SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

LABORATORY REPORT

To: Dr. Randy Hoover

From: Sam Pfenning, Pratik Kunkolienker

Subject: Lab Assignment 6: Counters and Ultrasonic Sensors

Date: April 22, 2019

Introduction

Servos are used in a variety of applications for actuation purposes. Hobby servos are a cheap way to add motion to projects that require precise positioning and not much torque. Such as actuating the pylons on a model airplane or waving hello using a robot hand. The servos being used in the lab are micro servos and cover a 0-180° arc. The standard micro servos require a 50Hz signal with a $\approx$ 1-2ms on time. Since timers 0 and 1 on the ATMEGA328P are taken up by the wheels and ultrasonic respectively , the only way to achieve this is by using the 8-bit Timer/counter2.

Equipment

The equipment for the lab was used as follows:

1. Arduino UNO R3
2. Ultrasonic Sensor
3. Micro servo
4. Waveform Generator (Used for testing)
5. Oscilloscope (Used for testing)

Implementation

As mentioned earlier the micro servos require a 50Hz signal and a 1-2ms on time. This can be done using the phase corrected PWM mode on Timer/counter2. Timer/counter2, unlike Timer/counter1, is an 8-bit timer. Due to this the servo only has ≈ 13 that it can take. A 16-bit timer is required for higher precision.

For the servos we're using a 50Hz signal is required. Since the longest duration in Fast PWM 16.39ms, this mode is not ideal for out application. Instead Phase corrected PWM is used. This mode basically doubles our longest pulse time which is now 32.64ms.

If we chose this mode of operation we will still need to slow down the timer a bit. To do this OCR2A is enabled to be TOP and is set to 156. The value if OCR2A is found using the following formula

$$\begin{aligned} \text{OCR2A} &= \frac{\text{f\_clk}}{2 * N * \text{f\_PWM}} \\ &= \frac{16e6}{2 * 1024 * 50} \\ &= \boxed{156.25} \end{aligned}$$

We will need to round this value down in order for the microcontroller to make any sense of it. We also need the signal to be high for 1ms-2ms. To do this we us OCR2B. Using the formula OCR2B = f_clk*desired high pulse (in ms)/(2*N) and setting the appropriate COMB bits to clear OC2B on match while counting up and set while counting down(COM2B1:0=10).

The Code

The *main.c* file initializes the ultrasonic, servo and the serial by calling their respective *init()* functions. It then proceeds to the servo position and distance data to through serial to the connected computer. After completing a sweep it demonstrates that the servo can move to the center, fully CCW and fully CW.

Most of the magic happens in the *servo.h* and the *servo.c* files. *servo.h* files defines the max and min values that OCR2B can take. These correspond to the fully CCW and fully CW servo positions. It also declares the functions being used.

The *servo.c* file defines the *servo_init()* function which sets the TCCR2A and TCCR2B registers to 0b10100001 and 0b00001111 respectively. Setting TCCR2A to the value mentioned enables the phase corrected PWM mode and sets the correct settings for the COMA1:0 and COMB1:0 bits. The majority of the TCCR2B register is used to set the pre scaler value which in this case would be 1024. Further, the *servo_position* and *map* functions check if the supplied angle is valid and proceed to map the angle in degree to a value between SERVO_MIN and SERVO_MAX using the following equation,

$$OCR2B = (angle) * (SERVO\_MAX - SERVO\_MIN)/180 + SERVO\_MIN$$

Since this equation uses integer math, all decimal values are truncated. This result in a slightly skewed 90° location. To get around this a round up function was implemented. It checks to see if the decimal value is greater than 0.5 and adds another 0.5 to round it up.

