PID Speed Control of DC Motor with Encoder

Eilat Avidan

**Goal:**

* Control the speed of the supplied DC motor with encoder using PID algorithm.
* Display the speed to the terminal using FTDI chip (USART).
* Receive speed value from user and try to adjust the motor accordingly.

**Deliverables:**

This project is divided to three different parts. The first part is to send the motor speed value to the terminal using FTDI chip. Then, the second part is to receive data from the user of the required speed and adjust the motor speed until getting the speed required. The third part is to press a pushbutton that will slow down the motor until complete stop.

# **Components:**

1. AVR ATMEGA328p – an eight-bit microcontroller, easy to use and can be programmed using C or assembly language. Total of 28 pins which 23 pins are programmable I/O Lines. There are three different timers in the chip, 2 timers are 8-bit and one of them is 16-bit timer. There is also one USART module that will be used for the project to send and receive data from and to the motor.
2. DRV8833 Dual Motor Driver- the device has two H-bridge drivers that provides a dual bridge motor solution for many applications.
3. 2,737 RPM- motor with encoder- a DC motor with encoder that can send and receive information regarding the motion of the DC motor. Maximum speed of this motor is 2737 RPM, with a maximum current of 4.9 amp (12-volt maximum voltage).
4. FTDI chip- Future Technology Devices International, used in the project to transfer data from the chip to the computer and vice versa.
5. Switch
6. Power supply (5V)

# [[1]](#footnote-1)

# Schematics

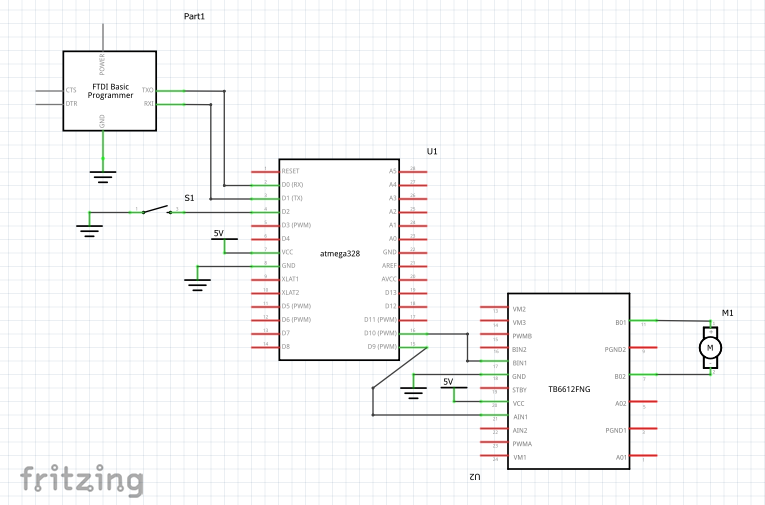


Figure 1: Figure caption centered

* *I could not find a DC motor with encoder in fritzing so I used a regular one.*

Code #1- sending the speed value to the terminal

#define *F\_CPU* 8000000UL

#define UBRR\_9600 51

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <avr/interrupt.h>

#include <stdint.h>

volatile unsigned int adc\_speed; //global variables

char outs[20];

void adc\_init(void)

{

ADMUX = (0<<REFS1)| // Reference Selection Bits

(1<<REFS0)| // AVcc - external cap at AREF

(1<<ADLAR)| // ADC left Adjust Result

(0<<MUX2)| // Analog Channel Selection Bits

(0<<MUX1)| // ADC0 Pin

(0<<MUX0);

ADCSRA = (1<<ADEN)| // ADC ENable

(1<<ADSC)| // ADC Start Conversion

(1<<ADATE)| // ADC Auto Trigger Enable

(0<<ADIF)| // ADC Interrupt Flag

(0<<ADIE)| // ADC Interrupt Enable

(1<<ADPS2)| // ADC Prescaler Select Bits

(0<<ADPS1)|

(1<<ADPS0);

}

void read\_adc(void)

{

unsigned char i =4; //set i to 4- make 4 readings

adc\_speed = 0; //initialize ADC\_TEMP

while (i--)

{

ADCSRA |= (1<<ADSC);

while((ADCSRA & (1<<ADIF)) == 0);

adc\_speed += ADCH; //sum up 4 readings

*\_delay\_ms*(50);

}

adc\_speed = adc\_speed / 4; // Average of four samples

}

void USART\_init( unsigned int ubrr )

{

UBRR0H = (unsigned char)(ubrr>>8); //set baud rate

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1 << TXEN0) | (1 <<RXEN0); // Enable receiver, transmitter

UCSR0C = (1 << UCSZ00) | (1 << UCSZ01); //asynchronous 8-bit data 1 stop bit

}

void USART\_tx\_string( char \*data )

