**SOEN 422/2 Fall 2016**

**LAB REPORT #4**

Daniel Isakov (26872425)

Jacqueline Luo (26938949)

Kenny Nguyen (27079427)

Table of Contents

[1. Introduction 1](#_Toc466323250)

[1.1 Problem Statement 2](#_Toc466323251)

[1.2 Abbreviation and Acronyms 2](#_Toc466323252)

[2. Resources 2](#_Toc466323253)

[2.1 Hardware Resources 2](#_Toc466323254)

[2.2 Hardware Setup 2](#_Toc466323255)

[2.3 Software Resources 2](#_Toc466323256)

[2.4 Software Setup 2](#_Toc466323257)

[3. Programs 3](#_Toc466323258)

[3.1 Moving Motors Forward and Backward 3](#_Toc466323259)

[Reference Code 3](#_Toc466323260)

[3.2 Changing direction and speed from console 3](#_Toc466323261)

[Reference Code 3](#_Toc466323262)

[3.3 Return analog values 3](#_Toc466323263)

[Reference Code 3](#_Toc466323264)

[4. Discussion and Conclusion 3](#_Toc466323265)

[5. Appendix 4](#_Toc466323266)

# Introduction

The point of this lab was to be able to integrate an Atmega328 into a circuit with with H-bridges to control motors. The integration was to allow a user to alter the direction and speed of the motors through console input.

## Problem Statement

The lab is divided into 5 sub-tasks: connect the Atmega 328 to the computer via USB, set up the H-bridge connections, write a program that will make a motor run in a forward and backward direction, modify the program so that the Atmega 328 can adjust its speed according to console inputs, make the program return analog values based on PWM of the motors and their direction.

## Abbreviation and Acronyms

PWM: Pulse Width Modulation. Affects the speed of the motors. By decreasing the duty cycle between pulses happening at high frequencies, it becomes possible to convert a digital value into an analog one, and thus it becomes possible to set the motor to different speeds.

UART: Universally Asynchronous Receiver/Transmitter. A form of hardware-based communication between the computer and the Atmega 328. Data transmission is done in bytes.

# Resources

## Hardware Resources

* An Atmega 328 microcontroller, with a custom Concordia University flash programming board, and a SparkFun PGM-09825 programmer.
* An RC-car chassis with 4 wheels attached each to a motor, a 5V battery pack, and a breadboard.
* Two SN754410 H-bridges.
* A PC or Laptop (interchangeable) to serve as console.

## Hardware Setup

* The programmer and programming board are attached to the PC or Laptop.
* H-bridges are each attached to 2 motors, with one motor for each side. That is, pins 3 and 6 are responsible for one wheel and pins 11 and 14 for another.
* The Atmega 328 is attached to both H-bridges via the breadboard (See Appendix figures).
* The 5V battery pack feeds the entire circuit.
* The programmer is attached to the Atmega 328 in order the enable UART communication.

## Software Resources

* Compiler, assembler, linker and Standard C & math libraries for C/C++ code for the Atmega 328 were taken from the Atmel AVR toolchain.
* AVRDUDE was a utility to upload and manipulate compiled code into the Atmega 328.

## Software Setup

* Programing was done both in a Linux (at the Lab) and Windows (at Home) environment.
* Command console was used from both environments to invoke compiler and upload code into the Atmega.
* Code was written in simple text editors: Notepad++ and gedit.

# Programs

## Moving Motors Forward and Backward

Write a program to set up each motor to run a motor in a “forward” direction, and the second PWM to run the other motor one in a “backward direction.

In this part of the lab, we had problems getting the motors to work properly. We were sending a high to one motor controller and a low to the other, but it did not make the motor work. It turned out that we simply lacked power as when we used the on board battery, the motors did indeed work as intended.

### Reference Code

#include <avr/io.h>

#define F\_CPU 1000000UL

#include <util/delay.h> // uses F\_CPU to achieve us and ms delays

//Functions that control the car.

