This was done by a repetition of dividing by 10 to get the correct format. The section in the code that does this is under “div16u” that uses a dividend and a divisor with the low and high byte of the DC value. After calling this subroutine to get the result of the Dc value it would go to the subroutine “displayDC” which would send this data to the LCD. For the DC value to actually be sent to the LCD it would need to be converted to ASCII. Since most of this code to get the DC value to the LCD was given to us, it was very simple to integrate it into our code. It was a repetition of calling “div16u” for each byte and then converting it to ASCII and then storing it into the RAM. To ensure this was working properly we tested it using the 127 value (50%) and it worked, the LCD now showed “DC = 050.0%”.

The next thing to do was implement the interrupts along with the RPG and PBS so that we could toggle the LEDs with the PBS and change the DC value by 5% (increase or decrease from 5% -100%). To set up the interrupt we started off by configuring PCICR register to allow for a control on the interrupts since PCICR is the pin change interrupt control register. We then needed three Pin change mask registers which were PCMSK0, PCMSK1, PCMSK2 which were going to be used for the PBS and the RPG. PCMSK0 was set to PCINT0 which was connected to the PBS. PCMSK1 was set to PCINT13 which was channel A of the RPG. PCMSK2 was set to PCINT23 which was channel B of the RPG. Next was to use the command “SEI” which would enable the interrupts globally. Since we decided to do a software debounce for the button and the RPG, we used the code from the previous labs but had to change certain specifications for them to meet this lab. One thing we added for this lab with the PBS was a subroutine that would display if the LEDs were on or off and print either “ON” or “OFF” on the LCD. This was simply just disconnecting OC2B to turn the lights off and setting OC2B to non-inverting for the lights to be turned on when the button was pressed and also displaying the new message. For the RPG debounce we needed either add 5% (CW) to the DC or subtract 5% (CCW). The way that we did this was we decided to either add 13 or subtract 13 since 13 was almost 5% of 255 (255 \* .05 = 12.75). This would change the DC value by 5% every time that the RPG was rotated. But there were two special numbers we had to watch for so that the DC value wouldn’t for over 100% and less than 5%. These numbers were 244 and 23, so when the DC value got to 244 from the rotation of the RPG, 255 would be loaded instead of adding 13 since that would give 257 which is above 255. And when the DC value got to 23 from the rotation of the RPG, 13 would be loaded instead of subtracting 13 since 13 would give the DC value of 5% and 10 wouldn’t give 5%. This was the last step in the lab, so we decided to test it and check to see if our code works. After testing we saw that our code was working properly with the correct message on the LCD displaying when the RPG was rotated and the PBS was pushed.

**4.Conclusion**

Overall, the goal of this lab was to gain some experience with advanced timer/counter functionality, pulse width modulation, interrupts, and LCDs. After completing this lab we would say that we have achieved that goal. During the lab there were some problems that we ran into when testing our code. One of them being when it came to working with the PBS debounce we had the wrong command to see if the button was pressed which was SBRC which should have been SBRS. This was a small problem but took a long time to figure out why our button was not functioning properly. Another problem we had was actually knowing how to do the interrupt properly. At first we didn’t know that the PCINTn were going to be set up using three ports, because at first we had the RPG in the same port along with the PBS. But after reading the datasheet we were able to figure out how to configure the interrupts. We also didn’t know that throughout this lab we needed to use “RETI” instead of “RET” because we were using interrupts. Another command that we didn’t know we needed to use at first and where to put it was “SEI” to enable the interrupts to start working. Overall, after completing this lab we can now construct a brightness adjustable LED light source that utilizes a LCD, rotary pulse generator (RPG), and a pushbutton switch (PBS).

