CPE301 – SPRING 2025

Design Assignment 5

Student Name: Carlos Funes

Student #: 5007995832

Student Email: [funesc2@unlv.nevada.edu](mailto:funesc2@unlv.nevada.edu)

Primary Github address: <https://github.com/CaFu0320>

Directory: <https://github.com/CaFu0320/submission_da/tree/main/DesignAssignments/DA5>

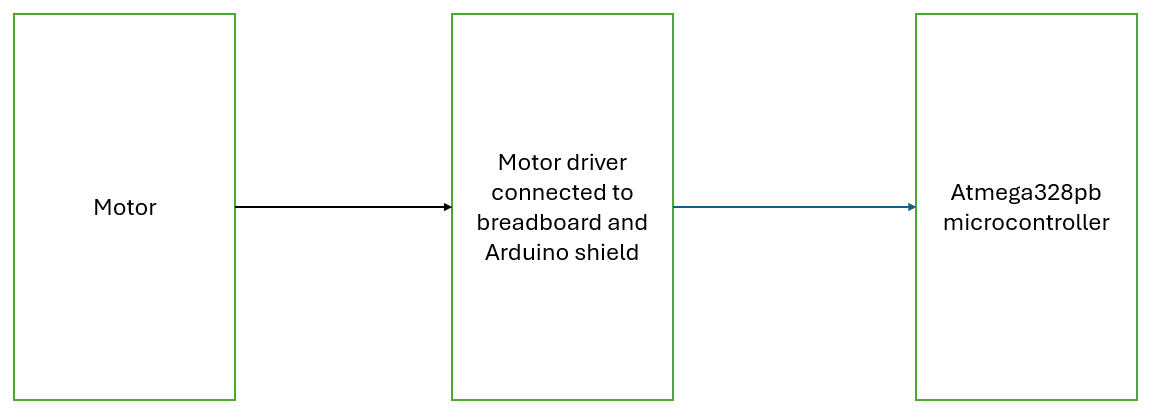
Video Playlist: <https://youtube.com/playlist?list=PLa3xS6s509hhSrDtIiRlyynXMlhipu13Z&si=vMDsBcMr7TgJm_Hq>

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

* ATMEGA328PB MINI BOARD
* ARDUINO SHIELD
* UART
* EXTERNAL POTENTIOMETER
* DC MOTOR
* POLOLU MOTOR DRIVER CHIP
* BREADBOARD



1. **C CODE TASKS**

/\*

\* DA5

\* Created: 4/19/2025 10:27:07 AM

\* Author : Carlos Funes

\*/

#define *F\_CPU* 16000000UL //defining CPU frequency

#include <avr/io.h> //including AVR I/O register definitions

#include <avr/interrupt.h> //including interrupt handling functions

#include <stdio.h> //including standard I/O functions for snprintf

#include <stdlib.h> //for functions

#include <util/delay.h> //for delay\_ms()

//global variables

volatile *uint8\_t* uartOverride = 0; //flag that decides if user should override ADC value

volatile *uint8\_t* uartPWMValue = 0; //PWM value set by the user input

volatile *uint32\_t* revCtr = 0; //number of pulses counted

char outs[72]; //buffer for messages to UART

#define PULSES\_PER\_REV 1 //pulses per one full motor rotation

//UART functions

void USART\_Init(unsigned int baud) { //setting baud rate to 9600

*uint16\_t* ubrr = *F\_CPU* / 16 / baud - 1; //calculating UART speed

UBRR0H = (*uint8\_t*)(ubrr >> 8); //setting baud rate high byte

UBRR0L = (*uint8\_t*)(ubrr); //setting baud rate low byte

UCSR0B = (1 << RXEN0) | (1 << TXEN0); //enabling transmitter and receiver

UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //8 data bits, 1 stop bit

}

void USART\_Transmit(char data) { //sending one character to UART

while (!(UCSR0A & (1 << UDRE0))); //waiting until transmit buffer is ready

UDR0 = data; //send data when ready

}

void USART\_SendString(const char\* str) { //sending strings to UART

while (\*str) USART\_Transmit(\*str++); //sending string characters

}

int USART\_Receive\_NonBlocking(void) { //checking if computer sent something

if (UCSR0A & (1 << RXC0)) return UDR0; //return if new data is received

return -1; //otherwise, don't

}

//ADC functions

void ADC\_Init() { //setting up ADC to read voltages

DDRC &= ~(1 << DDC0); //PC0 as input

ADMUX = 0x40; //using AVcc as ref voltage in channel 0

ADCSRA = 0x87; //enabling ADC and giving it a prescaler of 128

}

int ADC\_Read(char channel) { //reading ADC value from given channel

ADMUX = 0x40 | (channel & 0x07); //selecting ADC channel

ADCSRA |= (1 << ADSC); //starting ADC conversion

while (!(ADCSRA & (1 << ADIF))); //waiting until conversion is complete

ADCSRA |= (1 << ADIF); //clearing interrupt flag

return ADCW; //returning ADC values

}

//PWM Setup

void PWM\_Init() { //PWM output for motor control

DDRD |= (1 << DDD6); //PD6 for output

TCCR0A = (1 << WGM00) | (1 << WGM01) | (1 << COM0A1); //fast PWM mode

TCCR0B = (1 << CS00) | (1 << CS02); //prescaler of 1024

}

//motor direction control

void MotorDirectionInit() { //initializing motor control pins

DDRD |= (1 << DDD4) | (1 << DDD5); //setting PD4 and PD5 as outputs

PORTD |= (1 << PORTD5); //PD5 high

PORTD &= ~(1 << PORTD4); //PD4 low

}

//input capture setup

void InitTimer1(void) { //capturing encoder pulses

DDRB &= ~(1 << DDB0); //PB0 as input (ICP1)

PORTB |= (1 << PORTB0); //enabling pull-up resistor

TCNT1 = 0; //reset timer1 counter

TCCR1A = 0; //normal operation

TCCR1B = (1 << ICES1) | (1 << ICNC1); //capturing on rising edge and enable noise canceler

TIMSK1 = (1 << ICIE1); //enabling input capture interrupt

}

void StartTimer1(void) { //starting timer 1 without prescaler

TCCR1B |= (1 << CS10); //no prescaling, therefore timer1 counts at full speed

*sei*(); //enabling global interrupts

}

//ISRs

ISR(TIMER1\_CAPT\_vect) {

revCtr++; //incrementing pulse counter

USART\_SendString("CAPTURE\r\n"); //sending CAPTURE to UART

}

//UART commands

void HandleUARTCommand() { //reading user commands

static char buffer[16]; //buffer to store incoming characters

static *uint8\_t* idx = 0; //index for buffer

int c;

while ((c = USART\_Receive\_NonBlocking()) != -1) { //if something was received

if (c == '\r' || c == '\n') { //if character is enter key or newline

buffer[idx] = '\0'; //end string with a null character

if (idx > 0) { //making sure we have something typed

if (buffer[0] == 'S' || buffer[0] == 's') { //checking if command starts with S or s

*uint16\_t* val = 0; //storing PWM value from string

*uint8\_t* valid = 1; //assuming input is correct

for (*uint8\_t* i = 1; i < idx; i++) { //after S, converting each character into a number

if (buffer[i] >= '0' && buffer[i] <= '9') { //if it's a digit

val = val \* 10 + (buffer[i] - '0'); //build the number

} else {

valid = 0; //if its not a digit, we mark it as invalid

break;

}

}

if (valid && val <= 255) { //if number is valid and within 0-255 range

uartPWMValue = val; //saving PWM value

uartOverride = 1; //switching to UART control mode

*snprintf*(outs, sizeof(outs), "PWM:%u\r\n", val); //creating confirmation message

USART\_SendString(outs); //sending confirmation to UART

} else {

USART\_SendString("Invalid PWM!\r\n"); //error message for invalid input

}

} else if (buffer[0] == 'A' || buffer[0] == 'a') { //if command is A or a, switch to ADC control

uartOverride = 0;

