Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería Electrónica

DISEÑO E IMPLEMENTACION DE UN EQUIPO PORTATIL REGISTRADOR DE VIBRACIONES DE MOTORES ELECTRICOS

Tesis presentada por la Bachiller:

Ticona Saavedra, Juan Carlos

para optar el Título Profesional de

Ingeniero Electrónico con especialidad en automatización y control

Asesor:

Ing. Zegarra Gago, Henry Christian

Arequipa- Perú

2021

DICTAMEN APROBATORIO DE BORRADOR DE TESIS

UCSM-ERP

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

INGENIERIA ELECTRONICA

TITULACIÓN CON TESIS

DICTAMEN APROBACIÓN DE BORRADOR

Arequipa, 15 de Julio del 2021

Dictamen: 001122-C-EPIE-2021

Visto el borrador del expediente 001122, presentado por:

2013600931 - TICONA SAAVEDRA JUAN CARLOS

Titulado:

DISEÑO E IMPLEMENTACION DE UN EQUIPO PORTÁTIL REGISTRADOR DE VIBRACIONES DE MOTORES ELÉCTRICOS

Nuestro dictamen es:

APROBADO

1475 - MALAGA CHAVEZ CESAR EDUARDO
DICTAMINADOR

1767 - SULLA TORRES RAUL RICARDO
DICTAMINADOR

2465 - ZEGARRA GAGO HENRY CHRISTIAN
DICTAMINADOR

DEDICATORIA

Esta tesis está dedicada a mis padres a quienes con su amor, paciencia y esfuerzo me han permitido llegar a cumplir hoy, un logro más.

AGRADECIMIENTO

A Dios.

A mis padres Juan y María, por inculcar en mí, el ejemplo del esfuerzo y que a la vez me enseñaron que el mejor conocimiento que se puede tener es el que se aprende por sí mismo.

RESUMEN

Las vibraciones en la actualidad están aportando mucho en el mantenimiento predictivo, ya que mediante la toma de datos y el debido procesamiento se puede diagnosticar el fallo en un motor eléctrico y mediante datos estadísticos elaborar una tendencia de fallo. Todo esto permite tener una planificación de manteamiento más certero y disminuir los costos de mantenimiento.

Diagnosticar el fallo de un motor es importante ya que nos permite saber cuál de sus elementos estacionarios y rotativos están averiados, una de las técnicas más comunes en ser usada es el análisis de vibraciones, que nos muestra en el dominio de la frecuencia varios espectros correspondientes a las vibraciones producidas por el motor eléctrico.
En la presente tesis se describe el "Diseño e implementación de un equipo portátil registrador de vibraciones de motores eléctricos", este equipo cuenta con un acelerómetro que nos permite sensar los datos de aceleración del motor eléctrico, para luego ser procesada por varios filtros y aplicar la transformada rápida de Fourier para obtener la firma de vibración, posteriormente se visualiza el espectro para luego ser almacenada en una memoria SD extraíble.

Con este dispositivo el usuario es capaz de diagnosticar fallas como el desequilibrio, holguras mecánicas, desalineamiento y fallos en los cojinetes. Cabe mencionar que el análisis de estas vibraciones es más profundo ya que se tiene que evaluar las diferentes vibraciones producidas por el motor eléctrico y puede que se encuentren muchas más fallas.

PALABRAS CLAVES:
Motor eléctrico, vibraciones, transformada rápida de Fourier, espectros, desequilibrio, holguras mecánicas, desalineamiento y fallo en los cojinetes.

ABSTRACT

Vibrations are currently contributing a lot to predictive maintenance, since data collection and proper processing can diagnose the failure in an electric motor and using statistical data to develop a failure trend. All of this allows for more accurate maintenance planning and lower maintenance costs.

Diagnosing the failure of an engine is important since it allows us to know which of its stationary and rotating elements are damaged, one of the most common techniques to be used is vibration analysis, which shows us in the frequency domain several corresponding spectra to the vibrations produced by the electric motor.

This thesis describes the "Design and implementation of a portable vibration recorder for electric motors". This equipment has an accelerometer that allows us to sense the acceleration data of the electric motor, to then be processed by various filters and apply the fast Fourier transform to obtain the vibration signature; later the spectrum is visualized and then stored in a removable SD memory.

With this device the user is able to diagnose faults such as imbalance, mechanical backlash, misalignment and bearing failures. It is worth mentioning that the analysis of these vibrations is more in-depth since the different vibrations produced by the electric motor have to be evaluated and many more faults may be found.

KEY WORDS:

Electric motor, vibrations, fast fourier transform, spectrum, unbalance, mechanical backlash, misalignment and bearing failure.

INTRODUCCIÓN

Esta tesis consta del diseño e implementación de un equipo portátil registrador de vibraciones de motores eléctricos, con el fin de que el usuario pueda diagnosticar la falla del motor eléctrico. Los datos que captura el equipo son almacenados en una memoria extraíble. Si bien es cierto que en los últimos años la aplicación de los motores eléctricos en diferentes tipos de industria se ha incrementado, por ende el mantenimiento de los motores está evolucionando todos los días, implementado diferentes instrumentos electrónicos de medición, uno de ellos que es muy importante es el analizador de vibraciones que tiene como aplicación en motores trifásicos y monofásicos basados en el análisis del espectro de la señal de vibración. Con este instrumento electrónico se puede identificar el problema del motor, ya que se tiene un espectro característico para cada falla y así luego corregir el problema, también se puede usar para anticipar la falla, conocido también como mantenimiento predictivo. Con esta última descripción las empresas pueden tener el control de las máquinas y de los programas de mantenimiento, logrando que los gastos se reduzcan para refacciones y mano de obra. El equipo que se describe en esta tesis es de bajo costo a comparación de los que se encuentran en el mercado. Este equipo cuenta con componentes electrónicos del mercado local y de fácil acceso a ellos.

Esta tesis consta de seis capítulos, el primer capítulo es el marco teórico, el segundo capítulo son las especificaciones técnicas del equipo, el tercer capítulo describe el proceso de diseño del hardware y software, el cuarto capítulo presenta las pruebas y resultados del equipo.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

CAPITULO I

1. PLANTEAMIENTO TEORICO

1.1 DETERMINACION DEL PROBLEMA

Actualmente en el mercado se vienen presentando dos tipos de equipos de analizadores de vibración, algunos que funcionan midiendo directamente la velocidad, mientras que otros más sofisticados y costoso presentan un análisis triaxial con pantallas donde se visualizan las firmas de vibración.

Dichos equipos mencionados son costosos a comparación de un multímetro u otros equipos, lo que lleva a la pequeña industria a reconsiderar la compra de estos equipos.

Por eso se diseñara e implementara un equipo registrador de vibraciones, que permita un análisis en el lugar de trabajo, para que el operador pueda visualizar la firma de vibración del motor y compararlo con la norma 10816 de severidad de vibraciones y espectros característicos de las posibles fallas mecánicas.

1.2 DESCRIPCION DEL PROBLEMA

El principal problema es la complejidad de la señal de vibración, dado que esta esta se compone por una suma indeterminada de componentes generadas por cada parte mecánica del motor. Por ende es necesario tener un transductor de alta definición para que pueda adquirir las señales más débiles de la vibración del motor, también se requiere de un controlador con alta capacidad de memoria para almacenar la forma de onda discreta, a esto también se suma la capacidad de cálculo del controlador que para el típico análisis en tiempo real, se requiere una potencia de cálculo similar a la de un FPGA (circuitos integrados reconfigurables compuestos de interconexiones programables que combinan bloques lógicos programables).

Pero utilizar un FPGA como fuente de análisis matemático, se estaría elevando el costo de forma excesiva del equipo.

1.3 JUSTIFICACION

En la actualidad todas las industrias, requieren de motores eléctricos, por lo tanto, se requiere de un equipo que permita registrar los espectros producidos por las vibraciones que son consecuentes de alguna falla de algún motor, este equipo puede ser utilizado para realizar un mantenimiento predictivo, ya que con este equipo podemos visualizar el espectro y luego analizar qué tan grave es la vibración. Cabe mencionar que el operador encargado en diagnosticar debe tener certificación de categoría 2.

1.4 OBJETIVOS

1.4.1 Generales

- Diseñar e implementar un equipo portátil registrador de vibraciones de motores eléctricos.

1.4.2 Específicos

- Seleccionar los componentes adecuados para el diseño del equipo.
- Diseñar una placa impresa para el acondicionamiento de la señal de vibración.
- Aplicar la transformada rápida de Fourier con el fin de analizar los diferentes espectros y a aplicación de filtros digitales para obtener una señal de salida óptima.

1.5 APORTACIONES

En esta tesis se está desarrollando un equipo portátil registrador de vibraciones de motores eléctricos en donde se da las siguientes aportaciones en la ingeniería peruana.

- Este equipo es implementado con un bajo costo a diferencia de los equipos que se encuentran en el mercado actualmente.
- Aplica principalmente el procesamiento digital de señales para:
  - ❖ Ejecución del valor cuadrático medio.
  - ❖ Análisis en el dominio de frecuencia.

- ❖ Ejecución del teorema de Nyquist para el muestreo de la señal de vibración.
- ❖ Ejecución de la transformada rápida de Fourier para el cálculo de magnitud y fase del espectro de la señal de vibración.
- ❖ Ejecución de filtros digitales.

- Graficas utilizando pantallas TFT.
  - ❖ Graficas en los ejes X y Y.
  - ❖ Control de la pantalla TFT.

- Desarrollo con el microcontrolador ARM CORTEX.

CAPITULO II

2. MARCO TEORICO

2.1 MOTORES DE INDUCCIÓN

Los motores de inducción son importantes en la industria ya que son construidos de una manera fácil y económica, estos motores tienen muchas aplicaciones por lo que en la actualidad es necesario realizar un mantenimiento adecuado para su preservación. Estos motores constan de un rotor, estator, carcasa y rodamientos, esto se aprecia en la figura 1.

En Chapman (2012) indica que la corriente que se induce en las bobinas del motor genera un campo electromagnético pudiendo realizar así el giro del rotor.

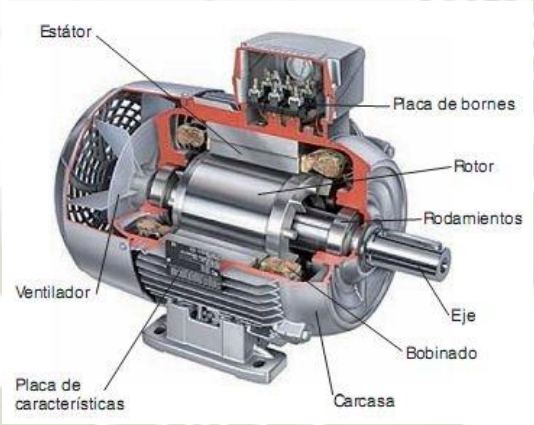


Figura 1: Partes de un motor de inducción

Fuente:https://sites.google.com/site/279motoreselectricos/partes-fundamentales-de-un-motor-electrico, (2011)

2.2 MICROCONTROLADOR RM46L852 – TEXAS INSTRUMENTS

En la actualidad existen varios tipos de microcontroladores, de diferentes marcas, características, precios, etc. En esta tesis la parte fundamental es el microcontrolador ya que con este podemos programar las diferentes tareas que implican las vibraciones producidas por un motor eléctrico de inducción. Este microcontrolador ya que posee las características que implican realizar cálculos pesados como la transformada rápida de

Fourier. Como se menciona en Texas Instruments (2018) este microcontrolador posee funciones de reloj, reinicio, conversores ADC, protocolo de comunicación I2C, entradas analógicas y digitales, configuración de la señal PWM, etc. También posee una, dos interfaces con el usuario, una de ellas permite la configuración de los módulos internos del microcontrolador y la otra es para la programación. Se muestra en la figura 2 el microcontrolador RM 46l852.

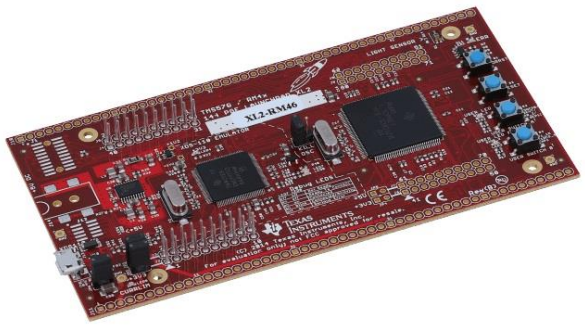


Figura 2: Microcontrolador RM 46l852

Fuente:https://www.ti.com/tool/LAUNCHXL2-RM46#tech-docs, (2021)

2.3 PLACAS IMPRESAS

Las placas impresas son fabricadas con el fin de mantener y conectar los componentes electrónicos, llevando a ser grabadas con rutas o pistas de material conductor, en Martinez (2015) se menciona que las placas impresas fueron creadas por un ingeniero austriaco Paul Eisler en 1936. En la actualidad son muy populares en los equipos electrónicos. Además, el diseño de placas electrónicas está bajo la norma IPC 2221. A continuación, se muestra en la figura 3 una placa impresa de ejemplo.

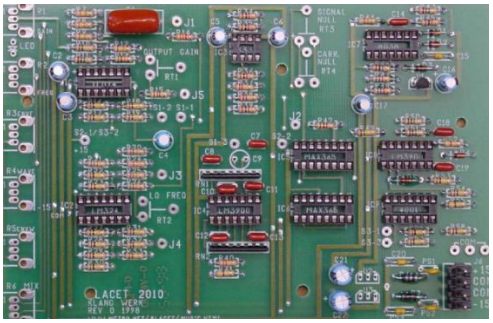


Figura 3: Placa Impresa ejemplo

Fuente: http://umh1783.edu.umh.es/wpcontent/uploads/sites/934/2013/02/Tema-2.pdf, (2013)

2.4 FALLAS COMUNES DE MOTORES DE INDUCCIÓN

2.4.1 Desequilibrio

El desequilibrio es un problema muy común en los motores eléctricos, ya que esto se produce a partir de una mala distribución de todas las partes que están sometidas a la carga del motor. Este problema conlleva a que se produzca una fuerza centrífuga debido a que el centro de las masas no coindice con el eje que produce el centro de giro del motor, esto produce una fuerza centrífuga.

En Power MI Cloud Condition Monitoring (2018) se explica como el desequilibrio producido se puede cuantificar en peso y distancia, en conclusión, a mayor fuerza centrífuga mayor será el desequilibrio. A continuación, se muestra la figura 4, que explica la ecuación de la fuerza centrífuga:

M=Masa de desequilibrio.
D=Radio de desequilibrio.
W=Velocidad angular.
F= Fuerza centrífuga.

$$F = MxDxW^2$$

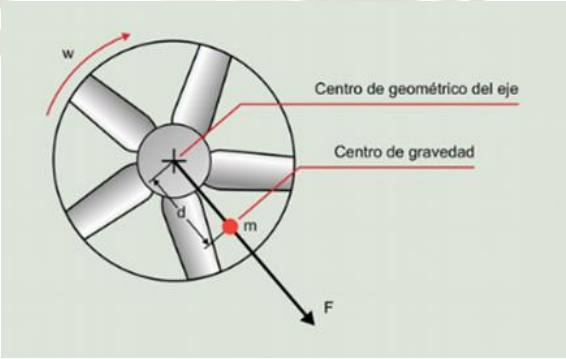


Figura 4: Fuerza centrífuga asociada a un motor desequilibrado
Fuente: (Power MI Cloud Condition Monitoring, 2018)

En Power MI Cloud Condition Monitoring (2018) muestra las diferentes causas que produce el desequilibrio, que se muestra a continuación.

- Aglomeración desigual de polvo en las palas de un ventilador.
- Erosión y corrosión desigual de los alabes de una bomba.
- Falta de homogeneidad en partes coladas, como burbujas, agujeros de soplado y partes porosas.
- Excentricidad del rotor.
- Distribución desigual en las barras del rotor de motores eléctricos o en el bobinado.
- Flexión de rodillo, especialmente en máquinas de la industria papelera.
- Pesos de equilibrado que faltan.
- Eje flexionado.
- Excentricidad.

En la figura 5 se muestra el espectro característico del desequilibrio.

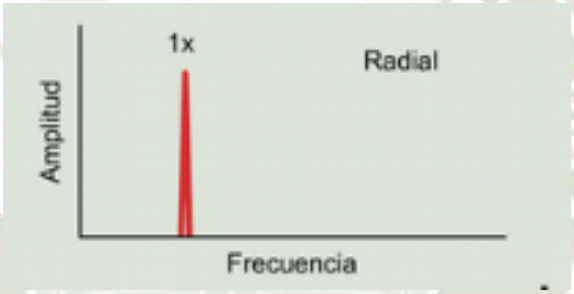


Figura 5: Espectro característico del desequilibrio

Fuente: (Power MI Cloud Condition Monitoring, 2018)

De esta figura se puede concluir que:

- La amplitud de la vibración es directamente proporcional a la cantidad de desequilibrio.
- Vibración radial en 1XRPM.
- Diferencia de fase entre la dirección horizontal y vertical de un cojinete de aproximadamente 90 grados, permitiendo una variación aceptable de ± 30 grados.
- No existen diferencias de fase significativas en las lecturas de fase entre ambos lados del eje en las mismas direcciones radiales. (único plano).
- La lectura de fase radial nos indicara que ambos lados del eje tienen un desfase de 180 grados (dos planos).

2.4.2 Holguras

Las holguras se producen por varias causas que se originan generalmente en los elementos rotativos del motor, en el anclaje del motor y el desgaste de algunas piezas del motor.

Por otro lado cabe mencionar que las holguras presenten varios armónicos que vienen hacer de la velocidad de giro del motor, estos armónicos presentan amplitudes 1x, 2x, 3x,..., amplitudes medios 1.5x, 2.5x, 3.5x… también sub armónicos 0.5x como indica en Power MI Cloud Condition Monitoring (2018). Es recomendable analizar estos armónicos en frecuencia ya que en el tiempo no presenta algún patrón para analizar es decir es totalmente errónea.