Discussion

Since most of the code was provided i9n the lecture slide no major issues were encountered. There might have been some issues in agreement between the slides and the provided code but

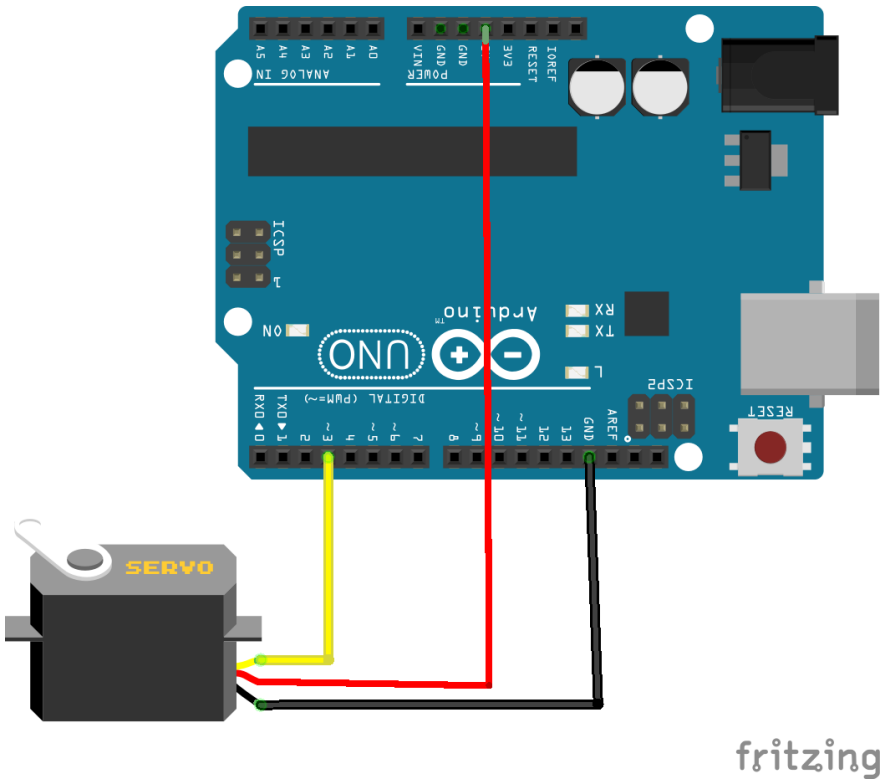

Figure 1: Circuit diagram for lab 7

these were cleared on checking the data-sheet. Unfortunately my servo broke and had to be replaced.

In the real world servos are used everywhere. Airplanes use servos to actuate their ailerons. Pressure control valves use them for precise pressure control.

Responses

1. the minimum pulse time for the servo was found to be $\boxed{0.642\text{ms}}$ and the max pulse time was found to be $\boxed{2.242\text{ms}}$
2. The Arduino Servo library uses Timer/counter 1 which is a 16-bit timer to accomplish fine control of the servo. It uses interrupts on output compare match to get facsimile of a software PWM. Since it uses interrupts, it can then use any pin to control the servo

Contributions

Contributions for this lab are scored "X/100" for each member of the team, where 100 means the team member contributed maximum efforts in completing the lab, and 0 means the team member contributed little to no effort.