USART\_SendString("ADC mode\r\n"); //sending confirmation message

}

}

idx = 0; //resetting buffer ready for next command

} else if (idx < sizeof(buffer) - 1) { //if character is not Enter, store in buffer

buffer[idx++] = c; //saving received character and move buffer index

} else { //if too many characters without enter key

idx = 0; //clear buffer

USART\_SendString("Cmd too long!\r\n"); //sending warning to user

}

}

}

int main(void) {

USART\_Init(9600); //UART communication at 9600 baud rate

ADC\_Init(); //preparing ADC to read potentiometer

PWM\_Init(); //setting up PWM for motor speed control

MotorDirectionInit(); //set motor to move in forward direction

InitTimer1(); //detecting motor pulses

StartTimer1(); //being pulse capture

USART\_SendString("System Ready\r\n"); //sending ready message

*uint32\_t* lastUpdate = 0; //timer for RPM calculations

*uint32\_t* lastRevCount = 0; //last pulse count

while (1) { //infinite loop

HandleUARTCommand(); //checking if user sent any commands

*uint16\_t* adcVal = ADC\_Read(0); //reading pot (ADC channel 0)

if (uartOverride) { //setting motor speed

OCR0A = uartPWMValue; //user given PWM if UART override is active

} else {

OCR0A = adcVal / 4; //using potentiometer (0-255) <<-- from 1024/255

}

if (++lastUpdate >= 1000) { //updating RPM reading every 1 second

lastUpdate = 0; //resetting counter

float rpm = 0;

*uint32\_t* pulses = revCtr - lastRevCount; //how many pulses since last update

lastRevCount = revCtr; //saving current pulse count for next update

if (pulses > 0) {

rpm = (pulses / (float)PULSES\_PER\_REV) \* 60.0; //calculating RPM

}

//creating message depending on control mode

if (uartOverride) {

*snprintf*(outs, sizeof(outs), "PWM:%u RPM:%.2f\r\n", OCR0A, rpm); //user given

} else {

*snprintf*(outs, sizeof(outs), "ADC:%u PWM:%u RPM:%.2f\r\n", adcVal, OCR0A, rpm); //potentiometer given

}

USART\_SendString(outs); //sending rpm and pwm to the terminal

}

*\_delay\_ms*(1); //delay to make loop timing better

}

}

****

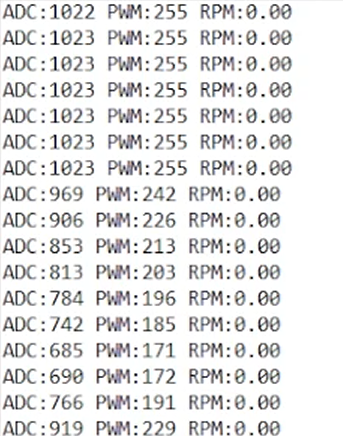
1. **SCHEMATIC**

**A computer screen shot of a computer

AI-generated content may be incorrect.**

1. **SCREENSHOTS OF EACH TASK OUTPUT (**

**TASK 1**

****

**TASK 2**

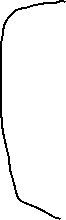
**A close up of numbers

AI-generated content may be incorrect.**

**TASK 3**

**A screenshot of a computer

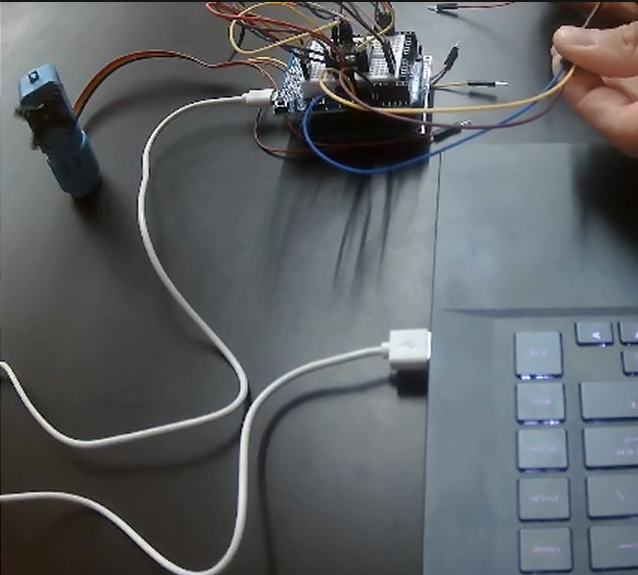
AI-generated content may be incorrect.**



**A graph with lines and numbers

AI-generated content may be incorrect.**

1. **SCREENSHOT OF DEMO**

****

1. **VIDEO LINK FOR DEMO**

Task: <https://youtu.be/Ntrmp7nqn6s>

1. **GITHUB LINK OF THIS DA**

Tasks c code: <https://github.com/CaFu0320/submission_da/tree/main/DesignAssignments/DA5>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Carlos Funes