TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Índice

Tesis publicada con autorización del autor
No olvide citar esta tesis

Anexo 1: Código fuente del programa para el microcontrolador PIC 18F2550.

```
' * Project name:   Simulador de Señales   // Nombre del proyecto
' * Test configuration:
'     MCU:            PIC18F2550
'     Dev.Board:      EasyPIC5
'     Oscillator:     HS 8.000 MHz  (USB osc. is raised with PLL to 48.000MHz)  //tenemos el oscilador de 8mhz en la tarjeta
'     Ext. Modules:    -
'     SW:             mikroBasic v7.1
'*****************************************************************************
program Simulador de Señales  // nombre del proyecto
'*****************************************************************************
include "USBdsc"  // módulo que incluye los descriptores para que funcione la comunicación HID entre el usb y la pc
'*****************************************************************************
 DIM ADC_A AS WORD
 DIM KOMANDA AS byte absolute $28
 DIM USB_START, IO_PRO, CH,I as  byte
 DIM userWR_buffer  as  byte[255]
 DIM userRD_buffer  as  byte[255]
const usbConfirmAction  = 0
const usbReadADC        = 1
'*****************************************************************************
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
' Main Interrupt Routine

'*********************************************************************************

sub procedure interrupt

  HID_InterruptProc       //se llama a la interrupción de HID para mantener la comunicación viva entre USB y PC

end sub

'*********************************************************************************

sub procedure CLER_WR_USB

 FOR I=0 TO 9 userWR_buffer[I]=0 NEXT I   // limpiamos los registros de escritura y  lectura del buffer (10 bits)

end sub

sub procedure CLER_RD_USB

 FOR I=0 TO 9 userRD_buffer[I]=0 NEXT I

end sub

'*********************************************************************************

sub procedure Init_Main     // procedimiento para configurar los puertos y se deshabilitan las interrupciones que no son necesarias

  '--------------------------------------

  ' Disable interrupts

  '--------------------------------------

    INTCON = 0              ' Disable GIE, PEIE, TMR0IE,INT0IE,RBIE //estan en todas las configuraciones

  INTCON2 = 0xF5

  INTCON3 = 0xC0
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
  RCON.IPEN = 0       ' Disable Priority Levels on interrupts

  PIE1 = 0   PIE2 = 0   PIR1 = 0   PIR2 = 0

  ADCON1 = ADCON1 or 0x0F ' Configure all ports with analog function as digital//configura todos los puertos con funciones analógicas como digitales

  ADCON2 = 0

  '--------------------------------------

  ' Ports Configuration

  '--------------------------------------

  TRISA = 0xFF // Configuramos portA como entrada

  LATA = 0

end sub

'*************************************************************************

' Main Program Routine

'*************************************************************************

main:

 Init_Main()  //llamamos a la subrutina de configuración de puertos e interrupciones

   CLER_RD_USB

   CLER_WR_USB

 HID_Enable(@userRD_buffer, @userWR_buffer)      // utilizamos las variables userRD y userWR en este proceso de habilitación de comunicación HID USB - habilitamos

 Delay_mS(1000)

 Delay_mS(1000)   // tiempo para estabilizar la comunicación

'*************************************************************************
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
 while true   // bucle infinito

    USB_START = HID_Read()       // esta variable devuelve la cantidad de caracteres recibidos del host, lo recibido se almacena en el buffer de lectura

    IO_PRO = 0                   // contador a 0

'*******************************************************************************

    while IO_PRO < USB_START  ''    //funciona cuando recibo un llamado del host

    KOMANDA = userRD_buffer[0]     // leemos la data del usb, komanda = al vendor id

 select case KOMANDA

'-------------------------------------------------------------------------------

 case usbReadADC  '    //rutina para mandar ADC por USB

   ADC_A = ADC_read(userRD_buffer[1])   //ADC_read () lee el canal que solicita el buffer , el [0] es el product id

   Delay_ms(4)  // para esperar el proceso de conversión del ADC

   IF  (ADC_A>1020)THEN  ADC_A=ADC_A-1020  ADC_B=255  ADC_C=255  ADC_D=255  ADC_E=255  END IF

   IF  (ADC_A>765)THEN  ADC_A=ADC_A-765  ADC_B=255  ADC_C=255  ADC_D=255  END IF

   IF (ADC_A>510)THEN ADC_A=ADC_A-510 ADC_B=255 ADC_C=255  END IF

   IF (ADC_A>255)THEN ADC_A=ADC_A-255 ADC_B=255  END IF

    userWR_buffer[0] = userRD_buffer[0] // escribimos lo que leemos en el byte de lectura para enviarlo a la PC  (identificador)

     userWR_buffer[1] = usbReadADC

     userWR_buffer[2] = ADC_A   //mando el ADC de cada canal

     userWR_buffer[3] = userRD_buffer[1] //identificador del canal que está en el ADC
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
 IF 0 = HID_Read()THEN   // si no has recibido nada --> escribimos en el buffer de salida

     HID_Write(@userWR_buffer, 8)   // escribimos en el buffer de salida que es un array de
8 bytes , envío lo que leo

     Delay_ms(1)

 ELSE Delay_ms(1)  END IF

  wend

'********************************************************************************

 wend

 HID_Disable() // deshabilitamos la comunicación

'********************************************************************************

end.

'********************************************************************************
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

<u>Anexo 2:</u> Código fuente del programa para la interfaz de usuario desarrollada en Visual Basic.

```
using Microsoft.Win32.SafeHandles;  son librerias

using System.Globalization;

using System.IO;

using System.Runtime.InteropServices;


using Microsoft.VisualBasic;

using System;

using System.Collections;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Timers;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Media;

using ZedGraph;

namespace USBScope

{
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
public partial class APP_MAIN : Form
{
    private IntPtr deviceNotificationHandle;
    private Boolean exclusiveAccess;
    private FileStream fileStreamDeviceData;
    private SafeFileHandle hidHandle;
    private String hidUsage;
    private Boolean myDeviceDetected;
    private String myDevicePathName;
    private Boolean transferInProgress = false;
    private DeviceManagement MyDeviceManagement = new DeviceManagement();
    private Hid MyHid = new Hid();
    private static System.Timers.Timer tmrReadTimeout;
    private static System.Timers.Timer tmrContinuousDataCollect;
    internal APP_MAIN FrmMy;

    byte m_USB_EndPoint_Command = 1;     // Lectura canales analogicos ... se envia un comando
    byte m_USB_EndPoint_AnalogCh = 0;     // Lectura canales analogicos

    // Plots
    double m_ADCValue_Ch00;
    double m_ADCValue_Ch01;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
double m_ADCValue_Ch02;

double m_ADCValue_Ch03;


double m_DataID;

bool m_EnableMonitoring;

string m_DataRead;

int m_Status;


GraphPane myPanelCh00;

GraphPane myPanelCh01;

GraphPane myPanelCh02;


LineItem myCurveCh00;

LineItem myCurveCh01;

LineItem myCurveCh02;


double[] x = new double[200];

double[] m_Table_Ch00 = new double[200];

double[] m_Table_Ch01 = new double[200];

double[] m_Table_Ch02 = new double[200];


private delegate void MarshalToForm(String action, String textToAdd);
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
public APP_MAIN()
{
    InitializeComponent();
}

private void APP_MAIN_Load(object sender, EventArgs e)
{
    try
    {
        FrmMy = this;
        Startup();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        DisplayException( this.Name, ex );
        throw;
    }

    // Plots
    m_Status = 0;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    InitGraphicPanel();
}

/// <summary>
/// Perform actions that must execute when the program starts.
/// </summary>

private void Startup()
{
    try
    {
        MyHid = new Hid();
        InitializeDisplay();

        tmrContinuousDataCollect = new System.Timers.Timer(50);
            tmrContinuousDataCollect.Elapsed += new
ElapsedEventHandler(OnDataCollect);
        tmrContinuousDataCollect.Stop();
        tmrContinuousDataCollect.SynchronizingObject = this;

        tmrReadTimeout = new System.Timers.Timer(5000);
        tmrReadTimeout.Elapsed += new ElapsedEventHandler(OnReadTimeout);
        tmrReadTimeout.Stop();
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            //  Default USB Vendor ID and Product ID:
            txtVendorID.Text = "0001";
            txtProductID.Text = "0020";
        }
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
            throw;
        }
    }

    ///  <summary>
    ///  Exchange data with the device.
    ///  </summary>
    /// <param name="source"></param>
    /// <param name="e"></param>
    ///  <remarks>
    ///  The timer is enabled only if cmdContinous has been clicked,
    ///  selecting continous (periodic) transfers.
    ///  </remarks>
    private void OnDataCollect(object source, ElapsedEventArgs e)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
{
    try
    {
        if (transferInProgress == false)
        {
            ReadAndWriteToDevice();
        }
    }
    catch (Exception ex)
    {
        DisplayException(this.Name, ex);
        throw;
    }
}


/// <summary>
/// ystem timer timeout if read via interrupt transfer doesn't return.
/// </summary>
/// <param name="source"></param>
/// <param name="e"></param>

private void OnReadTimeout(object source, ElapsedEventArgs e)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
{
    MyMarshalToForm("AddItemToListBox", "The attempt to read a report timed out.");

    CloseCommunications();

    tmrReadTimeout.Stop();

    //  Enable requesting another transfer.

    MyMarshalToForm("EnableCmdOnce", "");
    MyMarshalToForm("ScrollToBottomOfListBox", "");
}

///  <summary>
///  Initialize the elements on the form.
///  </summary>

private void InitializeDisplay()
{
    Int16 count = 0;
    String byteValue = null;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
try
{
   //  Create a dropdown list box for each byte to send in a report.
   //  Display the values as 2-character hex strings.

   for (count = 0; count <= 255; count++)
   {
      byteValue = String.Format("{0:X2} ", count);
      FrmMy.cboByte0.Items.Insert(count, byteValue);
      FrmMy.cboByte1.Items.Insert(count, byteValue);
   }

   //  Select a default value for each box

   FrmMy.cboByte0.SelectedIndex = 1;
   FrmMy.cboByte1.SelectedIndex = 0;

   //  Check the autoincrement box to increment the values each time a report is
sent.

   chkAutoincrement.CheckState = System.Windows.Forms.CheckState.Unchecked;

   //  Don't allow the user to select an input report buffer size until there is
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    //  a handle to a HID.


    cmdInputReportBufferSize.Focus();

    cmdInputReportBufferSize.Enabled = false;


    if (MyHid.IsWindowsXpOrLater())

    {

        chkUseControlTransfersOnly.Enabled = true;

    }

    else

    {

        //  If the operating system is earlier than Windows XP,

        //  disable the option to force Input and Output reports to use control transfers.


        chkUseControlTransfersOnly.Enabled = false;

    }

}

catch (Exception ex)

{

    DisplayException(this.Name, ex);

    throw;

}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
}

/// <summary>
/// Performs various application-specific functions that
/// involve accessing the application's form.
/// </summary>
///
/// <param name="action"> a String that names the action to perform on the form</param>
/// <param name="formText"> text that the form displays or the code uses for
/// another purpose. Actions that don't use text ignore this parameter. </param>

private void AccessForm(String action, String formText)
{
   try
   {
      // Select an action to perform on the form:

      switch (action)
      {
         case "AddItemToListBox":
            break;
         case "AddItemToTextBox":
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    txtBytesReceived.SelectedText = formText + "\r\n";


    break;
case "EnableCmdOnce":


    //  If it's a single transfer, re-enable the command button.


    if (cmdContinuous.Text == "Iniciar monitoreo")
    {
        cmdOnce.Enabled = true;
    }
    break;
case "ScrollToBottomOfListBox":
    break;
case "TextBoxSelectionStart":


    txtBytesReceived.SelectionStart = formText.Length;


    break;
default:
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                break;
            }
        }
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
            throw;
        }
    }

    /// <summary>
    /// Enables accessing a form's controls from another thread
    /// </summary>
    ///
    /// <param name="action"> a String that names the action to perform on the form </param>
    /// <param name="textToDisplay"> text that the form displays or the code uses for
    /// another purpose. Actions that don't use text ignore this parameter. </param>

    private void MyMarshalToForm(String action, String textToDisplay)
    {
        object[] args = { action, textToDisplay };
        MarshalToForm MarshalToFormDelegate = null;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        //  The AccessForm routine contains the code that accesses the form.

        MarshalToFormDelegate = new MarshalToForm(AccessForm);

        //  Execute AccessForm, passing the parameters in args.

        base.Invoke(MarshalToFormDelegate, args);
    }

    /// <summary>
    /// Provides a central mechanism for exception handling.
    /// Displays a message box that describes the exception.
    /// </summary>
    ///
    /// <param name="moduleName"> the module where the exception occurred. </param>
    /// <param name="e"> the exception </param>

    internal static void DisplayException(String moduleName, Exception e)
    {
        String message = null;
        String caption = null;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            //  Create an error message.


            message = "Exception: " + e.Message + "@" + "Module: " + moduleName + "@" +
"Method: " + e.TargetSite.Name;

            caption = "Unexpected Exception";


            MessageBox.Show(message, caption, MessageBoxButtons.OK);


            tmrContinuousDataCollect.Enabled = false;

            tmrContinuousDataCollect.Stop();
        }


        ///  <summary>

        ///  Initiates exchanging reports.

        ///  The application sends a report and requests to read a report.

        ///  </summary>


        private void ReadAndWriteToDevice()

        {

            try

            {
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
//  If the device hasn't been detected, was removed, or timed out on a previous attempt
//  to access it, look for the device.

if ((myDeviceDetected == false))
{
    myDeviceDetected = FindTheHid();
}

if ((myDeviceDetected == true))
{
    //  Get the bytes to send in a report from the combo boxes.
    //  Increment the values if the autoincrement check box is selected.

    if (Convert.ToDouble(chkAutoincrement.CheckState) == 1)
    {
        if (cboByte0.SelectedIndex < 255)
        {
            cboByte0.SelectedIndex = cboByte0.SelectedIndex + 1;
        }
        else
        {
            cboByte0.SelectedIndex = 0;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    }
    if (cboByte1.SelectedIndex < 255)
    {
        cboByte1.SelectedIndex = cboByte1.SelectedIndex + 1;
    }
    else
    {
        cboByte1.SelectedIndex = 0;
    }
}

//  An option button selects whether to exchange Input and Output reports
//  or Feature reports.

if ((optInputOutput.Checked == true))
{
    ExchangeInputAndOutputReports();
}
else
{
    ExchangeFeatureReports();
}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        }
    }
    catch (Exception ex)
    {
        DisplayException(this.Name, ex);
        throw;
    }
}

/// <summary>
/// Uses a series of API calls to locate a HID-class device
/// by its Vendor ID and Product ID.
/// </summary>
///
/// <returns>
///  True if the device is detected, False if not detected.
/// </returns>

private Boolean FindTheHid()
{
    Boolean deviceFound = false;
    String[] devicePathName = new String[128];
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
String functionName = "";

Guid hidGuid = Guid.Empty;

Int32 memberIndex = 0;

Int32 myProductID = 0;

Int32 myVendorID = 0;

Boolean success = false;


try

{

    myDeviceDetected = false;

    CloseCommunications();


    //  Get the device's Vendor ID and Product ID from the form's text boxes.


    GetVendorAndProductIDsFromTextBoxes(ref myVendorID, ref myProductID);


    //  ***

    //  API function: 'HidD_GetHidGuid


    //  Purpose: Retrieves the interface class GUID for the HID class.


    //  Accepts: 'A System.Guid object for storing the GUID.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
// ***

Hid.HidD_GetHidGuid(ref hidGuid);

functionName = "GetHidGuid";

// Fill an array with the device path names of all attached HIDs.

deviceFound = MyDeviceManagement.FindDeviceFromGuid(hidGuid, ref
devicePathName);

// If there is at least one HID, attempt to read the Vendor ID and Product ID
// of each device until there is a match or all devices have been examined.

if (deviceFound)
{
   memberIndex = 0;

   do
   {
      // ***
      // API function:
      // CreateFile
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
// Purpose:

// Retrieves a handle to a device.