{

while ((\*data != '\0'))

{

while (!(UCSR0A & (1 <<UDRE0))); //wait for the transmit buffer to empty

UDR0 = \*data; //put the data into the empty buffer, which sends the data

*\_delay\_ms*(125); // wait a bit

data++;

}

}

unsigned char USART\_Receive( void )

{

while ( !(UCSR0A & (1<<RXC)) )

{

// Wait for data

}

return UDR0;

}

int main()

{

adc\_init(); // Initialize the ADC (Analog / Digital Converter)

USART\_init(UBRR\_9600); // Initialize the USART (RS232 interface)

USART\_tx\_string("DC motor with encoder:\r\n");

*\_delay\_ms*(125); // wait a bit

TCCR0A |= (1 << WGM00); //fast pwm mode

TCCR0A |= (1 << WGM01); //fast pwm mode

TCCR0A |= (1 << COM0A0); //compare output mode

TCCR0A |= (1 << COM0A1); //compare output mode

TCCR0B |= (1 << CS00); //set timer counter

TCCR0B &= ~(1 << CS01);

TCCR0B &= ~(1 << CS02);

OCR0A = 128; //50 % duty cycle

sei();

while(1)

{

// wait for interrupt

}

}

ISR (TIMER0\_OVF\_vect) //overflow interrupt

{

TCCR0B = 0; //stop the counter

read\_adc();

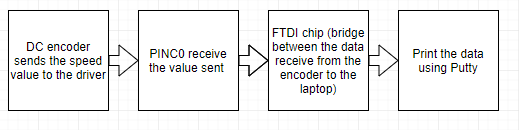
*snprintf*(outs,sizeof(outs),"%3d\r\n", adc\_speed); // print ADC value

USART\_tx\_string(outs);

*\_delay\_ms*(125); // wait a bit

TCNT0 = 128;

}



# part#2

#define *F\_CPU* 8000000UL

#define UBRR\_9600 51 // for 8Mhz with .2% error

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#include <avr/interrupt.h>

#include <stdint.h>

void read\_adc(void);

void adc\_init(void);

void USART\_init( unsigned int ubrr );

void USART\_tx\_string( char \*data );

volatile unsigned int adc\_temp;

char outs[20];

typedef struct encoder\_data

{

*int16\_t* direction;

*int32\_t* count;

} encoder;

typedef struct motor\_data

{

*int16\_t* velocity\_setpoint;

*int16\_t* velocity;

*int16\_t* integral\_error;

*int16\_t* previous\_error;

*int16\_t* kp;

*int16\_t* ki;

*int16\_t* kd;

} motor;

int encoder\_velocity (encoder \*e)

{

int velocity = *abs* (e->count);

e-> count = 0; //reset encoder's count

return velocity;

}

int do\_pid (motor \*m)

{

float p\_error; //difference between the velocity we want to the current one

float i\_error; //sum of errors

float d\_error;

float p\_output; //proportional component of the output

float i\_output;

float d\_output;

float output;

//calculate the errors

p\_error = m->velocity\_setpoint - m->velocity;

i\_error = m->integral\_error;

d\_error = p\_error - m->previous\_error;

//calculate the component of the PID output

p\_output = m->kp\*p\_error;

i\_output = m->ki\*i\_error;

d\_output = m->kd\*d\_error;

output = p\_output + i\_output + d\_output;

//update errors

m->integral\_error += p\_error;

m->previous\_error = p\_error;

return (int) output;

}

void encoder\_update (encoder \*e, int A, int B)

{

if (A == 1)

{

if (B == 1)

{

e-> direction = 1;

e-> count++;

}

else

{

e-> direction = 0;

e-> count--;

}

}

else

{

if (B == 1)

{

e-> direction = 0;

e-> count--;

}

else

{

e-> direction = 1;

e-> count++;

}

}

}

int main ()

{

TCCR0A |= (1 << WGM00); //fast pwm mode

TCCR0A |= (1 << WGM01); //fast pwm mode

TCCR0A |= (1 << COM0A0); //compare output mode

TCCR0A |= (1 << COM0A1); //compare output mode

TCCR0B |= (1 << CS00); //set timer counter

TCCR0B &= ~(1 << CS01);

TCCR0B &= ~(1 << CS02);

OCR0A = 128; //50% duty cycle

OCR0A = motor\_output (parameters, OCR0A, do\_pid(pRMotor));

//encoder initializations

encoder pREncoder = &REncoder; //pointer to right encoder structure

DDRC |= ~(1<<PC0); //encoder A input, channel A

DDRD |= ~(1<<PD3); //encoder B input, channel B

PORTC |= (1<<PC0); //resistor enable

PORTD |= (1<<PD3); //resistor enable

EICRA |= (1 << ISC10); //external interrupt, any logical change on INT1 generate an interrupt request

EIMSK |= (1 << INT1); //enable external interrupt

adc\_init(); // Initialize the ADC (Analog / Digital Converter)

USART\_init(UBRR\_9600); // Initialize the USART (RS232 interface)

USART\_tx\_string("Connected!\r\n"); // we're alive!