//Fnt = front wheels, Bak = back wheels

//fwd = forward, rev = reverse

void fwdFnt()

{

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void fwdFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm; //set the speed of the car, smaller the faster

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void revFnt()

{

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void revFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm;

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void fwdBak()

{

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void fwdBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void revBak()

{

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

revFnt();

revBak();

}

void fwdAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

fwdFnt();

fwdBak();

}

void revAll()

{

revFnt();

revBak();

}

void fwdAll()

{

fwdFnt();

fwdBak();

}

// Motor Initialization routine -- this function must be called

// before you use any of the above functions

void motors\_init()

{

// configure for inverted PWM output on motor control pins:

// set OCxx on compare match, clear on timer overflow

// Timer0 and Timer2 count up from 0 to 255

TCCR0A = TCCR2A = 0xF3;

//prescaler /1024

TCCR0B = TCCR2B = 0x05; //start the clock

// start clocks

OCR0A = OCR0B = OCR2A = OCR2B = 0;

// set pins as output

DDRD |= (1 << PD7) | (1 << PD4) | (1 << PD2) | (1 << PD3) | (1 << PD6) | (1 << PD0) | (1 << PD5);

DDRB |= (1<< PB0) |(1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4) | (1 << PB5);

}

//I took this code from online.

// delay for time\_ms milliseconds by looping

// time\_ms is a two-byte value that can range from 0 - 65535

// a value of 65535 (0xFF) produces an infinite delay

void delay\_ms(unsigned int time\_ms)

{

// \_delay\_ms() comes from <util/delay.h> and can only

// delay for a max of around 13 ms when the system

// clock is 20 MHz, so we define our own longer delay

// routine based on \_delay\_ms()

unsigned int i;

for (i = 0; i < time\_ms; i++)

\_delay\_ms(1);

}

int main()

{

motors\_init();

//fwdFnt();

//fwdFnt(100); //the smaller the nb the faster the speed. 200 seems to be too weak

fwdFnt(100);

revBak(100);

for(;;)

{

}

return 0;

}

## Changing direction and speed from console

Modify your program so that you can control the speed and direction of the motors by commands from the console.

We set 4 preset speeds. If the user enters a number from between 1-4 the speed of the car will change. 1 is slowest and 4 is fastest. If the user enters 0, the car will stop. After entering a number, the user enters ‘f’ or ‘b’ for forwards or backwards. The communication is done through UART/USB.

### Reference Code

#include <avr/io.h>

#define F\_CPU 1000000UL

#include <util/delay.h> // uses F\_CPU to achieve us and ms delays

#include <string.h>

#define USART\_BAUDRATE 4800

#define BAUD\_PRESCALE (((F\_CPU/(USART\_BAUDRATE\*16UL)))-1)

//Functions that control the car.

//Fnt = front wheels, Bak = back wheels

//fwd = forward, rev = reverse

void stop()

{

PORTB |= 0b00000000;

PORTD |= 0b00000000;

}

void fwdFnt()

{

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void fwdFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm; //set the speed of the car, smaller the faster

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void revFnt()

{

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void revFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm;

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void fwdBak()

{

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void fwdBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void revBak()

{

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

revFnt();

revBak();

}

void fwdAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

fwdFnt();

fwdBak();

}

void revAll()

{

revFnt();

revBak();

}

void fwdAll()

{

fwdFnt();

fwdBak();

}

// Motor Initialization routine -- this function must be called

// before you use any of the above functions

void motors\_init()

{

// configure for inverted PWM output on motor control pins:

// set OCxx on compare match, clear on timer overflow

// Timer0 and Timer2 count up from 0 to 255

TCCR0A = TCCR2A = 0xF3;

//prescaler /1028

TCCR0B = TCCR2B = 0x05; //start the clock

// start clocks

OCR0A = OCR0B = OCR2A = OCR2B = 0;

// set pins as output

DDRD |= (1 << PD7) | (1 << PD4) | (1 << PD2) | (1 << PD3) | (1 << PD6) | (1 << PD0) | (1 << PD5);

DDRB |= (1<< PB0) |(1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4) | (1 << PB5);

}

//I took this code from online.

// delay for time\_ms milliseconds by looping

// time\_ms is a two-byte value that can range from 0 - 65535

// a value of 65535 (0xFF) produces an infinite delay

void delay\_ms(unsigned int time\_ms)

{

// \_delay\_ms() comes from <util/delay.h> and can only

// delay for a max of around 13 ms when the system

// clock is 20 MHz, so we define our own longer delay

// routine based on \_delay\_ms()

unsigned int i;

for (i = 0; i < time\_ms; i++)

\_delay\_ms(1);

}

int main(void)

{

motors\_init();

int speed = 100;//Note: the smaller the number, the faster the wheel will spin.

UCSR0B |= (1<<RXEN0) | (1<<TXEN0); //Initialization for serial communication

UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01);

UBRR0H = (BAUD\_PRESCALE >> 8);

UBRR0L = BAUD\_PRESCALE;

for(;;)

{

// wait until a byte is ready to read

while( ( UCSR0A & ( 1 << RXC0 ) ) == 0 ){}

// grab the byte from the serial port

received\_byte = UDR0;

// wait until the port is ready to be written to

while( ( UCSR0A & ( 1 << UDRE0 ) ) == 0 ){}

if(received\_byte == '0')//If what was typed was 0

{

stop(); //make the car break

}

else if(received\_byte == '1')//If what was typed was 1

{

speed = 180; //slow speed

}

else if(received\_byte == '2')//If what was typed was 2

{

speed = 120; //medium speed

}

else if(received\_byte == '3')//If what was typed was 3

{

speed = 80; //fast speed

}

else if(received\_byte == '4')//If what was typed was 4

{

speed = 30; //fastest speed

}

if(received\_byte == 'f')

{

fwdAll(speed);

}

else if(received\_byte == 'b')

{

revBak(speed);

}

}

return 0;

}

## Return analog values

Modify your program to receive the PWM equivalent analog values, and the motor directions on the console at one second intervals using a timer.

Now the user enters ‘f’ or ‘b’ as before, but if they enter a number, the speed of the car will be set to that. At this point our car stopped working, hardware-wise. The motors were still functioning, and we checked and confirmed that the Hbridges motor controls were receiving a 1 and a 0. We could not figure out why the car would not work but here is the code we would have used.

### Reference Code

#include <avr/io.h>

#define F\_CPU 1000000UL

#include <util/delay.h> // uses F\_CPU to achieve us and ms delays

#include <string.h>

#define USART\_BAUDRATE 4800

#define BAUD\_PRESCALE (((F\_CPU/(USART\_BAUDRATE\*16UL)))-1)

//Functions that control the car.

//Fnt = front wheels, Bak = back wheels

//fwd = forward, rev = reverse

void stop()

{

PORTB |= 0b00000000;

PORTD |= 0b00000000;

}

void fwdFnt()

{

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void fwdFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm; //set the speed of the car, smaller the faster

PORTB |= 0b00100000;

PORTD |= 0b10000000;

}

void revFnt()

{

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void revFnt(unsigned char pwm)

{

OCR0A = OCR0B = pwm;

PORTB |= 0b00010001;

PORTD |= 0b00000000;

}

void fwdBak()

{

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void fwdBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000010;

PORTD |= 0b00010000;

}

void revBak()

{

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revBak(unsigned char pwm)

{

OCR2A = OCR2B = pwm;

PORTB |= 0b00000100;

PORTD |= 0b00000100;

}

void revAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

revFnt();

revBak();

}

void fwdAll(unsigned char pwm)

{

OCR0A = OCR0B = OCR2A = OCR2B = pwm;

fwdFnt();

fwdBak();

}

void revAll()

{

revFnt();

revBak();

}

void fwdAll()

{

fwdFnt();

fwdBak();

}

// Motor Initialization routine -- this function must be called

// before you use any of the above functions

void motors\_init()

{

// configure for inverted PWM output on motor control pins:

// set OCxx on compare match, clear on timer overflow

// Timer0 and Timer2 count up from 0 to 255

TCCR0A = TCCR2A = 0xF3;

//prescaler /1028

TCCR0B = TCCR2B = 0x05; //start the clock

// start clocks

OCR0A = OCR0B = OCR2A = OCR2B = 0;

// set pins as output

DDRD |= (1 << PD7) | (1 << PD4) | (1 << PD2) | (1 << PD3) | (1 << PD6) | (1 << PD0) | (1 << PD5);

DDRB |= (1<< PB0) |(1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4) | (1 << PB5);

}

//I took this code from online.

