MODULO DE ADQUISICION DE CONSTANTES PARA MODELAMIENTO MATEMATICO DE IN MOTOR DC

PRESENTADO POR

CRISTHIAN ALEXANDER CARDONA ARIAS

TECNOLOGIA EN ELECTRONICA

UNIVERSIDAD MINUTO DE DIOS

SOACHA

2011

1. Modulo de adquisición de constantes para modelamiento matemático de un motor DC

“Marco teórico”

El modulo de adquisición de constantes para modelamiento matemático de un motor DC consiste en un dispositivo capaz de excitar y hacer lectura de los valores que puede llegar a tomar un motor DC y enviar estos datos al software MATLAB el cual puede procesarlos y entregar al usuario un modelo matemático de tal motor.

Ya teniendo este modelo matemático el usuario puede hacer un uso más apropiado de su motor DC puede hacer un control más apropiado de este y aumentar su eficiencia acoplando módulos de control PID por ejemplo.

El modulo de adquisición de datos está compuesto por varias partes necesarias para el conjunto, está el micro controlador encargado de la excitación del motor, este se encarga simplemente de enviar pulsos de amplitud aleatoria al motor.

Esta el micro controlador principal que es el encargado de hacer la lectura de las variables, el control de las variables y la comunicación con el PC es la parte del modulo encargada del control en él se procesan todas las lecturas del motor y la lectura de la excitación que se le ha hecho al mismo y se envían vía USB al PC para el procesamiento de estas.

Para la lectura de la velocidad que puede llegar a tomar el motor se utiliza un modulo de conversión frecuencia-voltaje acoplando mecánicamente el motor a un encoder del cual se recibe un serie de pulsos que varían su frecuencia dependiendo de la velocidad del motor, y convirtiendo esta frecuencia en una tensión que pueda interpretar nuestro micro controlador principal.

Es así como ya teniendo estos datos en PC y por medio de algoritmo hecho en MATLAB tenemos tanto los valores de excitación como los valores de respuesta en una grafica que pueda interpretar el usuario.

Ya teniendo todos estos datos en la GUI de MATLAB la cual es como la ventana principal de trabajo en MATLAB en la cual se evidencian la mayoría de los procesos en el programa, podemos disponer de esta matriz de datos para hacer un análisis más adecuado y obtener el modelo matemático que buscamos.

Para hacer esto utilizamos una función de MATLAB llamada TOOLBOX que es la que hace el análisis de los datos de salida con respecto a los de entrada y así nos entrega el modelo matemático que se está buscando.

1.1 ¿Por qué el modulo de adquisición?

Este proyecto se inicio con el fin de darle al estudiante una herramienta para familiarizarse más con el modelamiento matemático.

Ya que es de notar que es un tema algo complicado de entender y siempre se maneja en su gran mayoría por la parte teórica y prácticamente nada en la parte practica, con este proyecto el estudiante puede ratificar lo que está haciendo en el papel con el software MATLAB y un motor de similares condiciones a las del cálculo escrito.

También está el caso de otro experimento en el que se necesite saber qué tipo de control es el apropiado para determinado motor, por ejemplo un robot seguidor de línea se necesita que el robot responda a determinada velocidad cuando se pase de un área obscura a un área clara, además de esto se necesita que el robot varíe su velocidad proporcionalmente a la intensidad de la luz que se recibe de la superficie y que cuando la superficie se a muy obscura estabilice rápidamente su velocidad máxima.

Para solucionar dicho problema muchos estudiantes que no tienen herramientas adecuadas se valen del muy utilizado método de prueba y error, el cual puede llegar a ser efectivo pero quita mucho tiempo al estudiante y no garantiza que los resultados tengan el grado de exactitud que se esperaba.

Pero si se tiene una herramienta como el modulo de adquisición de variables para el modelamiento matemático de un motor DC simplemente se acopla el motor a esta, se hace dicho análisis y ya teniendo el modelo matemático del motor solo faltaría hacer un análisis con variables como el peso del robot o la fricción de las ruedas del robot contra la superficie, y hacer un control PID que le dé al robot la inteligencia para variar proporcionalmente dependiendo de la superficie sobre la que este y la velocidad para estabilizarse y de responder rápidamente a cambios en el color de la superficie sobre la que esta.

2. Objetivo general

2.1 Diseñar una interfaz de adquisición de datos para el modelamiento y obtención de variables de motores D.C.

3. Objetivos específicos

3.1 Diseñar una interfaz amigable para el usuario, fácil de usar para el estudiante.

3.2 Realizar el modulo de adquisición de variables de tal manera que permita al usuario visualizar el proceso en tiempo real.

4. Justificación

4.1 Este proyecto es realizado con la intención de ayudar al estudiante en el tema de modelamiento matemático que es un tema muy importante para la industria de nuestros días y es tan difícil de comprender con tan solo un lápiz y un papel.

5. Alternativas de solución

A continuación se presentara de forma más detallada la forma en cómo se desarrollo el proyecto y cada una de sus partes.

5.1. El generador de pulsos

El generador de pulsos es la parte del modulo en donde se inicia todo el proceso, con este damos los pulsos de excitación al motor necesarios para iniciar el proceso de análisis.

Para hacer este generador de pulsos se opto por trabajar con un PIC12F629 es un PIC de gama baja y de tan solo 8 pines que se adecua a nuestra necesidad.

Se decidió hacer este generador en otro micro controlador y no en el principal para quitar carga al mismo ya que este tiene funciones con mucha más carga grafica y podría afectar las lecturas.

