LABORATORY REPORT

To: Dr. Randy Hoover

From: Sam Pfenning, Pratik Kunkolienker

Subject: Lab Assignment 6: Counters and Ultrasonic Sensors

Date: April 13, 2019

Introduction

Lab 6's goal was to generate AVR C code for initializing and configuring 16-bit timers, tracking time elapsed (semi-accurately, or as accurate as possible), and mapping the values taken from the sensor. The ultrasonic sensor was also utilized to track distance away from objects.

Equipment

The equipment for the lab was used as follows:

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

Implementation

The lab was implemented using AVR C. The ultrasonic sensor sends a high pulse whose length is proportional to the time it took for the sound wave to go from the transmitter to the receiver. An accurate timer must be used to obtain the pulse width from the echo pin. Timer1 , a 16-bit timer was chosen with a clk/8 pre-scaler. This specific pre-scaler was used because it is the largest pre-scaler that can be used to generate the 10ms trigger pulse.

This meant that in order to generate a 10ms pulse we'd have to count 20 ticks. To count the pulse width of the return pulse, a pin change interrupt was used. The pin change interrupt triggers on both falling and rising edges so accurate track must be kept of which event caused interrupt. A flag was used to do this. On a rising edge, the flag is set to 1 and TCNT1 is set to 0. On a falling edge, the flag is cleared and the value in TCNT1 is read.

While this is happening, The program first waits for the ECHO pin to read high, this is to make sure that the we are in receive mode, and then waits for the pin to go low again (The interrupt occurs during this wait time). It then returns the value that was read from TCNT1.

According to the data sheet, in order to get the distance in centimeters, the time in milliseconds must be divided by 58. Since Timer1 counts 1 tick per 0.5ms, the final value in TCNT1 must be divided by 2 before dividing it by 58 to get the distance reading.

The Code

The code starts off by including all the requisite files. These files contain functions and definitions that are needed for the the program to work. The majority of the essential code is in the *ultrasonic.c* file. This file contains functions to initialize the interrupts on PORTC, initialize Timer/Counter 1 and set the output bits on DDRC. The *distance_init()* function does this by setting the CS11 bit in the TCCR1B register, this not only enables the timer but also sets the prescaler to clk/8 which translates to 0.5ms/tick at 16MHz clock speed

To trigger the transmit on the ultrasonic, a 10ms pulse is required. The US_TRIG pin, as defined in the pin_map.h file is used to generate a semi-accurate 10ms pulse. On the *distance_trigger()* function call , the US_TRIG pin is set high. Timer1 is then used to count 20 ticks (each tick is 0.5ms) and then the US_TRIG pin is set low. This creates the required 10ms pulse.

It is required that the function *distance_receive()* be called immediately after *distance_trigger()* function. The receive function waits for the US_ECHO pin go high and successively waits for it to go low.This ensures that a pulse was actually registered. An interrupt is triggered whenever there is a pin change on the US_ECHO pin. This interrupt will be triggered while the *distance_receive()* function is waiting for the pin to go low. It then divides the number in pulse width by 116(=2*58) to get the distance in cm.

Even though different pins can be used for pin change interrupts, only 3 interrupt vectors are available, hence all the available pins are grouped together to form 3 groups. In this program PCINT1 is used to check the status of the US_ECHO pin. The interrupt in enebled in the *distance_init()* function by setting the *PCIE1* bit in the *PCICR* register.

The circuit diagram for the lab is shown in figure 1

Discussion

The biggest issue we ran into was debugging the sensor we used. Pratik's US sensor was not working properly, which we found through testing with a waveform generator, an oscilloscope, and a few hours of trouble-shooting. Our results were that it was definitely the sensor malfunctioning. Upon swapping to a different sensor our code ran as intended.

