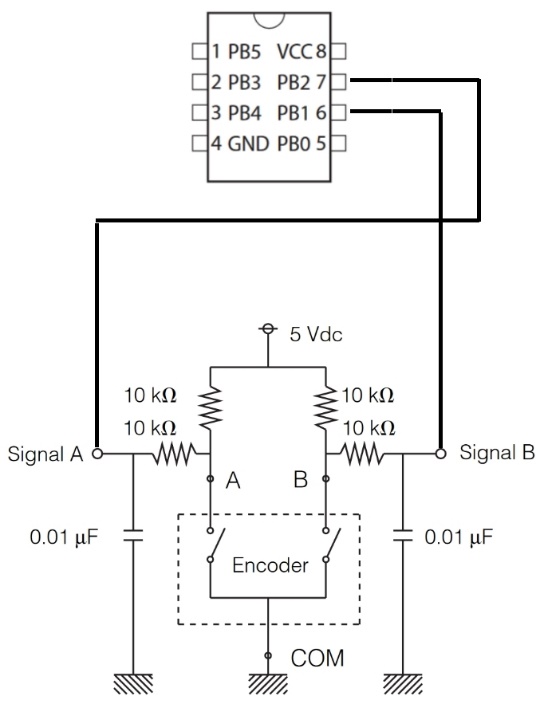
# Lab 3 – Rotary Pulse Generator Controlled Duty Cycle

## 1. Introduction

The goal of Lab 3 was to construct a system for controlling the active duty cycle of a five volts square wave, using a rotary pulse generator as an input signal. The device can vary the duty cycle of a 3.96 kHz square wave in a range from 30% to 70% in duty cycle increments of <= 1%. The duty cycle adjustment is done using the rotary pulse generator. Once outside of the range of 30% or 70%, any further rotation in an “out of range direction” has no effect on the active cycle.

## 2. Schematic

Figure 1 shows the “Test Circuit” sourced from the RPG Encoders manual (Appendix B [1]) connected with it’s B and A signal to PB1 and PB0 respectively. We choose to separately solder the “Test Circuit”, and dedicate PB2 to our output, for analysis on an oscilloscope.



(Figure. 1, RPB “Test Circuit” connected by A, B signals to pins PB0 and PB1 on the Tiny45)

## 3. Discussion

Our program implements the provided subroutine (“delay”) to issue a delay based on a 1/8 pre-scaled hardware timer for both the logical one and logical zero of our square wave. Our “main” loop continues to run endlessly and begins by calling “read\_rpg”.

Inside of “read\_rpg”, registers r28 and r29 are used temporarily to detect the current state of the rotary pulse generator pins (PB0 and PB1). This state is the stored in a register name “current\_state”, while the contents of “current\_state” are transferred to the “previous state” register.

Afterwards, “which\_direction” is called to determine directional rotation based on the contents of “current\_state” and “previous\_state”. Based on the determined direction, either the “clockwise” or “counter\_clockwise” subroutines will be called. Those subroutines will then update the “duty\_reg” accordingly within the range of 30% and 70%, determined by the “upper\_cycle\_limit” and “lower\_cycle\_limit”. Those limits were configured by repeatedly testing on the oscilloscope to configure the appropriate boundary points.

Equipped with the information of how long the active duty cycle should last, our “main” routine calls the provided “delay” function twice. First for the logical zero portion of the square wave, and then again for the logical one portion. Before each call, the “count” register is loaded with the appropriate value calculated from the previously stored “duty\_reg”.

This forms the intended control for the duty cycle in an endless loop, with its required features listed below.

Implemented Features:

|  |  |
| --- | --- |
| Frequency within +/- 1% of the nominal 3.96 kHz |  |
| Utilizes timer hardware |  |
| Generates duty cycles over the entire range from 30% through 70% in no more than 1% increments |  |
| Easy to adjust the device to any duty cycle within the specified range of operation |  |
| Doesn’t use interrupts |  |
| Don’t use the on-board PWM module |  |
| Utilizes subroutines to organize the program |  |
| Utilizes the external 10 MHz quartz crystal to achieve more accurate timing |  |

## 4. Conclusion

In this lab, we grew more comfortable navigating the flow of assembly branching and jumps, as well as managing memory and configuring built in hardware. Interacting with a timer object with a pre-scaling configuration made our program consistent and easy to debug. In addition, utilizing the RPG signals and writing the assembly needed to properly compare them was interesting, and we look forward to using these strategies in future labs, as well as on our final project.

## 5. Appendix A: Source Code

A-1: main.asm

////////////////////////////////////////////////////////////////////////////////

// Assembly language file for Lab 3 in ECE:3360 (Embedded Systems)

// Spring 2019, The University of Iowa.