La generación de pulsos se hace por medio de un algoritmo desarrollado en el software CCS el cual describimos a continuación.

```
//#include
"C:\Documents_and_Settings\cristian\Mis_documentos\prjectofinal\generador2\generador2.h"

#include <12F629.h>

#define RAND_MAX 30  // solamente 30 números

#include <STDLIB.H>  //libreria donde está la función rand();

#fuses NOWDT         //no watch dog timer

#fuses INTRC_IO      //oscilador rc interno

#fuses NOCPD

#fuses NOPROTECT     //codigo sin proteccion

#fuses NOMCLR        //no master clear

#fuses PUT
```

```
#fuses BROWNOUT
#use delay (clock=4000000) //reloj a 4 Mhz


void main()
{
char num=0;       //variable almacena numero aleatorio
char x=0;         //variable del proceso
srand(100);       //maximo hasta 100


while(true)
{
num=rand();       //genera numero pseudo-aleatorio
x=num*1000;       //multiplicamos el numero pseudo aleatorio por 1000
output_high(PIN_A1); //activamos el motor
delay_ms(x);         //tiempo aleatorio que dura el motor encendido
output_low(PIN_A1);  //desactivamos el motor
delay_ms(100);       //retardo para nuevo tiempo
}
}
```

Programación del PIC12f629 como generador de pulsos

De esta manera tenemos el generador de pulsos listo, pero no podemos conectar el motor directamente al micro controlador ya que este no soportaría la carga.

Para dar solución a esto se hace un driver para el motor construida con transistores TIP122 y dos resistencias de 1KΩ, de esta manera solucionamos tal problema y aseguramos que el pulso sea el mismo en las terminales del motor como en la salida del pin del micro controlador.

De esta manera nuestro circuito oscilador quedaría así:

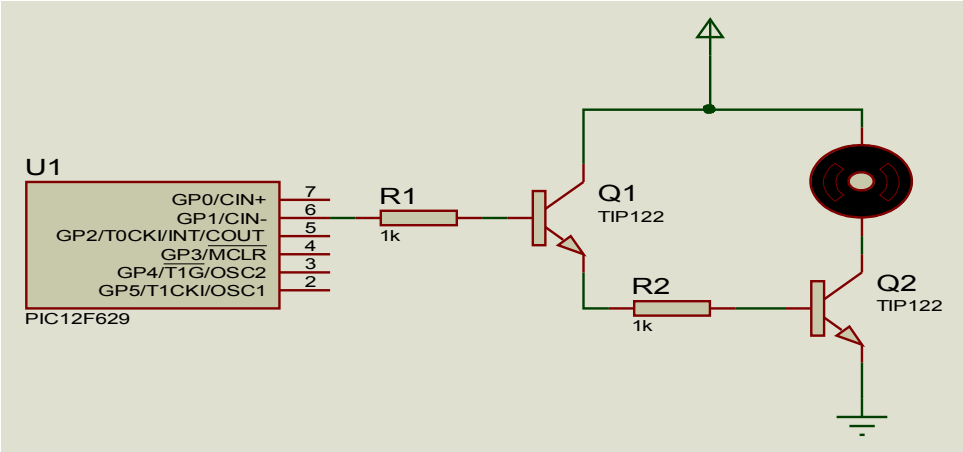

Figura 1 circuito de excitación

5.2. El micro controlador principal

El micro controlador principal es la parte del modulo que se encarga de procesar todas las señales y enviarlas al PC es muy importante ya que sin él no tendríamos las lecturas ni la comunicación USB.

Se utiliza un micro controlador PIC18F4550 es un PIC gama alta de 40 pines, se utiliza gracias a su capacidad de comunicación USB ya que esta comunicación es mucho más amigable y común para el usuario además la transferencia de datos es bastante robusta y por una comunicación serial probablemente colapsaría constantemente.

También se utilizo este micro controlador ya que posee 10 canales análogos de entrada, pues para el proyecto solo se necesitan 2, pero este reunía 2 características cruciales para el desarrollo de nuestro proyecto.

Las funciones del micro controlador principal se desarrollaron en un algoritmo hecho en el software CCS que explicamos a continuación:

```
#include <18F4550.h>                    //escogemos la libreria del pic 18f4550
#device ADC=10                          //usamos el ADC  de 10 BITS
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL1,CPUDIV1,VREGEN,MCLR,NOPBADEN

#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#BYTE ADCON0  = 0x0FC2                  // Registro de control del ADC

#BYTE ADCON1  = 0x0FC1                  // Registro de control del ADC

#define USB_HID_DEVICE     FALSE        //deshabilitamos el uso de las directivas HID
```

```
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK    //turn on EP1(EndPoint1) for IN
bulk/interrupt transfers
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK    //turn on EP1(EndPoint1) for OUT
bulk/interrupt transfers
#define USB_EP1_TX_SIZE    64                 //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE    64                 //size to allocate for the rx endpoint 1 buffer

#include <pic18_usb.h>       //Microchip PIC18Fxx5x Hardware layer for CCS's PIC USB driver
            #include <Descriptoreje1.h>        //descriptors del Pic USB
#include <usb.c>                  //handles usb setup tokens and get descriptor reports

#define  LEDV    PIN_B6
#define  LEDR    PIN_B7
#define  LED_ON  output_high
#define  LED_OFF output_low



#BYTE TRISA  = 0x0F92                      // Registro de control de E/S del puerto A
#BYTE TRISB  = 0x0F93                      // Registro de control de E/S del puerto B
#BYTE PORTA  = 0x0F80                       // Registro del puerto A
#BYTE PORTB  = 0x0F81                       // Registro del puerto B
#BYTE ADCON1  = 0x0FC1                       // Registro de control del ADC
#BYTE CMCON   = 0x0FB4                       // Registro del modulo comparador



int8 dato[64];
unsigned int16 entrada;
int8 Ciclo,offset;

void main(void) {

LED_ON(LEDV);                     //encendemos led en RB6 para indicar presencia de energía
LED_OFF(LEDR);

usb_init();                           // inicializamos el USB
usb_task();                            // habilita periferico usb e interrupciones
usb_wait_for_enumeration();      // esperamos hasta que el PicUSB sea configurado por el host

LED_OFF(LEDV);
LED_ON(LEDR);                    // encendemos led en RB7 al establecer contacto con la PC
TRISA  = 0x0FF;
TRISB  = 0x00;
ADCON1 = 0x0F;                   // Se configura al ADC para entradas digitales (apagar el ADC)
CMCON  = 0x07;                   // Se configuran los comparadores para entradas digitales
(apagar los comparadores)

setup_comparator(NC_NC_NC_NC);
setup_adc_ports(AN0);              // Se selecciona AN0 como entrada analógica y las demás
como digitales
setup_adc( VSS_VDD );            // Se indica el rango de voltaje que tendrá la entrada análoga
set_adc_channel( 0 );                // Indica de que pin se hará la conversión
setup_adc( ADC_CLOCK_DIV_64 );            // Indica la frecuencia que se usará el reloj del ADC
ADC_CLOCK_DIV_16
```

```
while (TRUE)
 {
          if(usb_enumerated())                    // si el Pic está configurado vía USB
         {
          if (usb_kbhit(1))                    // si el endpoint de salida contiene datos del host
         {
         usb_get_packet(1, dato, 64);        // cojemos el paquete de tamaño 8bytes del EP1 y
         almacenamos en dato
         portb = dato[0];                    // Se muestra al byte completo en el puerto B
         dato[1] = porta;                    // Se lee el puerto A y se almacena en el vector
         offset=0;
         set_adc_channel( 0 );          // escogemos el canal 0 para el primer canal de conversion
         for(Ciclo=0;Ciclo<=15;Ciclo++)
          {
           read_adc(ADC_START_ONLY);
           while (bit_test(ADCON0,1));
           entrada = read_adc(ADC_READ_ONLY);
           dato[2+offset]= make8(entrada,0);
           dato[3+offset] = make8(entrada,1);
           offset=offset+2;
         }

         set_adc_channel( 1 );      // escogemos el canal 1 para el segundo canal de conversion
         for(Ciclo=16;Ciclo<=28;Ciclo++)
          {
           read_adc(ADC_START_ONLY);
           while (bit_test(ADCON0,1));
           entrada = read_adc(ADC_READ_ONLY);
           dato[2+offset]= make8(entrada,0);
           dato[3+offset] = make8(entrada,1);
           offset=offset+2;
         }
```

```
        usb_put_packet(1, dato, 64, USB_DTS_TOGGLE);
         //y enviamos el mismo paquete de tamaño 64bytes del EP1 al PC
      }
     }
   }
}
```