// Accepts:

// A device path name returned by SetupDiGetDeviceInterfaceDetail

// The type of access requested (read/write).

//  FILE_SHARE attributes to allow other processes to access the device while this handle is open.

// A Security structure or IntPtr.Zero.

// A creation disposition value. Use OPEN_EXISTING for devices.

// Flags and attributes for files. Not used for devices.

// Handle to a template file. Not used.


// Returns: a handle without read or write access.

// This enables obtaining information about all HIDs, even system

// keyboards and mice.

// Separate handles are used for reading and writing.

// ***


// Open the handle without read/write access to enable getting information about any HID, even system keyboards and mice.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                hidHandle = FileIO.CreateFile(devicePathName[memberIndex], 0,
FileIO.FILE_SHARE_READ | FileIO.FILE_SHARE_WRITE, IntPtr.Zero,
FileIO.OPEN_EXISTING, 0, 0);

                functionName = "CreateFile";

                if (!hidHandle.IsInvalid)
                {
                    // The returned handle is valid,
                    // so find out if this is the device we're looking for.

                    // Set the Size property of DeviceAttributes to the number of bytes in the
structure.

                    MyHid.DeviceAttributes.Size = Marshal.SizeOf(MyHid.DeviceAttributes);

                    // ***
                    // API function:
                    // HidD_GetAttributes

                    // Purpose:
                    // Retrieves a HIDD_ATTRIBUTES structure containing the Vendor ID,
                    // Product ID, and Product Version Number for a device.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
// Accepts:
// A handle returned by CreateFile.
// A pointer to receive a HIDD_ATTRIBUTES structure.

// Returns:
// True on success, False on failure.
// ***

success = Hid.HidD_GetAttributes(hidHandle, ref MyHid.DeviceAttributes);

if (success)
{
    // Find out if the device matches the one we're looking for.

    if ((MyHid.DeviceAttributes.VendorID == myVendorID) &&
(MyHid.DeviceAttributes.ProductID == myProductID))
    {
        // Display the information in form's list box.
        ScrollToBottomOfListBox();

        myDeviceDetected = true;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                // Save the DevicePathName for OnDeviceChange().

                myDevicePathName = devicePathName[memberIndex];
            }
            else
            {
                // It's not a match, so close the handle.

                myDeviceDetected = false;
                hidHandle.Close();
            }
        }
        else
        {
            // There was a problem in retrieving the information.

            myDeviceDetected = false;
            hidHandle.Close();
        }
    }
```

// Keep looking until we find the device or there are no devices left to examine.

Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            memberIndex = memberIndex + 1;
        }
        while (!((myDeviceDetected || (memberIndex == devicePathName.Length))));
    }

    if (myDeviceDetected)
    {
        //  The device was detected.
        //  Register to receive notifications if the device is removed or attached.

        success                                                    =
MyDeviceManagement.RegisterForDeviceNotifications(myDevicePathName,
FrmMy.Handle, hidGuid, ref deviceNotificationHandle);

        //  Learn the capabilities of the device.

        MyHid.Capabilities = MyHid.GetDeviceCapabilities(hidHandle);

        if (success)
        {
            //  Find out if the device is a system mouse or keyboard.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                hidUsage = MyHid.GetHidUsage(MyHid.Capabilities);


                //  Get the Input report buffer size.


                GetInputReportBufferSize();

                cmdInputReportBufferSize.Enabled = true;


                //Close the handle and reopen it with read/write access.


                hidHandle.Close();

                hidHandle = FileIO.CreateFile(myDevicePathName, FileIO.GENERIC_READ
| FileIO.GENERIC_WRITE, FileIO.FILE_SHARE_READ | FileIO.FILE_SHARE_WRITE,
IntPtr.Zero, FileIO.OPEN_EXISTING, 0, 0);


                if (hidHandle.IsInvalid)

                {

                   exclusiveAccess = true;

                   ScrollToBottomOfListBox();

                }


                else

                {

                   if (MyHid.Capabilities.InputReportByteLength > 0)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            {
                //  Set the size of the Input report buffer.

                Byte[] inputReportBuffer = null;

                inputReportBuffer = new Byte[MyHid.Capabilities.InputReportByteLength];

                fileStreamDeviceData = new FileStream(hidHandle, FileAccess.Read | FileAccess.Write, inputReportBuffer.Length, false);
            }

            if (MyHid.Capabilities.OutputReportByteLength > 0)
            {
                Byte[] outputReportBuffer = null;

                outputReportBuffer = new Byte[MyHid.Capabilities.OutputReportByteLength];

            }

            //  Flush any waiting reports in the input buffer. (optional)

            MyHid.FlushQueue(hidHandle);

        }
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            }
        }
        else
        {
            //  The device wasn't detected.

            cmdInputReportBufferSize.Enabled = false;
            cmdOnce.Enabled = true;

            ScrollToBottomOfListBox();
        }
        return myDeviceDetected;
    }
    catch (Exception ex)
    {
        DisplayException(this.Name, ex);
        throw;
    }
}

/// <summary>
/// Sends a Feature report, then retrieves one.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
/// Assumes report ID = 0 for both reports.

/// </summary>


private void ExchangeFeatureReports()

{

    String byteValue = null;

    Int32 count = 0;

    Byte[] inFeatureReportBuffer = null;

    Byte[] outFeatureReportBuffer = null;

    Boolean success = false;


    try

    {

        inFeatureReportBuffer = null;


        if ((MyHid.Capabilities.FeatureReportByteLength > 0))

        {

            // The HID has a Feature report.


            // Set the size of the Feature report buffer.

            // Subtract 1 from the value in the Capabilities structure because

            // the array begins at index 0.
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        outFeatureReportBuffer = new
Byte[MyHid.Capabilities.FeatureReportByteLength];

        //  Store the report ID in the buffer:

        outFeatureReportBuffer[0] = 0;

        //  Store the report data following the report ID.
        //  Use the data in the combo boxes on the form.

        outFeatureReportBuffer[1] = Convert.ToByte(cboByte0.SelectedIndex);

        if (((int)outFeatureReportBuffer[1]) > 1)
        {
            outFeatureReportBuffer[2] = Convert.ToByte(cboByte1.SelectedIndex);
        }

        //  Write a report to the device

        success = MyHid.SendFeatureReport(hidHandle, outFeatureReportBuffer);

        if (success)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
{
    for (count = 0; count <= outFeatureReportBuffer.Length - 1; count++)
    {
        //  Display bytes as 2-character Hex strings.

        byteValue = String.Format("{0:X2} ", outFeatureReportBuffer[count]);
    }
}
else
{
    CloseCommunications();
}

//  Read a report from the device.

//  Set the size of the Feature report buffer.
//  Subtract 1 from the value in the Capabilities structure because
//  the array begins at index 0.

if ((MyHid.Capabilities.FeatureReportByteLength > 0))
{
    inFeatureReportBuffer = new
Byte[MyHid.Capabilities.FeatureReportByteLength];
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
}

// Read a report.

success = MyHid.GetFeatureReport(hidHandle, ref inFeatureReportBuffer);

if (success)
{
    // Display the report data received in the form's list box.
    txtBytesReceived.Text = "";

    for (count = 0; count <= inFeatureReportBuffer.Length - 1; count++)
    {
        // Display bytes as 2-character Hex strings.

        byteValue = String.Format("{0:X2} ", inFeatureReportBuffer[count]);

        // Display the received bytes in the text box.

        txtBytesReceived.SelectionStart = txtBytesReceived.Text.Length; //*
TRANSINFO: .NET Equivalent of Microsoft.VisualBasic NameSpace */
System.Runtime.InteropServices.Marshal.SizeOf( txtBytesReceived.Text );
        txtBytesReceived.SelectedText = byteValue + "\r\n";
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                }
            }
            else
            {
                CloseCommunications();
            }
        }
        else
        {
            ;
        }

        ScrollToBottomOfListBox();
        cmdOnce.Enabled = true;
    }
    catch (Exception ex)
    {
        DisplayException(this.Name, ex);
        throw;
    }
}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
/// <summary>
/// Sends an Output report, then retrieves an Input report.
/// Assumes report ID = 0 for both reports.
/// </summary>

private void ExchangeInputAndOutputReports()
{
    String byteValue = null;
    Int32 count = 0;
    Byte[] inputReportBuffer = null;
    Byte[] outputReportBuffer = null;
    Boolean success = false;

    try
    {
        success = false;

        // Don't attempt to exchange reports if valid handles aren't available
        // (as for a mouse or keyboard under Windows 2000/XP.)

        if (!hidHandle.IsInvalid)
        {
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
// Don't attempt to send an Output report if the HID has no Output report.

if (MyHid.Capabilities.OutputReportByteLength > 0)
{
    // Set the size of the Output report buffer.

    outputReportBuffer = new Byte[MyHid.Capabilities.OutputReportByteLength];

    // Store the report ID in the first byte of the buffer:

    outputReportBuffer[0] = 0;

    // Store the report data following the report ID.
    // Use the data in the combo boxes on the form.

    outputReportBuffer[1] = m_USB_EndPoint_Command;//
Convert.ToByte(cboByte0.SelectedIndex);

    if (((int)outputReportBuffer[1]) == m_USB_EndPoint_Command)
    {
        outputReportBuffer[2] = m_USB_EndPoint_AnalogCh;//
Convert.ToByte(cboByte1.SelectedIndex);
        m_USB_EndPoint_AnalogCh++;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            if (m_USB_EndPoint_AnalogCh >= 4) m_USB_EndPoint_AnalogCh = 0;
        }

        //  Write a report.

        if ((chkUseControlTransfersOnly.Checked) == true)
        {

            //  Use a control transfer to send the report,
            //  even if the HID has an interrupt OUT endpoint.

            success = MyHid.SendOutputReportViaControlTransfer(hidHandle,
outputReportBuffer);
        }
        else
        {
            // If the HID has an interrupt OUT endpoint, the host uses an
            // interrupt transfer to send the report.
            // If not, the host uses a control transfer.

            if (fileStreamDeviceData.CanWrite)
            {
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                    fileStreamDeviceData.Write(outputReportBuffer, 0,
outputReportBuffer.Length);

                    success = true;

                }

            }

            if (success)

            {

                txtBytesReceived.Text = "";

                for (count = 0; count <= outputReportBuffer.Length - 1; count++)

                {

                    //  Display bytes as 2-character hex strings.


                    byteValue = String.Format("{0:X2} ", outputReportBuffer[count]);

                }

            }

            else

            {

                CloseCommunications();

            }

        }

        else

        {

            ;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
}

// Read an Input report.

success = false;

// Don't attempt to send an Input report if the HID has no Input report.
// (The HID spec requires all HIDs to have an interrupt IN endpoint,
// which suggests that all HIDs must support Input reports.)

if (MyHid.Capabilities.InputReportByteLength > 0)
{
   // Set the size of the Input report buffer.

   inputReportBuffer = new Byte[MyHid.Capabilities.InputReportByteLength];

   if (chkUseControlTransfersOnly.Checked)
   {
      // Read a report using a control transfer.

      success = MyHid.GetInputReportViaControlTransfer(hidHandle, ref
inputReportBuffer);
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
if (success)
{
    txtBytesReceived.Text = "";

    for (count = 0; count <= inputReportBuffer.Length - 1; count++)
    {
        //  Display bytes as 2-character Hex strings.

        byteValue = String.Format("{0:X2} ", inputReportBuffer[count]);

        //  Display the received bytes in the text box.

        txtBytesReceived.SelectionStart = txtBytesReceived.Text.Length; ///*
TRANSINFO: .NET Equivalent of Microsoft.VisualBasic NameSpace */
System.Runtime.InteropServices.Marshal.SizeOf(txtBytesReceived.Text);
        txtBytesReceived.SelectedText = byteValue + "\r\n";
    }
}
else
{
    CloseCommunications();
}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            ScrollToBottomOfListBox();


            //  Enable requesting another transfer.


            AccessForm("EnableCmdOnce", "");
        }
        else
        {
            //  Read a report using interrupt transfers.
            //  To enable reading a report without blocking the main thread, this
            //  application uses an asynchronous delegate.

            IAsyncResult ar = null;
            transferInProgress = true;


            // Timeout if no report is available.


            tmrReadTimeout.Start();


            if (fileStreamDeviceData.CanRead)
            {
                fileStreamDeviceData.BeginRead(inputReportBuffer,                0,
inputReportBuffer.Length, new AsyncCallback(GetInputReportData), inputReportBuffer);
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
                }
                else
                {
                    CloseCommunications();
                }
            }
        }
        else
        {
            AccessForm("EnableCmdOnce", "");
        }
    }
    else
    {
        AccessForm("EnableCmdOnce", "");
    }
    ScrollToBottomOfListBox();
}
catch (Exception ex)
{
    DisplayException(this.Name, ex);
    throw;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    }
}

/// <summary>
/// Close the handle and FileStreams for a device.
/// </summary>
///
private void CloseCommunications()
{
    if (fileStreamDeviceData != null)
    {
        fileStreamDeviceData.Close();
    }

    if ((hidHandle != null) && (!(hidHandle.IsInvalid)))
    {
        hidHandle.Close();
    }

    // The next attempt to communicate will get new handles and FileStreams.

    myDeviceDetected = false;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    }

    /// <summary>
    /// Attempt to write a report and read a report.
    /// </summary>

    private void cmdOnce_Click(System.Object eventSender, System.EventArgs eventArgs)
    {
        try
        {
            // Don't allow another transfer request until this one completes.
            // Move the focus away from cmdOnce to prevent the focus from
            // switching to the next control in the tab order on disabling the button.

            fraSendAndReceive.Focus();
            cmdOnce.Enabled = false;

            ReadAndWriteToDevice();
        }
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            throw;
        }
    }

    /// <summary>
    /// Start or stop a series of periodic transfers.
    /// </summary>

    private void cmdContinuous_Click(System.Object eventSender, System.EventArgs eventArgs)
    {
        try
        {
            if (cmdContinuous.Text == "Iniciar monitoreo")
            {
                // Start doing periodic transfers.

                if (!(cmdOnce.Enabled))
                {
                    AccessForm("AddItemToListBox", "A previous transfer hasn't completed. Please try again.");
                }
                else
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    {
        cmdOnce.Enabled = false;

        //  Change the command button's text to "Cancel Continuous"

        cmdContinuous.Text = "Cancelar";

        //  Enable the timer event to trigger a set of transfers.

        tmrContinuousDataCollect.Enabled = true;
        tmrContinuousDataCollect.Start();
        ReadAndWriteToDevice();
    }

    // Inicia monitoreo (Plot)
    m_EnableMonitoring = true;
    APP_TIMER.Enabled = m_EnableMonitoring;
    RunningSetting();
}
else
{
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        //  Stop doing continuous transfers.
        //  Change the command button's text to "Iniciar monitoreo"

        cmdContinuous.Text = "Iniciar monitoreo";

        // D isable the timer that triggers the transfers.


        tmrContinuousDataCollect.Enabled = false;
        tmrContinuousDataCollect.Stop();

        cmdOnce.Enabled = true;

        // Detiene monitoreo (Plot)
        m_EnableMonitoring = false;
        APP_TIMER.Enabled = m_EnableMonitoring;
        StopSetting();
    }
}
catch (Exception ex)
{
    DisplayException(this.Name, ex);
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            throw;
        }
    }

    /// <summary>
    /// Scroll to the bottom of the list box and trim as needed.
    /// </summary>

    private void ScrollToBottomOfListBox()
    {
        try
        {
            Int32 count = 0;

            // If the list box is getting too large, trim its contents by removing the earliest data.
        }
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
            throw;
        }
    }
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        /// <summary>
        /// Retrieves a Vendor ID and Product ID in hexadecimal
        /// from the form's text boxes and converts the text to Int16s.
        /// </summary>
        ///
        /// <param name="myVendorID"> the Vendor ID</param>
        /// <param name="myProductID"> the Product ID</param>

        private void GetVendorAndProductIDsFromTextBoxes(ref Int32 myVendorID, ref Int32
myProductID)
        {
            try
            {
                myVendorID = Int32.Parse(txtVendorID.Text, NumberStyles.AllowHexSpecifier);
                myProductID = Int32.Parse(txtProductID.Text, NumberStyles.AllowHexSpecifier);
            }
            catch (Exception ex)
            {
                DisplayException(this.Name, ex);
                throw;
            }
        }
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
/// <summary>
/// Finds and displays the number of Input buffers
/// (the number of Input reports the host will store).
/// </summary>

private void GetInputReportBufferSize()
{
    Int32 numberOfInputBuffers = 0;
    Boolean success;

    try
    {
        // Get the number of input buffers.

        success = MyHid.GetNumberOfInputBuffers(hidHandle, ref
numberOfInputBuffers);

        // Display the result in the text box.

        txtInputReportBufferSize.Text = Convert.ToString(numberOfInputBuffers);
    }
    catch (Exception ex)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    {
        DisplayException(this.Name, ex);
        throw;
    }
}

/// <summary>
/// Retrieves Input report data and status information.
/// This routine is called automatically when myInputReport.Read
/// returns. Calls several marshaling routines to access the main form.
/// </summary>
///
/// <param name="ar"> an object containing status information about
/// the asynchronous operation. </param>

private void GetInputReportData(IAsyncResult ar)
{
    String byteValue = null;
    Int32 count = 0;
    Byte[] inputReportBuffer = null;
    Boolean success = false;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
try
{
    inputReportBuffer = (byte[])ar.AsyncState;

    fileStreamDeviceData.EndRead(ar);

    tmrReadTimeout.Stop();

    if ((ar.IsCompleted))
    {
        MyMarshalToForm("AddItemToListBox", "An Input report has been read.");
        MyMarshalToForm("AddItemToListBox", " Input Report ID: " +
String.Format("{0:X2} ", inputReportBuffer[0]));
        MyMarshalToForm("AddItemToListBox", " Input Report Data:");

        // Plots
        Int32 vADCValue = 0;
        vADCValue += Convert.ToInt32(inputReportBuffer[3]);
        vADCValue += Convert.ToInt32(inputReportBuffer[4]);
        vADCValue += Convert.ToInt32(inputReportBuffer[5]);
        vADCValue += Convert.ToInt32(inputReportBuffer[6]);
        vADCValue += Convert.ToInt32(inputReportBuffer[7]);
        switch (inputReportBuffer[8])
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        {
            case 0: m_ADCValue_Ch00 = vADCValue; break;  //variable del tipo double
            case 1: m_ADCValue_Ch01 = vADCValue; break;
            case 2: m_ADCValue_Ch02 = vADCValue; break;
            case 3: m_ADCValue_Ch03 = vADCValue; break;
            default: break;
        }

        for (count = 0; count <= inputReportBuffer.Length - 1; count++)
        {
            //  Display bytes as 2-character Hex strings.

            byteValue = String.Format("{0:X2} ", inputReportBuffer[count]);

            MyMarshalToForm("AddItemToListBox", " " + byteValue);
            MyMarshalToForm("TextBoxSelectionStart", txtBytesReceived.Text);
            MyMarshalToForm("AddItemToTextBox", byteValue);
        }
    }
    else
    {
        MyMarshalToForm("AddItemToListBox", "The attempt to read an Input report
has failed.");
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            }

            MyMarshalToForm("ScrollToBottomOfListBox", "");

            //  Enable requesting another transfer.

            MyMarshalToForm("EnableCmdOnce", "");
            transferInProgress = false;
        }
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
            throw;
        }
    }

    private void APP_MAIN_FormClosed(object sender, FormClosedEventArgs e)
    {
        try
        {
            Shutdown();
        }
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
        catch (Exception ex)
        {
            DisplayException(this.Name, ex);
            throw;
        }
    }

    /// <summary>
    /// Perform actions that must execute when the program ends.
    /// </summary>

    private void Shutdown()
    {
        try
        {
            CloseCommunications();

            // Stop receiving notifications.

MyDeviceManagement.StopReceivingDeviceNotifications(deviceNotificationHandle);
        }
        catch (Exception ex)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    {
        DisplayException(this.Name, ex);
        throw;
    }
}

/// <summary>
/// Search for a specific device.
/// </summary>

private void cmdFindDevice_Click(System.Object sender, System.EventArgs e)
{
    try
    {
        FindTheHid();
    }
    catch (Exception ex)
    {
        DisplayException(this.Name, ex);
        throw;
    }
}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
private void chkAutoincrement_CheckedChanged(object sender, EventArgs e)
{

}

private void cmdInputReportBufferSize_Click(object sender, EventArgs e)
{

}

private void InitGraphicPanel()
{
    APP_VIEWER_0.GraphPane.CurveList.Clear();
    myPanelCh00 = APP_VIEWER_0.GraphPane;
    myPanelCh00.XAxis.Scale.Min = 0;
    myPanelCh00.XAxis.Scale.Max = 200;
    myPanelCh00.YAxis.Scale.Min = 0;
    myPanelCh00.YAxis.Scale.Max = 40;
    myPanelCh00.YAxis.MajorGrid.IsVisible = true;
    myPanelCh00.YAxis.MinorGrid.IsVisible = true;
    myPanelCh00.XAxis.Title.Text = "Tiempo (Segundos)";
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
myPanelCh00.YAxis.Title.Text = "Temperatura °C";

myPanelCh00.Title.Text = " ";


APP_VIEWER_1.GraphPane.CurveList.Clear();

myPanelCh01 = APP_VIEWER_1.GraphPane;

myPanelCh01.XAxis.Scale.Min = 0;

myPanelCh01.XAxis.Scale.Max = 200;

myPanelCh01.YAxis.Scale.Min = -50;

myPanelCh01.YAxis.Scale.Max = 200;

myPanelCh01.YAxis.MajorGrid.IsVisible = true;

myPanelCh01.YAxis.MinorGrid.IsVisible = true;

myPanelCh01.XAxis.Title.Text = "Tiempo (Segundos)";

myPanelCh01.YAxis.Title.Text = "Presión kPa";

myPanelCh01.Title.Text = " ";


APP_VIEWER_2.GraphPane.CurveList.Clear();

myPanelCh02 = APP_VIEWER_2.GraphPane;

myPanelCh02.XAxis.Scale.Min = 0;

myPanelCh02.XAxis.Scale.Max = 200;

myPanelCh02.YAxis.Scale.Min = 0;

myPanelCh02.YAxis.Scale.Max = 20;

myPanelCh02.YAxis.MajorGrid.IsVisible = true;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
myPanelCh02.YAxis.MinorGrid.IsVisible = true;

myPanelCh02.XAxis.Title.Text = "Tiempo (mS)";

myPanelCh02.YAxis.Title.Text = "Pos. del Pedal";

myPanelCh02.Title.Text = " ";



for (int i = 0; i < x.Length; i++)

{

   x[i] = i;

   m_Table_Ch00[i] = 0;

   m_Table_Ch01[i] = 0;

   m_Table_Ch02[i] = 0;

}


// PointPairList holds the data for plotting, X and Y arrays

PointPairList splCh00 = new PointPairList(x, m_Table_Ch00);

PointPairList splCh01 = new PointPairList(x, m_Table_Ch01);

PointPairList splCh02 = new PointPairList(x, m_Table_Ch02);


// Add cruves to myPanelCh00 object

myCurveCh00 = myPanelCh00.AddCurve("ADC Value", splCh00, Color.Yellow,
SymbolType.None);

myCurveCh00.Clear();
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            myCurveCh00.Line.Width = 2.0F;


            myCurveCh01 = myPanelCh01.AddCurve("ADC Value", splCh01, Color.Yellow,
SymbolType.None);
            myCurveCh01.Clear();
            myCurveCh01.Line.Width = 2.0F;


            myCurveCh02 = myPanelCh02.AddCurve("ADC Value", splCh02, Color.Yellow,
SymbolType.None);
            myCurveCh02.Clear();
            myCurveCh02.Line.Width = 2.0F;


            APP_VIEWER_0.GraphPane.Chart.Fill.Type = ZedGraph.FillType.Solid;
            APP_VIEWER_0.GraphPane.Legend.Fill.Type = ZedGraph.FillType.Solid;


            APP_VIEWER_1.GraphPane.Chart.Fill.Type = ZedGraph.FillType.Solid;
            APP_VIEWER_1.GraphPane.Legend.Fill.Type = ZedGraph.FillType.Solid;


            APP_VIEWER_2.GraphPane.Chart.Fill.Type = ZedGraph.FillType.Solid;
            APP_VIEWER_2.GraphPane.Legend.Fill.Type = ZedGraph.FillType.Solid;


            StopSetting();
        }
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
private void Plot()
{
    int PWM_Width;
    int PWM_Periode = 60;
    int PWM_Freq;
    double PWM_Duty = 0;
    double Signal_0;
    double Signal_1;
    PWM_Freq = (int)(1500.00*m_ADCValue_Ch03/1024.00);
    if (PWM_Freq < 10)
        PWM_Freq = 10;

    PWM_Periode = (int)(20000.00 / PWM_Freq);

    PWM_Width = (int)((m_ADCValue_Ch02 * (double)PWM_Periode) / 1024.00);
    PWM_Duty = (m_ADCValue_Ch02 * 100) / 1024.00;
    Signal_0 = (m_ADCValue_Ch00 * 40.00) / 1024.00; // Canal 0, valor maximo 40,
1023 valor maximo que toma la variable
    Signal_1 = (m_ADCValue_Ch01 * 200.00) / 1024.00;

    double VoltageCH0;
    double VoltageCH1;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
VoltageCH0 = (m_ADCValue_Ch00 * 5.00) / 1023.00;

VoltageCH1 = (m_ADCValue_Ch01 * 5.00) / 1023.00;

if (VoltageCH1 > 2.00)

   Signal_1 = 151;

else

{

   if (VoltageCH1 < 0.90)

      Signal_1 = 0;

   else

   {

      Signal_1 = 151.00 * (VoltageCH1 - 0.90) / 1.1;

   }

}


VoltageCH0 = 4.585 - Signal_0 / 20.00;

LBL_CH0.Text = String.Format("{0:0.00} ", VoltageCH0) + "V";

LBL_CH1.Text = String.Format("{0:0.00} ", VoltageCH1) + "V";

LBL_CH2.Text = String.Format("{0:0.00} ", PWM_Duty) + "%";

LBL_CH3.Text = String.Format("{0:0.00} ", PWM_Freq) + "Hz";


if (VoltageCH0 > 4.8)

{
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    myPanelCh00.Title.Text = "FMI 03";
    System.Media.SystemSounds.Beep.Play();
}
else
{
    if (VoltageCH0 < 0.2)
    {
        myPanelCh00.Title.Text = "FMI 04";
        System.Media.SystemSounds.Beep.Play();
    }
    else
        myPanelCh00.Title.Text = " ";
}

if (VoltageCH1 > 4.8)
{
    myPanelCh01.Title.Text = "FMI 03";
    System.Media.SystemSounds.Beep.Play();
}
else
{
    if (VoltageCH1 < 0.2)
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    {
        myPanelCh01.Title.Text = "FMI 04";
        System.Media.SystemSounds.Beep.Play();
    }
    else
        myPanelCh01.Title.Text = " ";
}

if ((PWM_Duty > 95) || (PWM_Freq > 1000))
{
    myPanelCh02.Title.Text = "FMI08";
    System.Media.SystemSounds.Beep.Play();
}
else
{
    if ((PWM_Duty < 7) || (PWM_Freq < 150))
    {
        myPanelCh02.Title.Text = "FMI08";
        System.Media.SystemSounds.Beep.Play();
    }
    else
        myPanelCh02.Title.Text = " ";
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
}

myCurveCh02.Clear();

int vXIndex = 0;

while (vXIndex < x.Length)

{

    int pC = 0;

    while (pC < PWM_Periode)

    {

        if (PWM_Width > pC)

        {

            myCurveCh02.AddPoint(vXIndex, 8);

        }

        else

        {

            myCurveCh02.AddPoint(vXIndex, 0);

        }

        vXIndex++;

        if (vXIndex >= x.Length)

            break;

        pC++;

    }

}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
if (m_DataID > (x.Length - 1))
{
    IPointList plotPointsCh00;
    IPointList plotPointsCh01;
//   IPointList plotPointsCh02;

    plotPointsCh00 = myCurveCh00.Points;
    plotPointsCh01 = myCurveCh01.Points;
//    plotPointsCh02 = myCurveCh02.Points;
    for (int pX = 0; pX < (x.Length - 1); pX++)
    {
        plotPointsCh00[pX].Y = plotPointsCh00[pX + 1].Y;
        plotPointsCh01[pX].Y = plotPointsCh01[pX + 1].Y;
//        plotPointsCh02[pX].Y = plotPointsCh02[pX + 1].Y;
    }

    plotPointsCh00[x.Length - 1].Y = Signal_0;
    plotPointsCh01[x.Length - 1].Y = Signal_1;
//    plotPointsCh02[x.Length - 1].Y = m_ADCValue_Ch02;

    myCurveCh00.Points = plotPointsCh00;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    myCurveCh01.Points = plotPointsCh01;
//    myCurveCh02.Points = plotPointsCh02;
}
else
{
    myCurveCh00.AddPoint(m_DataID, Signal_0);
    myCurveCh01.AddPoint(m_DataID, Signal_1);
//    myCurveCh02.AddPoint(m_DataID, m_ADCValue_Ch02);
}

m_DataID++;

APP_VIEWER_0.AxisChange();
APP_VIEWER_0.Invalidate();
APP_VIEWER_0.Refresh();

APP_VIEWER_1.AxisChange();
APP_VIEWER_1.Invalidate();
APP_VIEWER_1.Refresh();

APP_VIEWER_2.AxisChange();
APP_VIEWER_2.Invalidate();
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    APP_VIEWER_2.Refresh();
}

private void APP_TIMER_Tick(object sender, EventArgs e)
{
    APP_TIMER.Enabled = false;
    try
    {
        //RefreshStatusParameters();
        Plot();
    }
    catch (Exception ex)
    {
        ;
    }
    APP_TIMER.Enabled = m_EnableMonitoring;
}

void RunningSetting()
{
    myCurveCh00.Color = Color.Yellow;
    myCurveCh01.Color = Color.Yellow;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
myCurveCh02.Color = Color.Yellow;


APP_VIEWER_0.GraphPane.Fill.Color = Color.Gray;

APP_VIEWER_0.GraphPane.Chart.Fill.Color = Color.Red;

APP_VIEWER_0.GraphPane.Legend.Fill.Color = Color.Blue;


APP_VIEWER_1.GraphPane.Fill.Color = Color.Gray;

APP_VIEWER_1.GraphPane.Chart.Fill.Color = Color.Red;

APP_VIEWER_1.GraphPane.Legend.Fill.Color = Color.Blue;


APP_VIEWER_2.GraphPane.Fill.Color = Color.Gray;

APP_VIEWER_2.GraphPane.Chart.Fill.Color = Color.Red;

APP_VIEWER_2.GraphPane.Legend.Fill.Color = Color.Blue;


APP_VIEWER_0.AxisChange();

APP_VIEWER_0.Invalidate();

APP_VIEWER_0.Refresh();


APP_VIEWER_1.AxisChange();

APP_VIEWER_1.Invalidate();

APP_VIEWER_1.Refresh();
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
    APP_VIEWER_2.AxisChange();
    APP_VIEWER_2.Invalidate();
    APP_VIEWER_2.Refresh();
}

void StopSetting()
{
    myCurveCh00.Color = Color.LightGray;
    myCurveCh01.Color = Color.LightGray;
    myCurveCh02.Color = Color.LightGray;

    APP_VIEWER_0.GraphPane.Fill.Color = Color.Gray;
    APP_VIEWER_0.GraphPane.Chart.Fill.Color = Color.LightGray;
    APP_VIEWER_0.GraphPane.Legend.Fill.Color = Color.Gray;

    APP_VIEWER_1.GraphPane.Fill.Color = Color.Gray;
    APP_VIEWER_1.GraphPane.Chart.Fill.Color = Color.LightGray;
    APP_VIEWER_1.GraphPane.Legend.Fill.Color = Color.Gray;

    APP_VIEWER_2.GraphPane.Fill.Color = Color.Gray;
    APP_VIEWER_2.GraphPane.Chart.Fill.Color = Color.LightGray;
    APP_VIEWER_2.GraphPane.Legend.Fill.Color = Color.Gray;
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