*\_delay\_ms*(125); // wait a bit

TIMSK1 = (1<<TOIE1); //timer1 interrupt register

TCNT1 = 34286; //timer counts to this value

TCCR1A = 0;

TCCR1B = (1<<CS12); //256 prescaler

sei();

while(1)

{

// wait for interrupt

}

}

ISR(INT1\_vect)

{

encoder\_update(&REncoder, (PIND & 8) >> 3, (PINC & 1));

}

ISR (TIMER0\_OVF\_vect) //overflow interrupt

{

TCCR0B = 0; //stop the counter

read\_adc();

*snprintf*(outs,sizeof(outs),"%3d\r\n", adc\_temp); // print ADC value

USART\_tx\_string(outs);

*\_delay\_ms*(125); // wait a bit

TCNT1 = 125;

//TCCR1B = (1<<CS12);

}

void adc\_init(void)

{

/\*\* Setup and enable ADC \*\*/

ADMUX = (0<<REFS1)| // Reference Selection Bits

(1<<REFS0)| // AVcc - external cap at AREF

(1<<ADLAR)| // ADC left Adjust Result

(0<<MUX2)| // Analog Channel Selection Bits

(0<<MUX1)| // ADC0 Pin

(0<<MUX0);

ADCSRA = (1<<ADEN)| // ADC ENable

(1<<ADSC)| // ADC Start Conversion

(1<<ADATE)| // ADC Auto Trigger Enable

(0<<ADIF)| // ADC Interrupt Flag

(0<<ADIE)| // ADC Interrupt Enable

(1<<ADPS2)| // ADC Prescaler Select Bits

(0<<ADPS1)|

(1<<ADPS0);

}

/\* READ ADC PINS\*/

void read\_adc(void)

{

unsigned char i =4; //set i to 4- make 4 readings

adc\_temp = 0; //initialize ADC\_TEMP

while (i--)

{

ADCSRA |= (1<<ADSC);

while((ADCSRA & (1<<ADIF)) == 0);

adc\_temp += ADCH\*2; //sum up 4 readings

*\_delay\_ms*(50);

}

adc\_temp = adc\_temp / 4; // Average of four samples

}

/\* INIT USART (RS-232) \*/

void USART\_init( unsigned int ubrr )

{

UBRR0H = (unsigned char)(ubrr>>8);

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1 << TXEN0) | (1 <<RXEN0); // Enable receiver, transmitter

UCSR0C = (1 << UCSZ00) | (1 << UCSZ01); //asynchronous 8-bit data 1 stop bit

}

/\* SEND A STRING TO THE RS-232\*/

void USART\_tx\_string( char \*data )

{

while ((\*data != '\0')) //loop until the end of the string

{

while (!(UCSR0A & (1 <<UDRE0))); //wait until flag set

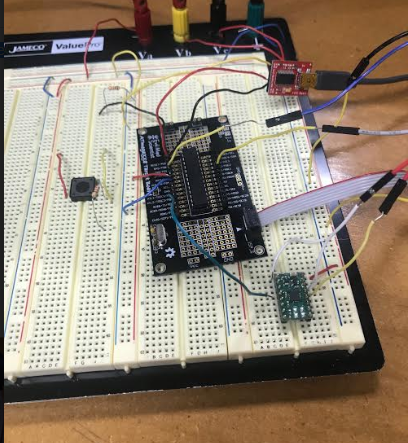
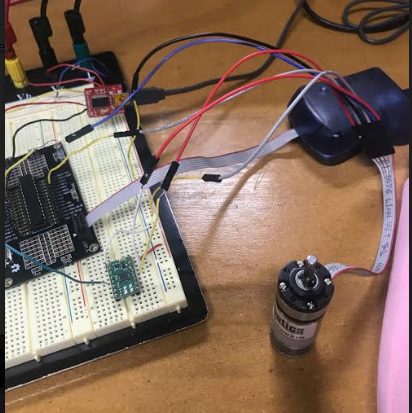
UDR0 = \*data;

*\_delay\_ms*(125); // wait a bit

data++;

}

}



# Conclusion

The project combines many elements that need to work together to perform the required output. Divide it to two parts help a lot to design the project. Even though the code works, I had problem with the encoder that did not work as expected.

1. [↑](#footnote-ref-1)