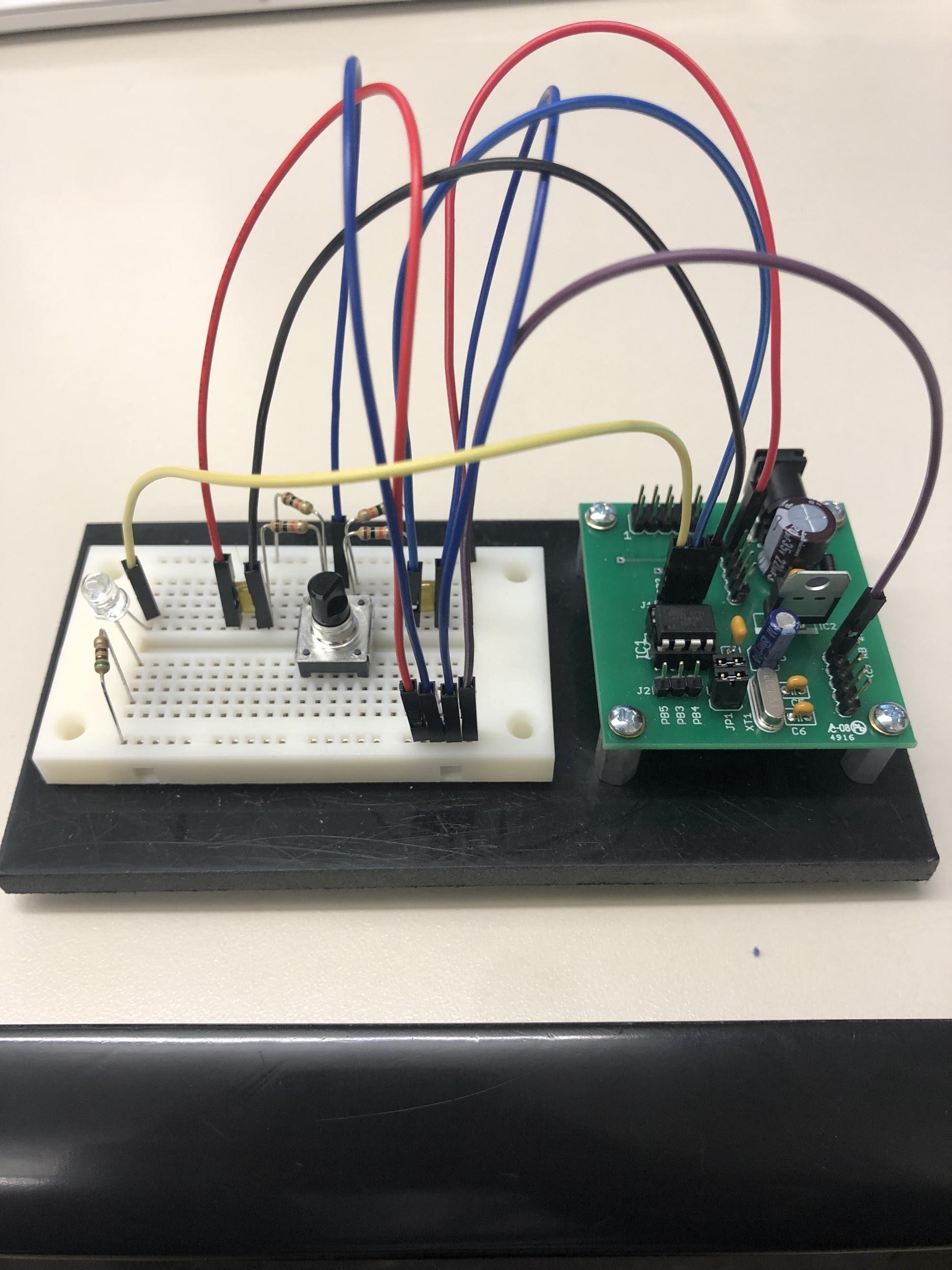
**Lab Report 3**

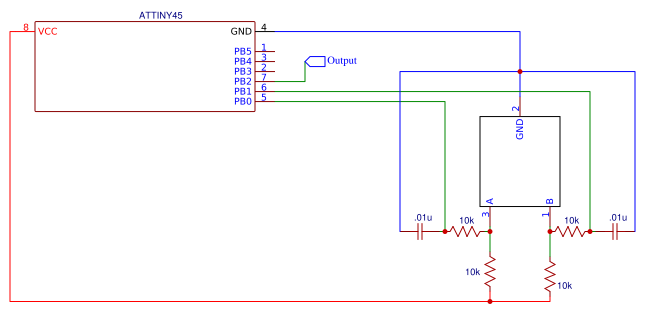
**Introduction**

For this lab we constructed a system to control the duty cycle of a 3.96kHz square wave. Our device was capable of varying the duty cycle through an rotary pulse generator (RPG). This device consists of a circuit board and an assembly program. By turning the RPG clockwise, the duty cycle increased to a limit of 70% of the full duty cycle. When the RPG is turned counter clockwise, the duty cycle is made smaller until it reaches a limit of 30% of a full duty cycle. One detent has less than a 1% effect on the duty cycle. 

This device uses a timer to determine the required timer values necessary to time the waveform’s on time and off-time for a duty cycle. Next we reload the timer to begin timing a new duty cycle. This timer was found natively on the ATTiny45 microcontroller. Afterwards,to test this device we were able to monitor the generated waveform on an oscilloscope.

**Schematic**

In the following wiring diagram you can see how we connected the RPG to the microcontroller. We used an RC filter to debounce the RPG when it rotates and then connected each of the pins of the RPG into PB0 and PB1.



**Discussion**

The process of creating this circuit consisted of three pieces: creating the circuit, writing the program in assembly code, and testing our program.

Creating the circuit was simple. We followed the wiring diagram shown above. There were no major problems with creating the circuit. We initially did not have the RPG for the first check off, but then replaced it with the correct RPG when it arrived at the shop.

The circuit itself is relatively simple. There is an RPG connected to a microcontroller with capacitors used as RC filters. The rotary encoder in the rpg has three brushes. When these switches are all closed the circuit obtains a 00 value. When one brush is on and the other is off it obtains a 01 or 10 value. When they are all on it obtains a 11 value. This information is necessary as it allowed us to determine how to see if the RPG knob is being turned clockwise or counterclockwise. The ‘A’ and ‘B’ pins of the RPG knob are connected to the two I/O ports in the microcontroller. These two pins enable the information from the rotary encoder to the program.

Some notes of the circuit are as follows. When the RPG is turned it has a bounce. To avoid an RPG debounce there are two RC filters connected to ground. This limits the contact bounce and helps avoid error.

The next part of the device was developing the software. Our program utilizes six subroutines and is implemented in assembly. The program works as follows. When a detent in the RPG is detected, the two bits are both read in at the same time using the “in” command. We omitted using the “sbic” and “sbis: commands as it reads the information one bit at a time. Initially we used this approach until we learned that while each bit is being read simultaneously, one of the two bits could change and we could receive an incorrect reading of the RPG. From here we separate the two bits and place them into a “newA” and “newB” register. Simultaneously we store the old two bits into new registers as well. If results of the xor of the opposite bits are 10 then it's rotating one of the ways, if its 01 it's the other way. 11 and 00 results are both no change.

After the direction of the rotation is determined an incDC and decDC subroutine are called. We calculated how many ticks are necessary to be loaded onto the count register to avoid overflow. When the device is being turned clockwise an increment subroutine is called and increments the count register 256 - n times. N is the maximum amount of ticks available when we are utilizing 70% of the duty cycle. We decrement the count register when it is turned counter clockwise. From here we check to see if the value of the count register is over or under the max and the minimum respectively. If it is outside of the range of values then the program does nothing and the duty cycle does not increase. If it is inside the range then the necessary arithmetic is performed. In the two functions a “startwave” subroutine is called.