Pratik - 100/100

Sam - 0/100

Appendices

A Code

A.1 main.c

```
/**
 * @file    main.c
 * @author  Pratik Kunkolienker
 * @date    22 April 2019
 * @brief   This is the main file for Lab 6.
 *
 * @details This is the entry point for the code. It initializes the required
 *      peripherals and the timers.
 */

/** @mainpage
 * @section Introduction
 *          This will serve as documentation for lab 7 for the CENG447: Embedded
 *      systems classes for the Spring of 2019.
 *
 *          Servos are used in a variety of applications for actuation
 *      purposes. Hobby servos are a cheap way to add motion to projects
 *      that require precise positioning and not much torque. Such as
 *      actuating the pylons on a model airplane or waving hello using a
 *      robot hand. The servos being used in the lab are micro servos and
 *      cover a 0-180 degree arc. The standard micro servos require a 50Hz
 *      signal with a approximately 1-2ms on time. Since timers 0 and 1 on the
 *      ATMEGA328P are taken up by the wheels and ultrasonic respectively
 *      the only way to achieve this is by using the 8-bit Timer/counter2.
 *      As always the source code can be found
 *          <a href="https://github.com/warlock31415/Embedded-CENG447/tree/master/Lab07">here</a>
 *
 *
 *
 * @section Video
 * Click
 * <a href="https://photos.app.goo.gl/kkpiPfTJJEt94HqN7">here</a> for the link
 * to the video
 *
 * @section Document
 * Download the PDF by clicking <a href="./Lab_7.pdf"> here</a>
 *
 * @section Issues
 *-# Broken servo
 *    + The servo cracked during testing so had to be replaced.
 */


#include<avr/io.h>
#include "servo.h"
#include<util/delay.h>
#include "ultrasonic.h"
#include "serial.h"


/**
* @details Initializes the Serial, ultrasonic sensor and the servo. Then does
*     a 0-180 degree sweep whilst taking distance measurements.And then
*     goes to the centered, CCW and CW positions on the servo.
```

```
*
* @returns void
**/


 int main()
{

  int i=0;
  servo_init();
  distance_init();
  ioinit();
  while(1)
  {
    for(i=0;i<=180;i++){
      printf("Position:%d",i);
      servo_position(i);
      printf(" sensor value: %d\n",distance_receive());
    }
  servo_position(90);
  _delay_ms(1000);
  servo_position(180);
  _delay_ms(1000);
  servo_position(0);
  _delay_ms(1000);
  }
  return 1;
}
```

A.2 ultrasonic.h

```
/**
 * @file     ultrasonic.h
 * @author   Pratik Kunkolienker
 * @date     27 March 2019
 * @brief    This file contains functions declarations for using the ultrasonic sensor
 *
 */
#ifndef __US
#define __US

  #include <avr/io.h>
  #include <util/delay.h>
  #include <avr/interrupt.h>

  void distance_init();
  void distance_trigger();
  int distance_receive();

#endif
```

A.3 ultrasonic.c

```
1
2  /**
3   * @file     ultrasonic.c
4   * @author   Pratik Kunkolienker
5   * @date     27 March 2019
6   * @brief    This file contains functions declarations for using the ultrasonic.
7   *
8   */
9
10 #include "ultrasonic.h"
11 #include "pin_map.h"
12 #include<avr/io.h>
13 #include <avr/interrupt.h>
14
15
16 /// Structure that contains the pulse length and the flag for PCINT1
17 struct timer_stat
18   {
19   // Stores the pulse 2*pulse length in ms
20   volatile unsigned int timer;
21   /// Checks the PCINT was triggered by a falling edge or a rising edge
22   volatile unsigned int flag;
23  };
24
25 /// Initialized the echo status
26  struct timer_stat stat = {0,0};
27
28
29
30  /**
31   * @details This function initializes Timer1 with a clk/8 prescaler. It also
32   *        the US_TRIG pis as outpu on PORTC. Further, the function
33   *        initializes pin change interrupt on the US_ECHO pin.
34   * @returns void
35   *
36   *
37   */
38 void distance_init(){
39
40   TCCR1B |= (1<<CS11); // Prescaler clk/8
41   DDRC |= (1<<US_TRIG); //set pinmode(output)
42
43   PCICR |= (1<<PCIE1);
44   PCIFR &= ~(1<<PCIF1);
45   PCMSK1 |= (1<<PCINT12);
46 }
47
48
49  /**
50   * @details Sends a 10ms pulse to trigger the ultrasonic sensor.
51   * @returns void
52   */
53 void distance_trigger(){
54   sei();
55   TCNT1 = 0;
56   PORTC |= (1<<US_TRIG);
57   while(TCNT1 < 20){  }
```

```
58   PORTC &= ~(1<<US_TRIG);
59 }
60
61  /**
62  * @details Waits for the pin to go high and then low. Once low get the amount
63  *       of time elapsed and divide it by 116 (2*58)
64  * @returns int distance
65  *
66  *
67  */
68 int distance_receive(){
69     loop_until_bit_is_set(PINC,US_ECHO);
70     loop_until_bit_is_clear(PINC,US_ECHO);
71   return stat.timer/116;
72 }
73
74
75  /**
76  * @details Handles pin change interrupt on the US_ECHO pin. TCNT1 is set to 0
77  *       on the rising edge. TCNT1 is read on the falling edge. This assures
78  *       accurate timing.
79  * @returns void
80  *
81  *
82  */
83 ISR(PCINT1_vect){
84   cli();
85   if (stat.flag == 0)
86   {
87     TCNT1=0;
88     stat.flag = 1;
89   }
90   else {
91     stat.timer = TCNT1;
92     stat.flag=0;
93   }
94   sei();
95 }
```