**5.Appendix A: Source Code**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Lab4.asm

;

; Created: 3/27/2021 11:31:09 PM

; Author : Hongy Zeng, Louis M Solovy

;

.include "m328Pdef.inc"

;\*\*\*\*\* Subroutine Register Variables

.def drem16uL=r14

.def drem16uH=r15

.def dd16uL =r16

.def dd16uH =r17

.def dv16uL =r18

.def dv16uH =r19

.cseg

.org 0x0000

rjmp start

.org 0x0006 ; button

rjmp button

.org 0x0008 ; A

rjmp rpg

.org 0x000A ; B

rjmp rpg

.org 0x0034

;LCDstr:.db 0x33,0x32,0x28,0x01,0x0c,0x06

;ldi r26, low(LCDstr)

;ldi r27, high(LCDstr)

msg1: .db "DC = ", 0x00

msg2: .db "(%)", 0x00

msg3: .db "LEDs:", 0x00

msg4: .db "ON ",0x00

msg5: .db "OFF", 0x00

; PB5 -> RS

; PB3 -> ENABLE

; PB0 -> BUTTON PCINT0

; PD7 -> CHANNEL B PCINT23

; PC5 -> CHANNEL A PCINT13

; PC0 -> D4

; PC1 -> D5

; PC2 -> D6

; PC3 -> D7

; PD3 -> PWM OC2B

start:

; set up I/O

ldi r16, (1<<DDB5) | (1<<DDB3)

out DDRB, r16

ldi r16, (1<<DDC3) | (1<<DDC2) | (1<<DDC1) | (1<<DDC0)

out DDRC, r16

ldi r16, (1<<DDD3)

out DDRD, r16

ldi r16, 0x01

out PORTB, r16 ; PORTB0-2 set to high

ldi r16, 0b00100000

out PORTC, r16

ldi r16, 0b10000000

out PORTD, r16

; setup interrupt

ldi r16, 0b00000111

sts PCICR, r16

ldi r16, 0x01

sts PCMSK0, r16

ldi r16, 0b00100000

sts PCMSK1, r16

ldi r16, 0b10000000

sts PCMSK2, r16

;set up TC2 and OCR2B

ldi r16, 0b00100011 ; set to noninvert OC2B, fast PWM, compare value is OCR2B

sts TCCR2A, r16

ldi r16, 0b00000010 ; /8

sts TCCR2B, r16 ; cannot use out as out of range

ldi r16, 127 ; when OCR2B match , controls DC

sts OCR2B, r16

rcall initialization

.set dcval = 500 ; controls the digits for DC display

ldi r28, low(dcval)

ldi r29, high(dcval)

ldi r19, 0b10000000 ; bit 7 for if LED should be on, 6 is button pressed , 3 for ccw, 2 for cw

main:

sei

rcall display

refresh:

rjmp refresh

display:

; display 1st line

ldi r30,LOW(2\*msg1) ; Load Z register low

ldi r31,HIGH(2\*msg1) ; Load Z register high

rcall displayCString

;display DC

rcall displayDC

; display %

ldi r30, low(2\*msg2)

ldi r31, high(2\*msg2)

rcall displayCString

; move cursor to second line

ldi r20, 0x0C

out PORTC, r20

rcall LCDStrobe

rcall delay\_100us

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

rcall delay\_100us

ldi r30,LOW(2\*msg3) ; Load Z register low

ldi r31,HIGH(2\*msg3) ; Load Z register high

rcall displayCString

rcall displayONF

ret

initialization: ; Z overwritten

push r20

; 8 bit mode

cbi PORTB, 5 ; RS = 0 , commands

.set count = 1000

ldi r30, low(count)

ldi r31, high(count)

rcall delay ; delay 100ms

ldi r20, 0x03

out PORTC, r20

rcall LCDStrobe

.set count = 50

ldi r30, low(count)

ldi r31, high(count)

rcall delay ; 5ms

ldi r20, 0x03

out PORTC, r20

rcall LCDStrobe

rcall delay\_100us

rcall delay\_100us ; 200us

ldi r20, 0x03

out PORTC, r20

rcall LCDStrobe

rcall delay\_100us

rcall delay\_100us ; 200us

ldi r20, 0x02

out PORTC, r20

rcall LCDStrobe

.set count = 50

ldi r30, low(count)

ldi r31, high(count)

rcall delay ; 5ms

; 4bit mode

ldi r20, 0x02

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay ; 1.7ms

ldi r20, 0x08

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x08

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x01

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x06

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x0C

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

pop r20

ret

displayCString: ; modify r0

sbi PORTB, 5

lpm r0, Z+ ; r0 <-- first byte

tst r0 ; Reached end of message ?