```
            APP_VIEWER_0.AxisChange();
            APP_VIEWER_0.Invalidate();
            APP_VIEWER_0.Refresh();

            APP_VIEWER_1.AxisChange();
            APP_VIEWER_1.Invalidate();
            APP_VIEWER_1.Refresh();

            APP_VIEWER_2.AxisChange();
            APP_VIEWER_2.Invalidate();
            APP_VIEWER_2.Refresh();
        }

        private void LBL_CH0_Click(object sender, EventArgs e)
        {

        }
    }
}
```


Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

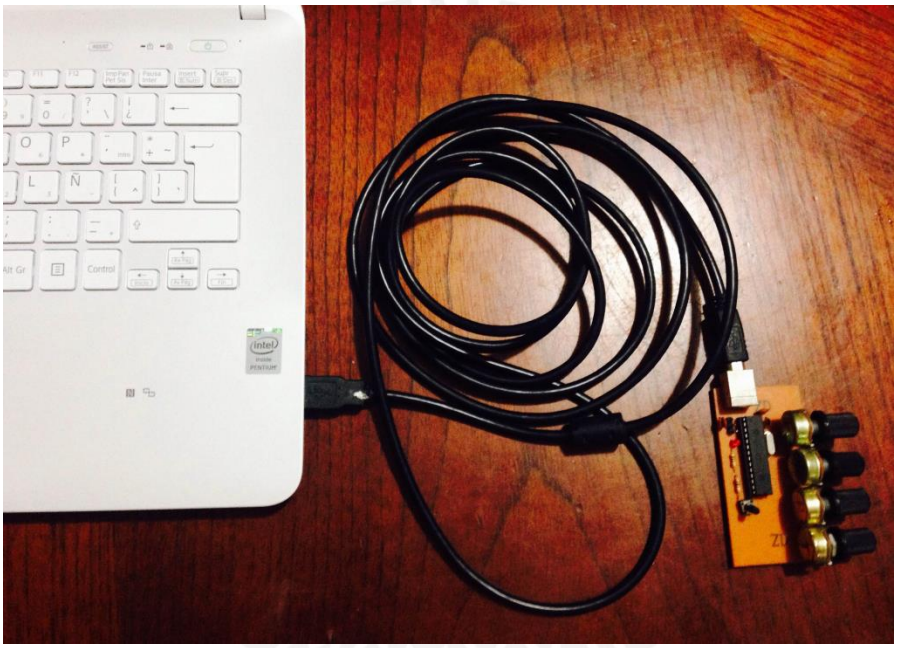
Anexo 3: **Manual de usuario - Simulador de Sensores de Maquinaria Caterpillar**

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

SIMULADOR DE SENSORES DE MAQUINARIA CATERPILLAR

Manual de Usuario



Material educativo dirigido a los instructores de cursos de Electricidad aplicada a Maquinaria Pesada Caterpillar

Tesis publicada con autorización del autor
No olvide citar esta tesis

Contenido

Tesis publicada con autorización del autor
No olvide citar esta tesis

Introducción

En este documento se describirá de forma clara el propósito del Sistema Simulador de Señales de Maquinaria Caterpillar y cómo el facilitador de las capacitaciones debe utilizar esta herramienta en los cursos que brinda a los técnicos mecánicos.

Para facilitar la comprensión del manual, se incluyen fotografías.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Descripción del Sistema Simulador de Señales de Maquinaria Caterpillar

El Sistema Simulador de Señales de Maquinaria Caterpillar ha sido diseñado e implementado con la finalidad de proporcionar una herramienta de enseñanza al facilitador de los cursos de electricidad.

El Sistema permite a los participantes de los cursos observar la forma de onda de los principales sensores que se encuentran en los motores electrónicos: sensor de temperatura del aire de admisión, sensor de presión del aire de admisión y sensor de posición del pedal.

El sistema Simulador consta de una tarjeta electrónica, un conector USB para impresora y un CD que contiene la aplicación.

Fig 1: Tarjeta Electrónica del Sistema Simulador de Señales

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