A.4 Servo.h

```
/**
 * @file    servo.h
 * @author  Pratik Kunkolienker
 * @date    22 April 2019
 * @brief   This is the main file for Lab 6.
 *
 * @details This is the header file for the servo library. Change the max and
 *      min servo values here as required.
  */
///Checks if the required dependancies were included
#ifndef SERVO_h
#define SERVO_H

  #include<avr/io.h>
  ///Minimum allowable OCR2B value
  #define SERVO_MIN 5.0
  ///Maximum permissible OCR2B value. If 19 is used a noticeable voltage drop
  ///is seen.
  #define SERVO_MAX 18.0

  void servo_init();
  void servo_position(uint8_t angle);
#endif
```

A.5 Servo.c

```
/**
 * @file     servo.c
 * @author   Pratik Kunkolienker
 * @date     22 April 2019
 * @brief    This is the main file for Lab 6.
 *
 * @details This file contains the required functions for the servo to work
 */

#include "servo.h"
#include "pin_map.h"

static uint8_t map(uint8_t angle);


/**
* @details Initializes Timer/counter2 to phase corrected PWM mode at 1024
*      prescaler and sets output compare registers A and B to 156 and 16
*      respectively.
* @returns void
**/
void servo_init(){
  DDRB = 0xFF;
  PORTB = 0xFF;
  DDRD = 0xFF;

  TCCR2A = 0xA1;
  TCCR2B = 0x0F;

  TCNT2=0;

  OCR2A = 156;
  OCR2B =16;
}

/**
* @details Checks to see if the position supplied is valid and sets default
*         values if not.
* @param [in] angle Specifies the servo position in degrees
* @returns void
**/
void servo_position(uint8_t angle){
  if (angle<0) angle =0;
  if (angle>180) angle =180;
  OCR2B = map(angle);

}


/**
* @details Maps the input angle to the corresponding OCR2B value
* @param [in] angle Specifies the servo position in degrees
* @returns uint8_t
**/
static uint8_t map(uint8_t angle){
  double servo_angle = (angle)*(SERVO_MAX-SERVO_MIN)/180+SERVO_MIN;
  if (servo_angle-(int)servo_angle>=0.5){
```

```
58      servo_angle = servo_angle+0.5;
59    }
60    return servo_angle;
61  }
```

A.6 Serial.h

```
#ifndef SERIAL_H
#define SERIAL_H


  #include <stdio.h>
  #include <avr/io.h>
  #include <util/delay.h>

  #define BAUD 9600
  #define MYUBRR F_CPU/16/BAUD-1


  void ioinit();
  static int uart_putchar(char c, FILE *stream);
  char uart_getchar(void);


  static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
  static FILE mystdin = FDEV_SETUP_STREAM(NULL, uart_getchar, _FDEV_SETUP_READ);
#endif
```

A.7 Serial.c

```
#include "serial.h"
#include <avr/io.h>
#include <stdio.h>

void ioinit()
{
  DDRD = 0b11111010;

  UBRR0H = ((MYUBRR)>>8);
  UBRR0L = MYUBRR;
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);

  stdout = &mystdout;
  stdin = &mystdin;
}

static int uart_putchar(char c, FILE *stream)
{
  if(c=='\n') uart_putchar('\r',stream);
  loop_until_bit_is_set(UCSR0A,UDRE0);
  UDR0 = c;
  return 0;

}

char uart_getchar(void)
{
    while(!(UCSR0A &(1<<RXC0)));
    return(UDR0);
}
```

Glossary

ATMEGA328P "A microcontroller chip manufactured by Microchip as a part of their AVR line of microcontroller chips" 1

PWM Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. 1–3