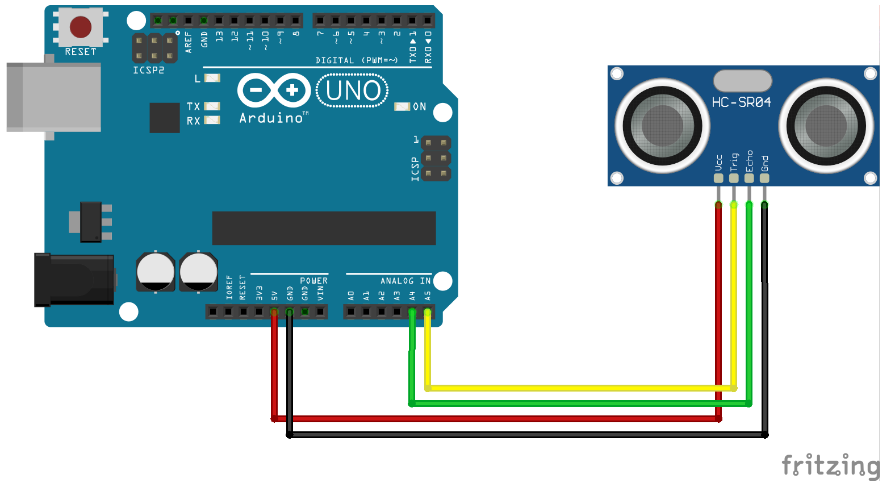

Figure 1: Circuit diagram for lab 6

A real world application of ultrasonic sensors is sonar. Sound waves are used in place of radar systems under water. Since water is a very lossy medium for electromagnetic waves to travel, sound waves are used to detect depth and other objects around the source.

Responses

1. ”Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds. Gives up and returns 0 if no pulse starts within a specified time out.

   The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length”[Geo]

2. ”The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU viathe 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations.Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bitaccess. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.”[Mic]

   Since both the read operations occur in the same clock cycle, data integrity is maintained.

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 - 100/100

Appendices

A Code

A.1 main.c

```c
/**
 * @file     main.c
 * @author   Pratik Kunkolienker
 * @date     27 March 2019
 * @brief    This is the main file for Lab 6.
 *
 * @details This is the entrypont for the code. It calls all the other functions
 *      to get the robot moving. In this case it implements a pid loop that
 *      has the robot maintian a certain distance from an object.
 */

/** @mainpage
 * @section Introduction
 *          This will serve as documentation for lab 6 for the CENG447: Embedded
 *      systems classes for the Spring of 2019.
 *
 *          Lab 6's goal was use Timer1 to count the duration of the pulse width
 *      of the echo pulse on the ultrasonic sensor and to generate an
 *      accurate 10ms trigger signal. As always the source code can be found
 *      <a href="https://github.com/warlock31415/Embedded-CENG447/tree/master/Lab06">here</a>
 *
 *      A clk/8 prescaler was used to get a tick every 0.5ms. This meant
 *      that in order to generate a 10ms pulse we'd have to count 20 ticks.
 *      To count the pulse width of the return pulse, a pin change interrupt
 *      was used. The pin change interrupt triggers on both falling and
 *      rising edges so accurate track must be kept of which event caused
 *      interrupt. A flag was used to do this. On a rising edge, the flag is
 *      set to 1 and TCNT1 is set to 0. On a falling edge, the flag is
 *      cleared and the vlaue in TCNT1 is read.
 *
 *      While this is happening, The program first waits for the ECHO pin to
 *      read high and then waits for the pin to go low again (The interrupt
 *      occures during this wait). It then returns the value that was read
 *      from TCNT1 and divides it  by 116.

 *      The division factor was obtained from the datasheet of the sensor.
 *      But since every tick is 0.5ms we must first divide it by 2 to get
 *      the time elapse in ms and then divide it by 58 (datasheet value) to
 *      get the distance.
 *
 *
 * @section Video
 * The robot can be seen measuring distances
 * <a href="https://photos.app.goo.gl/2ECBMUaLxQM8WiUg8">here</a>
 *
 * The robot can be seen keep it's distance from the wall
 *  <a href="https://photos.app.goo.gl/sDKsXncgCjGFD5kz8">here</a>
 * @section Document
 * Download the PDF by clicking <a href="./Lab_6.pdf"> here</a>
 *
 * @section Issues
 *-# Broken US
 *    + The initial ultrasonic sensor was broken and had to replaced.
 */
```

```
#include <avr/io.h>
#include <util/delay.h>
#include "ultrasonic.h"
#include<avr/interrupt.h>
#include "pin_map.h"
#include "L298.h"

/// P gain of the PID controller
#define kp 2

/// I gain of the PID controller
#define ki 0

/// D gain of the PID controller
#define kd 1

/// Set point for the PID loop
#define set_point 8

void pid();


/**
* @details initializes the timer and the motor controller
* @returns void
**/


int main()
{
  int distance;
  //ioinit();
  sei();
  distance_init();
  motor_init(5);

  while(1){
    _delay_ms(250);
    pid();
  }
}


 /**
 * @details This function handles the pid control. The function makes
      sure that the robot stays at the set point.
 * @returns void
 *
 *
 */
void pid(){
  unsigned short integral =0;
  unsigned short derivative =0;
  unsigned short last_error =0;
  char pwm;
  distance_trigger();
  int current_position = distance_receive();
  while(current_position!=set_point){