A continuación, se clasifican las holguras en tres tipos que son:

- **Holgura Tipo A o Holgura de flexibilidad estructural**

Se debe a una flexibilidad estructural de las base de las máquinas de la placa base o cimiento, también se debe a un mortero deteriorado, pernos de sujeción sueltos en la base y la distorsión del armazón o de la base, la diferencia de fase puede revelar una diferencia de fase aproximadamente 90 y 180 grados entre las lecturas verticales en el perno en la base de la máquina, en la placa base y en la base en sí, como indica Technical Associates of Charlotte (1996).

En la figura 6 se muestra el espectro característico de la holgura tipo A.

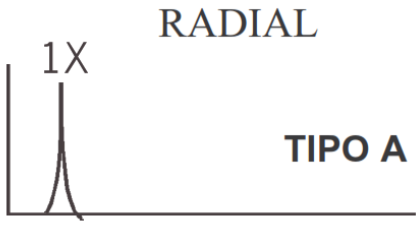


Figura 6: Espectro característico de la holgura tipo A
Fuente: (Technical Associates of Charlotte, 1996)

- **Holgura Tipo B o soltura en el pedestal**

Generalmente se debe a pernos flojos de soporte flojos, a fracturas en la estructura del armazón o en el pedestal del rodamiento según indica Technical Associates of Charlotte (1996).

En la figura 7 se muestra el espectro característico de la holgura tipo B.

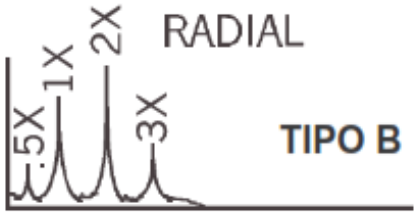


Figura 7: Espectro característico de la holgura tipo B"
Fuente: (Technical Associates of Charlotte, 1996)

- **Holgura Tipo C o Holgura rotacional**

Normalmente se genera a causa de un ajuste inadecuado entre las partes componentes originados muchas armónicas debido a la respuesta no lineal de las partes sueltas a las fuerzas dinámicas del rotor. Causa un truncamiento de la forma de onda y un piso de ruido mayor en el espectro. Con frecuencia el tipo C se debe a que el aro exterior del rodamiento este flojo en su tapa, a un rodamiento suelto y queda vueltas en su eje, a un claro excesivo en cojinetes planos y rodamientos, o por un impulsor suelto en su eje, etc. Con frecuencia la fase del tipo C es inestable y puede variar entre lectura y lectura, sobre todo si el rotor cambia de posición en el eje de un arranque a otro. A menudo, la holgura mecánica es altamente direccional y puede provocar lecturas notablemente diferentes si se comparan los niveles en incrementos de 30 grados en dirección radial en toda la caja del rodamiento. En Power MI Cloud Condition Monitoring (2018) menciona que se tenga en cuenta que la holgura con frecuencia provoca múltiples sub armónicas a exactamente ½x o 1/3x (0.5x, 1.5x, 2.5x, etc.).

En la figura 8 se muestra el espectro característico de la holgura tipo C.

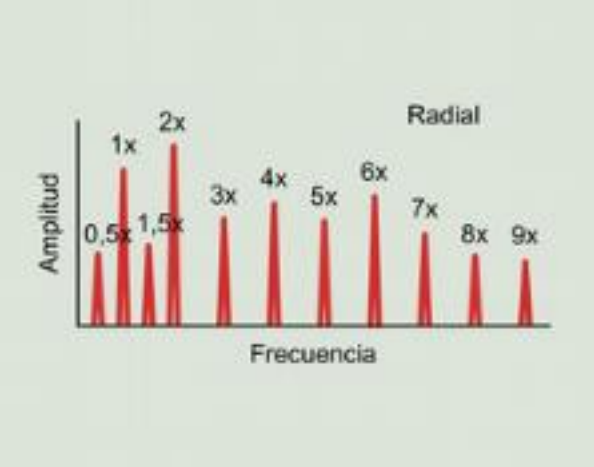


Figura 8: Espectro característico de la holgura tipo C
Fuente: (Power MI Cloud Condition Monitoring, 2018)

2.4.3 Desalineación

No cabe duda que la desalineación es un problema muy común en las maquinas rotativas, ya que presenta una desalineación en dos rotores. Esta desalineación es producida por colocar acoplamientos elásticos y rodamientos autoalineables, haciendo que tengan un desgaste y luego se produzca dicha desalineación.

La desalineación produce una fuerte vibración en las direcciones axiales y radiales, las lecturas axiales presentan armónicos de la velocidad de giro 1x, 2x, 3x RPM, y las lecturas radiales se presentan en 1x, 2x RPM. En Power MI Cloud Condition Monitoring (2018) menciona que la vibración es abruptamente alta tiene lugar en 1x y puede ser confundida con desequilibrio.

A continuación, se presentan tipos de desalineación que se producen en motores eléctricos.

- **Desalineación angular:**

Esta desalineación se presenta cuando las líneas centrales de los ejes forman un ángulo, la vibración produce 1x RPM en la dirección axial y a veces con armónicos en 2x y 3x, en el armónico 2x presenta una característica importante ya que puede tener la misma amplitud o superarla a 1x, según menciona Power MI Cloud Condition Monitoring (2018).

Las medidas de fase axial en ambos lados del acoplamiento se encuentran desfasadas 180 grados.

En la figura 9 se muestra el espectro característico de la desalineación angular.

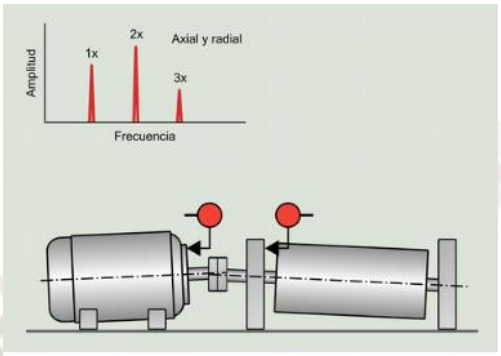


Figura 9: Espectros característicos de la desalineación angular
Fuente: (Power MI Cloud Condition Monitoring, 2018)

- **Desalineación paralela:**

Se dice que la desalineación paralela es cuando dos ejes son paralelos y están separados a una determinada distancia. En su espectro característico presenta en dirección radial una fuerte vibración en 1x RPM con armónicos en 2x y 3x RPM. Como se menciona en Power MI Cloud Condition Monitoring (2018) la dirección angular también puede igualar o incluso superar en amplitud el armónico 2x RPM y la medida de la fase radial a ambos lados del acoplamiento se encuentran desfasadas en 180 grados.

En la figura 10 se muestra el espectro característico de la desalineación paralela.

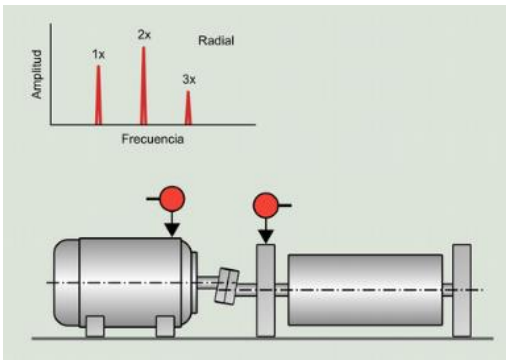


Figura 10: Espectro característico de la desalineación paralela
Fuente: (Power MI Cloud Condition Monitoring, 2018)

2.4.4 Fallo en cojinetes

Los cojinetes están conformados por chumaceras y rodamientos, en esta parte solo hablaremos de rodamientos de bolas ya que son muy comunes en la industria. En Power MI Cloud Condition Monitoring (2018) menciona que los rodamientos son muy importantes ya que reducen la fricción entre el eje y las piezas que están conectas para que el eje tenga más libertad al momento de girar.

Las causas más comunes por las que se malogra un rodamiento son las siguientes:

- Colocación defectuosa en alojamientos y flechas.
- Desalineamiento y desbalance.
- Técnicas de montaje inapropiadas.
- Ajustes inapropiados en flechas y alojamientos.
- Lubricación inadecuada.
- Manejo inadecuado de la maquina durante el transporte o la instalación.
- Paso de corriente eléctrica a través del rodamiento.

Para identificar las fallas en el rodamiento, se realiza mediante tonos de rodamiento que son las siguientes:

- BPFO: En Power MI Cloud Condition Monitoring (2018) indica que la frecuencia de deterioro de la pista exterior, son el número de bolas que pasan por un punto de la pista exterior, cada vez que la flecha de un giro completo.

- BPFI: En Power MI Cloud Condition Monitoring (2018) menciona que la frecuencia de deterioro de la pista interior, son el número de bolas que pasan por un punto de la pista interior, cada vez que la flecha de un giro completo.

- FTF: En Power MI Cloud Condition Monitoring (2018) indica que frecuencia de deterioro de la jaula, corresponde físicamente con el número de giros que realiza la jaula del rodamiento cada vez que el eje realiza un giro completo.

- BSF: En Power MI Cloud Condition Monitoring (2018) menciona que la frecuencia de deterioro de los elementos rodantes o bolas, corresponde físicamente con el número de giros que realiza una bola del rodamiento cada vez que el eje realiza un giro completo.

A continuación, se muestra en la figura 11 las fórmulas que acompañan a las definiciones ya comentadas líneas arriba.

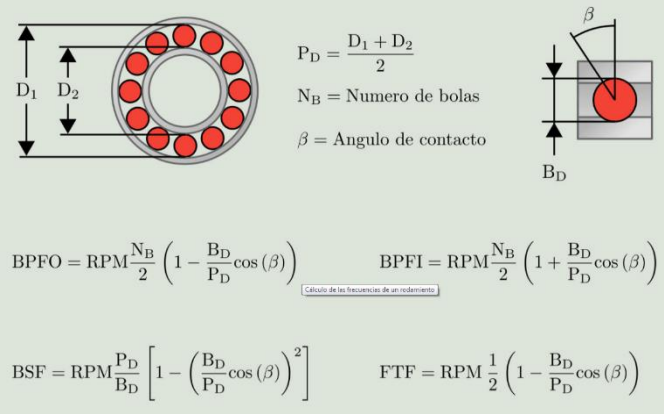


Figura 11: Cálculo de frecuencias de los tonos de rodamientos

Fuente: (Power MI Cloud Condition Monitoring, 2018)

Estas fórmulas son aplicables si se tiene todos los datos, pero generalmente todas estas frecuencias el fabricante de rodamientos las brinda, siempre y cuando se tenga el modelo y marca del rodamiento. En la figura 12 se describe las características de los rodamientos.

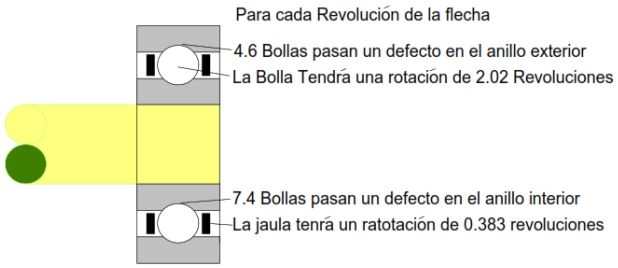


Figura 12: Aproximación de frecuencias de tonos de rodamientos más comunes

Fuente: (White, 1990)

De la figura 12 se ilustra una aproximación de frecuencias de tonos de rodamientos para los rodamientos más comunes que se encuentran en industria. Consecuentemente se

generan espectros característicos de las fallas y desgaste de los rodamientos que se muestra a continuación en las figuras 13, 14, 15 y 16.

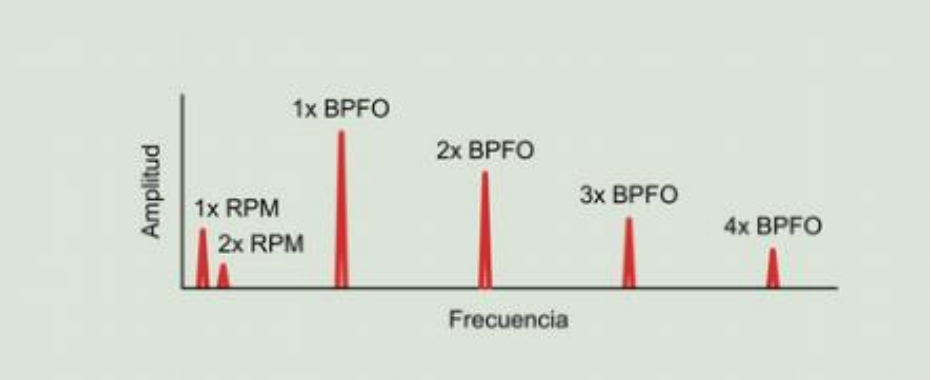


Figura 13: Espectro característico del fallo en la pista exterior de un rodamiento

Fuente: (Power MI Cloud Condition Monitoring, 2018)

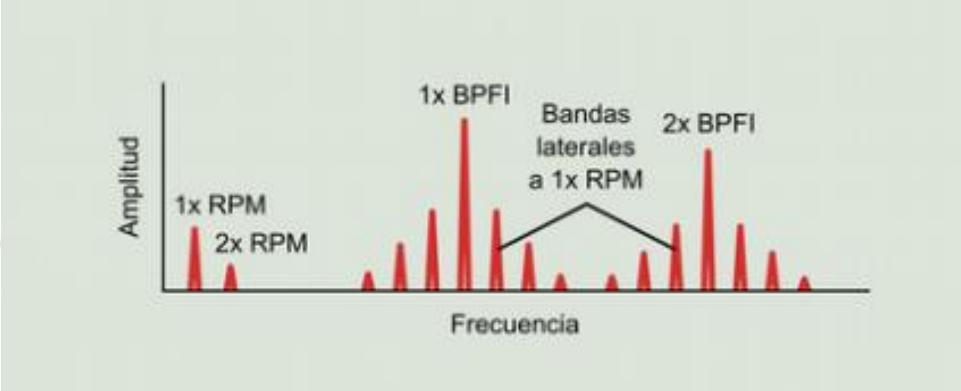


Figura 14: Espectro característico del fallo en la pista interior de un rodamiento

Fuente: (Power MI Cloud Condition Monitoring, 2018)

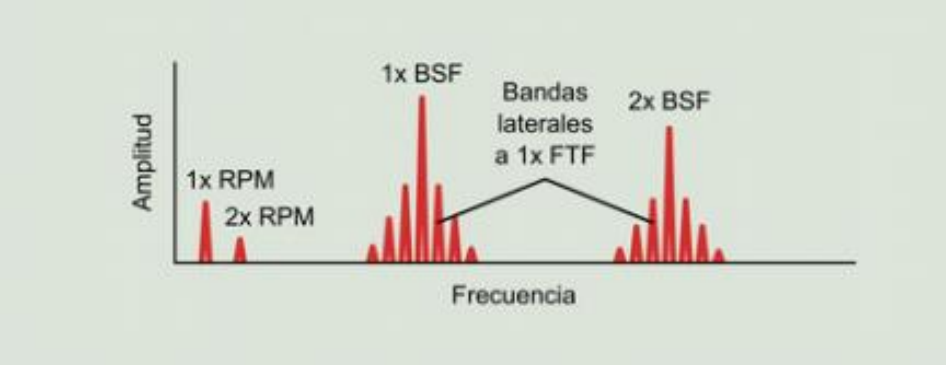


Figura 15: Espectro característico del fallo de un elemento rodante o bola del rodamiento

Fuente: (Power MI Cloud Condition Monitoring, 2018)

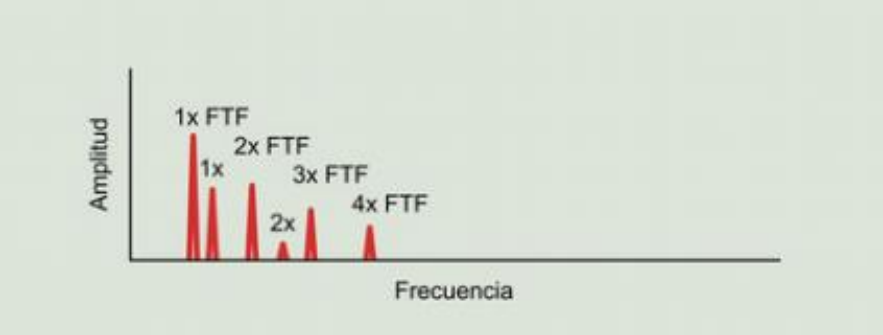


Figura 16: Espectro característico del fallo en la jaula de un rodamiento

Fuente: (Power MI Cloud Condition Monitoring, 2018)

2.5 UNIDADES DE VIBRACIÓN

2.5.1 Frecuencia

La frecuencia es básicamente la cantidad de ciclos que pasan en un solo segundo, la unidad de medida es el Hz. Esta medida fue creada por Heinrich Hertz, quien tuvo aportes en las investigaciones de las ondas de radio.

2.5.2 Valor RMS

El valor RMS es la medida más relevante de la amplitud porque tiene en cuenta la historia de la señal y proporciona un valor que está directamente relacionado con el contenido energético de la vibración. En la programación el algoritmo RMS se puede extraer de las librerías DSP, este algoritmo tiene alta precisión y velocidad cuando se calcula el valor RMS, cabe mencionar que trabaja internamente con un solo punto flotante. Este valor es requerido que se muestre en el equipo, ya que según la norma ISO 10816, recomienda evaluar los resultados en valores RMS en velocidad.

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_N^2}{N}}$$

2.6 UNIDADES DE AMPLITUD

Para las vibraciones mecánicas se considera tres variables, ya que en todo movimiento vibratorio se encuentra el desplazamiento, la velocidad y la aceleración. Al integrar la señal de vibración que nos da el acelerómetro, podemos conseguir dos variables más que son la velocidad y el desplazamiento, esto se lleva a cabo gracias a la derivada o la integración, en este caso, emplearemos la integración, ya que tenemos como única variable a la aceleración para luego pasarla a velocidad y posteriormente al desplazamiento. A continuación, se describe cada una de estas unidades.

2.6.1 Desplazamiento:

En White (1990) menciona que el desplazamiento se mide generalmente en micrómetros, es útil para ver frecuencias bajas por debajo de los 10Hz aproximadamente y además son visibles por el ojo humano, ya que el desplazamiento enfatiza en su amplitud en bajas frecuencias.

En la figura 17 se muestra el espectro de la señal de vibración vista desde el desplazamiento.

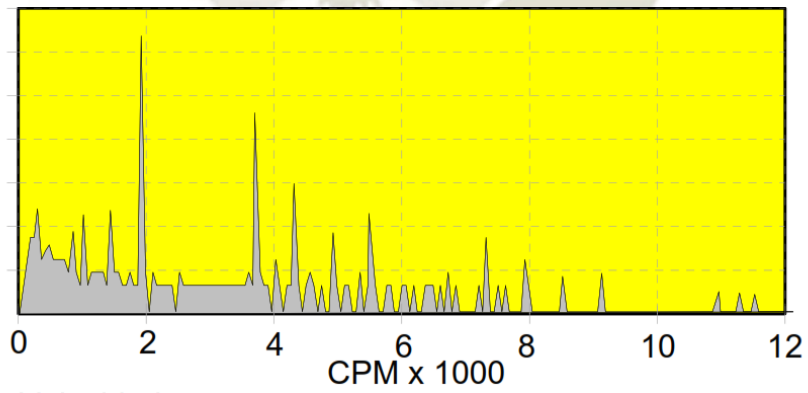


Figura 17: Espectro de la señal de vibración, vista desde el desplazamiento

Fuente: (White, 1990)

2.6.2 Velocidad

En White (1990) menciona que la velocidad en su mayoría se mide en milímetros por segundo, este parámetro es más uniforme al mostrar la amplitud de las frecuencias, también es el más utilizado en la práctica. Sus valores están aproximadamente entre los 10 y 1000Hz.

En la figura 18 se muestra el espectro de la señal de vibración vista desde la velocidad.

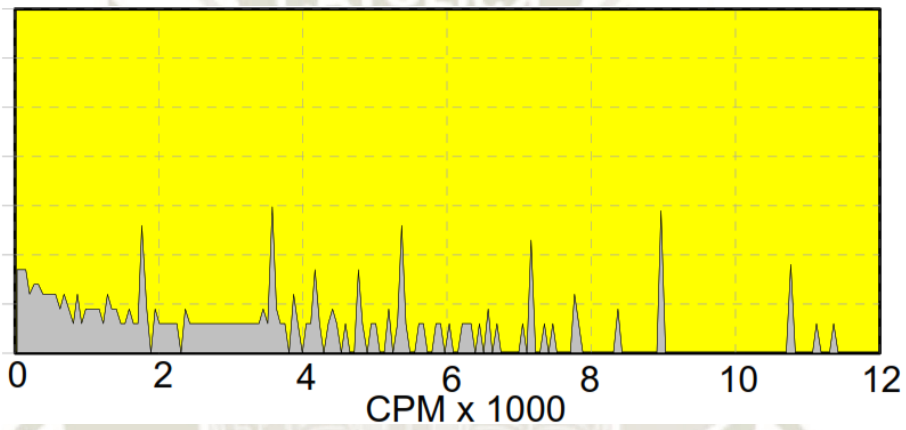


Figura 18: Espectro de la señal de vibración vista desde la velocidad

Fuente: (White, 1990)

2.6.3 Aceleración

En White (1990) menciona que la aceleración en su mayoría se mide en milímetros por segundo al cuadrado, en este caso la aceleración enfatiza más en su amplitud las altas frecuencias, y están por encima de los 1000Hz aproximadamente.

En la figura 19 se muestra el espectro de la señal de vibración vista desde la aceleración.

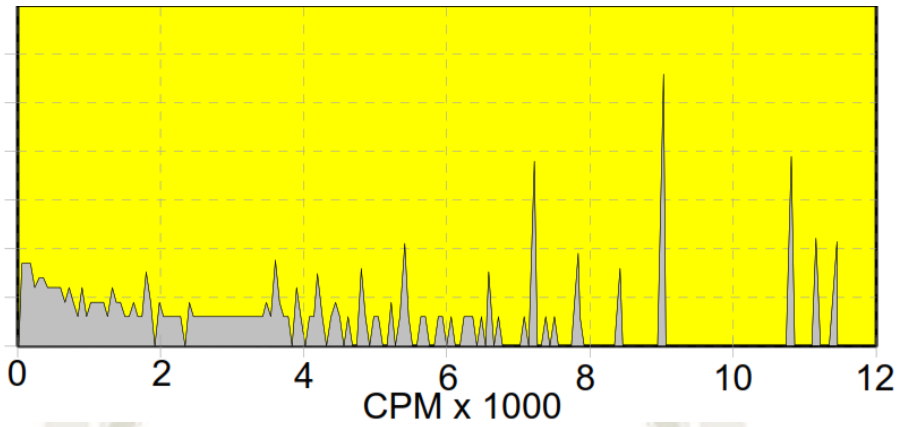


Figura 19: Espectro de la señal de vibración vista desde la aceleración

Fuente: (White, 1990)

2.7 SENSOR DE VIBRACIÓN

Para medir la vibración existen varios sensores que nos permite captar estas señales como por ejemplo sensores de proximidad, de velocidad que nos permite captar frecuencias bajas hasta los 1000Hz aproximadamente y los acelerómetros que son usados mayormente por su mayor rango de frecuencia, incluso llegan a 15KHz según indica White (1990). En esta parte solo hablaremos de los acelerómetros.

2.7.1 Acelerómetro piezoeléctrico

En White (1990) menciona que el acelerómetro piezoeléctrico tiene la característica principal de que puede captar frecuencias bajas, medias y altas, es por ello que es bastante usado en la industria. El acelerómetro está construido por un botón de montaje, un elemento de cristal, masa sísmica, resorte precargado y un amplificador. Todos sus elementos interactúan para que se emita una señal que es proporcional a la vibración producida por el motor eléctrico para luego ser procesada por una computadora o microcontrolador.

En la figura 20 se muestra las partes del acelerómetro piezoeléctrico.

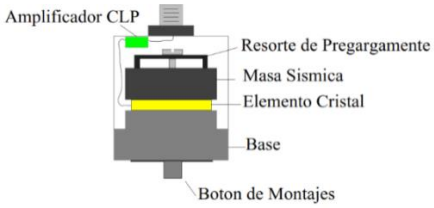


Figura 20: Partes del acelerómetro piezoeléctrico

Fuente: (White, 1990)

2.7.2 Montaje del acelerómetro

En la práctica hay varias formas de realizar el montaje del acelerómetro, este influye en el rango de frecuencias ya que nos limita en el proceso de medición, uno de los montajes más utilizados es con el uso de un imán en la base, ya que está en los rangos de frecuencia donde más problemas se encuentran en los motores eléctricos.

Holek (2005) menciona que todas las formas son usadas, ya que esto varía dependiendo en qué situación nos encontremos. Por ejemplo, si un motor está caliente y sobrepasa los valores permisibles del acelerómetro en temperatura, lo más practico es usar la forma de montaje de una varilla, ya que con esta forma de montaje se tardará un tiempo determinado en llegar el calor al acelerómetro.

En la figura 21 se muestra los varios tipos de montaje del acelerómetro.

Figura 21: Tipos de montaje del acelerómetro

Fuente: (Holek, 2005)

2.8 INTEGRACIÓN DE LA SEÑAL DE VIBRACIÓN

2.8.1 Algoritmo radix 4 FFT

Aplicar la transformada de Fourier para obtener datos en frecuencia, en la práctica los microcontroladores lo pueden hacer, pero se demoran mucho y ocupan demasiada memoria, para esto salió un algoritmo llamado Radix 4.

Cheng (200) menciona que al reutilizar los resultados de cálculo intermedios más pequeños para calcular múltiples salidas de frecuencia, logra que el algoritmo reordene la ecuación de la transformada discreta de Fourier en cuatro partes, entre muestras pares e impares, por lo que la longitud de las muestras se establece con lo siguiente (16, 64, 256, 1024, 2048). Esto quiere decir que si queremos que nos devuelva 512 muestras tendríamos que muestrear con cuatro veces más.

2.8.2 Periodo de muestreo.

White (1990) menciona que el criterio de Nyquist indica que una señal se puede muestrear si es mayor a 2 veces el ancho de banda de la señal, por ejemplo, si tenemos un acelerómetro que tiene como rango de frecuencias hasta 10KHz es recomendable usar mayor que el doble.

En la figura 22 se muestra el muestreo de una señal.

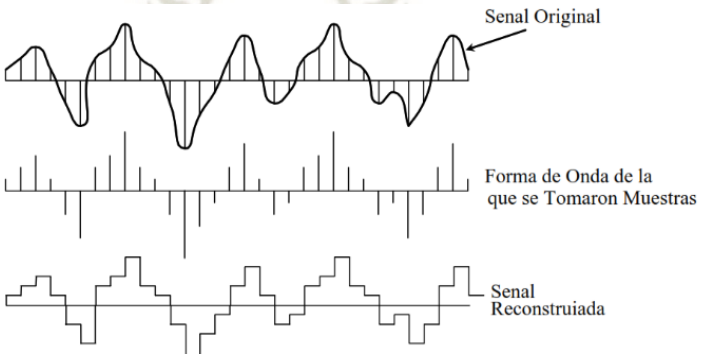


Figura 22: Muestreo de una señal

Fuente: (White, 1990)

2.8.3 Resolución de frecuencia y tiempo de adquisición de datos.

La resolución nos indica el espacio que va a tener entre cada línea, en el espectro, se calcula de la siguiente manera.

$$\Delta F = \frac{\text{Rango de frecuencia}}{\text{Numero de lineas}}$$

El tiempo de adquisición de datos es básicamente el tiempo que se va a demorar la computadora o microcontrolador en la adquisición de datos y se calcula sacado el reciproco de la resolución de frecuencia según indica White (1990).

En la figura 23 se muestra la resolución de frecuencia de una señal.

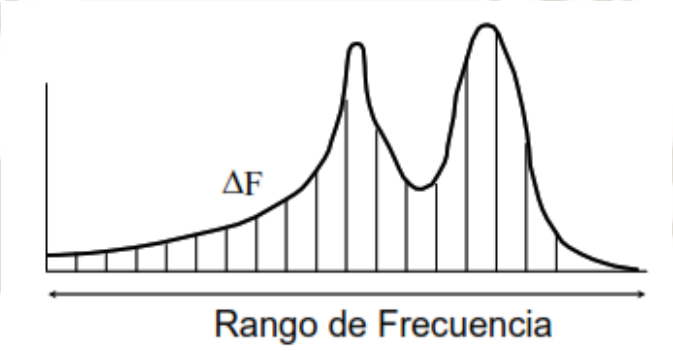


Figura 23: Resolución de frecuencia

Fuente: (White, 1990)

2.8.4 Máximo y promediado espectral

- Promediado espectral: Se almacena en un vector y además calcula el promedio de los datos de un vector. Y además calcula el promedio de los datos tomados nos sirve para disminuir un poco el ruido espectral.

- Máximo: En este algoritmo se busca el valor máximo en un vector y la posición del vector, todo esto devuelve dicha función. Este valor nos ayuda a ubicar rápidamente la frecuencia con mayor amplitud.

2.8.5 Ventana Hanning

Al aplicar la transformada de Fourier, se requiere que la señal muestreada comience y termine en cero. Con el uso de la ventana hanning, fuerza a la señal muestreada a que empiece en cero y termine en cero, pero también distorsiona a la forma de onda, toma una forma de modulación de amplitud esto menciona White (1990).

En la figura 24 se muestra la aplicación de la ventana hanning a una señal.

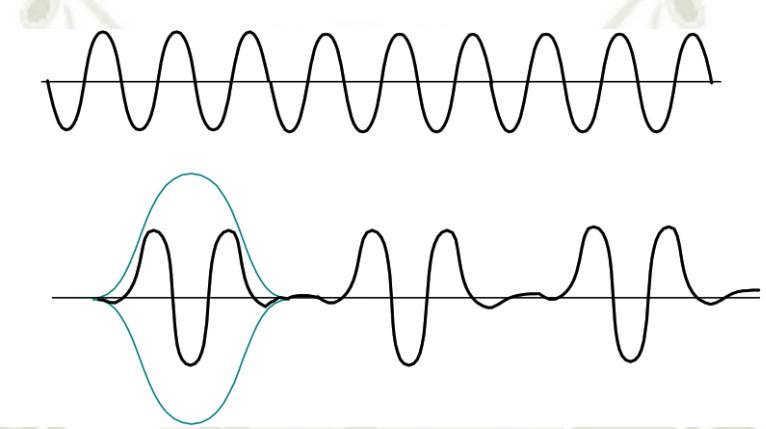


Figura 24: Ventana Hanning

Fuente: (White, 1990)

2.8.6 Conversión de unidades

La conversión de unidades es primordial para el desarrollo de nuestro equipo ya que el sensor de vibración nos da valores en aceleración (g o mm/s2), como se comentó líneas arriba en las unidades de vibración se comentó que para tener una mejor apreciación de la firma de vibración se tiene que tener valores en velocidad (mm/s).

Para ello se tiene que transformar las unidades de aceleración en velocidad según indica White (1990), mediante la siguiente ecuación se puede realizar dicha transformación.

$$Velocidad = \frac{87.75\ x\ Aceleracion}{frecuencia}$$

2.8.7 Filtros digitales

Gomez Gil (2017), indica que los filtros digitales en la actualidad son importantes para el desarrollo de procesamiento digital de señales, ya que estos filtros realizan diferentes actividades como por ejemplo para suavizar señales. Como en los filtros análogos existen gran variedad de filtros (pasa bajos, pasa altos, rechaza banda, pasa banda) y de diferentes órdenes (segundo, tercer, cuarto, etc.). En los filtros digitales es de igual forma. A continuación, se presentan dos tipos de filtros digitales.

- Filtro FIR: Según indica Gómez Gil (2017) el filtro FIR es conocido como filtro de respuesta al impulso finito, este tipo de filtro se basa en la acumulación múltiple secuencial, como se describe en la imagen cada h[n] se multiplica con la variable de entrada X[n], luego se suma todas ellas para luego tener el filtro digital, la característica principal de estos filtros son que pueden llegar a tener una fase lineal, son estables pero por lo contrario para llegar a un a fase líneas y a la estabilidad se tienen que diseñar con un orden mayor y eso provoca un mayor uso de memoria en el procesador en cual se esté diseñando. En esta tesis se aplicará este tipo de filtro.

  En la figura 25 se muestra el diagrama de bloques del filtro digital FIR.

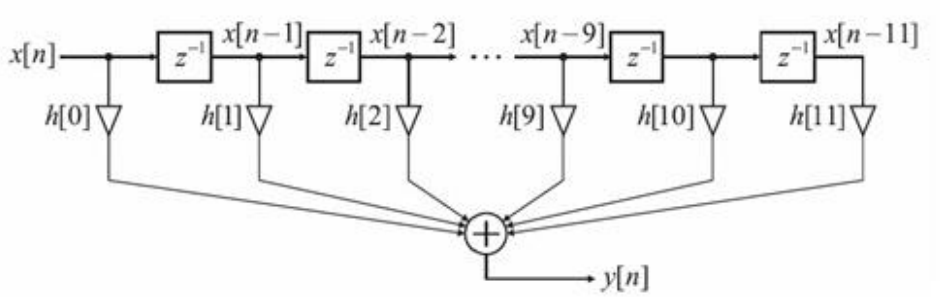


Figura 25: Diagrama de bloques del filtro digital FIR
Fuente: http://monipima.blogspot.com/2010/10/matlab-filtro-fir.html, (2010)

- IIR: Según indica Gómez Gil (2017) el filtro IIR es conocido como el filtro de respuesta al impulso infinito, este filtro cumple la misma función que la anterior, pero con la característica de que se puede diseñar con un menor orden y así no ocupa mucha

memoria en el procesador, pero es inestable este filtro. En esta tesis no se implementará este filtro ya que se busca estabilidad en la señal de vibración.

2.8.8 Relación señal / ruido

Como su nombre lo indica la relación señal / ruido, es la relación de la intensidad de la señal y el ruido de fondo que acompaña a la señal.

Muchos expertos comentan que la relación señal ruido desde los 25 dB hasta los 40dB, se puede lograr una transmisión de datos a una velocidad rápida según menciona Vega (2019). Este concepto se usará en el momento de aplicar el filtro digital FIR, ya que debemos tener un tope en el filtro.

2.8.9 Librería DSP de Texas instruments

La librería DSP consta de varios módulos de cálculos matemáticos como por ejemplo multiplicación, suma, resta, funciones trigonométricas, funciones de estadística (Máximo, promediado, Valor RMS), filtros digitales FIR, cálculo de la transformada de Fourier y muchas más. Esta librería la utilizaremos para desarrollar el motor de cálculo de la señal de vibración.

2.9 NORMAS Y RECOMENDACIONES GENERALES PARA EVALUAR VALORES DE VIBRACIÓN

Power MI Cloud Condition Monitoring (2018) indica que para el análisis de vibraciones se recomienda utilizar la norma de severidad ISO-10816, cabe mencionar que existen varias normas por ejemplo la ISO 2372, la carta de Rathbone.

2.9.1 Recomendaciones

Según Holek (2005), recomienda lo siguiente.

- Contrastar los datos obtenidos con la Norma ISO 10816 de severidad.

- Estar monitoreando el motor eléctrico cada cierto tiempo y realizar una comparación de tendencia, con el objetivo de ver los cambios significativos a través del tiempo.
- Comparar las mediciones con un motor parecido.
- Tener en cuenta la forma de fabricación del motor y la máquina, ya que en algunas aplicaciones estos motores manejan cargas que pueden proporcionar ruido en el espectro y es normal que trabaje de esa forma.

2.9.2 Norma ISO-10816

Esta norma de severidad está en unidades de velocidad y valores en RMS, ya que nos indica la energía total de firma vibración. Esta norma está caracterizada por tener rangos de máquinas según la potencia e instalación.

En la figura 26 se muestra la norma ISO 10816.