The “startwave” subroutine sets the TCCR0B register to 1 which enables a 8-bit prescaler. The TCNT0 is the counter for the timer and is loaded during each cycle.

**Conclusion**

Our device worked very well, our duty cycle, and frequency were very stable and did not bounce around at all. When demoing our lab to the TA we were able to prove each of the requirements of the lab were fulfilled successfully. Our minimum duty cycle was at 30%, and our maximum duty cycle was at 70%. When turning the RPG only one detent our duty cycle was changed by less than 1%. Our software successfully stopped the duty cycle at the maximum and minimum values. Overall the lab was a success, we learned how the timer hardware on the microcontroller, and we learned how to use the RPG as an input device to our microcontroller.

**Source Code**

;

; AssemblerApplication1.asm

;

; Created: 2/22/2019 12:58:25 PM

; Author : Daniel Nunez & Ben Weinberg

;

; Replace with your application code

; connected to TPIC6C595 as output

cbi DDRB,0; B connection

cbi DDRB,1; A connectoin

sbi DDRB,2; LED

; start main program

.def oldA = r16 ; holding old value of rpg

.def oldb = r17 ; holding old value of rpg

.def newA = r18 ; holding new value of rpg

.def newB = r19 ; holding new value of rpg

.def eorB = r20 ; result of eor

.def eorA = r21 ; result of eor

.def tmp1 = r23 ; Use r23 for temporary variables

.def tmp2 = r24 ; Use r24 for temporary values

.def count = r25 ; preloaded value

.def variable = r26 ; value that stores how much to change duty cycle by

.def rpg = r27

ldi count, 150

ldi tmp1, 0x02

out TCCR0B,tmp1 ; Restart timer

ldi variable, 125

start:

in rpg,PINB ; read them in at the same time

mov oldA, newA ; copy new A to old A

mov oldB, newB ; copy old b to new b

sbrs rpg, 0 ; check bits to get a&b

ldi newB,1

sbrc rpg, 0

ldi newB,0

sbrs rpg,1

ldi newA,1

sbrc rpg,1

ldi newA,0

mov eorA, newA

mov eorB, newB

eor eorA, oldB ; eor bits to check for rotation

eor eorB, oldA ; eor bits to check for rotation

cp eorA,eorB

breq startWave ; start the wave, no rotation

cp eorA,eorB

brne change ; change the variable, there is a rotation

change:

cpi eorb,00000001 ; CW turn, skip to increment duty cycle

breq incDC

cpi eorb,00000010 ; CCW turn, skip to decrement duty cycle

brne decDC

decDC: ; CCW turn

cpi variable, 0

brlt end\_if ; check for bounds

dec variable

end\_if:

rjmp startWave

incDC: ; CW turn

cpi variable, 126

brge end\_if2 ; check for bounds

inc variable

end\_if2:

rjmp startWave

startWave:

;37+var // how long high

mov tmp1, variable

subi tmp1, -37

mov count, tmp1

sbi PORTB,2

rcall delay

rcall wait

cbi PORTB,2

;166-var // how long low

ldi tmp1, 167

mov count, variable

sub tmp1, count

mov count, tmp1

rcall delay

rcall wait

rjmp start

; Wait for TIMER0 to roll over.

delay:

; Stop timer 0.

in tmp1,TCCR0B ; Save configuration

ldi tmp2,0x00 ; Stop timer 0

out TCCR0B,tmp2

; Clear over flow flag.

in tmp2,TIFR ; tmp <-- TIFR

sbr tmp2,1<<TOV0 ; Clear TOV0, write logic 1

out TIFR,tmp2

; Start timer with new initial count

out TCNT0,count ; Load counter

out TCCR0B,tmp1 ; Restart timer

ret

wait:

in tmp2,TIFR ; tmp <-- TIFR

sbrs tmp2,TOV0 ; Check overflow flag

rjmp wait

ret