```

```
    distance_trigger();
    current_position = distance_receive();
    int error = -(set_point-current_position);

    integral = integral+error;

    derivative = error - last_error;

    pwm = (kp*error)+(ki*integral)+(kd*derivative);
    //printf("pwm=%d -> distance = %d\n",pwm,current_position);

    if(pwm>100) pwm = 100;
    else if (pwm<-100) pwm = -100;


    if(pwm>0){
      forward(pwm);
    }
    else if (pwm<0)back(pwm);

    else motor_init(0);

    last_error = error;
    //_delay_ms(250);
  }



}
```

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

```

/**
 * @file     ultrasonic.c
 * @author   Pratik Kunkolienker
 * @date     27 March 2019
 * @brief    This file contains functions declarations for using the ultrasonic.
 *
 */

#include "ultrasonic.h"
#include "pin_map.h"
#include<avr/io.h>
#include <avr/interrupt.h>


/// Structure that contains the pulse length and the flag for PCINT1
struct timer_stat
  {
  // Stores the pulse 2*pulse length in ms
  volatile unsigned int timer;
  /// Checks the PCINT was triggered by a falling edge or a rising edge
  volatile unsigned int flag;
 };

/// Initialized the echo status
 struct timer_stat stat = {0,0};



 /**
 * @details This function initializes Timer1 with a clk/8 prescaler. It also
 *       the US_TRIG pis as outpu on PORTC. Further, the function
 *       initializes pin change interrupt on the US_ECHO pin.
 * @returns void
 *
 *
 */
void distance_init(){

  TCCR1B |= (1<<CS11); // Prescaler clk/8
  DDRC |= (1<<US_TRIG); //set pinmode(output)

  PCICR |= (1<<PCIE1);
  PCIFR &= ~(1<<PCIF1);
  PCMSK1 |= (1<<PCINT12);
}


 /**
 * @details Sends a 10ms pulse to trigger the ultrasonic sensor.
 * @returns void
 */
void distance_trigger(){
  sei();
  TCNT1 = 0;
  PORTC |= (1<<US_TRIG);
  while(TCNT1 < 20){  }
```

```
  PORTC &= ~(1<<US_TRIG);
}

 /**
 * @details Waits for the pin to go high and then low. Once low get the amount
 *       of time elapsed and divide it by 116 (2*58)
 * @returns int distance
 *
 *
 */
int distance_receive(){
    loop_until_bit_is_set(PINC,US_ECHO);
    loop_until_bit_is_clear(PINC,US_ECHO);
  return stat.timer/116;
}


 /**
 * @details Handles pin change interrupt on the US_ECHO pin. TCNT1 is set to 0
 *       on the rising edge. TCNT1 is read on the falling edge. This assures
 *       accurate timing.
 * @returns void
 *
 *
 */
ISR(PCINT1_vect){
  cli();
  if (stat.flag == 0)
  {
    TCNT1=0;
    stat.flag = 1;
  }
  else {
    stat.timer = TCNT1;
    stat.flag=0;
  }
  sei();
}
```

A.4 L298.h

```
/**
 * @file     L298.h
 * @author   Pratik Kunkolienker
 * @date     27 March 2019
 * @brief    This file contains functions declarations for using the L298 motor
 *       driver.
 *
 * @details This file declares the L298 functions.
 *
 */
#ifndef L298_H
#define L298_H
  #include <avr/io.h>
  #include <util/delay.h>
  #include "pin_map.h"

  /**
    The L298 class object that has all the motor related functions
  **/
    void motor_init(char clk);
    void turn_right(int speed);
    void forward(int speed);
    void back(int speed);
    void turn_left(int speed);

    void square_turn(int speed);
    void circ();

    char map(int d_cyc);


#endif
```