Programación del micro controlador principal

Esta es la programación principal del micro controlador pero adjunta a esta se encuentra un archivo llamado descriptor.h con este configuramos la comunicación USB en el PC, este archivo nos permite numerar el dispositivo y finalmente configurar el puerto para la comunicación.

El circuito del micro controlador principal es muy simple ya que solo lo utilizamos para hacer las lecturas de señal y para la comunicación USB, además de algunas partes necesarias como el circuito oscilador con el cristal, el circuito de RESET y 2 LEDS que indican el estado de la comunicación con el PC.

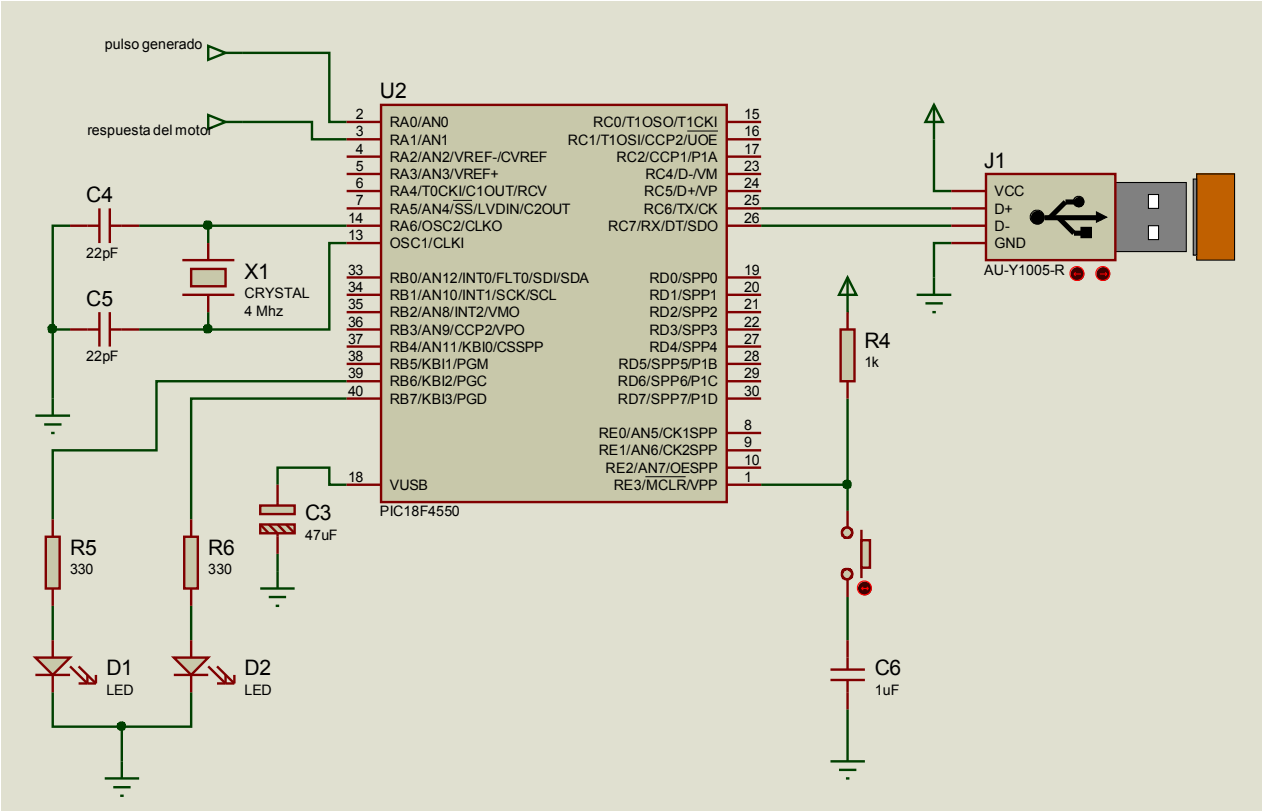

Figura 2 circuito del micro controlador principal

5.3. Driver del motor

El driver del motor es simplemente un circuito para aumentar la corriente en los terminales del motor, ya que si lo conectamos directamente al micro controlador la carga podría dañar el PIC además de que no sería suficiente corriente para el óptimo funcionamiento del motor.

Para dar solución a este problema se hace un circuito de 2 transistores NPN en cascada manteniendo la tensión en los terminales del motor y aumentando la corriente que alimenta al mismo.

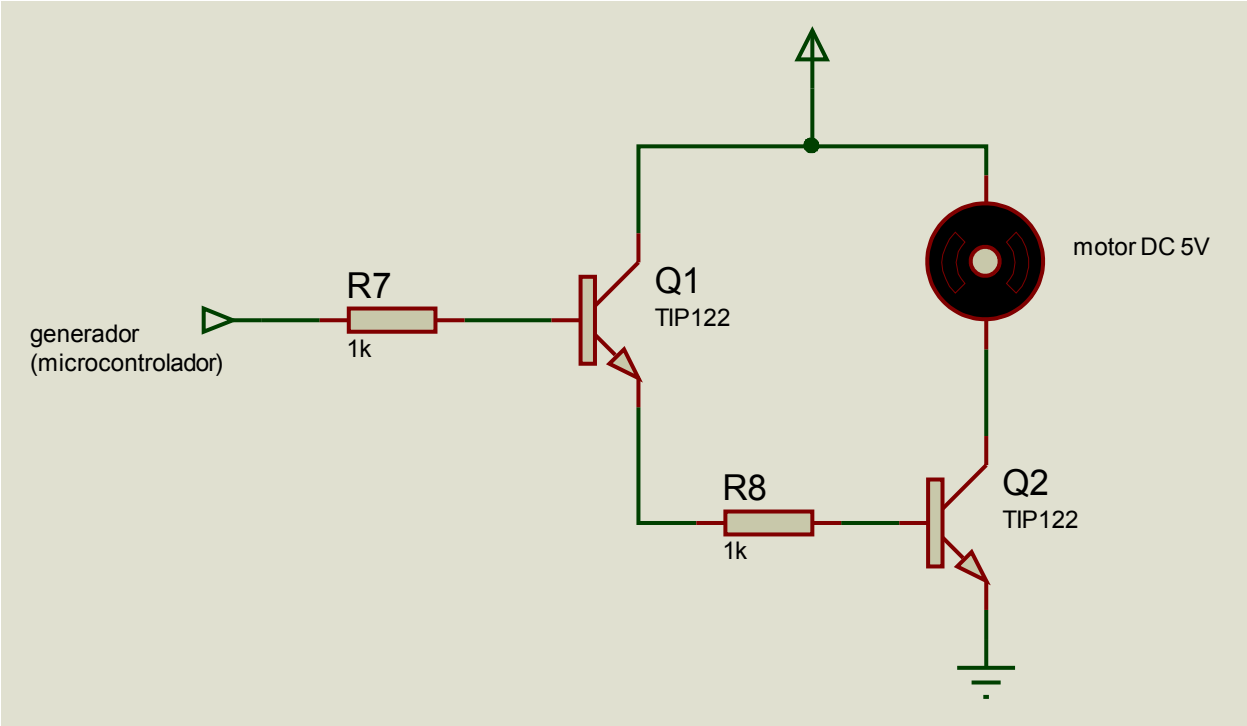

Figura 3 driver del motor

Para que el la velocidad del motor pueda ser convertida a frecuencia se necesita un encoder el cual dependiendo de la velocidad que tenga el motor entrega una serie de pulsos al conversor frecuencia-voltaje, a continuación se presenta una tabla comparativa entre la tensión de alimentación del motor y la frecuencia entregada por el encoder:

Tensión de alimentación	Frecuencia de salida
1.5v	57hz
1.6v	77hz
1.7v	97hz
1.8v	113hz
1.9v	124hz

2.0v	132hz
2.1v	143hz
2.2v	154hz
2.3v	165hz
2.4v	178hz
2.5v	194hz
2.6v	204hz
2.7v	214hz
2.8v	228hz
2.9v	237hz
3.0v	258hz
3.1v	264hz
3.2v	272hz
3.3v	287hz
3.4v	304hz
3.5v	316hz
3.6v	334hz
3.7v	350hz
3.8v	368hz
3.9v	380hz
4.0v	393hz
4.5v	450hz
5.0v	520hz

Tabla 1 tabla comparativa entre la tensión de alimentación del motor y la frecuencia de salida en el encoder.

5.4. Conversor frecuencia-voltaje

El conversor frecuencia-voltaje es la parte del modulo encargada de hacer una lectura a la respuesta del motor y convertirla a una tensión que pueda leer el conversor AD del micro controlador principal.

Este circuito está formado por un sensor de ranura que entrega una frecuencia de pulsos dependiendo de la velocidad que tome el encoder, el encoder es una rueda de 1.6 cm de diámetro con 65 ranuras en su borde exterior.

Esta rueda esta acoplada mecánicamente al eje de nuestro motor variando la frecuencia de bloqueo en que la radiación infrarroja pasa a través de las ranuras con la velocidad que tenga el eje del motor.

Los pulsos del sensor de ranura entran a un integrado encargado de hacer la conversión frecuencia-voltaje, el LM2907 se acomodaba a nuestras necesidades y con una configuración de capacitores y resistencias podemos graduar la tensión de salida con respecto a la frecuencia de entrada, el circuito es el siguiente.

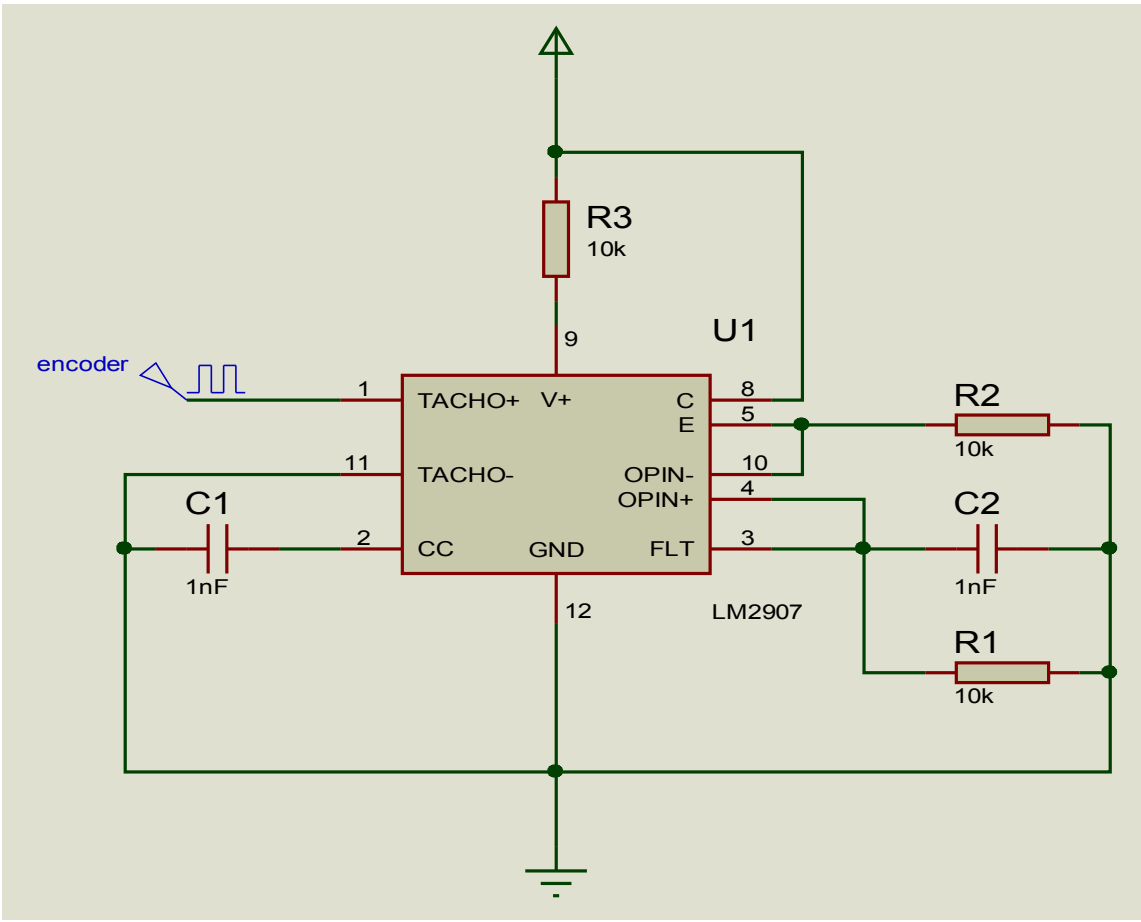

Figura 4 circuito de conversión frecuencia-tensión

Para este circuito se deben tener en cuenta algunos parámetros en los valores de los componentes teniendo en cuenta la siguiente fórmula:

$$Vo=VccxFinxC1XR1Xk$$

Ecuación 1

Donde K es la constante de ganancia típicamente de 1.0

De esta manera la tensión de salida estaría dada por:

$$Vo=5VxFinx1nFx10K\Omega$$

Ecuación 2

Una R1 de 10KΩ un condensador 1 de 1nF y dependiendo de la frecuencia que entregue el encoder al conversor frecuencia-voltaje será la tensión de salida.

En la siguiente tabla se evidencia la tensión de salida con respecto a la frecuencia de entrada.

Frecuencia de entrada	**Tensión de salida**
10hz	86.7mv
20hz	145.7mv
30hz	204.7mv
50hz	322.7mv
100hz	818.4mv
200hz	1207.2mv
300hz	1797.5mv
500hz	2.98v
1khz	3.44v
1.5khz	3.46v
2khz	3.48v

