Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

RAPORT

la Programarea aplicațiilor incorporate și independente de platformă

Lucrare de laborator Nr.4

Tema:”Pulse Width Modulation”

A efectuat st. gr. FAF-141: Bega Valeria

A verificat: Andrei Bragarenco

Chişinău 2017

Topic:

Actuators. Generate PWN signal.

Objectives:

- Learn how to program a microcontroller in Atmel Studio
- Learn to build a scheme in Proteus
- Study and understand PWN signals

Task:

Debelop an application that will read data when pressing a button and rotate the mothor either clockwise or counterclockwise.

Overview of the work:

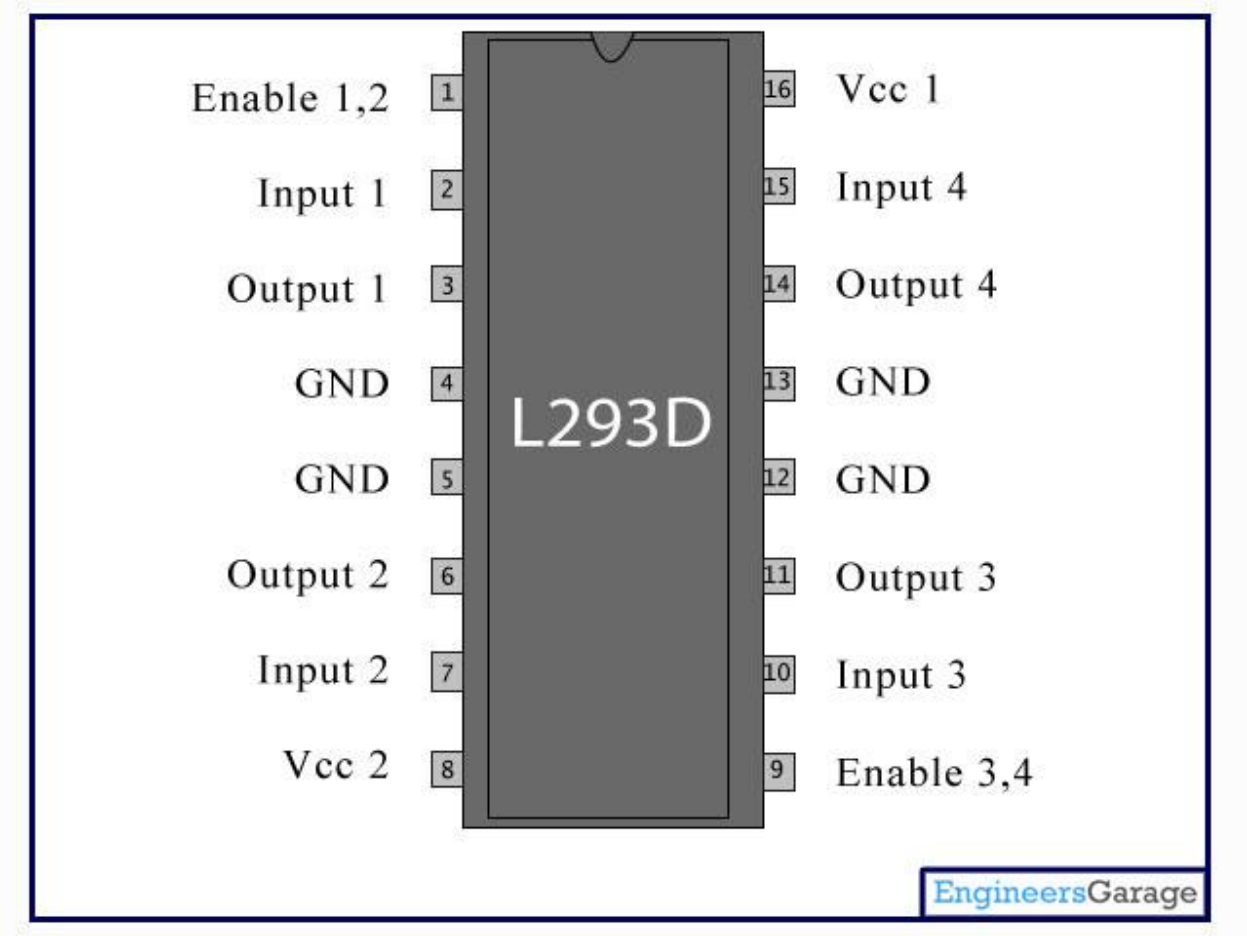
1. PWN signal

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

2. L293 Driver

The L293 is an integrated circuit motor driver that can be used for simultaneous, bi-directional control of two small motors. Small means small. The L293 is limited to 600 mA, but in reality can only handle much small currents unless you have done some serious heat sinking to keep the case temperature down. Unsure about whether the L293 will work with your motor? Hook up the circuit and run your motor while keeping your finger on the chip. If it gets too hot to touch, you can't use it with your motor.

The L293 comes in a standard 16-pin, dual-in line integrated circuit package.

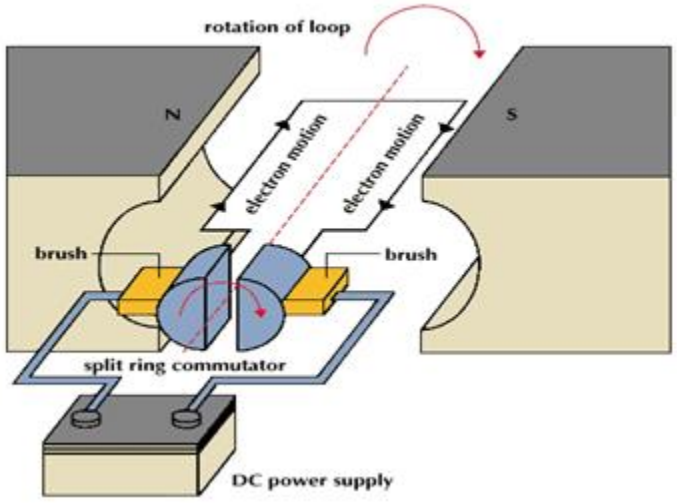


Pin No	Function	Name
1	Enable pin for Motor 1; active high	Enable 1,2
2	Input 1 for Motor 1	Input 1
3	Output 1 for Motor 1	Output 1
4	Ground (0V)	Ground
5	Ground (0V)	Ground
6	Output 2 for Motor 1	Output 2
7	Input 2 for Motor 1	Input 2
8	Supply voltage for Motors; 9-12V (up to 36V)	Vcc_2
9	Enable pin for Motor 2; active high	Enable 3,4
10	Input 1 for Motor 1	Input 3
11	Output 1 for Motor 1	Output 3
12	Ground (0V)	Ground
13	Ground (0V)	Ground
14	Output 2 for Motor 1	Output 4
15	Input2 for Motor 1	Input 4
16	Supply voltage; 5V (up to 36V)	Vcc_1

3. DC Motors

A motor is an electrical machine which converts electrical energy into mechanical energy. The principle of working of a DC motor is that "whenever a current carrying conductor is placed in a magnetic field, it experiences a mechanical force". The direction of this force is given by Fleming's left hand rule and it's magnitude is given by F = BIL. Where, B = magnetic flux density, I = current and L = length of the conductor within the magnetic field.

Fleming's left hand rule: If we stretch the first finger, second finger and thumb of our left hand to be perpendicular to each other AND direction of magnetic field is represented by the first finger, direction of the current is represented by second finger then the thumb represents the direction of the force experienced by the current carrying conductor.



Solution

Start with creating the L293 Driver which initializes the Motor.

```
///Begin ADCSRA
        ADCSRA =
               (1 << ADEN) | // ADC Enable
               (0 << ADSC) | // ADC Start Conversion
               (1 << ADATE)| // ADC Auto Trigger Enable
               (1 << ADIE) | // ADC Interrupt Enable
```

```
		// division factor set to 32
		(1 << ADPS2)|
		(0 << ADPS1)|
		(1 << ADPS0);
		///End ADCSRA

	///Begin ADMUX
	ADMUX =
		// AREF, Internal Vref turned off.
		(0 << REFS0) |
		(0 << REFS1) |

		// Set right adjusting.
		(0 << ADLAR) |

		// Get input from ADC4.
		(0 << MUX4)  |
		(0 << MUX3)  |
		(1 << MUX2)  |
		(0 << MUX1)  |
		(0 << MUX0)  ;
		///End ADMUX

	///Begin SFIOR
	SFIOR =
		// Set timer0 overflow interrupt.
		(1 << ADTS2) |
		(0 << ADTS1) |
		(0 << ADTS0) ;
	///End SFIOR
```

With L293 driver we can set Power

```
void L293_setPow(unsigned int power)
{
	PWM_set(power);
}
```

Set direction to clockwise and counterclockwise

```
void L293_clockwise()
{
	SET_1(PORTD, IN1);
	SET_0(PORTD, IN2);
}

void L293_antiClockwise()
{
	SET_0(PORTD, IN1);
	SET_1(PORTD, IN2);
}
```

Stop the motor and let it move free without any force

```
void L293_stop()
{
	SET_1(PORTD, IN1, IN2);
}

void L293_free()
{
	SET_0(PORTD, IN1, IN2);
}
```

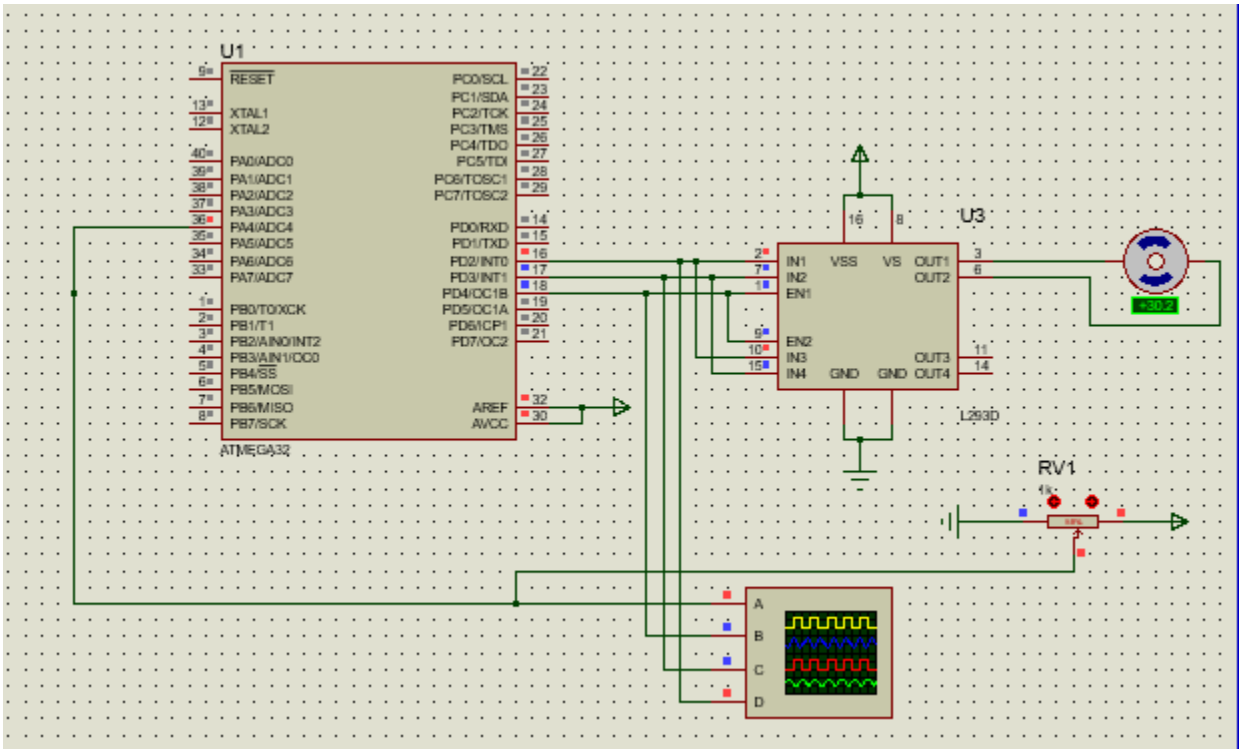
Using PWN we can set frequency in OCR1A register.

```
void PWM_set (unsigned int val)
{
	if (val > PWM_IT_TOP)
		val = PWM_IT_TOP;

	OCR1B = val;
}
```

In1	In2	meaning
0	0	Free
0	1	Clockwise
1	0	Anti-clockwise
1	1	Stop

Above is represented how the signals works and its meaning.

This is how the simulation in Proteus looks like:



Conclusion:

In this laboratory work I learned how to control a motor using PWN signal and how to make it move in two different directions using L293 driver. I got the chance to read more about motors, especially DC motors.

Appendix:

```
#include "ADC.h"
#include "utils.h"

void ADC_init()
{
	///Begin ADCSRA
	ADCSRA =
		(1 << ADEN) | // ADC Enable
		(0 << ADSC) | // ADC Start Conversion
		(1 << ADATE)| // ADC Auto Trigger Enable
		(1 << ADIE) | // ADC Interrupt Enable

		// division factor set to 32
		(1 << ADPS2)|
		(0 << ADPS1)|
		(1 << ADPS0);
		///End ADCSRA

	///Begin ADMUX
	ADMUX =
		// AREF, Internal Vref turned off.
		(0 << REFS0) |
		(0 << REFS1) |

		// Set right adjusting.
		(0 << ADLAR) |

		// Get input from ADC4.
		(0 << MUX4)  |
		(0 << MUX3)  |
		(1 << MUX2)  |
		(0 << MUX1)  |
		(0 << MUX0)  ;
		///End ADMUX

	///Begin SFIOR
	SFIOR =
		// Set timer0 overflow interrupt.
		(1 << ADTS2) |
		(0 << ADTS1) |
		(0 << ADTS0) ;
	///End SFIOR
}
```

```
#ifndef ADC_H_
#define ADC_H_

#include <stdint.h>
#include <avr/io.h>
#include <stdio.h>

#include "utils.h"

void ADC_init();
```

```
#endif /* ADC_H_ */
```

```
#include <avr/io.h>
#include "L293.h"
#include "pwm.h"
#include "utils.h"

#define IN1 2
#define IN2 3

void L293_init()
{
	PWM_init();

	SET_1(DDRD, IN1, IN2);

	L293_stop();
}

void L293_setPow(unsigned int power)
{
	PWM_set(power);
}

void L293_clockwise()
{
	SET_1(PORTD, IN1);
	SET_0(PORTD, IN2);
}

void L293_antiClockwise()
{
	SET_0(PORTD, IN1);
	SET_1(PORTD, IN2);
}

void L293_stop()
{
	SET_1(PORTD, IN1, IN2);
}

void L293_free()
{
	SET_0(PORTD, IN1, IN2);
}
```

```
#ifndef L293_H_
#define L293_H_


void L293_init();
void L293_setPow(unsigned int power);
void L293_antiClockwise();
void L293_clockwise();
void L293_stop();
```

```
void L293_free();


#endif /* L293_H_ */



#ifndef PWM_H_
#define PWM_H_

#define PWM_IT_TOP 1023

void PWM_init();
void PWM_set(unsigned int val);

#endif /* PWM_H_ */

/*
 * pwm.c
 */

#include <avr/io.h>
#include "utils.h"
#include "pwm.h"

void PWM_init()
{
	// We'll use pin OC1B
	// Setting timer 1.

	// Non-inverting mode
	SET_1(TCCR1A, COM1B1);
	SET_0(TCCR1A, COM1B0);

	// Fast PWM, 10-bit
	SET_0(TCCR1B, WGM13);
	SET_1(TCCR1B, WGM12);
	SET_1(TCCR1A, WGM11);
	SET_1(TCCR1A, WGM10);

	// Prescaling 64
	// it doesn't really matter but 64 is good
	// because it's easier to see on the oscilloscope
	// and the motor seems to work better
	SET_0(TCCR1B, CS12);
	SET_1(TCCR1B, CS11);
	SET_1(TCCR1B, CS10);

	// Set OC1A pin as output
	SET_1(DDRD, 4);
}

void PWM_set (unsigned int val)
{
	if (val > PWM_IT_TOP)
		val = PWM_IT_TOP;

	OCR1B = val;
}
```