//

// Desc: Lab3 Rotary Pulse Generator Controlled Duty-Cycle

//

// Authors: B. Mitchinson, A. Powers

////////////////////////////////////////////////////////////////////////////////

// definitions and preprocessor directives

//=============================================================================

// .inc include files

////////////////////////////////////////////////////////////////////////////////

.include "tn45def.inc"

////////////////////////////////////////////////////////////////////////////////

.cseg

// set DDRB

////////////////////////////////////////////////////////////////////////////////

cbi DDRB, 0

cbi DDRB, 1

sbi DDRB, 2

////////////////////////////////////////////////////////////////////////////////

// variables for current and previous state of the rpg

////////////////////////////////////////////////////////////////////////////////

.def current\_state = r16

.def previous\_state = r17

////////////////////////////////////////////////////////////////////////////////

// variables for duty-cycle range control

////////////////////////////////////////////////////////////////////////////////

.equ upper\_cycle\_limit = 162

.equ lower\_cycle\_limit = 30

.equ half\_duty\_cycle = 100

.def duty\_reg = r18

ldi duty\_reg, half\_duty\_cycle

////////////////////////////////////////////////////////////////////////////////

// variables for rotation speed

////////////////////////////////////////////////////////////////////////////////

.def rate\_reg = r19

ldi rate\_reg, 0x01

////////////////////////////////////////////////////////////////////////////////

// timer registers

////////////////////////////////////////////////////////////////////////////////

.def tmp1 = r23

.def tmp2 = r24

.def count = r25

ldi r30, 0x02

out TCCR0B, r30

ldi r30, 0x00

// rpg signal interpretation registers

////////////////////////////////////////////////////////////////////////////////

.equ BOTH\_ON = 0x00

.equ A\_ON = 0x01

.equ B\_ON = 0x02

.equ BOTH\_OFF = 0x03

// 0 - both on 0b00000000

// 1 - clockwise 0b00000001

// 2 - counter 0b00000010

// 3 - both off 0b00000011

// A is bit position 1

// B is bit position 0

// subroutines and program logic

//=============================================================================

// main method (infinite update loop)

////////////////////////////////////////////////////////////////////////////////

main:

nop

nop

rcall read\_rpg

nop

rcall which\_direction

nop

cbi PORTB, 2

ldi count, 206

sub count, duty\_reg

rcall delay

sbi PORTB, 2

mov count, duty\_reg

rcall delay

rjmp main

////////////////////////////////////////////////////////////////////////////////

// all subroutines here, grouped by functionality

////////////////////////////////////////////////////////////////////////////////

// reading from rotary pulse generator

////////////////////////////////////////////////////////////////////////////////

read\_rpg:

nop

push r28

push r29

ldi r28, 0x01

ldi r29, 0x02

mov previous\_state, current\_state

ldi current\_state, 0x00

sbis PINB, 0

add current\_state, r29 ; run if a is high

sbis PINB, 1

add current\_state, r28 ; run if b is high

pop r29

pop r28

ret

////////////////////////////////////////////////////////////////////////////////

// deciding directions and main subroutines for each direction

////////////////////////////////////////////////////////////////////////////////

// figure out the direction the rpg is being turned

which\_direction:

nop

cpi previous\_state, BOTH\_ON

breq which\_end

// if current state low

cpi current\_state, BOTH\_OFF

breq current\_low

rjmp which\_end

current\_low:

cpi previous\_state, A\_ON

breq counter\_clockwise

cpi previous\_state, B\_ON

breq clockwise

rjmp which\_end

which\_end:

ret

// clockwise

clockwise:

add duty\_reg, rate\_reg

cpi duty\_reg, upper\_cycle\_limit

brsh recover\_upper

rjmp end\_cwise

recover\_upper:

ldi duty\_reg, upper\_cycle\_limit

end\_cwise:

ret

// counter\_clockwise

counter\_clockwise:

sub duty\_reg, rate\_reg

cpi duty\_reg, lower\_cycle\_limit

brsh end\_ccwise

ldi duty\_reg, lower\_cycle\_limit

end\_ccwise:

ret

////////////////////////////////////////////////////////////////////////////////

delay:

; Stop timer 0

in tmp1, TCCR0B

ldi tmp2, 0x00

out TCCR0B, tmp2

; Clear over flow flag

in tmp2, TIFR

sbr tmp2, 1<<TOV0

out TIFR, tmp2

; Start timer with new initial count

out TCNT0, count

out TCCR0B, tmp1

; wait

wait:

in tmp2, TIFR

sbrs tmp2, TOV0

rjmp wait

ret

// exit main.asm

// (control should never reach this point as we have a main infinite loop)

.exit

## 6. Appendix B: References

*[1] Panasonic Encoders/EVEG/H/K/L :*

<https://industrial.panasonic.com/cdbs/www-data/pdf/ATC0000/ATC0000CE4.pdf>

*[2] Atmel 8-bit AVR Microcontroller:*

<http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf>