Lab 3 Report

Embedded Systems

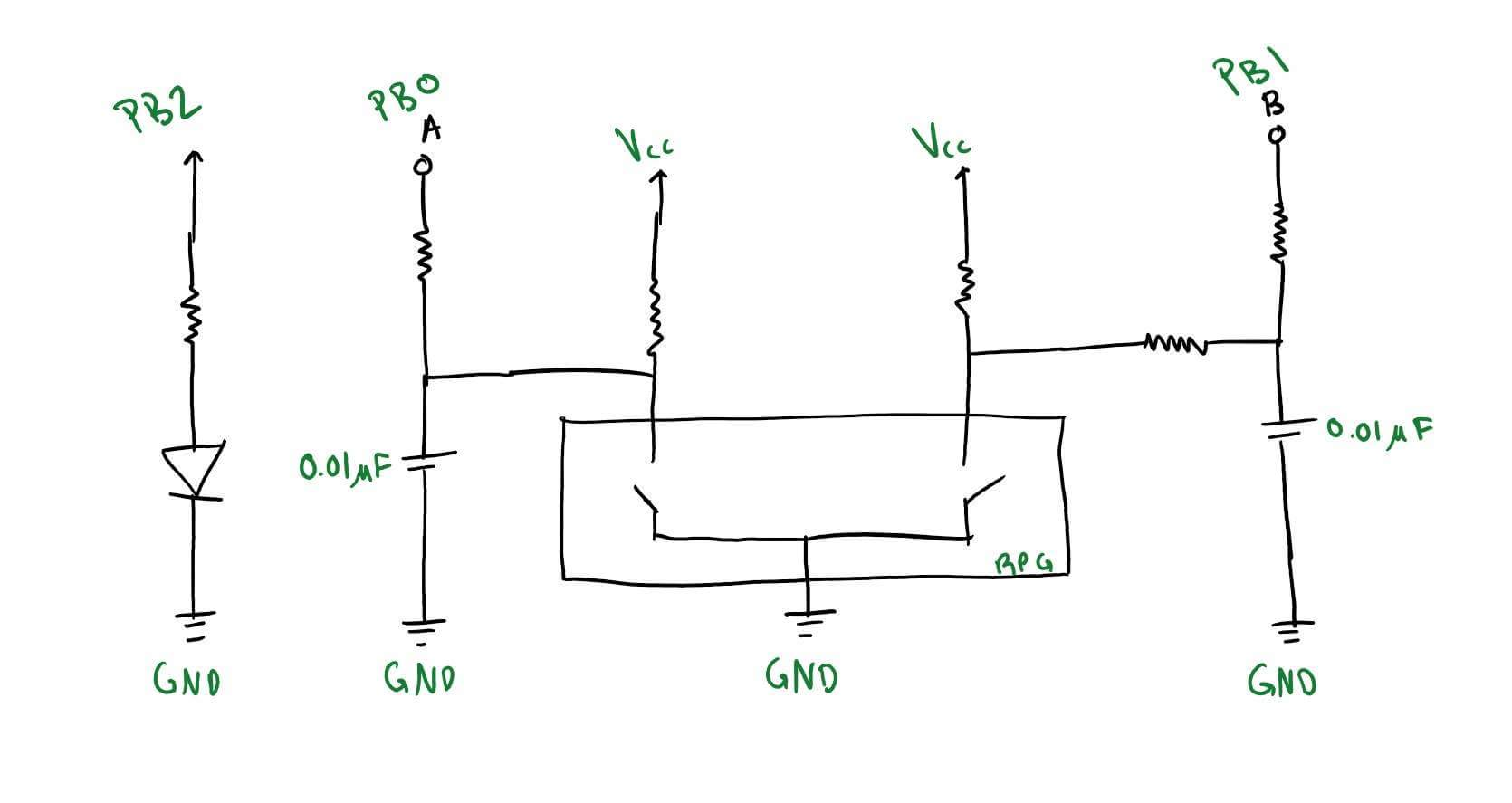
Daniel Machlab, Ted Paulsen

03/02/2018

**Introduction**

The goal of this lab was to use a rotary pulse generator RPG and an ATTiny45 microcontroller to provide a way to change the duty cycle of a 3.9 kHz square wave. We used timer routines to generate delays based upon user input from the RPG.

