UNIVERSIDAD
PANAMERICANA

Proyecto Final: Odómetro, Taquímetro y Tacómetro.

Alumno:
Andre Nicasio Romo

Carrera: Ingeniería Mecatrónica

Materia: Microcontroladores II

Profesor: María Teresa Orvañanos Guerrero

Fecha de entrega: 4 de Junio del 2024
Universidad Panamericana de México, Ags.

Índice:

Objetivo del Proyecto:

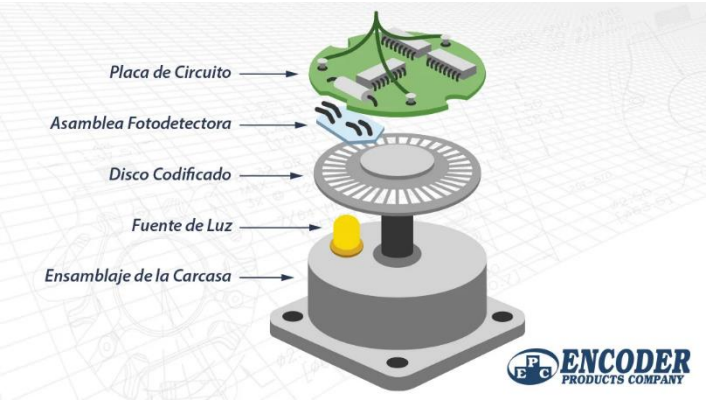
El objetivo principal de este proyecto es que nosotros como alumnos seamos capaces de implementar una solución para algún problema y/o simplemente innovar para tener un mejor registro de ciertas magnitudes físicas que presenta una máquina para ver su desempeño, en mi caso, fue el VeHimo, donde se pretendía medir el rendimiento del vehículo, donde para la implementación del dispositivo se me pidió que la mediciones fueran realizadas mediante un encoder rotativo y una pantalla oled, la cual ocupa un protocolo de comunicación distinto a los vistos en clase (I2C).

Planificación:

Antes de empezar como loco de lleno a programar, se tuvo que realizar una investigación previa acerca de como funciona un encoder rotativo y la pantalla oled, además del protocolo i2c.

Encoder Rotativo:

Navegando en la red se encontró que un encoder rotativo es un sensor (o dispositivo) capaz de convertir el movimiento rotativo en señales eléctricas, de esta clase de sensor existen 2 tipos de encoder, los encoder incrementales y los absolutos. Los Encoder incrementales generan una señal cada vez que este rota, mientras que los encoders absolutos generan multibit que nos indican la posición exacta de la rotación en cuestión.



Después de haber comprendido la diferencia de ambos encoder, se optó por elegir el encoder incremental, ya que en sí, no nos interesa tanto la posición exacta de la rueda del vehículo, sino, cuantas veces gira la rueda en un determinado tiempo (recordemos que v = d/t), por lo que el encoder rotativo es el más adecuado.

Cada encoder tiene distinta cantidad de pulsos por rotación, por lo que hay que tomar en cuenta que entre más ppr tenga un encoder, más preciso será.

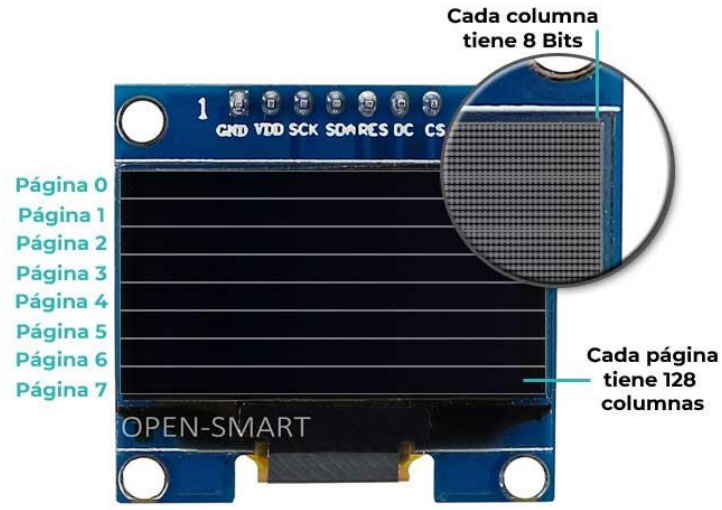
Encoder Products Company. (n.d.). ¿Qué es un Encoder? Encoder.com. Retrieved June 3, 2024, from https://www.encoder.com/article-que-es-un-encoder