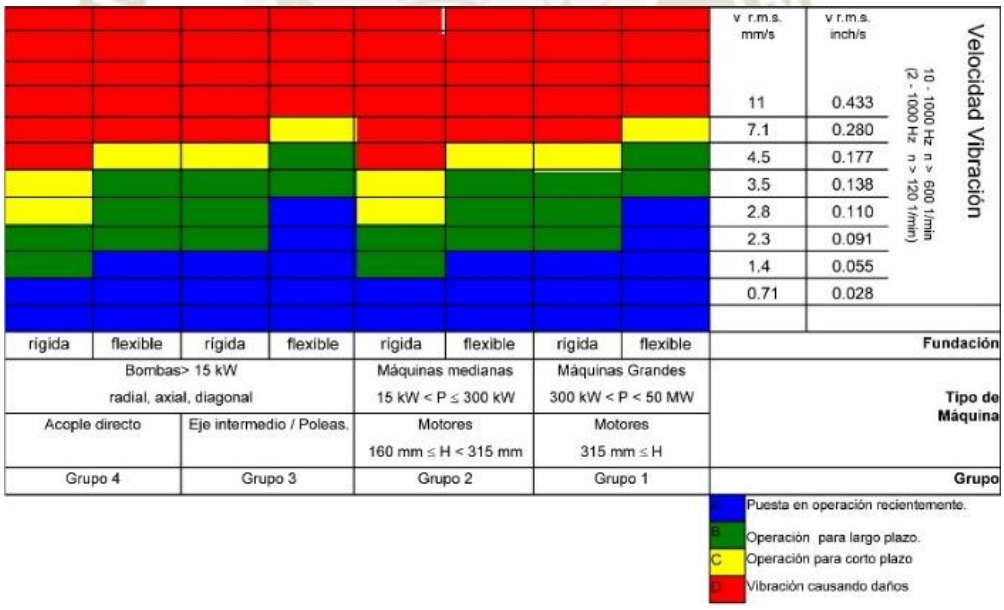


Figura 26: Norma ISO 10816

Fuente: https://pruftechnik.wordpress.com/2011/10/13/la-vibracion-confirma-la-norma/, (2011)

2.10 PRINCIPIO DE FUNCIONAMIENTO DEL PROTIPO REGISTRADOR DE VIBRACIONES DE MOTORES ELECTRICOS.

El equipo portátil registrador de vibraciones de motores eléctricos, registra el espectro producido por la vibración del motor eléctrico, luego muestra el resultado en la pantalla LCD, el valor máximo y el valor RMS del espectro, para que posteriormente el usuario pueda comparar los valores con la norma ISO 10816.

El procesamiento de los datos lo llevará a cabo el microcontrolador, el cual muestrea las vibraciones obtenidas del sensor de vibración y una vez almacenados los datos en la memoria del microcontrolador ejecutará con ellos algoritmos como el cálculo de la transformada de Fourier.

Una vez desarrollados los cálculos por el microcontrolador este procederá a mostrar los resultados en la pantalla LCD y luego almacenar datos de la frecuencia y amplitud del espectro en la memoria SD extraíble.

CAPITULO III

3. DISEÑO E INGENIERIA

En este capítulo se presenta las especificaciones técnicas y diseño e ingeniería del equipo registrador de vibraciones para motores eléctricos como se muestra en la tabla 1 y figura 27.

3.1 ESPECIFICACIONES DEL EQUIPO

Tabla 1
Especificaciones técnicas del equipo

Especificaciones generales	Especificaciones eléctricas
• Almacena datos en formato Excel hasta de 8GB. • Muestra valores máximo y valor RMS de la firma de vibración. • Grado de protección: Prototipo. • Temperatura de operación: 0-50°C.	• Adaptador de CA: Voltaje de entrada 220Vac y frecuencia de entrada de 60Hz. • Sistema monofásico. • Batería de litio: 5V/2000mAh. • Tiempo de carga de batería: una hora y media. • Tiempo de descarga de batería: tres horas en condiciones normales.
Especificaciones del sensor de vibración	**Especificaciones del sistema de medición y procesamiento**
• Sensibilidad: 100mV/g (±10%). • Rango de medición: ±50g. • Frecuencia de resonancia: 25KHz nominal. • Rango de frecuencia: 0.43KHz a 10Khz (±3dB). • Sensibilidad transversal: ≤5%. • Requisitos de alimentación: 18VDC a 24VDC/2 a 20mA. • Compatibilidad EMC: Cable apantallado de 50cm.	• Procesador: ARM CORTEX RM 46L. • Velocidad del procesador: 1.66DMIPS/220MHz. • Conversor ADC: 12 bits/3.3V. • Ventana de espectro: Hanning. • Resolución de la pantalla FFT: 512 líneas. • Ancho de banda: 5 a 10KHz.
Especificaciones del módulo de visualización	

• Visualizador: Pantalla LCD TFT a colores de 7”.
• Resolución: 800x480 pixeles.
• Entrada: Panel táctil

Fuente: Propia

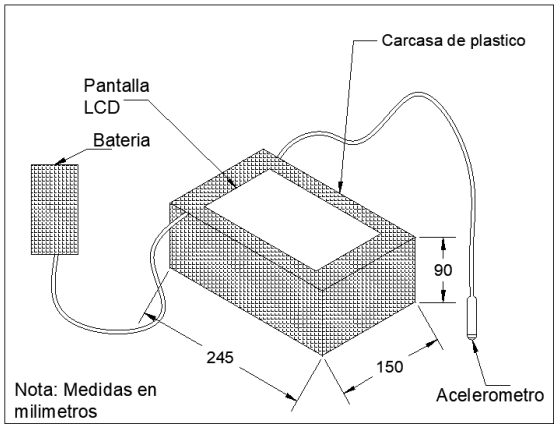


Figura 27: Dimensiones en milímetros del prototipo registrador de vibraciones

Fuente: Propia

3.2 DISEÑO DE HARDWARE

A continuación, se muestra en la figura 28 un diagrama de bloques generales donde se observa la relación de las diferentes etapas. La cual se compone por lo siguiente:

- Etapa de acondicionador de señal
- Etapa de circuito de visualización
- Etapa de batería y adaptador CA
- Etapa de sistema de reloj y memoria

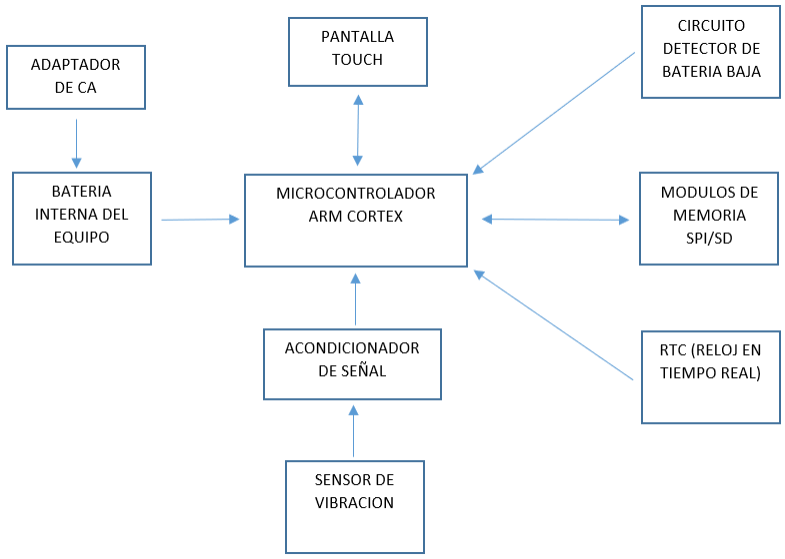


Figura 28: Diagrama de bloques general del equipo registrador de vibraciones

Fuente: Propia

A continuación, se detalla cada etapa de la figura 28.

3.2.1 Etapa de acondicionador de señal

En la figura 29 se muestra el diagrama de bloques de la etapa de acondicionador de señal.

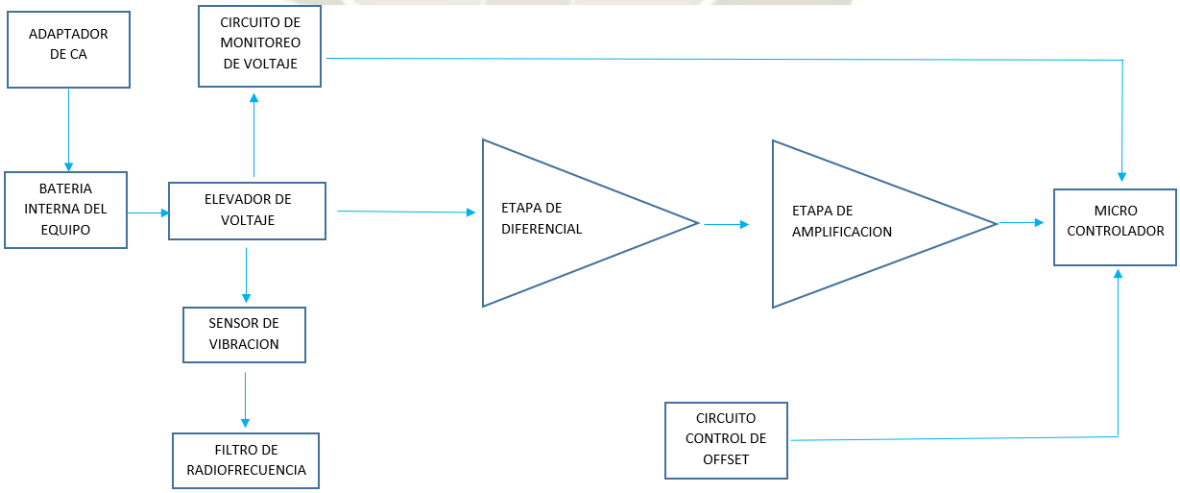


Figura 29: Diagrama de bloques de la etapa de acondicionador de señal.

Fuente: Propia

Como se aprecia en la Figura 29 el módulo de elevador de voltaje nos permite alimentar el sensor de vibración con 20Vdc, ya que la batería solo proporciona 5Vdc y no es suficiente para alimentar el sensor de vibración. Con este módulo también se tiene que

alimentar el amplificador operacional de la etapa del diferenciador de voltaje ya que tiene como alimentación de +20Vdc.

El Circuito de entrada de voltaje de sensor de vibración es requerido para monitorear el voltaje de alimentación del sensor de vibración que vendrían hacer 20Vdc. Ya que si disminuye podría apagarse el sensor de vibración. Este sensor de vibración es un acelerómetro que se escogió para sensar las vibraciones producidas por el motor eléctrico, para luego procesar la señal producida por este sensor de vibración e identificar el problema del motor eléctrico. Se está usando el filtro de radiofrecuencia para filtrar las señales de radiofrecuencia que se introducen en el circuito de acondicionamiento de señal.

En la etapa diferencial se está usando un amplificador operacional que se comporta como un conversor de corriente a voltaje, ya que la señal producida del sensor está en corriente, por lo que se tiene que convertir esta señal a voltaje.

En la etapa de amplificación la señal obtenida de la etapa diferencial es amplificada para que se pueda tener valores aceptables y nuestro microcontrolador pueda procesar correctamente la señal producida del sensor de vibración.

La etapa de control de offset es requerida para aplicaciones de amplificadores operacionales que solo necesitan una sola fuente de alimentación, está compuesto por un amplificador operacional en su configuración de seguidor de voltaje, un potenciómetro y un capacitor, básicamente lo utilizaremos como una entrada más al ADC para luego restarlo con el voltaje del sensor de vibración, para tener como resultado la señal de vibración.

A continuación, se procede a explicar los criterios y selección de componentes para la etapa de acondicionador de señal para sensor de vibración

3.2.1.1 Modulo elevador de voltaje

En la tabla 2 se aprecia las características de dos posibles alternativas para la selección del módulo elevador de voltaje.

Tabla 2

Selección de módulos de elevador voltaje

Item	Modelo 1	Modelo 2
Voltaje de entrada	5VDC	3.7 VDC
voltaje de salida	24VDC	32 VDC
Potencia de consumo	30W	10W
Foto de referencia		

Fuente: Propia

De acuerdo a las características vistas en la tabla 2 se escogió el modelo 1, porque se tiene como fuente de alimentación principal, una batería de 5Vdc. Además, por el tamaño, ya que nuestro prototipo debe ser lo más liviano posible.

3.2.1.2 Circuito de entrada de voltaje de sensor de vibración

Para el circuito de monitoreo de voltaje se consideró un divisor resistivo ya que en R5 debe concretarse el voltaje máximo que pueda recibir el microcontrolador para este caso son 3.3 voltios.

En la figura 30 se muestra el circuito de entrada de voltaje del sensor de vibración.

Cálculo de la resistencia R4 y R5.

$$Vo = Vi\frac{R5}{R4 + R5}$$

$$\frac{Vo}{Vi} = \frac{R5}{R4 + R5}$$

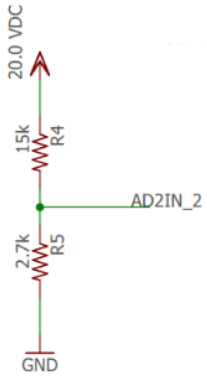


Figura 30: Circuito de entrada de voltaje del sensor de vibracion

Fuente: Propia

Como Vo es 3.3Vdc y es directamente proporcional a la resistencia R5 y Vi es 20Vdc y es directamente proporcional a la suma de la resistencia R4 y R5. Reemplazando en la formula tenemos lo siguiente.

$$\frac{3.3}{20} = \frac{R5}{R4 + R5}$$

Las resistencias comerciales escogidas son las siguientes:
R4: 15KΩ y R5: 2.7KΩ, que nos aproximan a dichos voltajes.

Para la potencia primero se calculó la corriente.

$$I = \frac{20}{15k + 2.7\text{k}} = 0.0013A$$

Luego a cada resistencia se aplicó la ecuación de potencia.

$$P = I^2 xR$$

Teniendo como resultado para R4 el valor de 0.019W y para R5 el valor de 0.003W, por lo que se consideró resistencias de ¼ de W para R4 y R5.

3.2.1.3 Sensor de vibración

Se tuvo tres opciones para elegir el sensor de vibración como se puede apreciar en la tabla 3.

Tabla 3

Selección de Sensores de vibración

Item	Acelerómetro Wilcoxon 766	Acelerómetro Wilcoxon 786A	Acelerómetro Emerson A0322RI
Sensibilidad	100mV/g	100mV/g	100mV/g
Rango de med.	±80g	±80g	±50g
Rango de frec.	2-10KHz (±5%), 1-12KHz (±12%), 0.6-15KHz (±3dB).	3-5KHz (±5%), 1-9KHz (±12%), 0.5-14KHz (±3dB).	1.2-5KHz (±5%), 0.87-7KHz (±10%), 0.43-10KHz (±3dB).
Al. V/I	18 a 30V/2-10mA	18 a 30V/2 a 10mA	18 a 24V/ 2 a 20mA
Voltaje Bias	12V	12V	8 a 12V
Impedancia de salida	< 100 Ωs	< 100 Ωs	<150 Ωs

Fuente: Propia

De la tabla 3 se escogió el acelerómetro de la marca EMERSON A0322RI, por el rango de medición, porque generalmente las vibraciones mecánicas en motores eléctricos no llegan a 50g. Además, este acelerómetro incluye cable de conexión y es más accesible en el mercado.

Este sensor tiene una alimentación de 18 a 24Vdc, impedancia de salida hasta de 150Ω y un voltaje de salida Bias de 8 a 12Vdc. Se está usando una alimentación de 20Vdc, por lo tanto, el voltaje Bias de salida debe ser aproximado a Vcc/2 es decir de aproximadamente de 10Vdc. Se necesita un circuito divisor de voltaje, para que el voltaje en R17 se convierta en la salida de voltaje del sensor de vibración, tomando en consideración la impedancia de salida es menor a 150 Ω.

$$Vo = Vi\frac{R17}{R17 + 150}$$

$$\frac{Vo}{Vi} = \frac{R17}{R17 + 150}$$
$$\frac{10}{20} = \frac{R17}{R17 + 150}$$

Como Vo es 10Vdc y es directamente proporcional a la resistencia R17 y Vi es 20Vdc y es directamente proporcional a la suma de la resistencia R17 y la impedancia de salida de 150Ω, dando como resultado que la resistencia R17 sea mayor a 150Ω.

En la figura 31 se muestra el circuito del sensor de vibración y la resistencia a calcular.

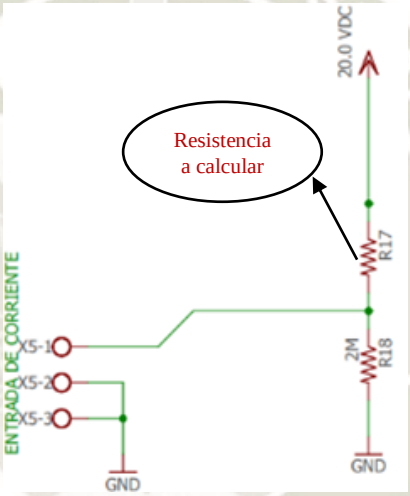


Figura 31: Circuito del sensor de vibración

Fuente: Propia

Se considera una resistencia de 200Ω para nuestro circuito ya que es fácil de conseguir en el mercado y es próxima a la impedancia de salida. Reemplazando la resistencia calculada en la ecuación siguiente para verificar que el voltaje en R17 sea aproximado a 10V, como se indicó líneas arriba.

$$Vo = Vi\frac{R17}{R17 + 150}$$
$$Vo = 10\frac{200}{200 + 150}$$
$$Vo = 11.4V$$

De la respuesta anterior tenemos un voltaje de salida de 11.4Vdc, con lo cual no vario mucho de los 10V estimados. La resistencia R18 se colocó para fijar a tierra el circuito, se escogió una resistencia muy grande que es de 2 Mega Ω.

Para la potencia primero se calculó la corriente.

$$I = \frac{20}{200 + 2\text{x}10^6} = 9.9x10^{-6}$$

Luego a cada resistencia se aplicó la ecuación de potencia.

$$P = I^2xR$$

Teniendo como resultado para R17 el valor de 1.1x10^-8W y para R18 el valor de 0.0002W, por lo que se consideró resistencias de ¼ de W para R18 y 1W para R17.

En la figura 32 se muestra el circuito de sensor de vibración y la resistencia para fijar a tierra el circuito.

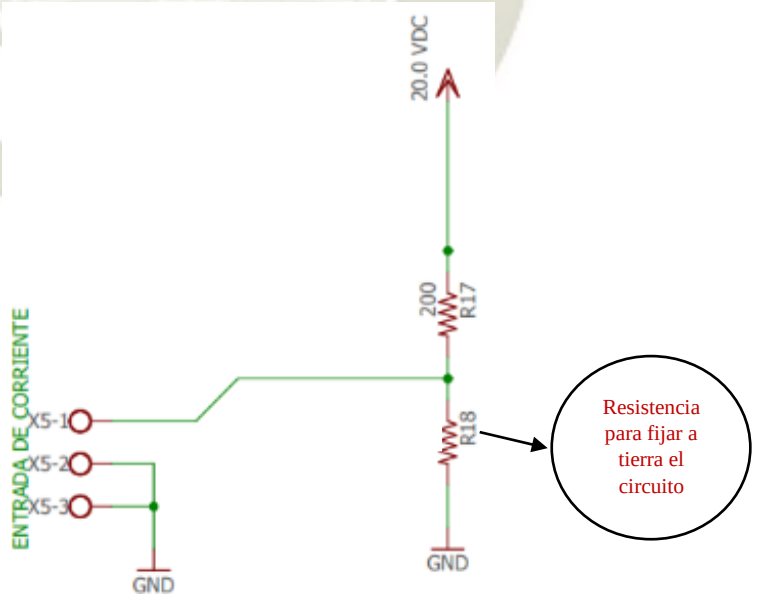


Figura 32: Circuito sensor de vibración

Fuente: Propia

3.2.1.4 Filtro de radiofrecuencia

Luego de obtener la señal de vibración, se da paso al filtro utilizado. Este filtro ayuda a filtrar las señales de radiofrecuencia que son emitidas por la red eléctrica. Cabe mencionar que este posee una topología de filtro tipo pi. Para escoger los capacitores C3 y C4 se realizó una prueba experimental para diferentes valores de capacitores, se empezó probando con capacitores de década en década desde 1uF hasta llegar hasta los 10nF. Se obtuvo mejores resultados con capacitores de 10nF, por lo tanto, asumimos que los valores de los capacitores serian de 10nF.

A continuación, se muestra las figuras 33, 34 y 35, donde se muestra la simulación en el software multisim.

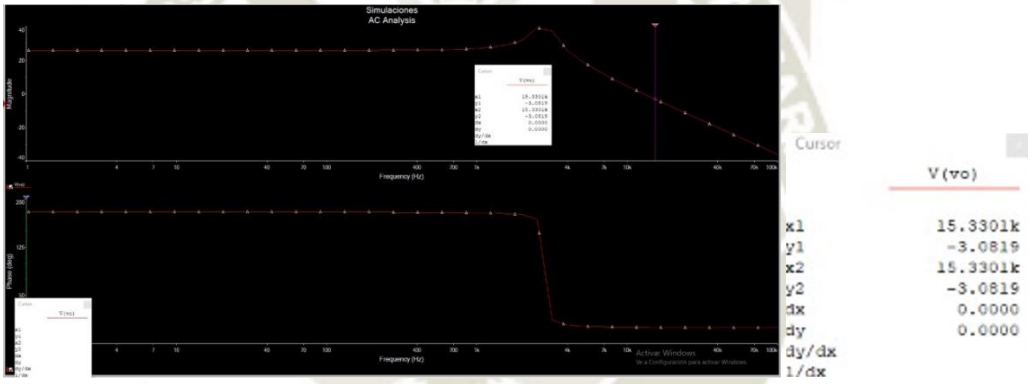


Figura 33: Simulación con 1uF

Fuente: Propia

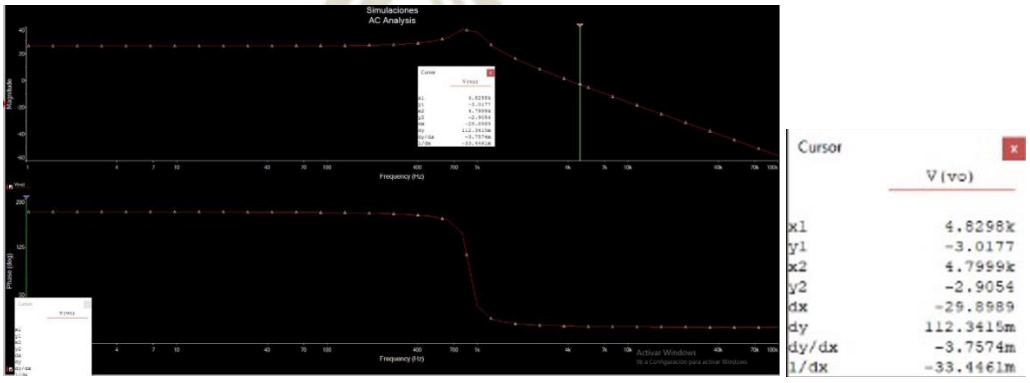


Figura 34: Simulación con 10 uF

Fuente: Propia

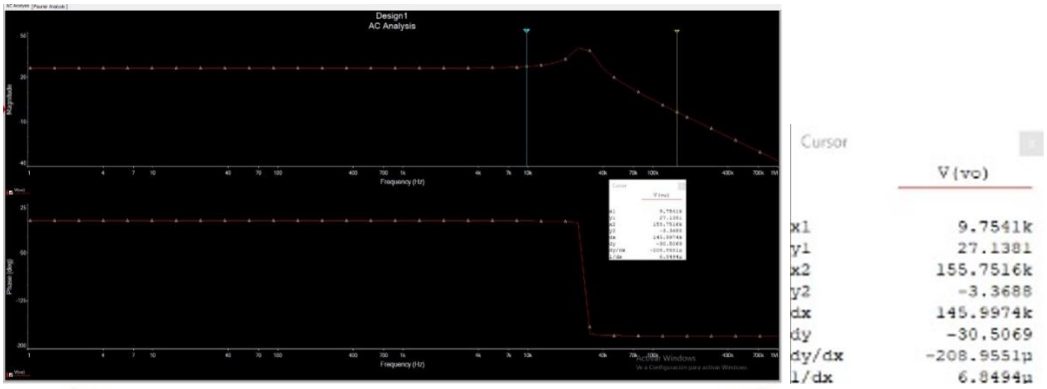


Figura 35: Simulación con 10nF

Fuente: Propia

En la figura 36 se muestra el circuito de filtro de radiofrecuencia a implementar

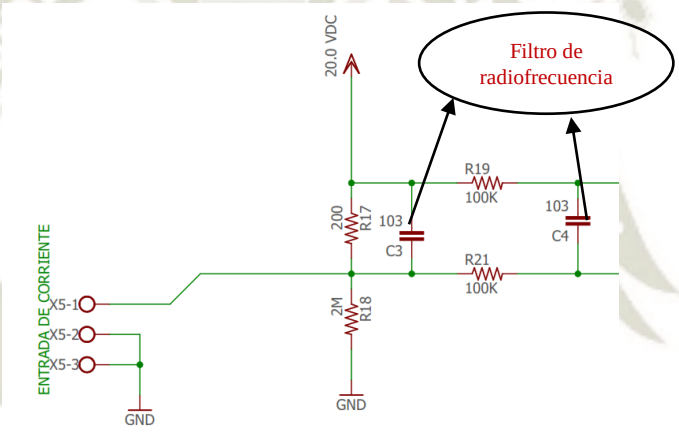


Figura 36: Circuito de filtro de radiofrecuencia

Fuente: Propia

3.2.1.5 Etapa diferencial

En esta etapa se tiene como objetivo obtener el voltaje en R17, con el fin de obtener los resultados en voltaje, porque del sensor se obtiene la señal en corriente. Para esta etapa se está considerando la configuración del amplificador operacional en modo diferenciador con ganancia 1.

En la figura 37 se muestra el circuito de la etapa diferencial.

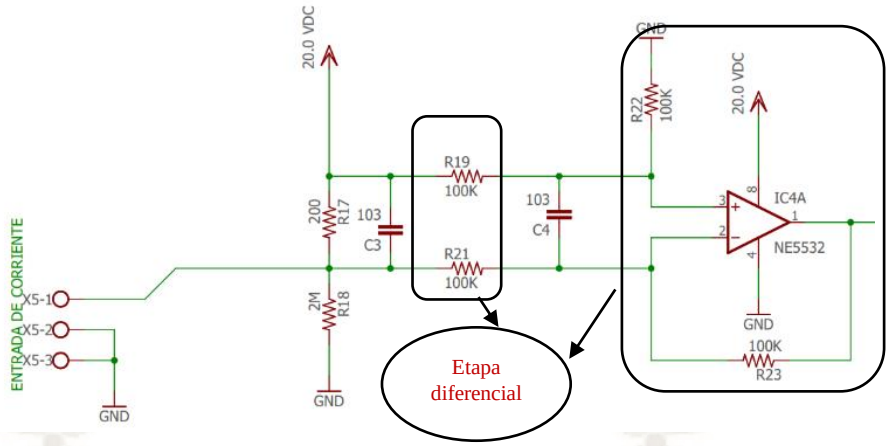


Figura 37. Circuito de la etapa diferencial

Fuente: Propia

Texas Instruments (2000) recomienda que deba considerarse R19, R21, R22, R23 iguales como se muestra en la figura 38, por lo que se escogió la resistencia de 100KΩ para tener una ganancia de 1.

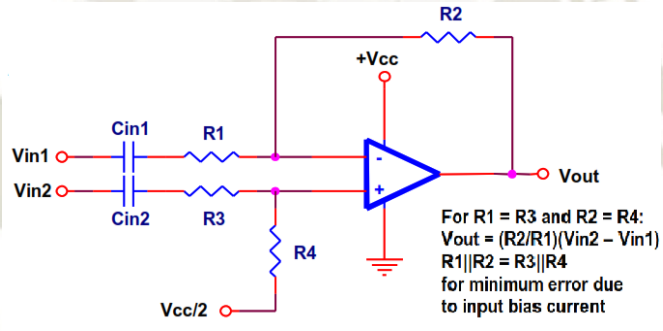


Figura 38: Circuito diferenciador con OPAMP

Fuente: (Texas Instruments, 2000)

La corriente que maneja está en orden de miliamperios por lo que se está considerando resistencias de 1/4W.

Para escoger el amplificador operacional se tuvo en consideración las opciones de la tabla 4.

Tabla 4

Selección de Amplificadores operacionales

Item	MCP 6022 – Microchip (02 canales)	OPA 316 – Texas instruments (02 canales)	NE5532 – Texas instruments (02 canales)
Rail to rail, entrada / salida	si	si	no
Ruido bajo hasta	8.7nV/√Hz a 10KHz	11nV/√Hz a 1KHz	5nV/√Hz a 1KHz
Ancho de banda	10 MHz	10 MHz	10 MHz
Alimentación	2.5V a 5.5V	1.8V a 5.5V	±22V
Temperatura	-40ºC a +85ºC	-40ºC a +125ºC	-65ºC a +150ºC

Fuente: Propia

En esta etapa se consideró el amplificador NE5532, ya que este amplificador está recibiendo la señal principal y se debe considerar la alimentación de 20Vdc del sensor de vibración y además que tiene menor ruido.

3.2.1.6 Etapa de amplificación

Esta etapa continúa después de la etapa diferencial, por lo que se agregó un condensador de acoplamiento C5 para eliminar la señal del voltaje Bias de 11.4Vdc. Luego de esto viene la parte del amplificador en modo inversor con una ganancia de 10. Se estuvo probando con una ganancia superior a 10 pero no se consiguió una buena visión de la señal. En la figura 39 se muestra el circuito de la etapa de amplificación.

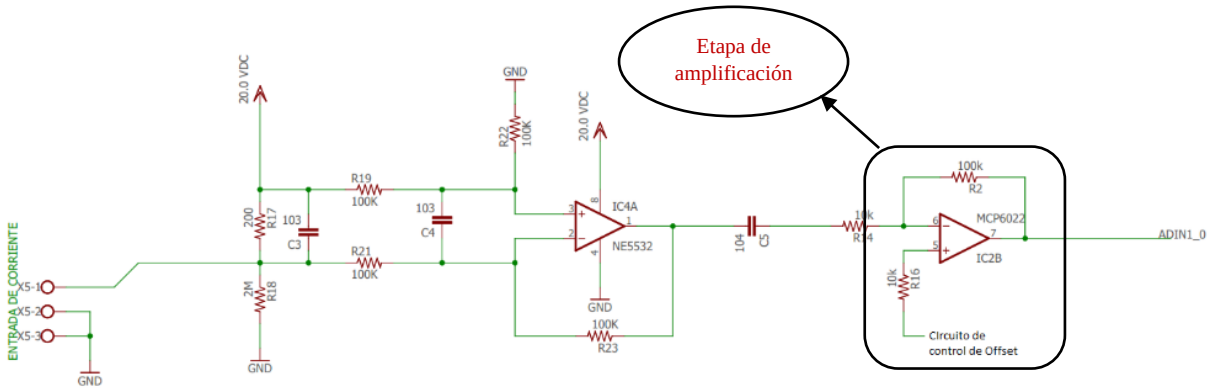


Figura 39: Circuito de la etapa de amplificación

Fuente: Propia

Texas Instruments (2000) recomienda que debe considerarse R16=R14//R2. Como se muestra en la figura 40, entonces las resistencias escogidas son: R14=10k, R2=100k, R16=10k. Para obtener una ganancia de 10.

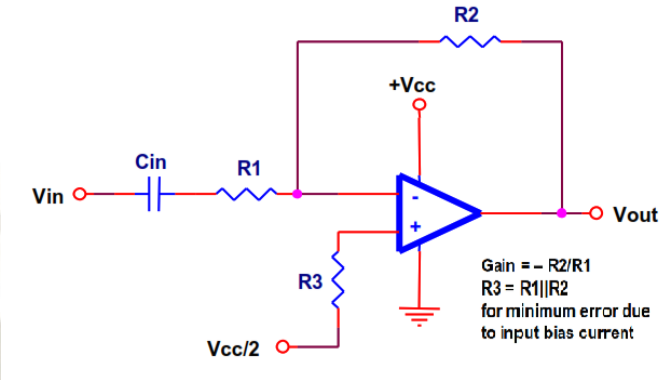


Figura 40: Circuito inversor amplificador con OPAMP

Fuente: (Texas Instruments, 2000)

La corriente que maneja está en orden de miliamperios por lo que se está considerando resistencias de 1/4W.

De la tabla 4, se consideró el amplificador MCP6022, para esta etapa de amplificación ya que esta etapa se necesita un operacional rail to rail.

3.2.1.7 Circuito de control de offset

El condensador C1 forma un filtro pasa bajo para suprimir el ruido introducido de la línea de tensión monofásica. Se colocó un potenciómetro de 50KΩ multivuelta, para fijar el voltaje offset de la etapa de amplificación. En la figura 41 se muestra el circuito de control de offset.

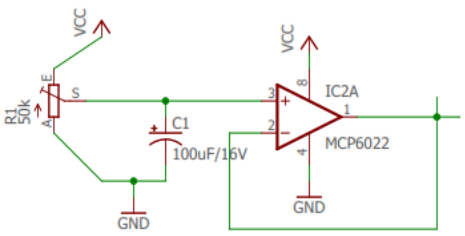


Figura 41: Circuito de control de offset

Fuente: Propia

Texas Instruments (2000) indica que el Opamp en configuración de seguidor de voltaje para acoplar la señal a la siguiente etapa. También cabe mencionar que los amplificadores operacionales de una sola fuente de alimentación necesitan un circuito de tierra virtual, generalmente de un voltaje de VCC/2.

Como se muestra en la figura 42. De igual forma que las resistencias anteriores son de 1/4W, las corrientes son pequeñas.

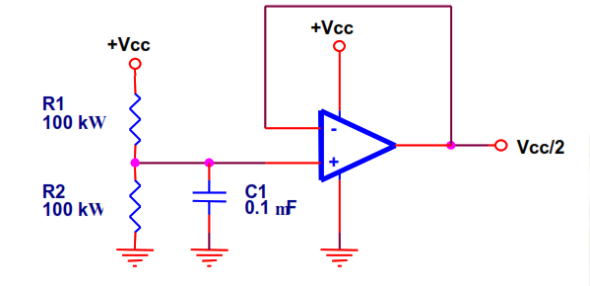


Figura 42: Circuito de control de offset con OPAMP

Fuente: (Texas Instruments, 2000)

De la tabla 4, se consideró el amplificador MCP6022, para esta etapa de circuito de control de offset, ya que esta etapa se necesita un operacional rail to rail y se debe considerar la alimentación del microcontrolador de 3.3Vdc.

3.2.1.8 Circuito esquemático de la etapa de acondicionador de señal

A continuación, se muestra en la figura 43 el circuito esquematico de etapa de acondicionador de señal para sensor de vibración.

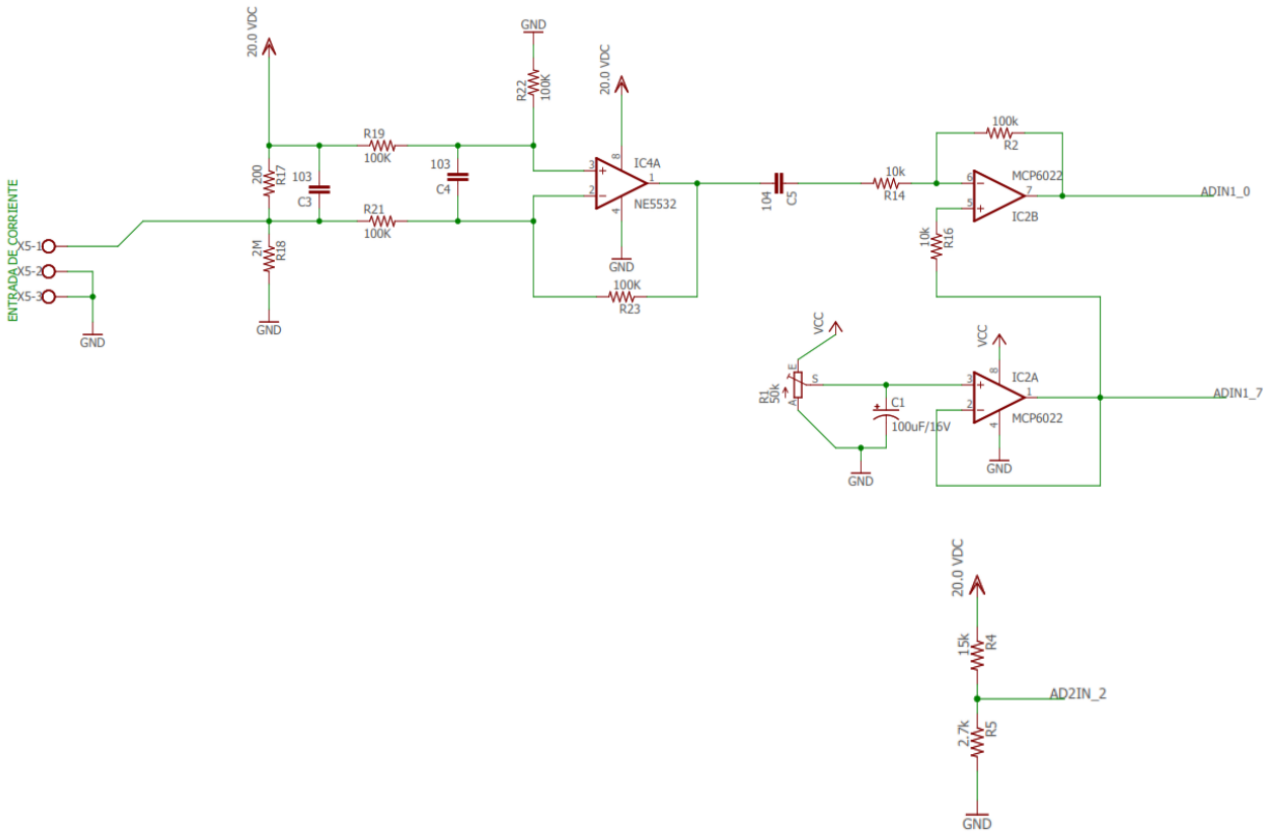


Figura 43: Esquemático de la etapa de acondicionamiento de señal

Fuente: Propia

3.2.2 Etapa del circuito de visualización

En la Figura 44 se muestra el diagrama de bloques del circuito de visualizacion

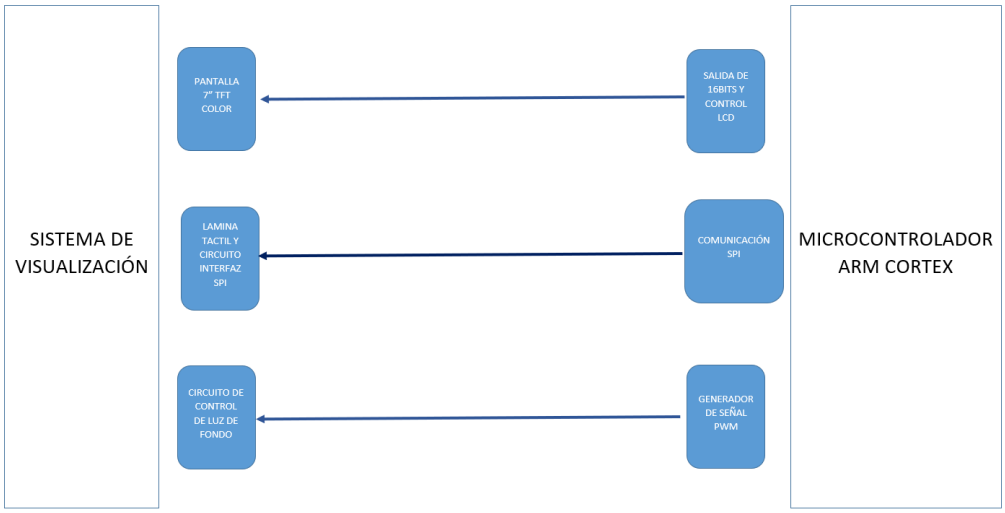


Figura 44: Diagrama de bloque del circuito de visualización

Fuente: Propia

Como se aprecia en la figura 44, este equipo cuenta con una pantalla LCD TFT táctil con código SSD1963, que es controlado por el microcontrolador y un dispositivo de registro de desplazamiento que es el 74HC595, este dispositivo nos ayuda a tener el control de la pantalla con tan solo 3 pines, en la pantalla se escogerán actividades que realice el equipo, adicionalmente muestra la información que ha sido procesada por el microcontrolador. Además, esta pantalla sirve como una interfaz HMI, para el ingreso y salida de datos.

A continuación, se describe las tareas que se puede realizar con la pantalla LCD.

- Visualización del espectro de vibración.
- Visualización de las características del motor eléctrico.
- Configuraciones del equipo.

3.2.2.1 Pantalla táctil TFT

A continuación, se muestra en la tabla 5 las diferentes pantallas TFT que se tuvo a consideración.

Tabla 5

Selección de la pantalla

Item	Nextion	SSD1963	SPITFT
Tamaño	7”	7”	5”
Alimentación	5v	5v	5v
Interfaz con el usuario	si	no	no
Acondicionamiento de librerías internas del microcontrolador, para contralar la pantalla	si	si	si

Fuente: Propia

Se escogió la pantalla SSD1963 de 7” de la tabla 5, ya que es posible el acondicionamiento de la interfaz con el usuario mediante la programación y además el tamaño es ideal para el equipo ya que es portable.

Esta pantalla TFT LCD tiene como principal característica que se conecta directamente a la fuente y se puede controlar la luz de fondo con la señal PWM. La señal de control de PWM es de 10KHz, esta frecuencia lo recomienda el fabricante. Por lo que solo se va a variar el dutty cicle, para tener una buena iluminación entre el 100%, 80%, 60% y 40%.

Ademas se esta incluyendo el 74HC959N que nos permite controlar la patanlla con tan solo 3 pines (LATCH/CLOCK/DATA), que tiene una entrada serial de 8 bit a una frecuencia de 100MHz, se alimenta a 5Vdc

3.2.2.2 Circuito esquemático de la etapa del circuito de visualización

A continuación, se muestra la figura 45 del circuito esquematico de etapa del circuito de visualización de señal para sensor de vibración.

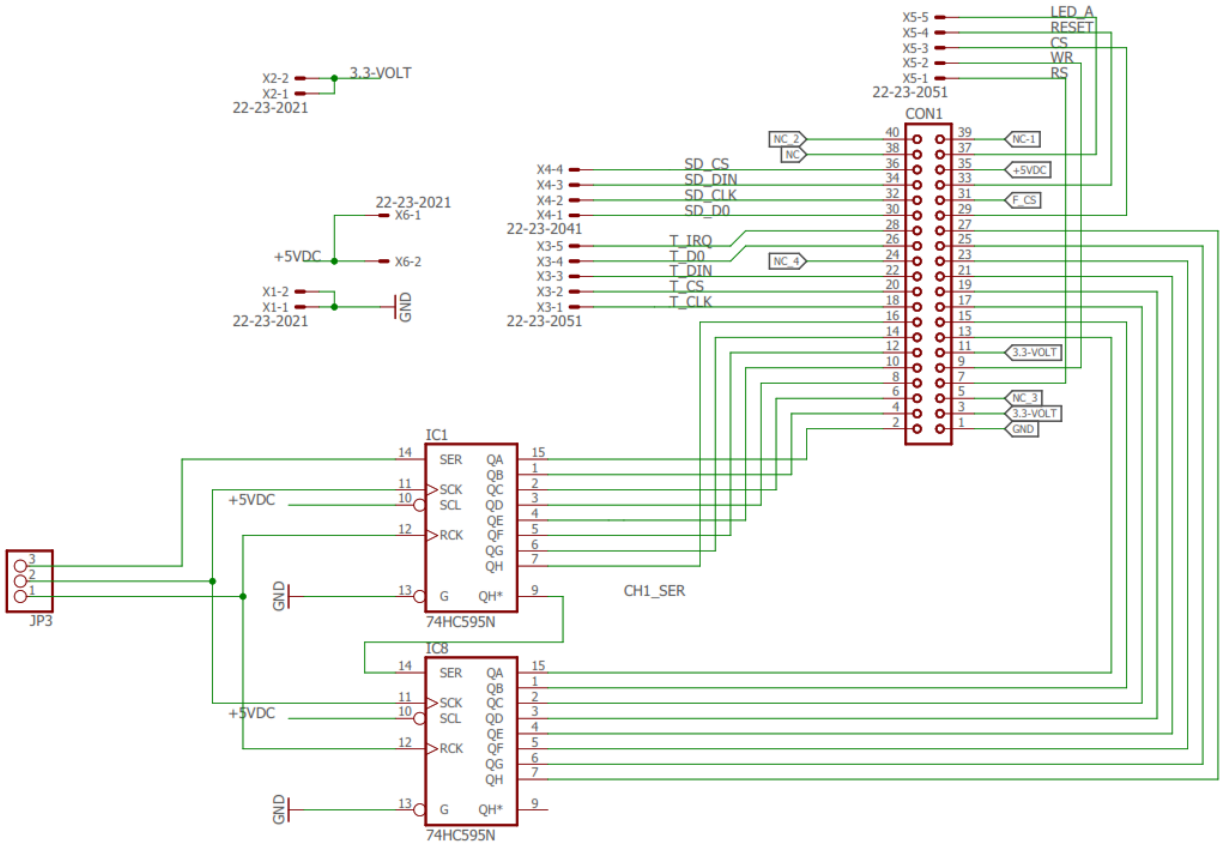


Figura 45: Esquemático de la etapa del circuito de visualización

Fuente: Propia

3.2.3 Etapa de batería y adaptador CA

En la Figura 46 se muestra el diagrama de bloques del circuito de visualizacion

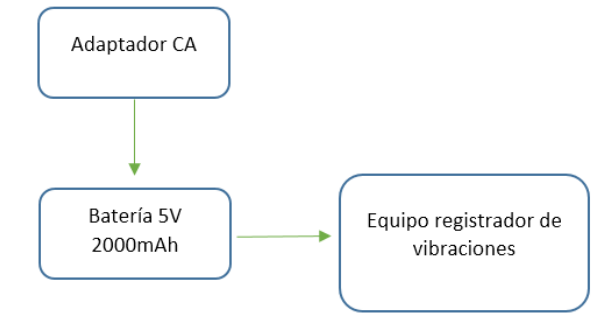


Figura 46: Diagrama de bloques de batería y adaptador de CA

Fuente: Propia

La parte de la alimentación del equipo registrador de vibraciones está conformada por un adaptador de CA y una batería de litio de 5V 4000mAh. Por lo que esto permite que el equipo sea portátil ya que solo se está usando una batería como fuente de alimentación. Esta batería se carga con el adaptador de CA a 220Vac.

3.2.3.1 Consumo del equipo registrador de vibraciones

En la tabla 6 se muestra el consumo eléctrico, todos los datos fueron extraídos de sus hojas de datos del fabricante.

Tabla 6

Consumo eléctrico general

Dispositivo	Consumo eléctrico
Hércules RM46L	880mA
LCD	200mA
Back light LCD	400mA*Dutty cycle
Otros	100mA
Total	1180+400*Dutty cycle

Fuente: Propia

La alimentación de la pantalla es directamente proporcional al dutty cycle al 100%, por lo que resulta un consumo de 400mA en el back light LCD. El consumo total es de 1580mA, tomando en consideración que el back light LCD este encendido siempre.

El consumo eléctrico del microcontrolador en ocasiones puede cambiar de 880mA a 1A, siempre y cuando el microcontrolador esté realizando operaciones matemáticas complejas como por ejemplo la transformada rápida de Fourier para visualizar la firma del espectro de la vibración.

En la tabla 7 se muestra las baterías que se tuvo en consideración en la selección.

Tabla 7

Selección de batería

Item	Xiaomi Power Bank	Parkman H1	Gmobile Modern H1
Capacidad	20000 mAh	4000 mAh	4000 mAh
Entrada	5V-2.1A / 9V-2.1A / 12V-1.5A	5V-1A	5V-2A
Salida con el usuario	5.1V-2.4A / 9V-2A / 12V-1.5A	5.1V-2.1A	5V-2.1A / 5V-1A

Fuente: Propia

Se escogió la batería Gmobile Modern H1, ya que posee las características mencionadas de la tabla 6, además es de fácil acceso a ella en el mercado.

3.2.3.2 Cálculo de la duración de la batería en alto consumo del equipo

Itotalequipo: Corriente total que consume el equipo.

Ibateria: Corriente suministrada desde la batería.

HT: Hora total de duración de batería.

$$HT = \frac{Ibateria}{Itotalequipo}$$

$$HT = \frac{4000mAh}{1580mA}$$

$$HT = 2.53\ h$$

Esto quiere decir que nuestra batería puede durar hasta 2 horas con 32 minutos considerando un alto consumo por el equipo.

Para la duración de batería en bajo consumo es necesario restar la corriente del blacklight LDC de la corriente total que consume el equipo.

Itotalequipo: Corriente total que consume el equipo.
Ibateria: Corriente suministrada desde la batería.
Ilcd: Corriente en blacklight LCD.
HT: Hora total de duración de batería.

$$HT = \frac{Ibateria}{Itotalequipo - Ilcd}$$

$$HT = \frac{4000mAh}{1580mA - 400mA}$$

$$HT = 3.39\ h$$

Esto quiere decir que nuestra batería puede durar hasta 3 horas con 23 minutos considerando un bajo consumo por el equipo.

3.2.4 Etapa del sistema de reloj y memoria

En la figura 47 se muestra el diagrama de bloques del sistema de reloj y memoria.

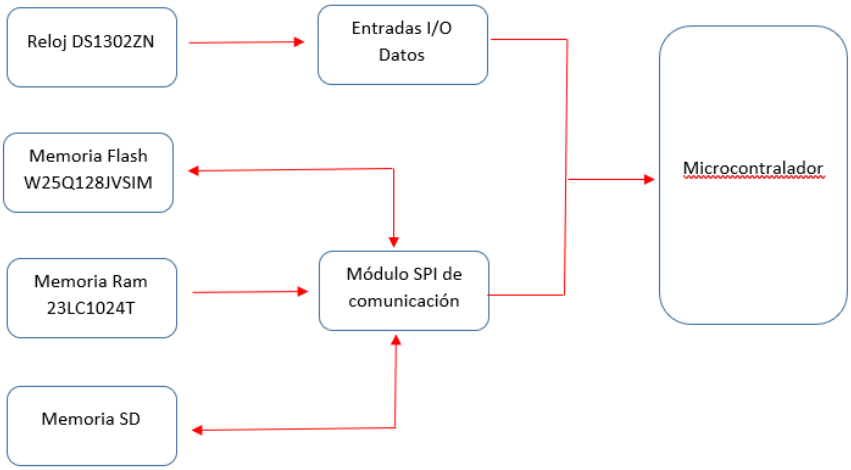


Figura 47: Diagrama de bloques del sistema de reloj y memoria

Fuente: Propia

Reloj en tiempo real, es muy importante ya que nos brinda información de la hora y calendario. Las memorias RAM y FLASH externas al microcontrolador donde se puede configurar lo siguiente: Valores de inicialización, Banderas de inicialización, las diferentes configuraciones del LCD, interrupciones. También guarda los nombres de los archivos de almacenamiento de la memoria SD, esta memoria SD extraíble puede contener información de la firma del espectro de vibración y se muestre en una tabla Excel, por ende, se consiguió una memoria amplia para almacenar dicha información. Y con el convertidor / Búfer no inversor lograremos amplificar la corriente emitida por las memorias FLASH y RAM, todo esto es para que el microcontrolador trabaje con normalidad y no tenga problemas en la transmisión de datos.

3.2.4.1 Estimación del tamaño de memoria Flash y RAM

Se tiene que priorizar los cálculos que va a realizar el microcontrolador, también es recomendable utilizar memorias externas para no sobrecargar el microcontrolador, por lo que empezaremos viendo la transformada de Fourier que lo realiza mediante el algoritmo de radix 4. El algoritmo radix 4, requiere de 2048 valores para que nos entregue 512 valores para ejecutar la transformada de Fourier, también se requiere realizar el promediado, cuando el número de promedios es alto es mejor, por lo que tomaremos entre 50 y 60 promedios aproximadamente, entonces la cantidad total que se requiere es de 122880 (50 promedios x 2048 valores). Además, se está considerando datos en int16 para

la programación que equivalen a 2Bytes por lo que en total se requiere 245.760 KBytes (122880 x 2Bytes).

3.2.4.2 Memorias internas del equipo portátil registrador de vibraciones

Memoria RAM

En la tabla 8 se muestra la selección de memoria RAM del equipo.

Tabla 8

Selección de memoria RAM

Item	23LC1024T-I_SN	PCF8570
Alimentación	2.5 a 5.5V	2 a 6V.
Frecuencia de reloj	20MHz	400KHz
Temperatura	-40 ° C a + 85 ° C.	-40 ° C a + 85 ° C.
Tamaño	128KByte.	128KByte

Fuente: Propia

Se escogió dos unidades de 23LC1024T-I_SN, por la cantidad de datos que se requieren, y también el rango de frecuencia de reloj es de 20MHz es alta y es acorde a nuestra aplicación y es más accesible en el mercado. Además, esta es la memoria más actual hasta la fecha para este tipo de aplicación, en un futuro se puede aplicar una ESP-PSRAM64H, que es de 8MBYTES.

Memoria FLASH

Se muestra en la tabla 9 características de la memoria flash a utilizar.

Tabla 9

Características de la memoria Flash W25Q128JVSIM

Item	Descripción
1	Rango de voltaje de alimentación: 2.5 a 5.5V de: 2.3 a 3.6V.
2	Rango de temperatura industrial: -40 ° C a + 85 ° C.
3	256K Byte por página programable.
4	Velocidad de transferencia continua de: 25MB/S.

Fuente: Propia

En el caso de la memoria flash W25Q128JVSIM solo se pudo obtener esta memoria ya que es la única que es compatible con el microcontrolador Texas instruments, también hay otras compatibles, se seleccionó por el tamaño de memoria.