**Schematic**



*Figure 1: Wiring schematic showing RPG connections to the microcontroller and output from microcontroller (PB2).*

Figure 1 above shows how we implemented the RPG and microcontroller. We modeled our implementation after the suggested diagram in the RPG data sheet. Capacitors are used to debounce the RPG signals.

**Discussion**

Our first milestone for lab 3 was to determine when the RPG was being turned and whether it was going clockwise or counterclockwise. We did this by constantly reading the value of PINB into a register named prev(for previous), delaying, and then reading the value of PINB into a register named curr(for current). If prev and curr were not equal, then the RPG was turned. We XORed the first bit of prev with the zeroth bit of curr and the zeroth bit of prev with the first bit of curr. If the result was 01, then the RPG was turned clockwise. If the result was 10, then the RPG was turned counter-clockwise.

Our second milestone for lab 3 was to change the duty cycle by turning the RPG. We confirmed the output of our system by connecting the output signal (PB2) to an oscilloscope. The RPG output signal has a frequency of 3.9 kHz and the duty cycle

Duty Cycle = Time on / Period

ranges from 0.3 to 0.7. This equates to 78 to 180 microseconds of on-time. When a turn from one detent to the next is recognized, the on-time is adjusted by 1.5 microseconds. This is 0.76% of the total wave period.

We thought about the timer problem in the following way. Since the duty cycle is always between 0.30 and 0.70. The first 30% of the wave is always on and the last 30% of the wave is always off. We have a routine called delay\_30\_percent which delays for 30% of the wave which is 78 microseconds. This delay routine is called at the beginning of the period and at the 70% mark. The middle 40% of the wave is variable based on the input from the RPG. We have two delay routines which combine to 40% of the wave. When the RPG is turned clockwise, the first of these delay routines is made longer and the second is made shorter. This ensures that the total delay remains 40% of the total wave.

**Conclusion**

From this lab we gained knowledge about timer routines and learned more about how to calculate timer offsets, timer pre-scalars. We learned more about the ATTiny45 and its assembly language. We are more experienced in writing assembly code and have learned more good practices in writing subroutines. Over the last two labs we have developed a good way to handle user events by using a sort of listener routine.

**Appendix A: Source Code**

;; Lab 3

;; Ted Paulsen, Daniel Machlab

;; Embedded Systems

; cbi is input sbi is output

cbi DDRB, 0 ; input - from A

cbi DDRB, 1 ; input - from B

sbi DDRB, 2 ; output - clockwise (A side) LED

; SETUP WORK

.def curr = R20 ; R20 is the current rpg reading

.def prev = R21 ; R21 is the previous rpg reading

.def count\_temp = R19

.def tmp1 = R23

.def tmp2 = R24

.def count\_30 = R25

.def count\_rpg = R22

.def count\_rpg\_2 = R16

//.def temp\_var = R29

ldi count\_30, 0xA3 ; preload count\_30 to 160

ldi count\_rpg, 140 ; preload count\_rpg to 140

ldi count\_temp, 1

ldi count\_rpg, 240

rcall timer\_config

; load both prev and curr with same initial readings

in curr, PINB ; load inputs into prev

andi curr, 0b00000011 ; mask out all signals but A & B

mov prev, curr ; copy contents of curr into prev

rcall delay ; delay a lil bit

rpg\_listener:

;rcall lightoff

in prev, PINB

andi prev, 0b00000011

rcall lighton

rcall delay\_30\_percent ; delay for 77 us

rcall delay\_rpg\_p1 //108us ; delay for 103 us

rcall lightoff

rcall delay\_rpg\_p2 //3 us

rcall delay\_30\_percent

;rcall delay

in curr, PINB

andi curr, 0b00000011

cp prev, curr

brne rpg\_handler

rjmp rpg\_listener

; This is the infinite loop which reads new inputs

; and handles the changes

rpg\_handler:

;rcall lighton

;rcall read\_input

;rcall delay

rcall test\_rpg

; check if AB == 00

cpi R17, 0b00000000 ; sets Z flag if R17 is 0

breq stationary ; branch if Z flag set, else continue

; check if AB == 01

cpi R17, 0b00000001 ; sets Z flag if the result of the dec operation is 0

breq counterclockwise; originally clockwise ; branch if Z flag set, else continue