// delay for time\_ms milliseconds by looping

// time\_ms is a two-byte value that can range from 0 - 65535

// a value of 65535 (0xFF) produces an infinite delay

void delay\_ms(unsigned int time\_ms)

{

// \_delay\_ms() comes from <util/delay.h> and can only

// delay for a max of around 13 ms when the system

// clock is 20 MHz, so we define our own longer delay

// routine based on \_delay\_ms()

unsigned int i;

for (i = 0; i < time\_ms; i++)

\_delay\_ms(1);

}

int main(void)

{

motors\_init();

int speed = 100;//Note: the smaller the number, the faster the wheel will spin.

UCSR0B |= (1<<RXEN0) | (1<<TXEN0); //Initialization for serial communication

UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01);

UBRR0H = (BAUD\_PRESCALE >> 8);

UBRR0L = BAUD\_PRESCALE;

for(;;)

{

// wait until a byte is ready to read

while( ( UCSR0A & ( 1 << RXC0 ) ) == 0 ){}

// grab the byte from the serial port

received\_byte = UDR0;

// wait until the port is ready to be written to

while( ( UCSR0A & ( 1 << UDRE0 ) ) == 0 ){}

if(received\_byte == 'f') //If what was typed was f

{

fwdAll(speed); //make the wheels move forwards

}

else if(received\_byte == 'b') //If what was typed was b

{

revBak(speed);//make the wheels go backwards

}

else

{

speed = received\_byte; //we assume the user is smart and enters a number

}

}

return 0;

}

## 3.4 Output PWM

Observe the input PWM signal to an Hbridge input, and the corresponding output signal from the Hbridge to a motor. Compare and comment.

Both signals should have the same shape. The difference is that the input signal and output signals have different voltages, the input is 5V and the output of the Hbridge to motor is 10V because we gave VCC2 10V.

# Discussion and Conclusion

This lab was for us to get familiar in using Hbridges and controlling motors. We had trouble powering the motors, but that was because the power from our laptops and computers were not enough. At the beginning, there was some trouble with exactly how to wire everything but ultimately, we figured out that an Hbridge takes PWM to their enable pins, 5V to VCC1, a voltage from between 5 to 10 to VCC2 and any digital pins go to the motor control pins.

# Appendix

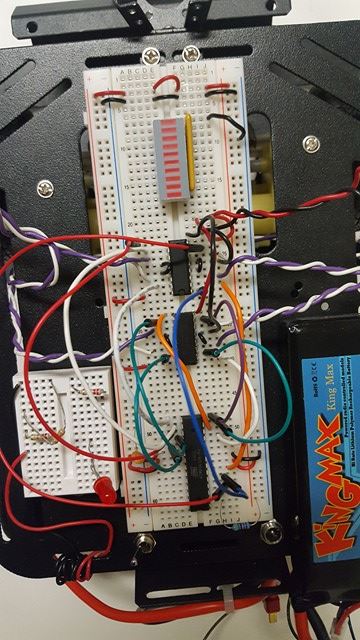


Figure 1: Completed Circuit

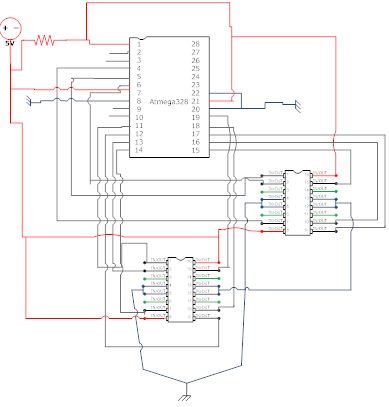


Figure 2: Circuit Diagram Atmega328 connected with digital I/O to two H-bridges