Tabla 2 Tabla comparativa del conversor frecuencia-tensión

5.5. El algoritmo en MATLAB y el descriptor

Ya teniendo la parte física de nuestro proyecto los circuitos de control y sus interfaces de potencia, el algoritmo que rige estos circuitos de control solo nos queda la interfaz con el PC, esto lo logramos gracias a un descriptor que enlaza nuestro micro controlador principal a un puerto del PC a continuación presentamos más específicamente dicho descriptor.

```
#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

////////////////////////////////////////////////////////////////
///
///   start config descriptor
///   right now we only support one configuration descriptor.
///   the config, interface, class, and endpoint goes into this array.
///
////////////////////////////////////////////////////////////////

  #DEFINE USB_TOTAL_CONFIG_LEN      32 //config+interface+class+endpoint

  //configuration descriptor
  char const USB_CONFIG_DESC[] = {
  //config_descriptor for config index 1
        USB_DESC_CONFIG_LEN,     //length of descriptor size
        USB_DESC_CONFIG_TYPE,         //constant CONFIGURATION (0x02)
        USB_TOTAL_CONFIG_LEN,0,  //size of all data returned for this config
        1,      //number of interfaces this device supports
        0x01,               //identifier for this configuration.  (IF we had more than one configurations)
        0x00,               //index of string descriptor for this configuration
        0xC0,               //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't), bits
0-4 reserved and bit7=1
        0x32,               //maximum bus power required (maximum milliamperes/2)  (0x32 =
100mA)
```

```
  //interface descriptor 0 alt 0
        USB_DESC_INTERFACE_LEN,  //length of descriptor
        USB_DESC_INTERFACE_TYPE,      //constant INTERFACE (0x04)
        0x00,               //number defining this interface (IF we had more than one interface)
        0x00,               //alternate setting
        2,       //number of endpoints, not counting endpoint 0.
        0xFF,               //class code, FF = vendor defined
        0xFF,               //subclass code, FF = vendor
        0xFF,               //protocol code, FF = vendor
        0x00,               //index of string descriptor for interface


  //endpoint descriptor
        USB_DESC_ENDPOINT_LEN, //length of descriptor
        USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
        0x81,               //endpoint number and direction (0x81 = EP1 IN)
        0x02,               //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
        USB_EP1_TX_SIZE & 0xFF,USB_EP1_TX_SIZE >> 8,         //maximum packet size
supported
        0x01,               //polling interval in ms. (for interrupt transfers ONLY)


  //endpoint descriptor
        USB_DESC_ENDPOINT_LEN, //length of descriptor
        USB_DESC_ENDPOINT_TYPE,     //constant ENDPOINT (0x05)
        0x01,               //endpoint number and direction (0x01 = EP1 OUT)
        0x02,               //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
        USB_EP1_RX_SIZE & 0xFF,USB_EP1_RX_SIZE >> 8,         //maximum packet size
supported
        0x01,               //polling interval in ms. (for interrupt transfers ONLY)


  };


  //****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
```

```
  //since we can't make pointers to constants in certain pic16s, this is an offset table to find
  //  a specific descriptor in the above table.

  //NOTE: DO TO A LIMITATION OF THE CCS CODE, ALL HID INTERFACES MUST START AT 0 AND BE SEQUENTIAL
  //      FOR EXAMPLE, IF YOU HAVE 2 HID INTERFACES THEY MUST BE INTERFACE 0 AND INTERFACE 1
  #define USB_NUM_HID_INTERFACES   0

  //the maximum number of interfaces seen on any config
  //for example, if config 1 has 1 interface and config 2 has 2 interfaces you must define this as 2
  #define USB_MAX_NUM_INTERFACES   1

  //define how many interfaces there are per config.  [0] is the first config, etc.
  const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};

  #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
     #error USB_TOTAL_CONFIG_LEN not defined correctly
  #endif
////////////////////////////////////////////////////////////////////
///
///   start device descriptors
///
////////////////////////////////////////////////////////////////////

  //device descriptor
  char const USB_DEVICE_DESC[] ={
       USB_DESC_DEVICE_LEN,         //the length of this report
       0x01,              //constant DEVICE (0x01)
       0x10,0x01,          //usb version in bcd
       0x00,              //class code (if 0, interface defines class.  FF is vendor defined)
       0x00,              //subclass code
       0x00,              //protocol code
```

```
        USB_MAX_EP0_PACKET_LENGTH,   //max packet size for endpoint 0. (SLOW SPEED
SPECIFIES 8)

        0xD8,0x04,          //vendor id (0x04D8 is Microchip)

        0x08,0x00,          //product id

        0x00,0x01,          //device release number

        0x01,               //index of string description of manufacturer. therefore we point to string_1
array (see below)

        0x02,               //index of string descriptor of the product

        0x00,               //index of string descriptor of serial number

        USB_NUM_CONFIGURATIONS   //number of possible configurations

  };




//////////////////////////////////////////////////////////////////

///

///   start string descriptors

///   String 0 is a special language string, and must be defined.  People in U.S.A. can leave this
alone.

///

///   You must define the length else get_next_string_character() will not see the string

///   Current code only supports 10 strings (0 thru 9)

///

//////////////////////////////////////////////////////////////////


//the offset of the starting location of each string.

//offset[0] is the start of string 0, offset[1] is the start of string 1, etc.

const char USB_STRING_DESC_OFFSET[]={0,4,12};


#define USB_STRING_DESC_COUNT sizeof(USB_STRING_DESC_OFFSET)


char const USB_STRING_DESC[]={

  //string 0

        4, //length of string index

        USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
```

```
      0x09,0x04,   //Microsoft Defined for US-English
  //string 1
      8, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'C',0,
      'C',0,
      'S',0,
  //string 2
      18, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'C',0,
      'r',0,
      'i',0,
      's',0,
      't',0,
      'i',0,
      'a',0,
      'n',0,
};
#ENDIF
```

Algoritmo o driver para reconocimiento de la tarjeta en los puertos del PC “descriptor.h”

Como ya lo hemos dicho este algoritmo se encarga de comunicarse con nuestro micro controlador principal y el algoritmo hecho en CCS para este se reconoce inmediatamente se conecta el dispositivo y se encarga de numerarlo y de asignarle un puerto.

También está el algoritmo de MATLAB el cual se encarga de manejar todos los datos recolectados por nuestra tarjeta de adquisición procesar estos datos y visualizarlos de una manera apropiada, también de almacenar dichos datos para luego enviarlos al WORKSPACE y procesarlos en la TOOLBOX de MATLAB.