Memoria SD

En la tabla 10 se muestra las características de las memorias externas que se tuvo en consideración.

Tabla 10

Características de la memoria SD

Descripción	Velocidad	Voltaje de operación	Temperatura
Memoria de 16 Gb	Clase 10(10MB/seg)	3.3V	-25ºC a 85ºC
Memoria de 8 Gb	Clase 4(4MB/seg)	3.3V	-25ºC a 85ºC
Memoria de 4 Gb	Clase 4(4MB/seg)	3.3V	-25ºC a 85ºC

Fuente: Propia

Se escogió una memoria de 8Gb SD, ya que la capacidad de almacenamiento y velocidad es acorde a nuestra aplicación.

3.2.4.3 Convertidor / Búfer no inversor

En la tabla 11 se muestra la descripción del convertidor / búfer no inversor a utilizar.

Tabla 11

Características del convertidor búfer no inversor CD4050

Item	Descripción
1	Rango de voltaje de alimentación: 3 a 15Vdc.
2	Capacidad de corriente de alta fuente.
3	Impulsión directa a 2 cargas TTL a 5V a temperatura máxima.

Fuente: Propia

En este caso para el convertidor solo se pudo obtener el CD4050 y es más accesible en el mercado.

3.2.4.4 Reloj en tiempo real

En la tabla 12 se muestra la selección del reloj en tiempo real.

Tabla 12

Selección de reloj en tiempo real

Item	DS1302ZN	DS1307	DS3231
Reloj en tiempo real	hasta 2100(fecha).	hasta 2100(fecha).	hasta 2100(fecha).
# de cables	3	2	3
Alimentación	:2.0 V a 5.5 V.	5	2.0 V a 5.5 V.
Temperatura	-40 ° C a + 85 ° C.	-40 ° C a + 85 ° C.	-40 ° C a + 85 ° C.

Fuente: Propia

Se escogió el DS130ZN ya que es fácil en la conexión y es más accesible en el mercado.

3.2.4.5 Circuito esquemático de la etapa del sistema de reloj y memoria

A continuación, se muestra la figura 48 del circuito esquematico de la etapa del sistema de reloj y memoria.

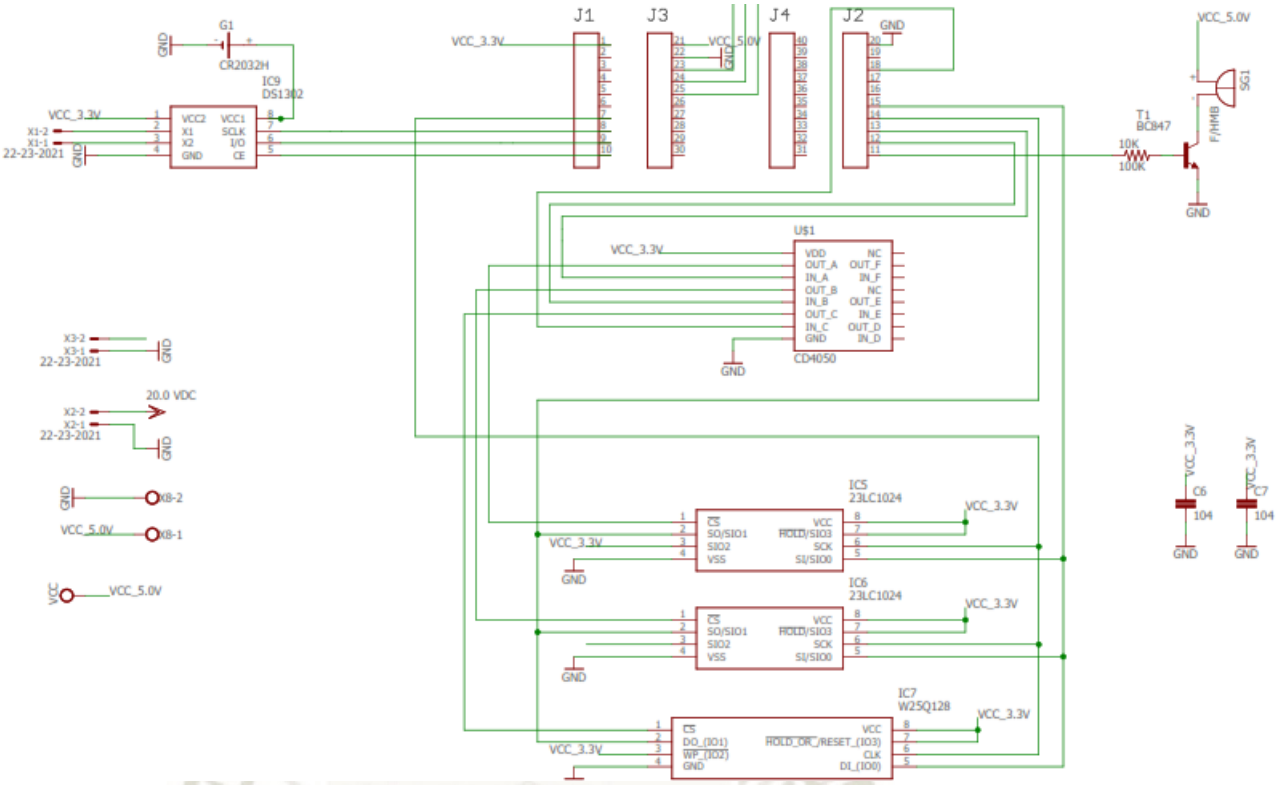


Figura 48: Circuito esquemático de la etapa del sistema de reloj y memoria

Fuente: Propia

3.2.5 Microcontrolador

El microcontrolador tiene como función principal realizar operaciones matemáticas como la transformada de Fourier, realizar el procesamiento de la señal de vibración, para luego transmitirla a través de la pantalla LCD.

En la tabla 13 se muestra la selección del microcontrolador.

Tabla 13

Selección de Microcontrolador

Item	ARM CORTEX R4F RM46L852	ARM CORTEX R5F RM57L843	ARM CORTEX R4 RM42L432
Reloj	220MHz	330MHz	100MHz
Punto flotante	si	si	si
Memoria FLASH	1.25MB	4 MB	384 KB
Memoria RAM	192KB	512 KB	32 KB
Memoria EEPROM	64KB	128 KB	16 KB
Alimentación	3 a 3.6V	3 a 3.6V	3.3V

Fuente: Propia

Se escogió el microcontrolador ARM CORTEX R4F RM46L852 ya que con el podemos realizar los cálculos matemáticos como la transformada de Fourier sin ningún problema, no se consideró los PICS porque con ello no podemos realizar estos cálculos rápidamente.

3.3 DISEÑO DE LA PLACA DE CIRCUITO IMPRESO (PCB)

Para el diseño se está considerando la norma ANSI IPC 2221

3.3.1 Circuito impreso de la etapa de acondicionamiento de señal.

En la tabla 14 se estima cada componente el consumo de corriente máxima.

Tabla 14

Estimación de corriente de consumo de la etapa de acondicionador de señal

Item	Característica
Elevador de tensión	1.5 A
Sensor de vibración	20 mA
OPAMS	10 mA / 1 mA
Memorias FLASH y RAM	3 mA
CD4050	1 uA
DS1302	300nA

Fuente: Propia

Vamos a considerar dos corrientes para el cálculo ya que el PCB está divido en dos partes, la primera es de alimentación y la segunda es de control.

Para estos cálculos siempre debemos colocarnos en la peor situación, para la primera parte consideramos el doble de la corriente del elevador de tensión que es de 3A y para la segunda consideramos también el doble de la suma del resto de componentes 200 mA.

Se muestra en la tabla 15 la temperatura máxima de los componentes de la etapa de acondicionador de señal.

Tabla 15

Estimación de temperatura de los componentes de la etapa de acondicionador de señal

Item	Característica
Elevador de tensión	150ºC
Sensor de vibración	-54ºC a 121ºC
OPAMS	150ºC
Memorias FLASH y RAM	150ºC
CD4050	150ºC
DS1302	125ºC

Fuente: Propia

De la tabla 15 se considera el valor de 150ºC para ambos casos, ya que es la temperatura máxima a la que pueden llegar casi todos los componentes.

A continuación, se muestra las ecuaciones recomendadas en la norma ANSI IPC 2221, se tiene la presencia de constantes, cuyos valores son las siguientes.

K1=0.0647
K2=0.4281
K3=0.6732

ΔT=Temperatura máxima – temperatura ambiente

$$\text{Area} = \left(\frac{\text{Corriente}}{(\text{k1} * \Delta\text{T}^{\text{K2}})}\right)^{\frac{1}{\text{K3}}} (\text{pulgadas2})$$

Reemplazando en la ecuación:

Con la corriente máxima de 3A, temperatura máxima de 150Cº y temperatura ambiente de 25ºC.

$$\text{Area} = \left(\frac{3}{(\text{k1} * 125^{\text{K2}})}\right)^{\frac{1}{\text{K3}}} (\text{pulgada}^2)$$

$$\text{Area} = 13.85\ (\text{pulgada}^2)$$

Ahora con la corriente máxima de 500mA, temperatura máxima de 150Cº y temperatura ambiente de 25ºC.

$$\text{Area} = \left(\frac{0.5}{(\text{k1} * 125^{\text{K2}})}\right)^{\frac{1}{\text{K3}}} (\text{pulgada}^2)$$

$$\text{Area} = 0.96\ (\text{pulgada}^2)$$

Ahora calcularemos el ancho con las diferentes áreas calculadas, estimando que el grosor del cobre de la PCB es de 1onza/pie2 siempre poniéndonos en el peor escenario.

$$\text{Ancho} = \frac{\text{Area}}{\text{Grosor} * 1.378} \text{(milesimas de pulgada)}$$

Con el área de 13.85 pulgada2

$$\text{Ancho} = \frac{13.85}{1 * 1.378} \text{(milesimas de pulgada)}$$

$$\text{Ancho} = 10.05 \text{ (milesimas de pulgada)}$$

Con el área de 0.96 pulgada2

$$\text{Ancho} = \frac{0.96}{1 * 1.378} \text{(milesimas de pulgada)}$$

$$\text{Ancho} = 0.70 \text{ (milesimas de pulgada)}$$

El grosor de línea debe ser como mínimo 10.05 milésimas de pulgada para la parte de alimentación y para la parte de control debe ser como mínimo de 0.7 milésimas de pulgada, por lo que teniendo en consideración los datos calculados se escogieron de 24 milésimas de pulgada para la parte de alimentación y para la parte de control 12 milésimas de pulgada.

También se está considerando las siguientes recomendaciones presentadas por la norma ANSI IPC 2221.

- Los componentes deben estar por lo menos a 1.2mm del borde del PCB
- Las pistas deben estar por lo menos a 0.75mm de cualquier elemento mecánico (tornillo, tuerca, arandela, etc.)
- Los componentes altos y bajos deben estar por lo menos a 8 a 10mm de separación.
- Los componentes bajos deben estar por lo menos a 2 a 4mm de separación.
- Las esquinas de las pistas deben tener 45º de curvatura.

En la figura 49 y 50, se muestra las características de la placa de la etapa de acondicionamiento de señal.

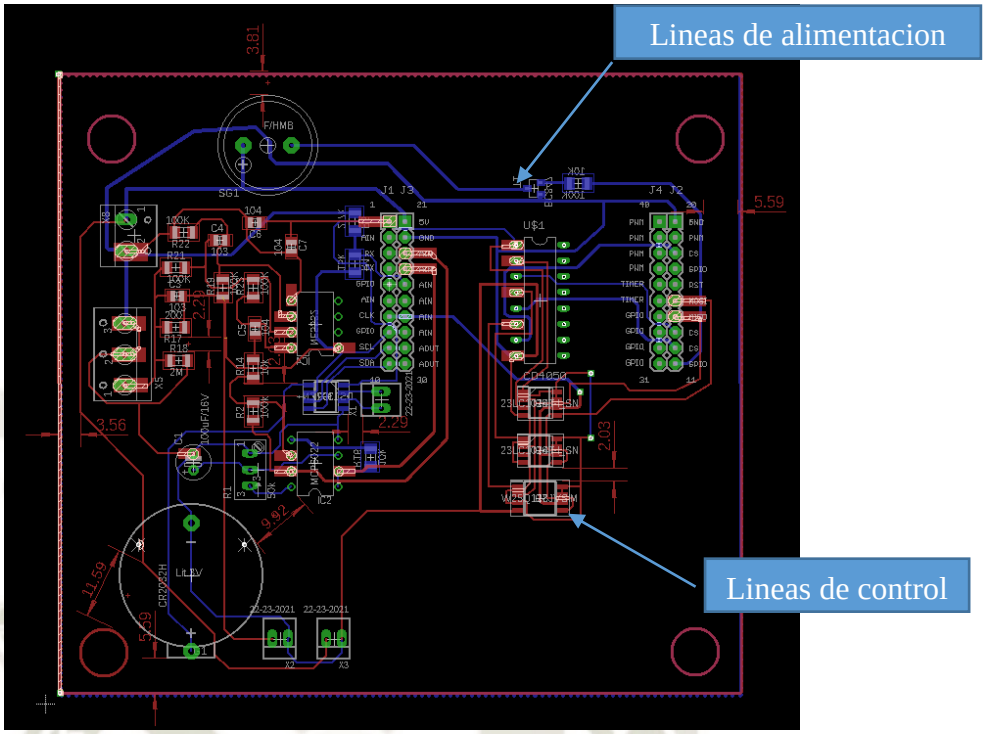


Figura 49: Dimensiones de la placa de la etapa de acondicionamiento de señal

Fuente: Propia

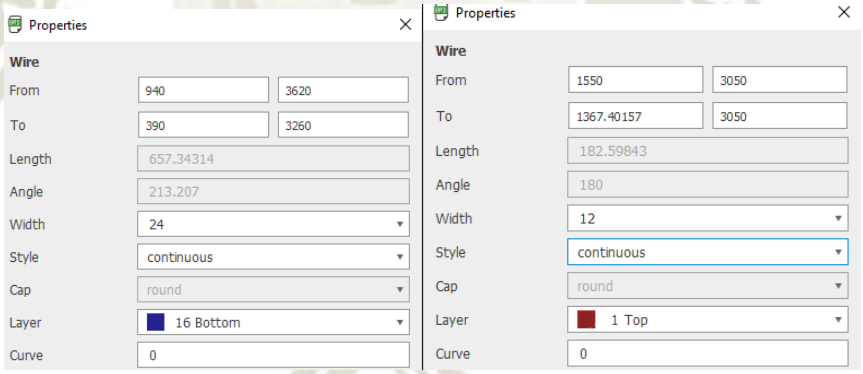


Figura 50: Propiedades del grosor de línea del PCB de la etapa de sensor de vibración y memorias

Fuente: Propia

Como se muestra en las figuras 49 y 50 se cumple con las recomendaciones de la norma ANSI IPC 2221.

3.3.2 Circuito impreso de la etapa de circuito de visualización

Para el cálculo de corriente vamos a estimar el componente que consume mayor cantidad de corriente.

Tabla 16

Estimación de corriente de consumo de la etapa circuito de visualización

Item	Característica
74HC595N	1 uA
Pantalla LCD	600 mA

Fuente: Propia

Siempre colocándonos en el peor de los escenarios vamos a considerar el doble de la corriente de la pantalla LCD, por lo que sería de 1A. En la tabla 17 se muestra la estimación de temperatura de los componentes de la etapa de circuito de visualización.

Tabla 17

Estimación de temperatura de los componentes de la etapa de circuito de visualización

Item	Característica
74HC595N	-65ºC a 150ºC
Pantalla LCD	-30ºC a 85ºC

Fuente: Propia

Siempre colocándonos en el peor de los escenarios vamos a considerar la temperatura mayor, que es del componente 74HC959N, por lo que sería de 150ºC.

A continuación, se muestra las ecuaciones recomendadas en la norma ANSI IPC 2221, se tiene la presencia de constantes, cuyos valores son las siguientes.

K1=0.0647

K2=0.4281

K3=0.6732

ΔT=Temperatura máxima – temperatura ambiente

$$\text{Area} = \left(\frac{\text{Corriente}}{(\text{k1} * \Delta\text{T}^{\text{K2}})}\right)^{\frac{1}{\text{K3}}}(\text{pulgadas2})$$

Reemplazando en la ecuación:

Con la corriente máxima de 1A, temperatura máxima de 150ºC y temperatura ambiente de 25ºC.

$$\text{Area} = \left(\frac{1}{(k1 * 125^{K2})}\right)^{\frac{1}{K3}} (\text{pulgada}^2)$$

$$\text{Area} = 2.71\ (\text{pulgada}^2)$$

Ahora calcularemos el ancho con el área calculada, estimando que el grosor del cobre de la PCB es de 1onza/pie2 siempre poniéndonos en el peor escenario.

$$\text{Ancho} = \frac{\text{Area}}{\text{Grosor} * 1.378} (\text{milesimas de pulgada})$$

Con el área de 2.71 pulgada2

$$\text{Ancho} = \frac{2.71}{1 * 1.378} (\text{milesimas de pulgada})$$

$$\text{Ancho} = 1.96\ (\text{milesimas de pulgada})$$

El grosor de línea debe ser como mínimo 1.96 milésimas de pulgada, por lo que teniendo en consideración los datos calculados se escogió de 14 milésimas de pulgada. También se está considerando las siguientes recomendaciones presentadas por la norma ANSI IPC 2221.

- Los componentes deben estar por lo menos a 1.2mm del borde del PCB.
- Las pistas deben estar por lo menos a 0.75mm de cualquier elemento mecánico (tornillo, tuerca, arandela, etc.).
- Los componentes altos y bajos deben estar por lo menos a 8 a 10mm de separación.
- Los componentes bajos deben estar por lo menos a 2 a 4mm de separación.
- Las esquinas de las pistas deben tener 45º de curvatura.

En las figuras 51 y 52, se muestra las características de la placa de la etapa de circuito de visualización.