Pantalla Oled:

La definición corta de una pantalla Oled seria:

"Se trata de pantallas que utilizan diodos orgánicos de emisor de luz capaces de consumir muy poca energía, son muy delgadas, se comunican por I2C o SPI y producen una imagen más brillante y nítida que una pantalla LCD." ***del Valle Hernández, L. (2020, February 18). SSD1306 pantalla OLED con Arduino y ESP8266 I2C. Programarfacil Arduino y Home Assistant. https://programarfacil.com/blog/arduino-blog/ssd1306-pantalla-oled-con-arduino/***

Normalmente, este tipo de pantallas suelen ser pequeñas, pero su desempeño se podría decir que es superior al de una pantalla LCD, si esta es superior, también es un tanto mas compleja, ya que utiliza los protocolos i2c y spi



Este tipo de pantallas, como se dijo anteriormente, cuenta con cierta cantidad de diodos (leds) para su funcionamiento, normalmente las pantallas suelen medir 0.96" con 128x64, por lo que cada píxel representara un bit.

Esto nos quiere decir que para mostrar un carácter se le tiene que enviar un conjunto de bytes (en las columnas de 8 bits) a lo largo de la página para poder dibujar el carácter deseado.

del Valle Hernández, L. (2020, February 18). SSD1306 pantalla OLED con Arduino y ESP8266 I2C. Programarfacil Arduino y Home Assistant. https://programarfacil.com/blog/arduino-blog/ssd1306-pantalla-oled-con-arduino/

La Pantalla Oled cuenta con un controlador SSD1306, el cual es el driver para que nosotros podamos mandar datos e imprimirlos. Se utilizo la siguiente Librería para el uso de la pantalla Oled.

Protocolo i2c:

El protocolo bien conocido como i2c es un protocolo que solo requiere de dos cables para realizar la transmisión y la recepción de datos de uno o más esclavos, mediante los pines SDA (serial data) y SCL (serial clock). Los datos enviados mediante este protocolo pueden ser transmitidos a 100kbps (el mas lento) a 400kbps (el más rápido).

La comunicación entre esclavos es posible debido a que cada uno de ellos cuenta con una única dirección de esclavo, por lo que le basta al maestro enviar mediante el bus la dirección del esclavo con el que quiera comunicarse.

Usualmente existen 4 modos en este protocolo de comunicación:

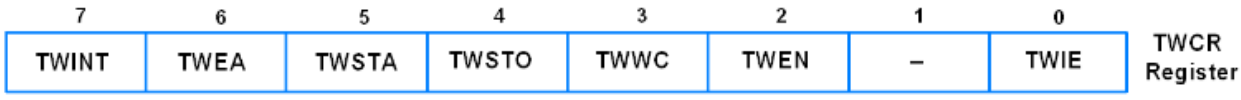
- Master transmitter: cuando el maestro envia datos.
- Master Receiver : cuando el maestro recibe datos.
- Slave transmiter: Cuando el esclavo envia datos.
- Slave Receiver: cuando el esclavo recibe datos.

Para que la comunicación entre dispositivos pueda ser llevada a cabo se requieren de ciertos registros, pero viéndolo de una forma sencilla, se sigue un cierto patrón de comunicación:

1. Una condición de inicio.
2. Enviar y/o seleccionar la dirección del esclavo con el cual se quiere establecer la comunicación.
3. Enviar/recibir el(los) dato(s).
4. Una condición de parada.

Para este tipo de comunicación, como anteriormente se mencionó, se requieren varios registros a ser configurados, pero los principales son 4:

- TWBR(bit Rate Register) para configurar la velocidad de transferencia de datos, con el TWPS bit en el registro TWSR (TWBR tiene valores de 0 a 255).
- TWCR (Control Register) par controlar tanto la condición de salida (TWSTA), la de parada(TWSTO), ack, enable (TWEN), interrupt (TWINT).



Bit 7- TWINT: TWI Interruption, genera una interrupción cuando completa alguno de los eventor principales (Condición de inicio, de Parada, Transmiscion, recepción, etc.)

Bit 6 -TWEA: enable acknowledgment bit (no nos interesa ahorita)

Bit 5 – TWSTA: TWI START condition bit: al tener un 1 genera la condición de inicio y el maestro o esclavo toman control del bus.

Bit 4 – TWSTO: TWI STOP condition bit: Genera la condicion de parada y el maestro y/o esclavo deja de tener el control sobre el bus.

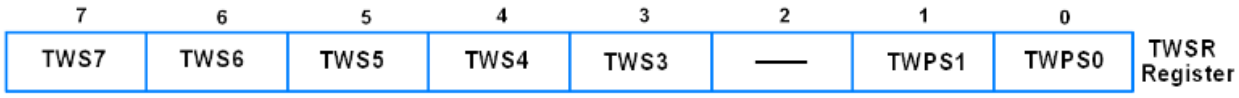
Bit 3 – TWWC: TWI write collision

Bit 2 – TWEN: TWI enable bit: Habilita la interfaz i2c, tomando control de los pines I/O

Bit 1 - Reserved

Bit 0 – TWIE: TWI interrupt enable: Habilita la interrupción.

- TWSR (Status register) Para ver el estado de la transmisión.



Bit 7:Bit 3 - TWS7: TWS3: TWI status bits (No nos interesan).

Bit 1:0 - TWPS1:TWPS0: TWI pre-scaler bits (Para la formula que viene mas adelante.

TWPS1	TWPS0	Exponent	Pre-scaler value
0	0	0	1
0	1	1	4
1	0	2	16
1	1	3	64

- TWDR (Data Register) registro donde se almacena el dato enviado y/o el dato recibido.

I2C in AVR ATmega16/ATmega32. (n.d.). Electronicwings.com. Retrieved June 4, 2024, from https://www.electronicwings.com/avr-atmega/atmega1632-i2c

Para Configurar la velocidad con la que trabajara el SCL (serial clock) se tiene la sig formula:

$$f_{SCL} = \frac{f_{cpu}}{16+2.TWBR.4^{TWPS}}$$

Donde TWBR y TWPS(en TWSR) son registros configurables.

La Configuración más común para el SCL suele ser:

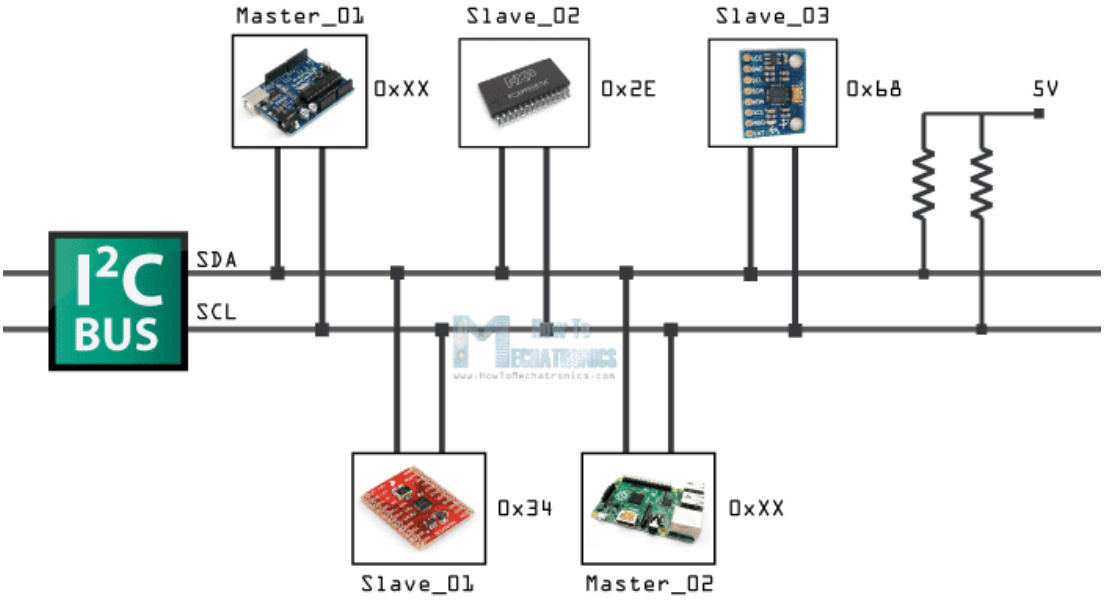
Velocidad	Fcpu	TWBR	TWPS
100kbps	8Mhz	32 (0b00100000)	0
400kbps	8Mhz	2 (0b00000010)	0

Pugalia, A. K. (n.d.). ATmega16 I2C programming. Sysplay.In. Retrieved June 4, 2024, from https://sysplay.in/blog/tag/atmega16-i2c-programming/

Sabiendo Esto se puede empezar a hacer una librería con lo visto, para poder establecer de una forma más rápida este protocolo.

Por cuestiones de tiempo (debido al Proyecto VeHimo) no se realizó una librería. Se encontró una librería en la red.

(link de la librería).



(Ejemplo conexión master-slave en i2c)

Programación:

Conociendo bien como es funcionan los dispositivos a usar, se comenzó a ver de que forma seria más optima la programación para el velocímetro.

Primero se buscó como se van a medir las magnitudes, la solución mas viable fue el tomar una rueda de patinaje de 100mm la cual ira conectada directamente al encoder, esta rueda ira presionada y girara por fricción directamente con la rueda del vehimo, por lo que partiremos de esa rueda de 100mm para los cálculos de Velocidad, Rpm y distancia.

Para la programación, se usó simplemente la interrupción 1 (la cual va directamente conectada al encoder), la idea es que el programa tome muestras de la cantidad de pulsos que manda el encoder cada segundo (entre varias pruebas esta fue la mejor Opción), por lo que partiendo de ahí se escribió el siguiente código:

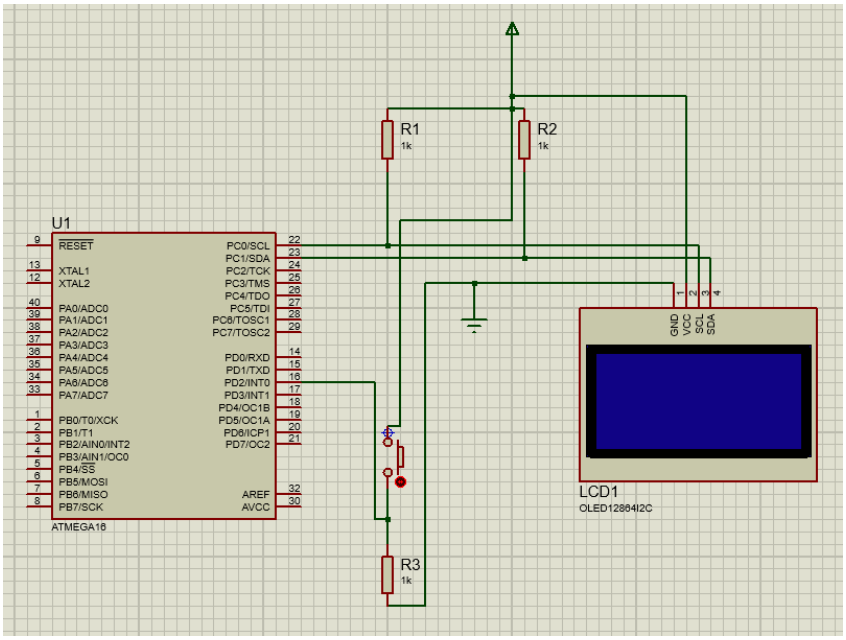
```
/*
 * OledTestEncoder.c
 *
 * Created: 28/05/2024 12:14:32 a. m.
 * Author : Dell
 */

#define  F_CPU 8000000UL
#include <avr/io.h>                     //Librería general del Microcontrolador
#include <stdint.h>                     //Para poder declarar varialbes especiales
#include <util/delay.h>                 //Para poder hacer delays
#include <avr/pgmspace.h>
//#include "lcd.h"
#include <avr/interrupt.h>
#include "SSD1306.h"
#include "i2c.h"

long cnt = 0;
//long timem = 0;
long speed = 0;
//int auxspeed = 0;
long rpm =0;
double kmh = 0;
long rad = 0;
float kms = 0;


uint8_t cero_en_bit(volatile uint8_t *LUGAR, uint8_t BIT)
{
	return (!(*LUGAR&(1<<BIT)));
}

uint8_t uno_en_bit(volatile uint8_t *LUGAR, uint8_t BIT)
{
	return (*LUGAR&(1<<BIT));
}

void Traba(volatile uint8_t *Lugar, uint8_t Bit){
	_delay_ms(10);
	_delay_ms(10);
```

```
	while(cero_en_bit(&*Lugar, Bit)){}
	_delay_ms(10);
	_delay_ms(10);
}

int main(void) {
	sei();

	DDRD|=0;
	PORTD=0;

	//Configuracion Int0
	GIFR|=(7<<5); //todas las interrupciones
	MCUCR|=(1<<1); //flanco de bajada
	GICR|=(1<<6); //para usar la interrucpion 0

	/*
	//Configuracion Timer
	TCNT0 = 0;
	TIFR = 3;
	TIMSK = 2; // sin overfow
	TCCR0 = 0b00001101; //prescaler 1024 modo cnt
	TCCR0-=5;
	OCR0 = 243;*/


	OLED_Init();  //initialize the OLED
	OLED_Clear(); //clear the display (for good measure)
	OLED_EnableInversion();
	OLED_EnableInversion();
	OLED_SetCursor(0,0);
	OLED_Printf("Velocidad:");


	while (1) {
		/*
		TCNT0 = 0;
		TCCR0 +=5;
		while(timem<4){}
		TCCR0 -=5;
		timem = 0;*/
		cnt = 0;
		sei();// se active la interrupción.
		for(int i = 0; i <= 100; i++) _delay_ms(10); // Se entretiene 1 segundo sin
hacer nada
		cli(); //se desactiva la interrupción.
		rpm = (cnt*60)/20; //formula para sacar el # de rppm
		rad = (rpm*3.1416)/30; //Conversion a Rad/s
		kmh = (rad*0.00005)*3600; //Conversion a km/h
		speed = (int)(kmh*10); //Agregamos un decimal y casteamos a un int
		kms += (rad*0.05);// para la medición de la distancia.
		cnt = 0;

		OLED_SetCursor(0,0);
		OLED_GoToNextLine();// pasa a la sigiente fila del oled
		OLED_DisplayChar((speed/1000)+48);
		OLED_DisplayChar(((speed/100)%10)+48);
```

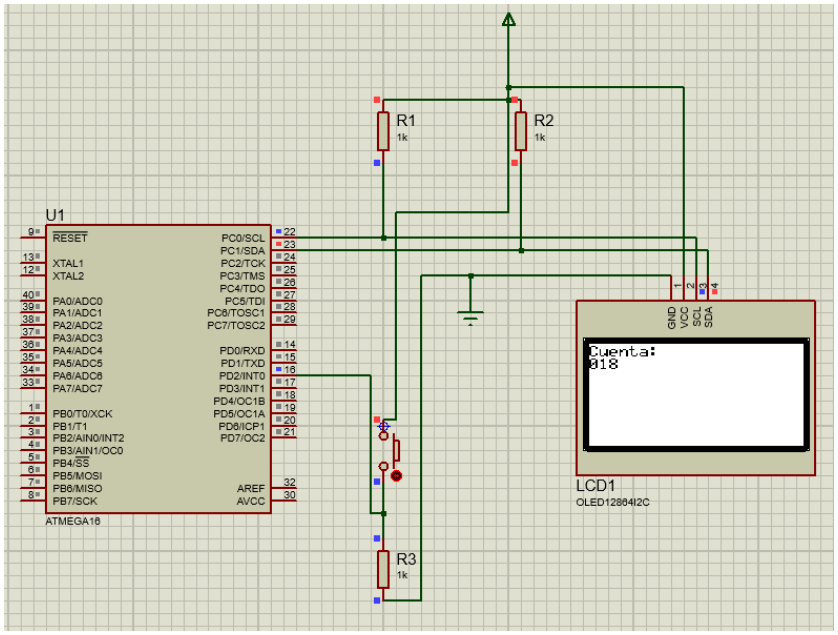
```
                OLED_DisplayChar(((speed/10)%10)+48);
                OLED_DisplayChar('.');
                OLED_DisplayChar((speed%10)+48);
                OLED_Printf(" km/h");
                OLED_GoToNextLine();
                OLED_GoToNextLine();
                OLED_Printf("Rpm:");
                OLED_GoToNextLine();

                OLED_DisplayNumber(E_DECIMAL, (int)rpm, 4); //imprime un numero entero.

                OLED_GoToNextLine();
                OLED_GoToNextLine();
                OLED_Printf("Distancia:");
                OLED_GoToNextLine();
                OLED_DisplayNumber(E_DECIMAL, (int)kms, 5);
                OLED_Printf(" Mts");


        }

        return 0; // never reached
}

ISR (INT0_vect)
{
        cnt+=1; //en la interrupcion simplemente aumenta el Contador.
        //Traba(&PIND,2);

}

/*
ISR (TIMER0_COMP_vect)
{
        timem +=1;

}
*/
```

*El código presenta varias partes del código omitidas, debido a que se realizaron diversas pruebas para que nos diera una medición exacta de la cantidad de pulsos (véase en el apartado de prueba y errores)

Conexiones y simulación:



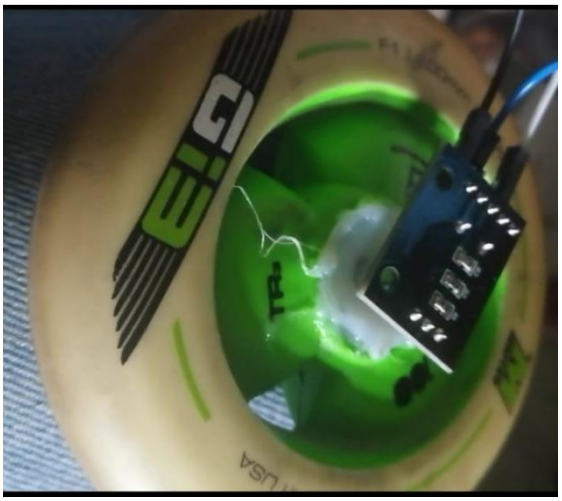
Debido a que no encontré un encoder en proteus 8 Utilice como prueba un botón, para verificar que si llegara la INT0, además este proteus tiene un código distinto, ya que fue el de prueba para ver si el encoder al momento de girar mostraba correctamente el número de pulsos enviados.



(Codigo de prueba)

Implementación:

Después de haberse serciorado de que el encoder envio los pulsos de manera correcta se procedió a implementar el prototipo. A la hora de la implementación, se tuvo que soldar y triangular una placa al chasis, el cual seria el soporte del encoder y la rueda.



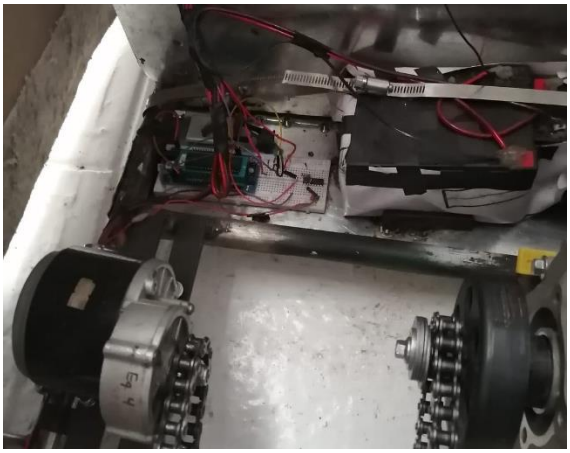
Rueda con encoder

Triangulacion con el encoder

(se perdió la rueda en la carrera)

Pantalla inicial imágenes de prueba

Implementacion volante



Implementacion de la circuiteria

Race Day

Resultados:

Los resultados obtenidos fueron satisfactorios, ya que una de las primeras pruebas que se hizo fue manualmente, ya que al usar una rueda de 100mm, girarla 3 veces seria aproximadamente 1m. y jalo de forma satisfactoria, otra de las pruebas que se hizo fue con el motor eléctrico, que al usar una relación de 2:1 en los sprockets, teóricamente hablando (y sin carga en el vehículo) tiene una Velocidad máxima de 10km/h, y las mediciones presentaron resultados de alrededor 11 y 9 km/h

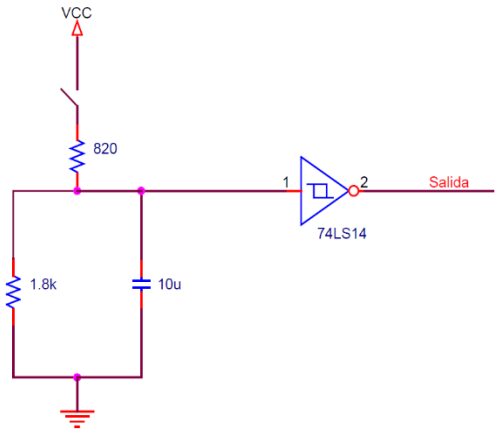
Mientras que en las pruebas con el motor de combustión, (prueba sin carga), presento una velocidad de alrededor 58km/h, el cual se acerca bastante a los cálculos realizados para la transmisión que fuero de alrededor de 60km/h.

Lamentablemente el día de la carrera se tuvo algunas complicaciones, donde la rueda que iba directamente al encoder se salió.

Pruebas y análisis:

No todo fue color de rosas al momento de hacer el proyecto, Uno de los primeros problemas que tuve fue básicamente las mediciones de tiempo, ya que en un principio quería usar la INT0 y el Timer0, Para medir el tiempo que tardaba en completar una vuelta, esta forma de programarlo no me sirvió, ya que se tenia que medir el timer precisamente en ms para una medición clara de la velocidad, pero lo que pasaba era que no media bien el tiempo o la cantidad de pulsos, esto se debe a que el micro se volvía loco sobre a cual interrupción debía atender, es por ello que se recurrió a la programación anteriormente mencionada, la cual, se hace menos 1 segundo, y durante ese segundo toma cuantas veces entra a la interrupción.

Otro de los problemas que tuve fue el famoso fenómeno del rebote, el cual, no podía ser corregido mediante un antirrebote mediante software, ya que como hemos visto el antirrebote, consume alrededor de 100ms, y en mi caso, las mediciones de los pulsos ocurren en cuestión de milisegundos, es por ello que, para poder resolver ese problema, recurrí a un circuito anti-rebote, donde fue necesaria la intervención del integrado 7414 (Schmith trigger).



Básicamente este Circuito nos ayuda a eliminar los rebotes indeseados del encoder, dándonos así mediciones más exactas.

Conclusión:

Este proyecto me ha ayudado a saber investigar bien, ya que en el mundo laboral existen una gran diversidad de dispositivos y sensores que no sabemos como funcionan, pero sabiendo leer e investigando de forma adecuada podemos echarlos a andar.

Otra de las enseñanzas que me da este proyecto, fue ver desde distintos ángulos un mismo problema, ya que hay una gran variedad de métodos para resolver un problema, ya que, si uno te puede dar mejores resultados que otro método, y solamente uno sabe cual conviene mas mediante prueba y error.

Además de que los sensores mecánicos siempre van a tener fallas con los rebotes, por lo que debemos ser conscientes de ello y saberlo afrontar según se requiera, ya que si el sensor usado no se va a usar con alta frecuencia, sino, mas como de control, se puede implementar un antirrebote mediante software, mientras que si se va usar con alta frecuencia (como el encoder) se puede implementar un antirrebote mediante hardware.