A continuación se presenta dicho algoritmo:

```
function varargout = menu8var(varargin)
% Esto se ejecuta cada vez que se interactua con el menu8var
```

```
% MENU8VAR M-file for menu8var.fig
%      MENU8VAR, by itself, creates a new MENU8VAR or raises the
existing
%      singleton*.
%
%      H = MENU8VAR returns the handle to a new MENU8VAR or the handle
to
%      the existing singleton*.
%
%      MENU8VAR('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MENU8VAR.M with the given input
arguments.
%
%      MENU8VAR('Property','Value',...) creates a new MENU8VAR or
raises the
%      stop.  All inputs are passed to menu8var_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu8var.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help menu8var

% Last Modified by GUIDE v2.5 24-Sep-2011 15:26:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @menu8var_OpeningFcn, ...
                   'gui_OutputFcn',  @menu8var_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end

% --- Executes just before menu8var is made visible.
% Sólo se ejecuta al inicio
function menu8var_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to menu8var (see VARARGIN)

global t data_in data_out vid_pid_norm out_pipe in_pipe conectado
my_in_pipe my_out_pipe handles1
```

```
handles1 = handles;
% Choose default command line output for menu8var
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes menu8var wait for user response (see UIRESUME)
% uiwait(handles.figure1);


conectado = 0;

data_in = eye(1,64,'uint8');       % Se declara el vector de datos de
entrada (el que se recibe del PIC)
data_out = eye(1,64,'uint8');      % Se declara el vector de datos de
salida (el que se envia al PIC)
                                   % TODOS LOS DATOS SE DECLARAN COMO
                                   % UINT8 de lo contrario no hay
                                   % comunicación.

vid_pid_norm = libpointer('int8Ptr',[uint8('vid_04d8&pid_0008') 0]);
out_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);
in_pipe = libpointer('int8Ptr',[uint8('\MCHP_EP1') 0]);

%SE DECLARA LA RUTINA QUE DEBE DE EJECUTARSE AL CERRAR LA APLICACIÓN
set(handles.figure1,'CloseRequestFcn',@closeGUI);
%CREAMOS EL OBJETO TIMER Y SE ARRANCA
t = timer('TimerFcn',@tiempo, 'ExecutionMode',
'fixedSpacing' ,'BusyMode', 'queue' ,'Period', 0.055);
start (t);

loadlibrary mpusbapi _mpusbapi.h alias libreria
% Los archivos _mpusbapi.c y mpusbapi.dll deben de estar en la misma
% carpeta de trabajo y se obtienen de la descarga del driver en la
% pagina de microchip ("Microchip MCHPFSUSB v2.4 Installer.zip"),
% al instalarse queda ubicado en X:\Microchip Solutions\USB
% Tools\MCHPUSB Custom Driver\Mpusbapi\Dll\Borland_C, en caso de
descargar
% una version de driver más reciente, reemplace éstos archivos por los
más
% nuevos.

%libisloaded libreria               % Confirma que la librería ha sido
                                   % cargada
%libfunctions('libreria', '-full')  % Muestra en la línea de comandos
las
                                   % funciones de la librería
%libfunctionsview libreria          % Muestra en un cuadro lo mismo
que la
                                   % instrucción anterior

%calllib('libreria','MPUSBGetDLLVersion');

while (conectado == 0)
    [conectado] =
calllib('libreria','MPUSBGetDeviceCount',vid_pid_norm);
    if (conectado ==0)
        selection = questdlg('La tarjeta de evaluación no se
encuentra',...
                        'Tarjeta no encontrada',...
```

```
                        'Buscar de nuevo','Cancelar','Cancelar');
        switch selection,
            case 'Cancelar',
                stop (t);
                return
            case 'Buscar de nuevo'
                conectado = 0;
        end
    end
end
if conectado == 1   % Es importante seguir ésta secuencia para
comunicarse con el PIC:
                    % 1. Abrir tuneles, 2. Enviar/Recibir dato
                    % 3. Cerrar tuneles

    [my_out_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0),
vid_pid_norm, out_pipe, uint8(0), uint8 (0)); % Se abre el tunel de
envio
    [my_in_pipe] = calllib('libreria', 'MPUSBOpen',uint8 (0),
vid_pid_norm, in_pipe, uint8 (1), uint8 (0)); % Se abre el tunel de
recepción
end
end

% --- Outputs from this function are returned to the command line.
function varargout = menu8var_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
end

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton1
end
% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton2
end

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton3
end

% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton4
end

% --- Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton5
end

% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton6
end

% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton7
end

% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of radiobutton8
end

% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of checkbox1
end

% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2
end

% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox3
```

```
end

% --- Executes on button press in checkbox4.
function checkbox4_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox4
end

% --- Executes on button press in checkbox5.
function checkbox5_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox5
end

% --- Executes during object creation, after setting all properties.
function text2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
end

function closeGUI(source,eventdata)
%src is the handle of the object generating the callback (the source
of the event)
%evnt is the The event data structure (can be empty for some
callbacks)
global t my_in_pipe my_out_pipe conectado
selection = questdlg('Desea cerrar la aplicación?',...
                     'Confirmación',...
                     'Si','No','Si');
switch selection,
   case 'Si',
     stop (t);
     if conectado == 1
        calllib('libreria', 'MPUSBClose', my_in_pipe);  % Se cierra el
tunel de recepción
        calllib('libreria', 'MPUSBClose', my_out_pipe); % Se cierra el
tunel de envio
     end
     unloadlibrary libreria      % Importante descargar la librería de
memoria, de lo contrario genera errores
     delete(gcf)
     close all
     clear all
   case 'No'
     return
end
 end

 %TIMER EMPIEZA A REALIZAR TAREA ASIGNADA
function tiempo(hObject, eventdata) %(source,eventdata)

global conectado my_out_pipe my_in_pipe data_in data_out handles1
int16 data;
```

```
%a = a + 1
if conectado == 1
    calllib('libreria', 'MPUSBWrite',my_out_pipe, data_out, uint8(64),
uint8(64), uint8(200)); % Se envia el dato al PIC
    [aa,bb,data_in,dd] = calllib('libreria', 'MPUSBRead',my_in_pipe,
data_in, uint8(64), uint8(64), uint8(200)); % Se recibe el dato que
envia el PIC
    %data_out (1) = data_out (1)+ 1;     % Se incrementa el primer
dato del vector que se envia
    %data_in(1)                          % Se muestra el primer valor
del vector recibido
end

%Se actualizan los datos que el PIC nos envia en la interfaz de
usuario.
% Se recibe un número binario procedente del PIC, se analiza bit a bit
y se
% activa/desactiva el checkbox correspondiente en la interfaz de
usuario.
i=2;
while (i<7)
if (bitget(data_in(2), i) == 1)
    set(handles1.(strcat('checkbox',int2str(i-1))),'value',0);
else
    set(handles1.(strcat('checkbox',int2str(i-1))),'value',1);
end
i = i + 1;
end

% Se actualizan los datos que se envia al PIC y que son modificados
desde la interfaz de usuario.
% Se genera un número binario de 8 bit de acuerdo a la combinación de
los
% botones seleccionados
i = 1;
while (i<9)
Hint=get(handles1.(strcat('radiobutton',int2str(i))),'value');
%returns toggle state of radiobutton
if (Hint==1)
    data_out(1) = bitor(data_out(1), uint8(2 ^ (i-1)));
else
    %El byte 0 se le aplica un complemento a 1 de 2^7
    data_out(1) = bitand(data_out(1), bitcmp(uint8(2^(i-1))));
end
i = i + 1;
end

% Se actualiza la lectura analógica en el cuadro de texto

data = (uint16(data_in(4)) * uint16(256)) + uint16(data_in(3));
%set(handles1.text2,'String',(num2str(data)));

data1 = (uint16(data_in(6)) * uint16(256)) + uint16(data_in(5));
%set(handles1.text3,'String',(num2str(data1)));

data2 = (uint16(data_in(8)) * uint16(256)) + uint16(data_in(7));
%set(handles1.text4,'String',(num2str(data2)));

data3 = (uint16(data_in(10)) * uint16(256)) + uint16(data_in(9));
%set(handles1.text5,'String',(num2str(data3)));
```

```
data4 = (uint16(data_in(12)) * uint16(256)) + uint16(data_in(11));
%set(handles1.text6,'String',(num2str(data4)));

data5 = (uint16(data_in(14)) * uint16(256)) + uint16(data_in(13));
%set(handles1.text7,'String',(num2str(data5)));

data6 = (uint16(data_in(16)) * uint16(256)) + uint16(data_in(15));
%set(handles1.text8,'String',(num2str(data6)));

data7 = (uint16(data_in(18)) * uint16(256)) + uint16(data_in(17));
%set(handles1.text9,'String',(num2str(data7)));

data8 = (uint16(data_in(20)) * uint16(256)) + uint16(data_in(19));
%set(handles1.text10,'String',(num2str(data8)));

data9 = (uint16(data_in(22)) * uint16(256)) + uint16(data_in(21));
%set(handles1.text11,'String',(num2str(data9)));

data10 = (uint16(data_in(24)) * uint16(256)) + uint16(data_in(23));
%set(handles1.text12,'String',(num2str(data10)));

data11 = (uint16(data_in(26)) * uint16(256)) + uint16(data_in(25));
%set(handles1.text13,'String',(num2str(data11)));

data12 = (uint16(data_in(28)) * uint16(256)) + uint16(data_in(27));
%set(handles1.text14,'String',(num2str(data12)));

data13 = (uint16(data_in(30)) * uint16(256)) + uint16(data_in(29));
%set(handles1.text15,'String',(num2str(data13)));

data14 = (uint16(data_in(32)) * uint16(256)) + uint16(data_in(31));
%set(handles1.text16,'String',(num2str(data14)));

data15 = (uint16(data_in(34)) * uint16(256)) + uint16(data_in(33));
%set(handles1.text17,'String',(num2str(data15)));

x(1)= data
x(2)= data1
x(3)= data2
x(4)= data3
x(5)= data4
x(6)= data5
x(7)= data6
x(8)= data7
x(9)= data8
x(10)= data9
x(11)= data10
x(12)= data11
x(13)= data12
x(14)= data13
x(15)= data14
x(16)= data15

  end
```

