LABORATORY REPORT

To: Dr.Randy Hoover

From: Benjamin LeBrun, Benjamin Garcia

Subject: Lab Assignment 6: TC1 and Ultrasonic Sensors

Date: April 12, 2019

Introduction

This lab tasked us with configuring the provided HC-SR04 ultrasonic range sensor using the ATMega328P's Timer/Counter1. We then had to provide a linear mapping from $[3cm, 30cm] -> [0, 255]$ and control the robot's motor speed using this mapping. The sensor output and appropriate mapping was to then be displayed to the screen with every distance sample.

Equipment

While the lab used the entire robot kit assembled together, the primary devices we used were:

- Acrylic vehicle body with screws, assembled
- Elegoo Uno (chip: Atmega328p)
- 4 DC motors with wheels, screws
- L298 H bridge module dual channel
- 2 ICR18650 batteries with battery box
- Ribbon cables
- Host laptop with AVR-gcc 8-bit toolchain
- USB 2.0 A to B cable
- HC-SR04 ultrasonic range sensor

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.

The ultrasonic sensor, battery pack, and H bridge were connected to the marked locations on the Elegoo shield included in the kit. The wheel motors were connected two to a side with the H bridge in the lower portion of the robot.

While the motors require more power than the micro-controller can provide to operate, the ultrasonic sensor is capable of running off of the micro-controller's power. Both components are capable of running off of the kit-provided battery pack when it is switched on.

Implementation

For this assignment, we implemented some basic timer control and interfacing logic for the ultrasonic sensor. Additionally, we created the beginning of a pin-change interrupt library to help with controlling the robot later on as the complexity of the assignments increases.

In addition to the new code, the motor control code from the last lab assignment has been brought over to the current assignment, and implementation details are reproduced here from the prior assignment.

initPCINT

Initialization logic for the pin-change interrupts. Sets the pin change interrupt control registers *PCMSK1* and *PCICR* using the defined bit strings *PCMSK1_CONFIG* and *PCICR_CONFIG* provided in the file 'pcint.h'.

ISR(PCINT1_vect) (interrupt)

Interrupt handler for the second pin change interrupt vector on the board. This assignment only required the use of PCINT12 (PORTC4, pin labeled on the board as A4), which allowed the interrupt to be simplified from the more general approach of capturing the current pin state and comparing with the previous state. This also made the interrupt take less time, and therefore provide a more accurate reading for the distance.

initUltrasonic

Initialization logic for the ultrasonic sensor and TC1. Configures TC1's overflow interrupt as well as disabling its output pins. starts the timer in the 'off' (disabled) state. TC1 is run in normal mode.

turnoffTimer1

Helper function to provide a more readable mnemonic for disabling TC1. Works by assigning 0 to *TCCR1B*.

turnonTimer1

Helper function to provide a more readable mnemonic for enabling TC1. Works by assigning $0x05$ to *TCCR1B*.

triggerUltrasonic

Sends a $10\mu s$ pulse to the ultrasonic sensor to trigger a 'ping'. Waits $60ms$ after the ping is triggered as suggested in the device's documentation.

getOverflowStatus

'Getter' function to provide access to the TimerOverflow variable. This can be used to provide specialized behaviors in the event that TC1 overflows while measuring a distance.

receiveUltrasonic

Reads from TC1 and returns the distance in centimeters corresponding to the time count in TC1. The equation used for this is $(i * 64)/58$.

TIM16_ReadTCNT1

Function to read from TC1 as an atomic operation. Disables the global interrupt bit until the read is completed, stores the timer value into the 'i' local variable, re-enables interrupts, and finally returns 'i'.

TIM16_WriteTCNT1

Similar to the read operation, performs an atomic write by disabling the global interrupt bit. the unsigned integer parameter is written into the TCNT1 register, and then interrupts are re-enabled before leaving the function.

initMotor

All of our motor initializations with timer preparation and scaling, PWM modes and timer interrupt enable for turn functions.

setB

Sets the OCR0B PWM pin and one side of our motor, specifically the left side motors on our robot. Set speed and direction with included enumeration inside of this motor driver file.

setA

Sets the OCR0A PWM pin and one side of our motor, specifically the right side motors on our robot. Set speed and direction with included enumeration inside of this motor driver file.

turnLeft

Simple abstraction function, activates the proper setB and setA functions with speed and length of time to be inside the function then stops.

turnRight

Simple abstraction function, activates the proper setB and setA functions with speed and length of time to be inside the function then stops.

driveForward

Simple abstraction function, drives the wheels forward for a set amount of time, then stops.

driveBackward

Simple abstraction function, drives the whels backwards for a set amount of time, then stops.

enum WHEEL_DIRECTION

Enumeration to allow ease of programming and simple calls for our motor driver functions. Values are:

- forward
- back

ISR(TIMER0_COMPA_vect) (interrupt)

Increments the MAIC by one each time the interrupt is invoked. This allows a rough delay functionality with an accuracy based on the period of the interrupt. The program uses a 1024 prescaler, so the period of a single interrupt cycle is 16ms. We noticed possible errors with this timing method for the smallest prescaler values.

getNumInterruptsForDuration

Computes the number of PWM periods that need to pass before the specified time has passed, to the next full period. The targetCount global is set to this value and the MAIC (Motor A Interrupt Counter) variable is compared against it to determine when the delay is finished. MAIC is set to 0 at the end of the function to prevent the delay from being incorrect.

delayUntilTargetCount

After a call to getNumInterruptsForDuration, targetCount will be set and delayUntilTargetCount will spin wait until MAIC is greater than the targetCount. MAIC is incremented by the TIMER0_COMPA_vect interrupt so the loop will be escaped after the appropriate delay has gone by.

DELAY_COUNT

Macro function to simplify the computation for getNumInterruptsForDuration. Divides the requested speed by the PWM (interrupt) period and adds 1 if the period does not evenly divide the delay.

Discussion

The biggest challenge we faced with our approach to this lab was determining what was wrong with our pin change interrupts when they failed to get responses. We found after several experiments and some online research that overly long interrupt handling logic could cause us to miss the rising/falling edges of our sensor. We had written the handler in a general style so it could track the last state and handle changes to multiple pins, but this backfired by taking too much time. The second, and current, version assumed that only one pin would change, allowing us to simplify the logic significantly.

This lab provided a useful introduction to using timers and interrupts to provide semi-asynchronous sensor reading and control, which is useful for allowing the robot to move while it is reading in data. Otherwise, we would be required to have the robot stop, take measurements, and then move again.

Responses

Appendices

Table of contents:

Appendix A: main.c

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

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

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

    // configure pin change interrupts
    initPCINT();

    // configure ultrasonic range sensor
    initUltrasonic();

    // configure motors
    initMotor();

    // Enable global interrupts
    sei();
}

unsigned char motorSpeedMap(unsigned int USDistance) {
    unsigned int ans = ((USDistance - 3) * 255 / 27);
    return (ans > 255? 255 : ans);
}

int main()
{
    Init();
    while (1)
    {
        triggerUltrasonic();
        while (!responseAvailable)
        {
        }
        unsigned int dist = receiveUltrasonic();
        fprintf(&mystdout, "Sensor reading: %dcm , mapped value: %d\n",
            dist, motorSpeedMap(dist));
        _delay_ms(500);
    }
    return 1;
}
```