breq done ; Yes => quit

swap r0 ; Upper nibble in place

out PORTC,r0 ; Send upper nibble out

rcall LCDStrobe ; Latch nibble '

rcall delay\_100us

swap r0 ; Lower nibble in place

out PORTC,r0 ; Send lower nibble out

rcall LCDStrobe ; Latch nibble

rcall delay\_100us

rjmp displayCstring

done:

cbi PORTB, 5

ret

displayDstring: ; modify r0

sbi PORTB, 5

ld r0,Z+

tst r0 ; Reached end of message ?

breq done\_dsd ; Yes => quit

swap r0 ; Upper nibble in place

out PORTC,r0 ; Send upper nibble out

rcall LCDStrobe ; Latch nibble

rcall delay\_100us

swap r0 ; Lower nibble in place

out PORTC,r0 ; Send lower nibble out

rcall LCDStrobe ; Latch nibble

rcall delay\_100us

rjmp displayDString

done\_dsd:

cbi PORTB, 5

ret

LCDStrobe:

sbi PORTB, 3

nop

nop

nop

nop

cbi PORTB, 3

ret

displayONF:

sbrc r19, 7

rjmp displayON

rjmp displayOFF

displayON:

ldi r30,LOW(2\*msg4) ; Load Z register low

ldi r31,HIGH(2\*msg4) ; Load Z register high

rcall displayCString

ret

displayOFF:

ldi r30,LOW(2\*msg5) ; Load Z register low

ldi r31,HIGH(2\*msg5) ; Load Z register high

rcall displayCString

ret

displayDC:

.dseg

dtxt: .BYTE 6 ; Allocation

.cseg

push r14

push r15

push r16

push r17

push r18

push r19

push r30

push r31

mov dd16uL,r28 ; LSB of number to display

mov dd16uH,r29 ; MSB of number to display

ldi dv16uL,low(10)

ldi dv16uH,high(10)

; Store terminating for the string.

ldi r20,0x00 ; Terminating NULL

sts dtxt+5,r20 ; Store in RAM

; Divide the number by 10 and format remainder.

rcall div16u ; Result: r17:r16, rem: r15:r14

ldi r20,0x30

add r14,r20 ; Convert to ASCII

sts dtxt+4,r14 ; Store in RAM

; Generate decimal point.

ldi r20,0x2E ; ASCII code for .

sts dtxt+3,r20 ; Store in RAM

; Generate unit

rcall div16u

ldi r20,0x30

add r14, r20

sts dtxt+2, r14

; Generate tens

rcall div16u

ldi r20,0x30

add r14, r20

sts dtxt+1, r14

; genearte hundreds

rcall div16u

ldi r20,0x30

add r14, r20

sts dtxt, r14

ldi r30, low(dtxt)

ldi r31, high(dtxt)

rcall displayDstring

pop r31

pop r30

pop r19

pop r18

pop r17

pop r16

pop r15

pop r14

ret

delay: ; based on Z (r31 r30) value, delay for Z times 100us. 50000 for 5s, 12000 for 12 s

rcall delay\_100us

sbiw Z, 1 ;R30, R31 16 bit loop

brne delay

ret

delay\_100us: ; store Z to stack and reuse them

push r30

push r31

;stop timer

ldi r31, 0x00 ; stop timer

out TCCR0B, r31 ;

; clear overflow flag

in r31, TIFR0

sbr r31, 1<<TOV0 ; clear TOV0, write logic 1

; set count

out TIFR0, r31

ldi r31, 56

;start timer with new initial count

out TCNT0, r31 ; load counter

ldi r30, 2

out TCCR0B, r30 ; restart timer/8

wait:

in r31, TIFR0 ;

sbrs r31, TOV0 ; check overflow, if overflow then skip

rjmp wait

pop r31

pop r30

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* "div16u" - 16/16 Bit Unsigned Division

;\*

;\* This subroutine divides the two 16-bit numbers

;\* "dd8uH:dd8uL" (dividend) and "dv16uH:dv16uL" (divisor).