Programa desarrollado en MATLAB para el procesamiento de las señales recolectadas por la tarjeta

6. Conclusiones.

El modulo de adquisición de variables presta una visualización clara del proceso tanto de inyección como de lectura de señales en el motor, gracias al WORKSPACE de MATLAB el usuario puede detallar y comparar en tiempo real la respuesta en velocidad del motor con respecto a los pulsos de excitación que se aplican a este mismo en una misma grafica.

El modulo también está diseñado de una manera simple con el fin de facilitar la conexión al usuario, solo basta con conectar el modulo de adquisición vía USB al PC y alimentar un circuito complementario compuesto por el conversor frecuencia-tensión y el circuito de excitación, como se puede apreciar esta parte del modulo se debe alimentar por aparte ya que si alimentamos esta parte en conjunto con el modulo de adquisición USB sobrecargaríamos la energía que entrega el PC al micro controlador principal.

Se debe alimentar con una fuente de 5v externa ya que el puerto USB solo nos entrega 5v con 500 mA de corriente suficiente para alimentar únicamente el micro controlador principal.

7. Bibliografía

7.1 Determinación de los parámetros para el servomotor NXT LEGO mindstoms con Técnicas de Identificación de Sistemas.

Ing. María luisa pinto salamanca, MSc Giovanni Rodrigo Bermúdez Bohórquez.

7.2 Diseño e implementación de un sistema de control digital de posición para un motor DC, Edwin Alfonso Gonzales Querubín-Morgan Garabito Vásquez, universidad santo Tomas Bucaramanga(2006)

Web. http://es.scribd.com/cymonso/d/12408800-Control-de-Motor-Encoder

7.3 [PID] Control Systems Simulation using MATLAB and SIMULINK

UNIVERSIDAD DE CALIFORNIA, departamento de ingeniería mecánica ME134 sistemas de control automático (2002).

7.4 Analog and Digital Control System Design-Transfer-Function, State-Space, and
Algebraic Methods, Chi-Tsong Chen State University of New York at Stony Brook

7.5 Modern Control Systems Analysis and Design Usign MATLAB, Robert H Bishop, The University of Texas

7.6 Process Systems Analysis and Control, Donald R Coughanowr-second edition.

7.7 Sistemas de control automático séptima edición, Benjamin C Kuo, departamento de ingeniería eléctrica y sistemas, Universidad de Illinois.

7.8 Sistemas de control moderno decima edición Richard C Dorf Universidad de California, Davis – Robert H Bishop, Universidad de Texas.

7.9 Ingeniería de control moderna, tercera edición

Katsuhiko ogata.

7.10 Hoja técnica LM2907
www.datasheetcatalog.org/datasheet/nationalsemiconductor/DS007942.PDF

7.11 Hoja técnica PIC18f4550
www.datasheetcatalog.org/datasheet/microchip/39617a.PDF

7.12 hoja técnica PIC12f629

www.datasheetcatalog.org/datasheets2/50/5044095_1.pdf