; check if AB == 10

cpi R17, 0b00000010 ; sets Z flag if the result of the dec operation is 0

breq clockwise; originally counterclockwise ; branch if Z flag set, else continue

; check if AB == 11

cpi R17, 0b00000011 ; sets Z flag if the result of the dec operation is 0

breq stationary ; branch if Z flag set, else continue

rjmp rpg\_handler ; finally, continue the loop

lighton:

sbi PORTB, 2

ret

lightoff:

cbi PORTB, 2

ret

; subroutine which transfers curr into prev

; and then loads new reading into curr

;read\_input:

; mov prev, curr ; copy current readings into prev

; in curr, PINB ; load new readings

; andi curr, 0b00000011 ; mask out only signals A & B

; ret

test\_rpg:

cpi curr, 0b00000000 ; if curr is 00, immediately xor with prev

breq exor\_prev

cpi curr, 0b00000011 ; if curr is 11, immediately xor with prev

breq exor\_prev

mov R17, curr

ldi R18, 0b00000011

eor R17, R18 ; if curr had 10, R17 will be loaded with 01

exor\_prev:

eor R17, prev

ret

; subroutine to handle when the rpg is stationary

; currently it turns off both LEDs

stationary:

rjmp rpg\_listener

; subroutine to hande when rpg is turning clockwise

clockwise:

cpi count\_rpg, 130

breq rpg\_listener

subi count\_rpg, 1

rjmp rpg\_listener

; subroutine to hande when rpg is turning counter-clockwise

counterclockwise:

ldi count\_temp, 1

cpi count\_rpg, 255

breq rpg\_listener

add count\_rpg, count\_temp

rjmp rpg\_listener

; a delay routine

delay:

ldi r26, 2

t4: ldi r27, 25

t5: ldi r28, 50

t6: dec r28

nop

brne t6

dec r27

nop

brne t5

dec R26

brne t4

ret

timer\_config:

ldi R30, 0x02

out 0x33, R30

ret

delay\_30\_percent:

; Stop timer

in tmp1, TCCR0B ; Save configuration

ldi tmp2, 0x00 ; Stop timer 0

out TCCR0B, tmp2 ;

; Clear timer overflow flag

in tmp2, TIFR ; tmp <-- TIFR

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

out TIFR, tmp2 ; write config back to TIFR

; Set initial counter offset and start

out TCNT0, count\_30 ; load counter

out TCCR0B, tmp1 ; restart timer

wait\_30:

in tmp2, TIFR ; tmp <-- TIFR

sbrs tmp2, TOV0 ; check overflow flag

rjmp wait\_30

ret

delay\_rpg\_p1:

; Stop timer

in tmp1, TCCR0B ; Save configuration

ldi tmp2, 0x00 ; Stop timer 0

out TCCR0B, tmp2 ;

; Clear timer overflow flag

in tmp2, TIFR ; tmp <-- TIFR

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

out TIFR, tmp2 ; write config back to TIFR

; Set initial counter offset and start

out TCNT0, count\_rpg ; load counter

out TCCR0B, tmp1 ; restart timer

wait\_p1:

in tmp2, TIFR ; tmp <-- TIFR

sbrs tmp2, TOV0 ; check overflow flag

rjmp wait\_p1

ret

delay\_rpg\_p2:

; Stop timer

in tmp1, TCCR0B ; Save configuration

ldi tmp2, 0x00 ; Stop timer 0

out TCCR0B, tmp2 ;

; Clear timer overflow flag

in tmp2, TIFR ; tmp <-- TIFR

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

out TIFR, tmp2 ; write config back to TIFR

; Set initial counter offset and start

mov count\_temp, count\_rpg

ldi count\_rpg\_2, 255

subi count\_temp, 130

sub count\_rpg\_2, count\_temp

out TCNT0, count\_rpg\_2 ; load counter

out TCCR0B, tmp1 ; restart timer

wait\_p2:

in tmp2, TIFR ; tmp <-- TIFR

sbrs tmp2, TOV0 ; check overflow flag

rjmp wait\_p2

ret