A.5 L298.c

```
/**
 * @file     L298.c
 * @author   Pratik Kunkolienker
 * @date     27 March 2019
 * @brief    This file contains functions for using the L298 motor driver
 *
 */




#include "L298.h"
#include "pin_map.h"
#include <avr/io.h>
#include <util/delay.h>



/**
* @details  This function initializes Timer0 by setting the compare match
*      registers A and B to clear on compare match. Setting the
*      Waveform generation mode bits 1 and 0 and the PWM clock frquency.
*      This function also sets appropriate pins on PORTD as output.
* @returns  void
**/

void motor_init(char clk){
  TCCR0A |= (1<<COM0A1); //Clear OC0A on compare match
  TCCR0A |= (1<<COM0B1); //Clear OC0B on compare match

  TCCR0A |= ((1<<WGM01)|(1<<WGM00)); // Fast PWM mode 3

  TCCR0B |= clk;//(1<<CS02)|(0<<CS01)|(0<<CS00);

  OCR0A = map(0);
  OCR0B = OCR0A;

  DDRD |= (1<<H_A_EN);
  DDRD |= (1<<H_B_EN);

  TCNT0 = 0;


}

/**
* @details    This is a private function which is only available to functions
*        inside the class. It takes a percentage of speed as input 100
*        being max and 0 being the minimum and maps it from 0 to 255.
* @param [in]   duty_cyc Percentage of max speed
* @returns  char duty_cyc*255/100
**/
 char map(int duty_cyc)
{
  return duty_cyc*255/100;
}

```

```
/**
* @details  This function sets the direction of the motors such that the left
*   side motors turn forwards and the right side mmotors turn the opposite
*   direction. The function calls the \a map() such that a percentage is
*   mapped to a 0-255 range.
*@param [in] speed A percentage of max speed 0-100%
* @returns  void
**/
void turn_right(int speed){

  PORTD |= (1<<H_IN1);
  PORTB &= ~(1<<H_IN2);
  PORTB |= (1<<H_IN3);
  PORTB &= ~(1<<H_IN4);

  OCR0A = map(speed);
  OCR0B = OCR0A;
}

/**
* @details  This function sets the direction of the motors such that the both
*   run forwards. The function calls the \a map() such that a percentage is
*   mapped to a 0-255 range.
*@param [in] speed A percentage of max speed 0-100%
* @returns  void
**/
void forward(int speed){
  PORTD |= (1<<H_IN1);
  PORTB &= ~(1<<H_IN2);
  PORTB |= (1<<H_IN4);
  PORTB &= ~(1<<H_IN3);

  OCR0A = map(speed);
  OCR0B = OCR0A;
}

/**
* @details  This function sets the direction of the motors such that the right
*   side motors turn forwards and the left side mmotors turn the opposite
*   direction. The function calls the \a map() such that a percentage is
*   mapped to a 0-255 range.
*@param [in] speed A percentage of max speed 0-100%
* @returns  void
**/
void  turn_left(int speed){
  PORTD &= ~(1<<H_IN1);
  PORTB |= (1<<H_IN2);
  PORTB |= (1<<H_IN4);
  PORTB &= ~(1<<H_IN3);

  OCR0A = map(speed);
  OCR0B = OCR0A;
}

/**
* @details  This function sets the direction of the motors such that the both
*   run backwards. The function calls the \a map() such that a
*   percentage is mapped to a 0-255 range.
*@param [in] speed A percentage of max speed 0-100%
```

```
* @returns  void
**/
void back(int speed){
  PORTD &= ~(1<<H_IN1);
  PORTB |= (1<<H_IN2);
  PORTB &= ~(1<<H_IN4);
  PORTB |= (1<<H_IN3);

  OCR0A = map(speed);
  OCR0B = OCR0A;
}

/**
* @details  Calls the \a turn_right() function 4 times such that the robot
*      traces a square whose side length is as long as 3s in equivalent
*      distance. The spedd is set to 0 at the end.
*@param [in] speed A percentage of max speed 0-100%
* @returns  void
**/
void square_turn(int speed){
  forward(speed);
  _delay_ms(1000);
  turn_right(speed);
  _delay_ms(1000/2);
  forward(speed);
  _delay_ms(1000);
  turn_right(speed);
  _delay_ms(1000/2);
  forward(speed);
  _delay_ms(1000);
  turn_right(speed);
  _delay_ms(1000/2);
  forward(speed);
  _delay_ms(1000);
  motor_init(0);

}

/**
* @details  Sets the right side speed a little faster than the left speed
*        so that the robot traces a circle. The radius of the circle is
*      a relationship of the difference between the wheel speeds.
* @returns  void
**/
void circ()
{
  PORTD |= (1<<H_IN1);
  PORTB &= ~(1<<H_IN2);
  PORTB |= (1<<H_IN4);
  PORTB &= ~(1<<H_IN3);

  OCR0A = map(100);
  OCR0B = map(30);
}
```

References

[Geo] Francesco Georg. *Arduino*. URL: https://ccrma.stanford.edu/~fgeorg/250a/lab2/arduino-0019/reference/PulseIn.html. (accessed:Apr 13, 2019).

[Mic] Microchip. *ATmega48P/88P/168P/328P*. Version 8025I AVR 02/09.