LABORATORY REPORT

To: Dr.Randy Hoover

From: Benjamin LeBrun, Benjamin Garcia

Subject: Lab Assignment 7: TC2 and Servo Control

Date: April 23, 2019

Introduction

For this lab, we utilized second internal timer to the Atmega328p to produce a PWM signal to drive our robot car's servo. This required activating Timer 2 on control pin 3 and correctly setting it to output a 50 Hz control signal. This will be used to move the ultrasonic sensor 90 degrees left and right of our robot to effectively maneuvere around obsticles and walls.

Equipment

The primary devices we used were:

- Acrylic vehicle body with screws, assembled
- Elegoo Uno (chip: Atmega328p)
- HC-SR04 ultrasonic distance sensor
- SG 90 hobby servo motor
- 2 ICR18650 batteries with battery box
- Ribbon cables
- Host laptop with AVR-gcc 8-bit toolchain
- USB 2.0 A to B cable

Configuration

Our robot vehicle was assembled according to Elegoo's instructions which can be found on Elegoo's website at https://www.elegoo.com/download/. For this lab, we are using the V3.0 version of the robot kit.

While the timers and configuration are similar to the configuration and driving of the DC motors using PWM modes from the Arduino's built in timers, for our servo we require a different operating frequency than our DC motor needs. In our case, we needed to generate a 50 Hz PWM signal from the second Timer, which required using Phase corrected mode instead of the DC motor's fast PWM mode and then writing a correct pulse width to the count registers.

This allows us to call a single function with the degree angle between 0 and 180 degrees with 0 being the leftmost position.

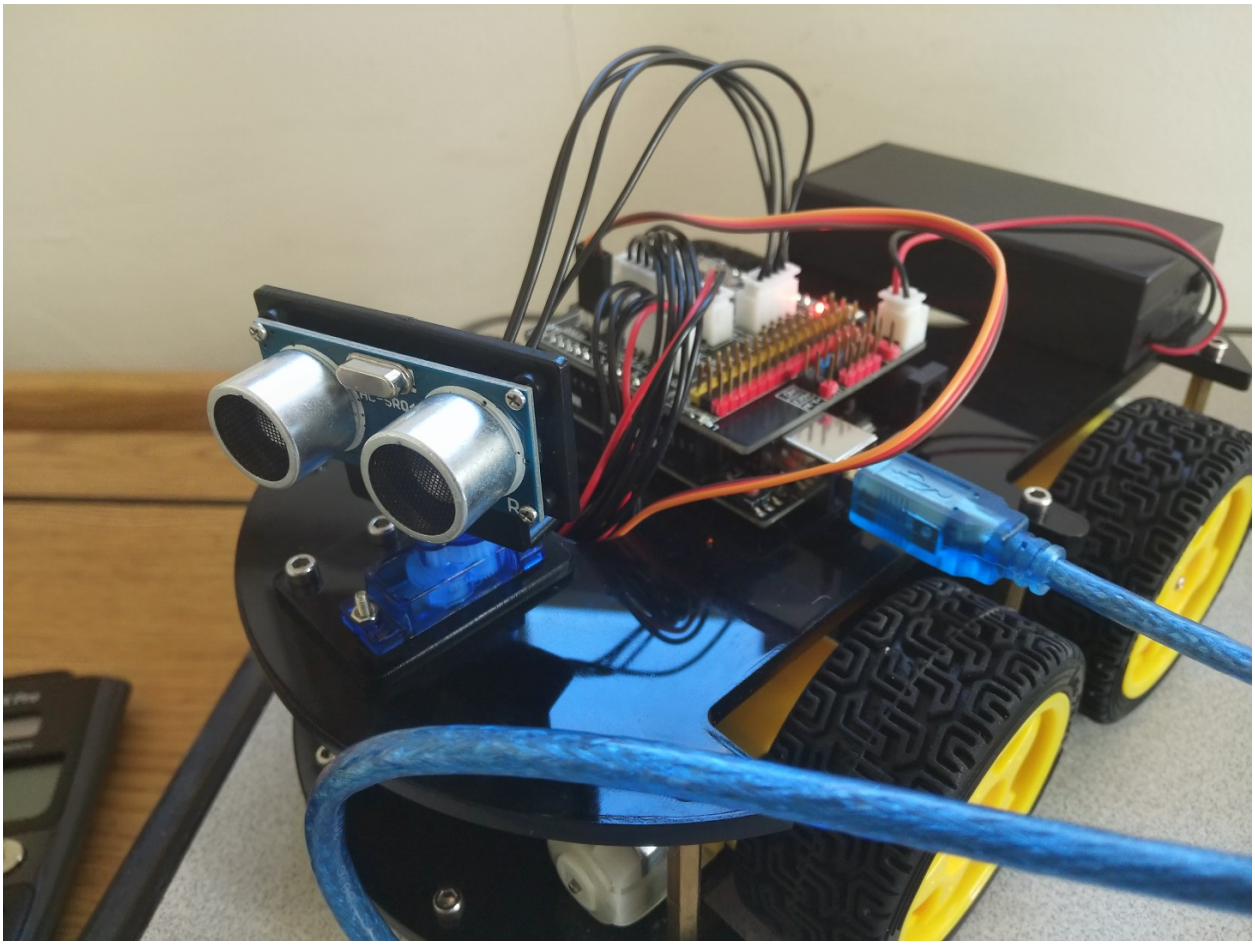

Figure 1: Servo with ultrasonic sensor assembly.

Implementation

For this lab we are writing our own servo controller that should simply convert and assign the correct pwm mode for an approximate direction.

initServo

Initializes timer 2 and sets the phase to the correct 50 Hz signal. Also sets any pins needed by the servo.

moveServo

Takes in a degree angle, maps to a PWM value then assigns to the PWM pin controlling the servo.

mapAngle

Takes in an angle and returns a mapped PWM value for the angle, exposed to allow us to read and debug.

Discussion

With this easier lab, we did some refactoring in our code to make it a little more portable and create fewer external dependancies between our different component files from previous labs. Involved a lot of using global headers and learning how the compiler attempts to link object files together containing shared global variables.

The next issue was with range mapping and how we wrote to the timers. Finding min and max ranges was largely just creating three defines, two constants and a calculation of those that found degree steps between those ranges. This was then inserted into the code where needed to streamline the process of finding the best minimum and maximum values to calculate into the timer control register.

Responses

1. Not all hobby servos are made equal. Yours may have different lower/upper bounds than 1ms or 2ms. What is your servo's pulse-width response range? How many discrete steps does this translate to in your code?

   Our range responds from approximately half a millisecond to approximately 2.3 milliseconds in pulse-width. This gives us approximately 14 steps between it's furthest left to right position with some wobble and innacuracy.

2. The Arduino libraries allow for any PWM pin to drive servos precisely, not just timer/-counter 1. How does the Arduino source code manage this feat? You will have to do some digging and exposition here.

   In the internal analogWrite() function inside the arduino libraries, the analog function only works on pins connected to a timer. Otherwise the arduino will switch to a digital in/out write.

   Otherwise, for each induvidual pin, the internal code attempts to read the correct pin/timer combo with a switch statement and will then configure quickly the correct PWM mode that takes in a value between 0 and 255 that can be directly written to the corresponding OCRXN register. To save time and create simplicity, in processing this correct pin/timer combo, if the values are 0 or 255, the analogWrite function defaults to just writing to those pins digitally high or low.

Appendices

Table of contents:

- main.c - entry, initialization and drive instructions
- bit_macros.h - bit manipulations macros
- globals.h - global variables needed by libraries using timers
- pcint.h - Pin change interrupt header for ultrasonic sensor
- pcint.c - Pin change interrupt code for ultrasonic sensor
- pin_map.h - Pin map header for device
- robotio.h - UART control header
- robotio.c - UART control code
- servo.h - Servo control library
- servo.c - Servo control code
- timers.h - Timers control library
- timers.c - Timers control code
- ultrasonic.h - Ultrasonic header file
- ultrasonic.c - Ultrasonic code file

Appendix A: main.c

```
#define F_CPU 16000000
#include "bit_macros.h"
#include "pcint.h"
#include "robotIo.h"
#include "servo.h"
#include "ultrasonic.h"
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

void printAngleDistance(unsigned char angle, unsigned int distance);

/* stdout stream */
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);

void Init()
{
    // configure UART and I/O
    initUART();

    // initialize pin change interrupts
    initPCINT();

    // configure ultrasonic range sensor
    initUltrasonic();

    // configure servo
    initServo();

    // Enable global interrupts
    sei();
}

int main()
{
    Init();
    while (1)
    {
        fprintf(&mystdout, "Starting tests:\r\n");
        // start centered
        moveServo(90);
        printAngleDistance(90, readUltrasonic());
        _delay_ms(500);
        // full CCW (left)
        moveServo(180);
        printAngleDistance(180, readUltrasonic());
        _delay_ms(500);
        // full CW (right)
        moveServo(0);
        printAngleDistance(0, readUltrasonic());
        _delay_ms(500);
        for (unsigned char i = 0; i <= 180; i += 15)
        {
            moveServo(i);
            printAngleDistance(i, readUltrasonic());
            _delay_ms(500);
        }
        _delay_ms(500);
    }
    return 1;
}

void printAngleDistance(unsigned char angle, unsigned int distance)
{
    fprintf(&mystdout, "Position: %d degrees | Distance: %d cm\r\n", angle,
            distance);
}
```

Appendix B: bit_macros.h

```
#ifndef _BIT_MACROS_H_
#define _BIT_MACROS_H

#define bitVal(bit) (1 << bit)
#define setBit(byte, bit) ((byte) |=(bitVal(bit)))
#define clearBit(byte, bit) ((byte) &= ~(bitVal(bit)))
#define togBit(byte, bit) ((byte) ^= (bitVal(bit)))

#endif
```

Appendix C: globals.h

```
#ifndef _GLOBALS_H_
#define _GLOBALS_H_
#include <stdbool.h>

/*****************************************************************************
 *
 * Overflow detection helpers
 *
 * Flag for overflow:
 * false = no overflow.
 * true = overflow happened in test
 */
volatile bool TimerOverflow;
volatile bool responseAvailable;

#endif
```

Appendix F: pcint.h

```
#include "pcint.h"

/* stdout stream */
// static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
// _FDEV_SETUP_WRITE);

void initPCINT()
{
    PCMSK1 |= PCMSK1_CONFIG;
    PCICR |= PCICR_CONFIG;
}

// specific pin change interrupt handler that is less general, but
// should be faster so less time is spent in the interrupt and
// measurements are more accurate.
//
// Expects only PCINT12 active (the ultrasonic sensor input)
ISR(PCINT1_vect)
{
    // used to detect whether pcint is going high->low or low->high
    // a value of false indicates we are low->high, a value of true indicates
    // high->low.
    static bool highEdge = false;

    if (highEdge)
    {
        turnoffTimer1();
        responseAvailable = true;
    }
    else
    {
        TIM16_WriteTCNT1(0);
        turnonTimer1();
        responseAvailable = false;
    }

    highEdge = !highEdge;
}
```

Appendix G: pcint.c

```
#include "pcint.h"

/* stdout stream */
// static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL,
// _FDEV_SETUP_WRITE);

void initPCINT()
{
    PCMSK1 |= PCMSK1_CONFIG;
    PCICR |= PCICR_CONFIG;
}

// specific pin change interrupt handler that is less general, but
// should be faster so less time is spent in the interrupt and
// measurements are more accurate.
//
// Expects only PCINT12 active (the ultrasonic sensor input)
ISR(PCINT1_vect)
{
    // used to detect whether pcint is going high->low or low->high
    // a value of false indicates we are low->high, a value of true indicates
    // high->low.
    static bool highEdge = false;

    if (highEdge)
    {
        turnoffTimer1();
        responseAvailable = true;
    }
    else
    {
        TIM16_WriteTCNT1(0);
        turnonTimer1();
        responseAvailable = false;
    }

    highEdge = !highEdge;
}
```

Appendix G: pin_map.h

```
#ifndef PIN_DEFS_H
#define PIN_DEFS_H
/*
>>> These are the ARDUINO pin mappings. <<<
#define US_RECV   12
#define US_ECHO   A4
#define US_TRIG   A5

#define H_A_EN  5
#define H_B_EN  6
#define H_IN1   7
#define H_IN2   8
#define H_IN3   9
#define H_IN4   11

#define IR_RECV  12

#define LED      13

#define LINE_R  10
#define LINE_M  4
#define LINE_L  2
*/

//<<< These are bit offsets for each pin, usable for PIN, DDR, PORT. >>>
#define US_RECV    4   // PORTB
#define US_ECHO    4   // PORTC
#define US_TRIG    5   // PORTC

// H bridge pins
#define H_A_EN     5   // PORTD PWM pin
#define H_B_EN     6   // PORTD PWM pin
#define H_IN1      7   // PORTD
#define H_IN2      0   // PORTB
#define H_IN3      1   // PORTB
#define H_IN4      3   // PORTB

#define IR_RECV    6   // PORTB

#define LED        5   // PORTB

#define LINE_R     2   // PORTB
#define LINE_M     4   // PORTD
#define LINE_L     2   // PORTD


#endif
```

Appendix H: robotio.h

```
#ifndef _ROBOT_IO_H_
#define _ROBOT_IO_H_
#include <avr/io.h>
#include <stdio.h>

/* baudrate configuration */
#define USART_BAUDRATE 9600
/* baudrate prescaler configuration */
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

/* configurations for the UART device register B*/
#define UCSR0B_CONFIG                                                   \
    (0 << RXCIE0) | (0 << TXCIE0) | (0 << UDRIE0) | (1 << RXEN0) |      \
        (1 << TXEN0) | (0 << UCSZ02) | (0 << RXB80) | (0 << TXB80);
/* configurations for the UART device register C*/
#define UCSR0C_CONFIG                                                   \
    (0 << UMSEL01) | (0 << UMSEL00) | (0 << UPM01) | (0 << UPM00) |     \
        (0 << USBS0) | (1 << UCSZ01) | (1 << UCSZ00) | (0 << UCPOL0);

/* handles inserting characters into the output stream */
int uart_putchar(char c, FILE* stream);
/* initializes UART communications using the defines above*/
void initUART();

#endif
```

Appendix H: robotio.c

```
#include "robotIo.h"

void initUART()
{
    // init uart
    // UCSR0B |= 0x98;
    // UCSR0C |= 0x06;
    UCSR0B |= UCSR0B_CONFIG;
    UCSR0C |= UCSR0C_CONFIG;

    UBRR0L = BAUD_PRESCALE;
    UBRR0H = (BAUD_PRESCALE >> 8);
}

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;
}
```

Appendix H: servo.h

```
#ifndef _SERVO_H_
#define _SERVO_H_
#include "bit_macros.h"
#include "pin_map.h"
#include "robotIo.h"
#include "timers.h"
#include <avr/io.h>

#define DEG_MAP_MAX 18
#define DEG_MAP_MIN 4
#define DEG_PER_UNIT (180 / (DEG_MAP_MAX - DEG_MAP_MIN))

void initServo();
void moveServo(unsigned char deg);
unsigned char mapAngle(unsigned char angleDeg);

#endif
```

Appendix H: servo.c

```
#include "servo.h"

/* stdout stream */
static FILE mystdout = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);

void initServo()
{
    // set pin and timers
    // set to 1 as output
    setBit(DDRD, 3);

    // init TC2

    initTimer2();
}

void moveServo(unsigned char deg) { OCR2B = mapAngle(deg); }

// mapping assumes 0 degrees is full CW (right) and 180 degrees is full CCW
// (left)
unsigned char mapAngle(unsigned char angleDeg)
{
    unsigned char result;
    if (angleDeg <= 0)
    {
        result = DEG_MAP_MIN;
    }
    else if (angleDeg >= 180)
    {
        result = DEG_MAP_MAX;
    }
    else
    {
        result = DEG_MAP_MIN + (angleDeg / DEG_PER_UNIT);
    }

    fprintf(&mystdout, "result map: %d\r\n", result);

    return result;
}
```

Appendix H: timers.h

```
#ifndef _TIMERS_H_
#define _TIMERS_H_
#include "globals.h"
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdbool.h>
#include <stdio.h>

void turnoffTimer1();
void turnonTimer1();
void initTimer0();
void initTimer1();
void initTimer2();
void TIM16_WriteTCNT1(unsigned int i);
unsigned int TIM16_ReadTCNT1();
bool getOverflowStatus();

#endif
```

Appendix H: timers.c

```
#include "timers.h"

// Getter for overflow status
bool getOverflowStatus() { return TimerOverflow; }

// Overflow vector
ISR(TIMER1_OVF_vect)
{
    // Timer 1 overflow
    TimerOverflow = true;
}

void initTimer0()
{
    // set both motors to start off
    OCR0A = 0;
    OCR0B = 0;

    // start TCNT0 at 0
    TCNT0 = 0x00;

    // Enable fast pwm mode for DC motor output
    TCCR0A = 0xA3;
    TCCR0B = 0x05; // 1024 prescaler
    // TCCR0B = 0x04; // 256 prescaler
    // TCCR0B = 0x03; // 64 prescaler
    // TCCR0B = 0x02; // 8 prescaler
    // TCCR0B = 0x01; // 1 prescaler

    // Enable counter match interrupt for counter A
    TIMSK0 = 0x02;
}

void initTimer1()
{
    /*
     * Timer 1
     * Pins 7:4 zeroes
     * tccr1a high byte zeroes
     * 16 us
     */
    OCR1A = 0;
    OCR1B = 0;
    // Start tcount at 1
    TCNT1 = 0x00;
    // Normal mode
    TCCR1A = 0x00;

    // enable overflow interrupt
    TIMSK1 |= (1 << TOIE1);

    // ensure we start with the timer off
    turnoffTimer1();
}

void initTimer2()
{
    // phase correct with 1024 prescaler gets us a high enough max time
    // use OCR2A to set TOP (WGM 5) such that we interrupt as close to 2ms
    // as possible

    // OCR2A = 16MHz/(2*1024*50Hz)
    // we choose 50Hz because 50Hz==20ms
    // OCR2A = 156.25, we can't have fractions so we use 156
    OCR2A = 156;
    // OCR2B determines the width of the pulse
    // OCR2B = 16MHz*(pulse width in ms)/(2*1024)
    // 1ms = 8, 1.5ms = 12, 2ms = 16
    // start facing forward
    OCR2B = 12;

    // start TC2 at 0
    TCNT2 = 0x00;

    // phase corrected PWM mode with 1024 clock prescale
    TCCR2A |= (0 << COM2A1) | (0 << COM2A0) | (1 << COM2B1) | (0 << COM2B0) |
              (0 << WGM21) | (1 << WGM20);
    TCCR2B |= (1 << WGM22) | (1 << CS22) | (1 << CS21) | (1 << CS20);
}

/*****************************************************************************
 *
 * Magical Timer land
 *
 * Prescaler value goes in TCCR1B
 * 0 - off
 * 1 - no prescaling
 * 2 - clock /8
 * 3 - clock /64
```

```
 * 4 - clock /256 - 16us
 * 5 - clock /1024 - 64us
 * source -
 * https://sites.google.com/site/qeewiki/books/avr-guide/timers-on-the-atmega328
 */

// convenience function to turn the timer off
void turnoffTimer1() { TCCR1B &= 0x00; }

// convenience function to turn the timer back on with a default prescaler
void turnonTimer1() { TCCR1B |= 0x05; }

// Reads from timer 1 counter
unsigned int TIM16_ReadTCNT1()
{
    unsigned char sreg;
    unsigned int i;
    // Save global interrupt flag
    sreg = SREG;
    // Disable interrupts
    cli();
    // Read TCNT1 into i
    i = TCNT1;
    // Restore global interrupt flag
    SREG = sreg;
    return i;
}

// Sets timer 1 counter
void TIM16_WriteTCNT1(unsigned int i)
{
    unsigned char sreg;
    // Save global interrupt flag
    sreg = SREG;
    // Disable interrupts
    cli();
    // Set TCNT1  to i
    TCNT1 = i;
    // Restore global interrupt flag
    SREG = sreg;
}
```

Appendix H: ultrasonic.h

```
#ifndef _ULTRASONIC_H_
#define _ULTRASONIC_H_
#include "bit_macros.h"
#include "globals.h"
#include "pcint.h"
#include "pin_map.h"
// #include "robotIo.h"
#include "timers.h"
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdbool.h>
#include <util/delay.h>

/*
 *
 * Ultrasonic sensor: A4 A5 (portc 4 5)
 * Servo Motor: 3 (portd 3)
 */

void initUltrasonic();
void triggerUltrasonic();
unsigned int readUltrasonic();

#endif
```

Appendix H: ultrasonic.c

```
#include "ultrasonic.h"

/*****************************************************************************
 *
 * Ultrasonic sensor functions
 *
 *
 * Sets the pins for ultrasonic sensor,
 * then will setup the necessary timer 1
 * to properly time the sensor
 */
void initUltrasonic()
{
    // Trig on A5
    setBit(DDRC, US_TRIG);
    // Echo on A4
    clearBit(DDRC, US_ECHO);

    clearBit(PORTC, US_TRIG);
    clearBit(PORTC, US_ECHO);

    // setBit(PORTC, US_ECHO); // pull-up for echo

    initTimer1();
}

unsigned int readUltrasonic()
{
    // Trigger the sensor
    triggerUltrasonic();
    // Spin while we either timeout or wait
    while (!TimerOverflow && !responseAvailable)
    {
    };

    unsigned int i = TIM16_ReadTCNT1();
    // 64 us per count in i
    unsigned int result = ((i * 64) / 58);

    return result;
}

// Triggers ultrasonic sensor, then waits 60 ms
void triggerUltrasonic()
{
    TimerOverflow = false;
    setBit(PORTC, US_TRIG);
    _delay_us(9);
    clearBit(PORTC, US_TRIG);
    // reset counter 1
    // TIM16_WriteTCNT1(0);
    // Delay while pulse is sent
    _delay_us(60);
}
```