Fig 2: Cable USB de impresora

Tesis publicada con autorización del autor
No olvide citar esta tesis

Instalación y uso del Sistema Simulador de Señales

A continuación se describen los pasos a seguir para utilizar el Sistema Simulador en las capacitaciones:

1. Conectar la tarjeta electrónica a una computadora utilizando el cable USB de impresora. El conector USB debe ir para el lado de la computadora.

Fig 3: Conexión entre la tarjeta Simuladora de Señales y computadora mediante el uso de un cable USB de impresora

2. Colocar el CD que contiene la aplicación y dar doble click al icono del programa con nombre USB_SCOPE; a continuación aparecerá la ventana de inicio del software. Para iniciar la visualización de las señales dar click en el recuadro “Iniciar Monitoreo”.

Tesis publicada con autorización del autor
No olvide citar esta tesis

Fig 4: Ventana de inicio del software Simulador de señales de sensores

3. Para variar la forma de onda de las señales simuladas se debe girar el potenciómetro del sensor que se desea observar. La tarjeta simuladora tiene 4 potenciómetros:

- 1 potenciómetro para la señal de temperatura del aire de admisión
- 1 potenciómetro para la señal de presión del aire de admisión
- 1 potenciómetro para la frecuencia y 1 potenciómetro para el ciclo de trabajo de la señal de posición de pedal.

Tesis publicada con autorización del autor
No olvide citar esta tesis

TESIS PUCP

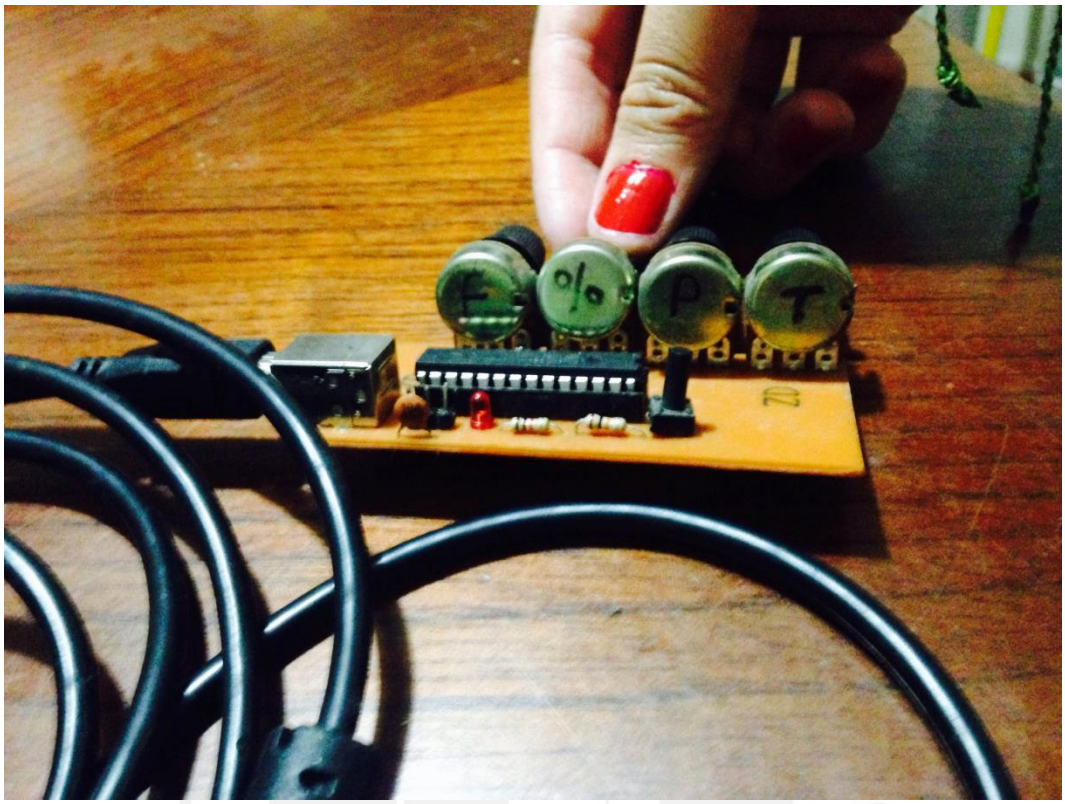


Fig 5: Tarjeta Simuladora de Señales con 4 potenciómetros que simulan tres sensores comunes de los motores Caterpillar

Tesis publicada con autorización del autor
No olvide citar esta tesis

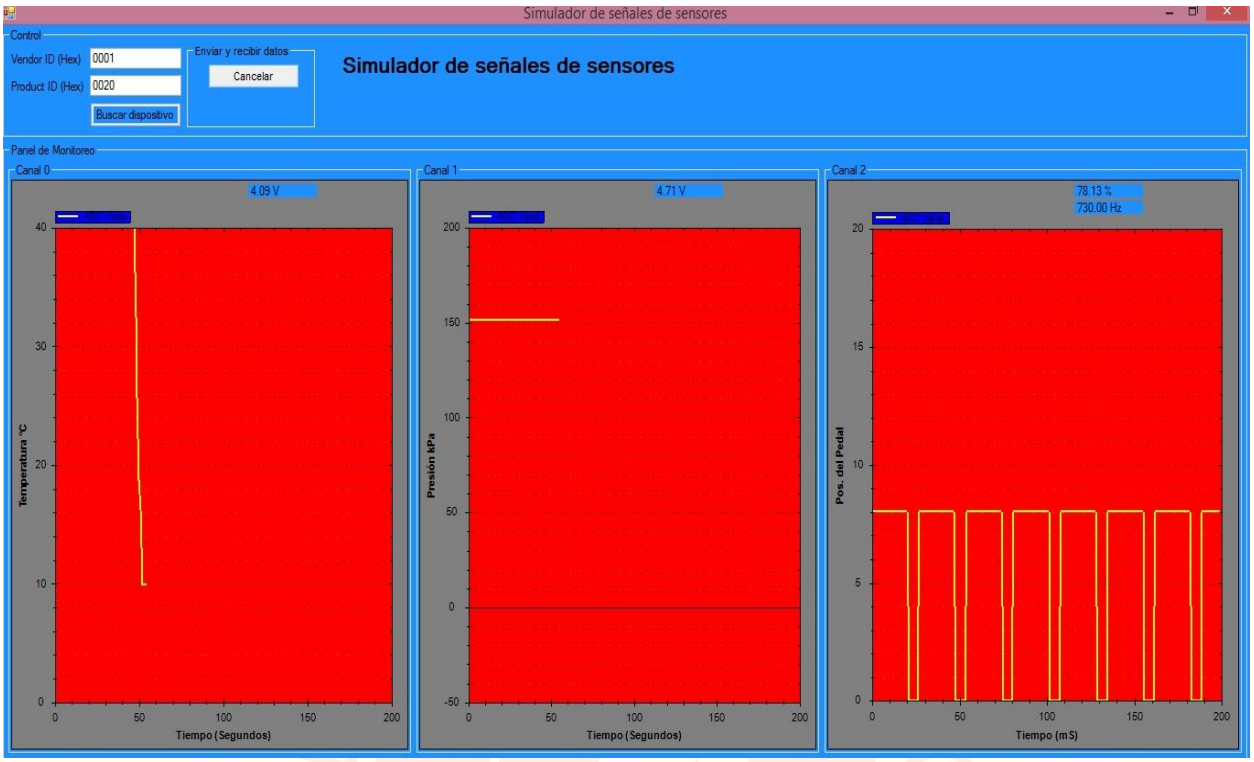


Fig 6: Pantalla de monitoreo de las señales simuladas por la tarjeta electrónica

4. Cuando se ha girado el potenciómetro hasta obtener un valor de falla (ejemplo 5,00 voltios) aparecerá en la ventana el nombre de la falla producida.

Tesis publicada con autorización del autor
No olvide citar esta tesis

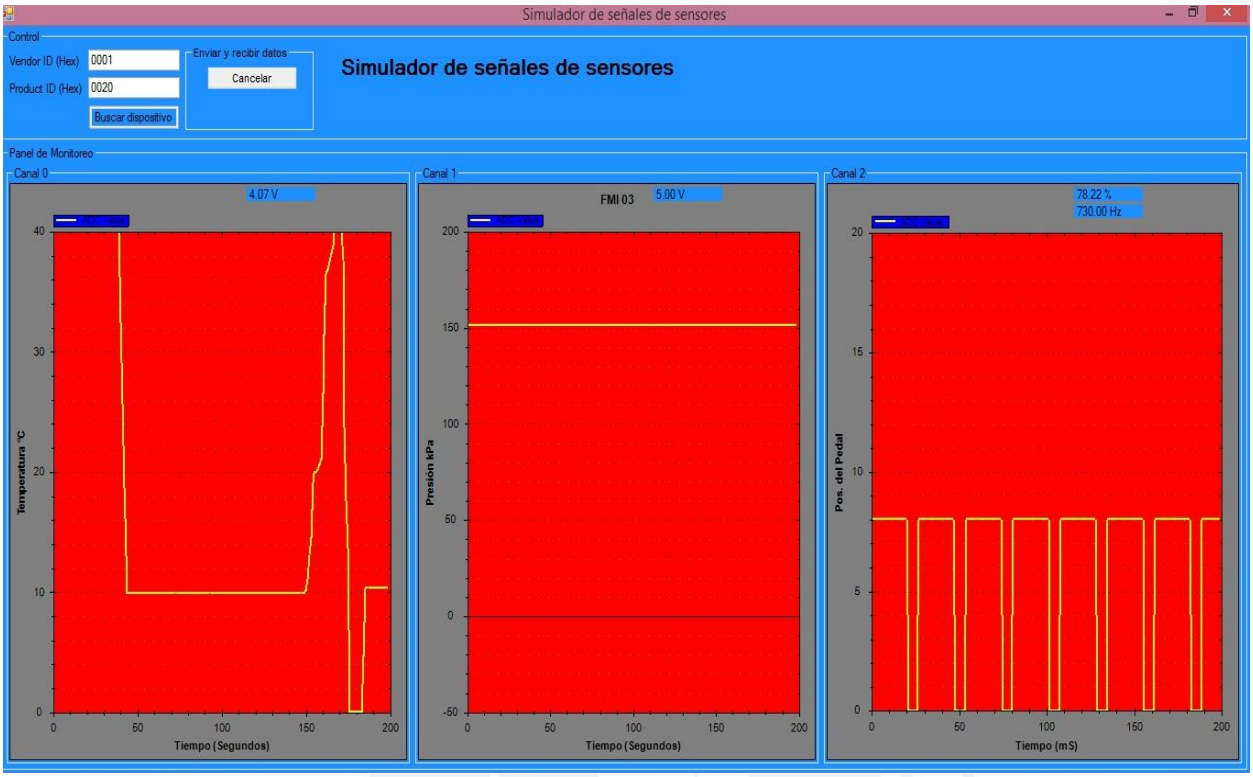


Fig 7: Falla en la señal de Presión, se muestra el FMI 03

Tesis publicada con autorización del autor
No olvide citar esta tesis