;\* The result is placed in "dres16uH:dres16uL" and the remainder in

;\* "drem16uH:drem16uL".

;\*

;\* Number of words :196 + return

;\* Number of cycles :148/173/196 (Min/Avg/Max)

;\* Low registers used :2 (drem16uL,drem16uH)

;\* High registers used :4 (dres16uL/dd16uL,dres16uH/dd16uH,dv16uL,dv16uH)

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\*\* Code

div16u:

clr drem16uL ;clear remainder Low byte

sub drem16uH,drem16uH;clear remainder High byte and carry

rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_1 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_2 ;else

d16u\_1: sec ; set carry to be shifted into result

d16u\_2: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_3 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_4 ;else

d16u\_3: sec ; set carry to be shifted into result

d16u\_4: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_5 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_6 ;else

d16u\_5: sec ; set carry to be shifted into result

d16u\_6: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_7 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_8 ;else

d16u\_7: sec ; set carry to be shifted into result

d16u\_8: rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_9 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_10 ;else

d16u\_9: sec ; set carry to be shifted into result

d16u\_10:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_11 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_12 ;else

d16u\_11:sec ; set carry to be shifted into result

d16u\_12:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_13 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_14 ;else

d16u\_13:sec ; set carry to be shifted into result

d16u\_14:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_15 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_16 ;else

d16u\_15:sec ; set carry to be shifted into result

d16u\_16:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_17 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_18 ;else

d16u\_17: sec ; set carry to be shifted into result

d16u\_18:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_19 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_20 ;else

d16u\_19:sec ; set carry to be shifted into result

d16u\_20:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_21 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_22 ;else

d16u\_21:sec ; set carry to be shifted into result

d16u\_22:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_23 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_24 ;else

d16u\_23:sec ; set carry to be shifted into result

d16u\_24:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_25 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_26 ;else

d16u\_25:sec ; set carry to be shifted into result

d16u\_26:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_27 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_28 ;else

d16u\_27:sec ; set carry to be shifted into result

d16u\_28:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_29 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_30 ;else

d16u\_29:sec ; set carry to be shifted into result

d16u\_30:rol dd16uL ;shift left dividend

rol dd16uH

rol drem16uL ;shift dividend into remainder

rol drem16uH

sub drem16uL,dv16uL ;remainder = remainder - divisor

sbc drem16uH,dv16uH ;

brcc d16u\_31 ;if result negative

add drem16uL,dv16uL ; restore remainder

adc drem16uH,dv16uH

clc ; clear carry to be shifted into result

rjmp d16u\_32 ;else

d16u\_31:sec ; set carry to be shifted into result

d16u\_32:rol dd16uL ;shift left dividend

rol dd16uH

ret

buttonDeb: ; if pressed, set r19 bit 6 to 1, else 0

push r22

push r23

push r26

ldi r23, 9 ; reset sample counter

ldi r26, 0 ;

bDeb\_loop:

in r22, PINB

sbrs r22, 0

inc r26 ; number of times button is pressed

rcall delay\_100us

dec r23

brne bDeb\_loop

subi r26, 4

brpl setbuttonP

andi r19, 0b10111111 ; set to not pressed if less than 5

rjmp bDeb\_done

setbuttonP:

ori r19, 0b01000000 ; set to pressed if more than 5

bDeb\_done:

pop r26

pop r23

pop r22

ret

button:

rcall buttonDeb

sbrs r19, 6

reti

button\_pressed:

rcall buttonDeb ; 0.9 ms + someother loop so 1ms

sbrc r19, 6

rjmp button\_pressed

andi r19, 0b10111111 ; button no longer pressed

sbrs r19, 7 ; don't skip if was light off

rjmp lighton

lightoff:

ldi r20, 0b00000000 ; disconnect OC2B

sts TCCR2A, r20

andi r19, 0b01111111

rcall clearDisplay

rcall display

reti

lighton:

ldi r20, 0b00100011 ; set to noninvert OC2B

sts TCCR2A, r20

ori r19, 0b10000000

rcall clearDisplay

rcall display

reti

rpg:

push r25

sbrs r19, 7

rjmp aftercycle

andi r19, 0b11110011; reset rotating direction

rcall rpgdebounce

sbrs r25, 6

rcall rotating

sbrc r29, 2

rjmp aftercycle

sbrs r25, 7

rcall rotating

aftercycle:

rcall clearDisplay

rcall display

pop r25

reti

rpgdebounce:

push r22

push r23

push r26

push r27

ldi r23, 9 ; reset sample counter

ldi r26, 0

ldi r27, 0

ldi r25, 0b11000000 ; load initial value

rpgdebounce\_loop:

in r22, PINC

sbrs r22, 5

inc r26 ; number of times A is active 0

in r22, PIND

sbrs r22, 7

inc r27

rcall delay\_100us

dec r23

brne rpgdebounce\_loop

subi r26, 4

brpl A\_on

rjmp B

A\_on:

andi r25, 0b10111111

B: subi r27, 4

brpl B\_on

rjmp rpgdebounce\_done

B\_on:

andi r25, 0b01111111

rpgdebounce\_done:

pop r27

pop r26

pop r23

pop r22

ret

rotating:

;determine direction

sbrs r25, 6

ori r19, 0b00000100 ; clockwise

sbrs r25, 7

ori r19, 0b00001000 ; ccw

rotating\_helper: ; loop till detent

rcall rpgdebounce ; reads the value

cpi r25, 0b11000000 ; compare to detent

brne rotating\_helper ; incpmlete cycle

sbrc r19, 2 ;don't skip if cw

rcall maybeincrease

sbrc r19, 3 ; if cww

rcall maybedecrease

ret

maybeincrease: ;

cpi r16, 255

brne maybeincreaseH

ret

maybeincreaseH:

cpi r16, 244

breq totop

cpi r16, 13

breq incto23

ldi r20, 13

add r16, r20

sts OCR2B, r16

rcall incDC

ret

totop:

ldi r16, 255

sts OCR2B, r16

ori r19, 0b00000010

rcall incDC

ret

incto23:

ldi r16, 23

sts OCR2B, r16

rcall incDC

ret

maybedecrease:

cpi r16, 13

brne maybedecreaseH

ret

maybedecreaseH:

cpi r16, 23

breq tobottom

cpi r16, 255

breq decto244

subi r16, 13

sts OCR2B, r16

rcall decDC

ret

tobottom:

ldi r16, 13

sts OCR2B, r16

rcall decDC

ret

decto244:

ldi r16, 244

andi r19, 0b11111101

sts OCR2B, r16

rcall decDC

ret

incDC:

adiw Y, 50;

ret

decDC:

sbiw Y, 50;

ret

clearDisplay:

push r20

push r30

push r31

ldi r20, 0x00

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

ldi r20, 0x01

out PORTC, r20

rcall LCDStrobe

.set count = 17

ldi r30, low(count)

ldi r31, high(count)

rcall delay

pop r31

pop r30

pop r20

ret

**6.Appendix B: References**

Atmel, “Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash”, <<https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf>> August 2013.

Pighixxx, “Arduino Uno Pinout Diagram” <<https://www.circuito.io/blog/arduino-uno-pinout/>> May 2013.

Bourns,”PEC12R - 12 mm Incremental Encoder” , <<https://www.mouser.com/datasheet/2/54/PEC12R-777795.pdf> > September 2019.

Longtech Display, “LCM1602A”, <<https://www.electronicoscaldas.com/datasheet/LCM-1602A_Longtech.pdf>>July 2011.

Fairchild, “PN2222A / MMBT2222A / PZT2222A NPN General Purpose Amplifier” <<https://cdn-shop.adafruit.com/datasheets/PN2222A.pdf>> August 2010.