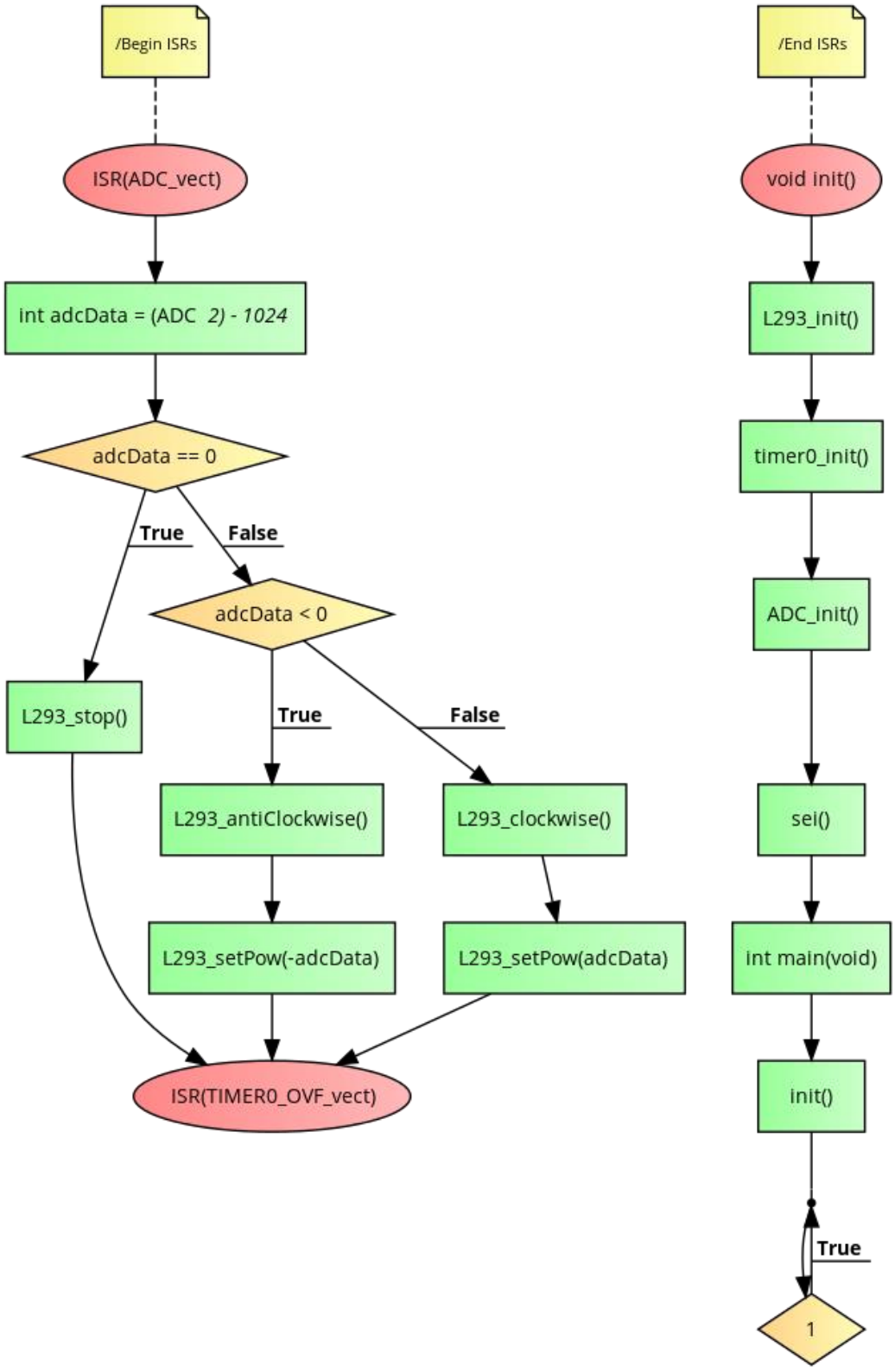
```
#include <avr/io.h>
#include "timer0.h"
#include "utils.h"

///Begin timer.init
void timer0_init()
{
	// Set timer0 with 1024 prescaler.
	TCCR0 =
		(1 << CS02) |
		(0 << CS01) |
		(1 << CS00) ;

	// Enable overflow interrupt.
	SET_1(TIMSK, TOIE0);

	// Set counter to 0.
	TCNT0 = 0;
}
///End timer.init
```

```
#ifndef TIMER_H_
#define TIMER_H_

void timer0_init();



#endif /* TIMER_H_ */
```

```
#include "utils.h"

void strcpylen( char * to, char * from, uchar len ) {
	while ( len-- ) {
		to[ len ] = from[ len ];
	}
}

void nops( int num ) {
	while ( num-- )
		asm( "nop" );
}
```

```
#include "utils.h"

void strcpylen( char * to, char * from, uchar len ) {
	while ( len-- ) {
		to[ len ] = from[ len ];
	}
}

void nops( int num ) {
	while ( num-- )
		asm( "nop" );
}

float getKelvinFromCelsius(float celsius)
```

```
{
	return celsius + 273.15f;
}

float getFahrenheitFromCelsius(float celsius)
{
	return celsius * 1.8 + 32.0;
}

#define F_CPU 8000000ul
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "L293.h"
#include "utils.h"
#include "ADC.h"
#include "timer0.h"

unsigned int count = 0;
unsigned int printOnEvery = 15;

///Begin ISRs
ISR(ADC_vect)
{
	int adcData = (ADC * 2) - 1024;


	if (adcData == 0) {
		L293_stop();
	}
	else if (adcData < 0) {
		L293_antiClockwise();
		L293_setPow(-adcData);
	}
	else {
		L293_clockwise();
		L293_setPow(adcData);
	}
}


ISR(TIMER0_OVF_vect)
{}
///End ISRs

void init()
{
	L293_init();
	timer0_init();
	ADC_init();


	sei();
}

int main(void)
{
	init();

    while(1);}
```


/Begin ISRs
ISR(ADC_vect)
int adcData = (ADC 2) - 1024
adcData == 0
True
False
adcData < 0
L293_stop()
True
False
L293_antiClockwise()
L293_clockwise()
L293_setPow(-adcData)
L293_setPow(adcData)
ISR(TIMER0_OVF_vect)
/End ISRs
void init()
L293_init()
timer0_init()
ADC_init()
sei()
int main(void)
init()
True
1