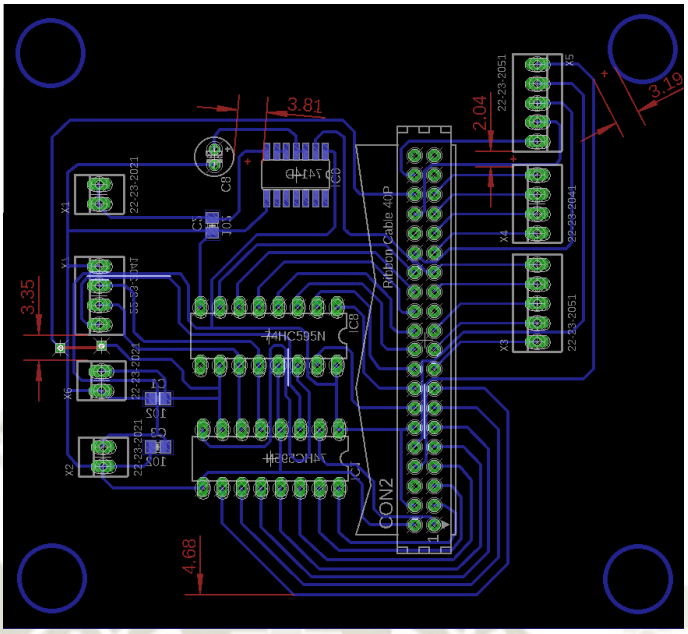


Figura 51: Dimensiones de la placa impresa de la etapa de circuito de visualización

Fuente: Propia

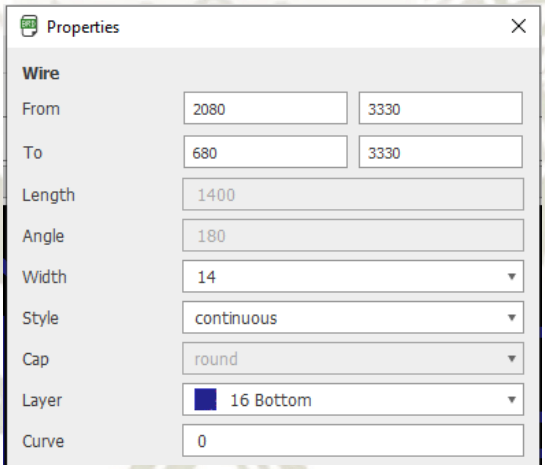


Figura 52: Propiedades de la PCB de la etapa de circuito de visualización

Fuente: Propia

3.4 DISEÑO DE SOFTWARE

A continuación, se describe la programación del equipo portátil registrador de vibraciones, como se comentó en el capítulo anterior no se está utilizando los PICS. Se está utilizando el microcontrolador RM46L de Texas instruments, el lenguaje para este microcontrolador es el C, pero se tiene varios entornos de programación que se detallaran a continuación.

3.4.1 ENTORNO DE PROGRAMACION

Para la selección del entorno de programación se tuvo en consideración lo siguiente: El ARM KEIL es un entorno de programación para los microcontroladores arm cortex de Texas instruments, en la actualidad se utiliza mucho ya que la interfaz con el usuario es muy agradable, pero se tiene una desventaja que se tiene que pagar la licencia de funcionamiento y operatividad. El CODE COMPOSER es un entorno de programación para los microcontroladores arm cortex de Texas instruments, esta aplicación se puede descargar desde la misma página de Texas instruments y es accesible a todos ya que no se tiene que pagar por una licencia. El entorno de programación que se escogió es el CODE COMPOSER de Texas instruments ya que no se necesita realizar ningún pago de licencia.

3.4.2 ACTIVACIÓN DE LOS PERIFÉRICOS DEL MICROCONTROLADOR

3.4.2.1 Interfaz grafica

Como se indicó anteriormente se está utilizando un microcontrolador de la marca Texas instruments RM46L. Este microcontrolador tiene una interfaz gráfica llamada halcogen, que junto con el composer studio ide code, nos permite programar y configurar el microcontrolador. Esta interfaz gráfica nos permite configurar los periféricos internos del microcontrolador, posteriormente ya con el programa composer studio se puede programar el microcontrolador para que realice las actividades específicas del equipo de registrador de vibraciones.

En la figura 53 se muestra el diagrama de bloques del microcontrolador, donde se puede apreciar los módulos internos del microcontrolador.

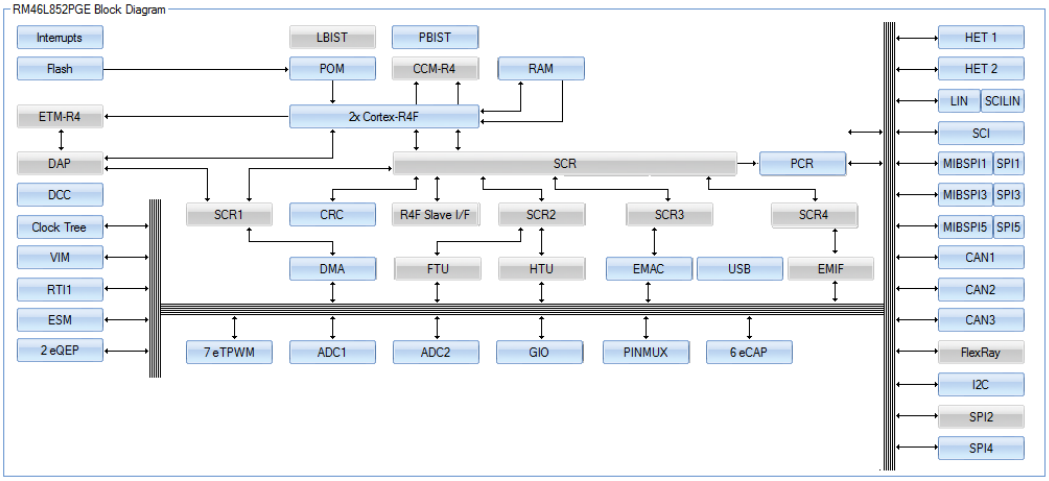


Figura 53: Diagramas de bloques del microcontrolador

Fuente: Propia

De la figura 53, los bloques de color celestes nos indican que están activados en el chip y los bloques de color gris nos indican que son reservados y realizan control de registros.

3.4.2.2 Módulo Conversor análogo digital (ADC)

Como se muestra en la Figura 53, el microcontrolador cuenta con dos módulos independientes que son: ADC1 y ADC2. El cual solo utilizaremos el ADC1. Este módulo conversor análogo digital del microcontrolador cuenta con una resolución de hasta 12 bits pero puede modificarse a 8 bits y 10bits.

El tiempo que demora el microcontrolador en total es de 2.491629us esta estimación de tiempo ya lo da el microcontrolador según muestra la Figura 55.

Se muestra en la figura 54 la distribución del módulo ADC1, que consta de 3 señales de entrada que cuenta el equipo registrador de vibraciones.

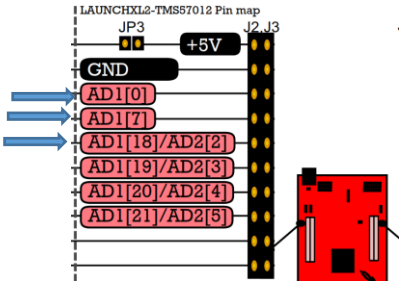


Figura 54: Distribución de canales ADC"

Fuente: Propia

Distribución de pines activos.

AD1 (0): En este canal ingresa la señal de vibración + offset

AD1 (7): En este canal ingresa la señal del circuito de offset

AD1 (18): Señal del circuito de entrada de voltaje de sensor de vibración

Nota: El voltaje offset es de 1.5 voltios aproximadamente [10].

Entonces la señal de vibración estaría compuesta por:

$$\text{Señal de vibracion}[N] = AD1[0] - AD1[7]$$

En consecuencia, los voltajes de offset se restarán y solo quedara el voltaje emitido por la señal de vibración.

El AD1 [18] se está utilizando para monitorear el voltaje del equipo portátil registrador de vibraciones.

En la figura 55 se muestra el diagrama de bloques del ADC 1.

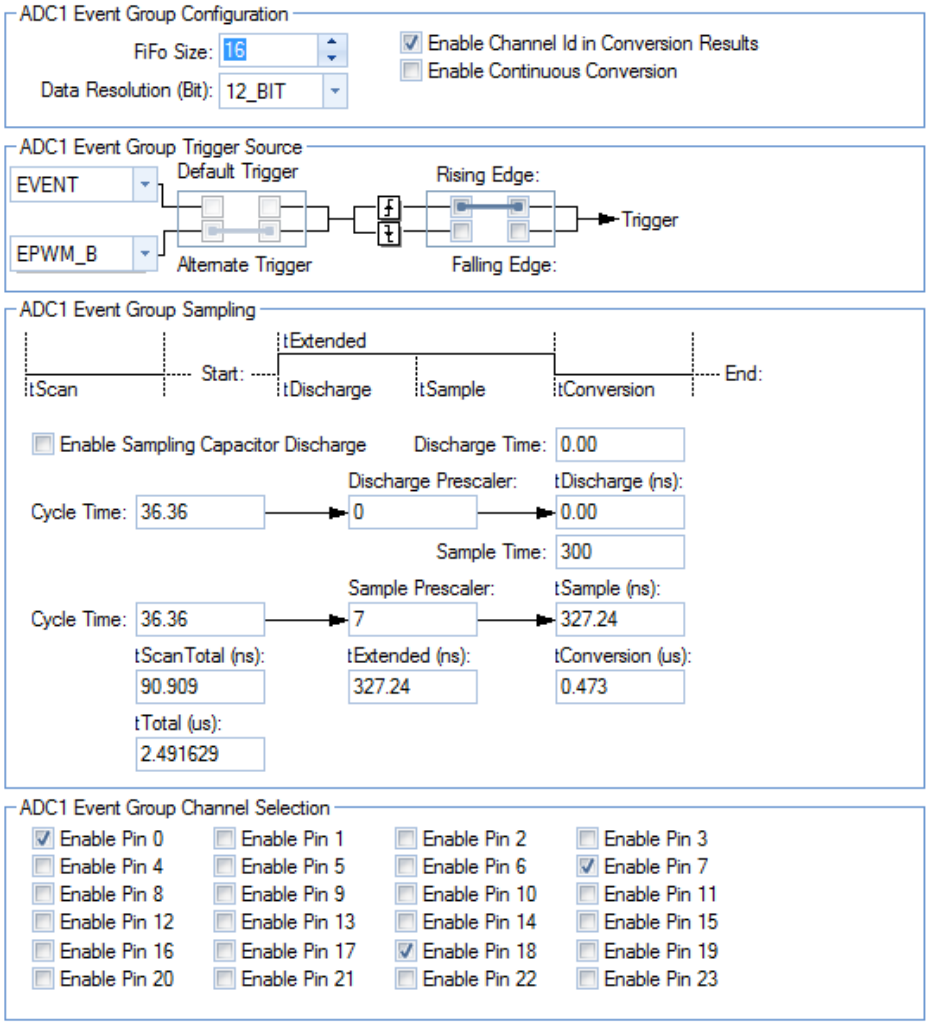


Figura 55: Diagrama de bloques del ADC 1

Fuente: Propia

Ahora escogeremos una resolución de 12 bits, ya que en White (1990) recomienda usar esa resolución. Ya que más bajo será el nivel de ruido y más grande será el rango dinámico. El voltaje máximo es de 3.3 Vdc, para calcular el bit menos significativo (LSB).

$$LSB = \frac{Vmax}{2^n}$$

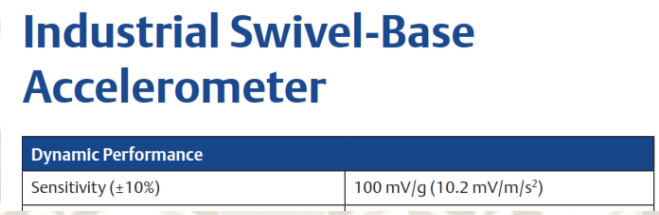
$$LSB = \frac{3.3}{2^{12}}$$

$$LSB = 805.66uV$$

3.4.2.3 Sensibilidad de la vibración

La sensibilidad del sensor de vibración está definida por el fabricante para este caso de la marca Emerson, que es un acelerómetro. A continuación, se muestra la sensibilidad del sensor.

Se muestra en la figura 56 la sensibilidad del acelerómetro.



Industrial Swivel-Base Accelerometer

Dynamic Performance	
Sensitivity (±10%)	100 mV/g (10.2 mV/m/s²)

Figura 56: Sensibilidad del acelerómetro

Fuente: Hoja de datos del acelerómetro EMERSON

3.4.2.4 Módulo de ETPWM de alta precisión para el ADC

El modulo ETPWM sirve para controlar el ADC, mediante el periodo de muestreo, que quiere decir que cada frecuencia de muestreo que pase, captura una cierta cantidad de datos con el ADC. Es por eso que se tiene que definir la frecuencia de muestreo.

Cálculo del periodo de muestreo

Como dato tenemos la frecuencia en la que se va a muestrear los datos que es de 40KHz.

$$Ts = \frac{1}{Fs}$$

$$Ts = \frac{1}{40KHz}$$

$$Ts = 25us$$

Se escogió 40 KHz, ya que nuestro acelerómetro tiene un ancho de banda de 10KHz, pero como estos datos se van a utilizar con librerías DSP para calcular la transformada de Fourier, esto implica que para realizar el cálculo se multiplique por 4 veces el ancho de banda, ya que la transformada de Fourier se realiza con el algoritmo

radix4. Entonces cumple con el criterio de Nyquist, que dice que la frecuencia de muestreo sea mucho más grande que el ancho de banda.

En la figura 57 se muestra el diagrama de bloques del ETPWM.

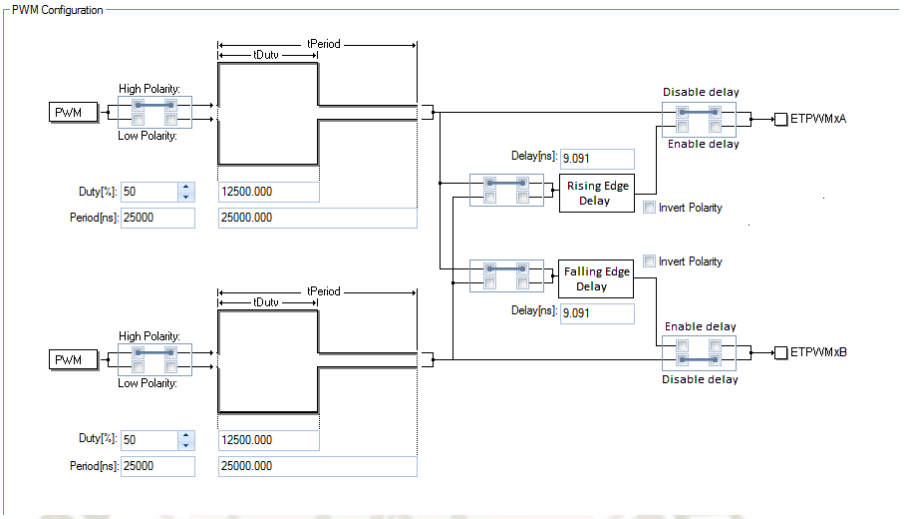


Figura 57: Diagrama de bloques del ETPWM

Fuente: Propia

3.4.2.5 Módulo temporizador de alta precisión HET y GPIO

Módulo HET

El microcontrolador RM46L multiplexa el módulo HET y se puede modificar como entradas y salidas de datos digitales (GPIO). Cada uno de estos pines se puede modificar como temporizadores, contadores y salida del PWM. Con estas entradas y salidas digitales estamos controlando la pantalla táctil TFT. Como se comentó en el capítulo 3 de esta tesis, la frecuencia de control es de 10KHz, por lo que el periodo de muestreo es de 100us.

En la figura 58 se muestra el módulo HET del diagrama de bloques del PWM.

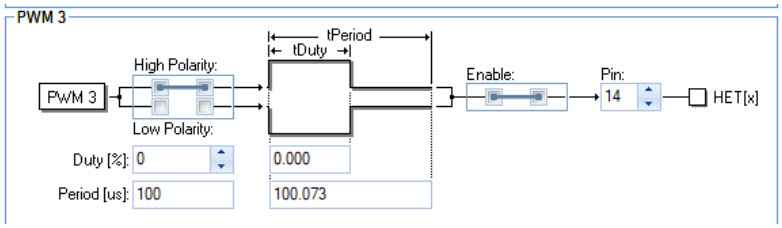


Figura 58: Módulo HET del diagrama de bloques del PWM

Fuente: Propia

En la figura 59 se muestra el diagrama de configuraciones como I/O digital.

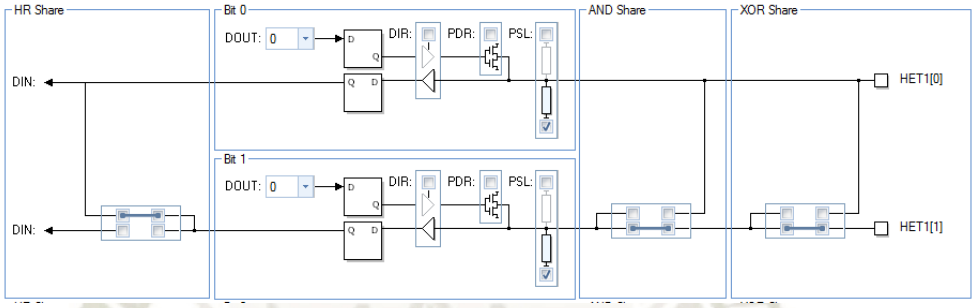


Figura 59: Diagrama de configuraciones como I/O digital

Fuente: Propia

3.4.2.6 Módulo de interrupción en tiempo real (RTI)

Este módulo consta de varias funciones de temporización y por diferentes contadores que tiene el microcontrolador.

Diagrama de módulo RTI general

Este módulo contiene dos módulos contadores y un módulo comparador. Todos los datos ya vienen pre configurados, ya que va a depender de la frecuencia de trabajo del microcontrolador. Después de realizar la temporización se activa la bandera de interrupción y se ejecuta la actividad para la que fue programada.

En la figura 60 se muestra el diagrama de bloque RTI general.

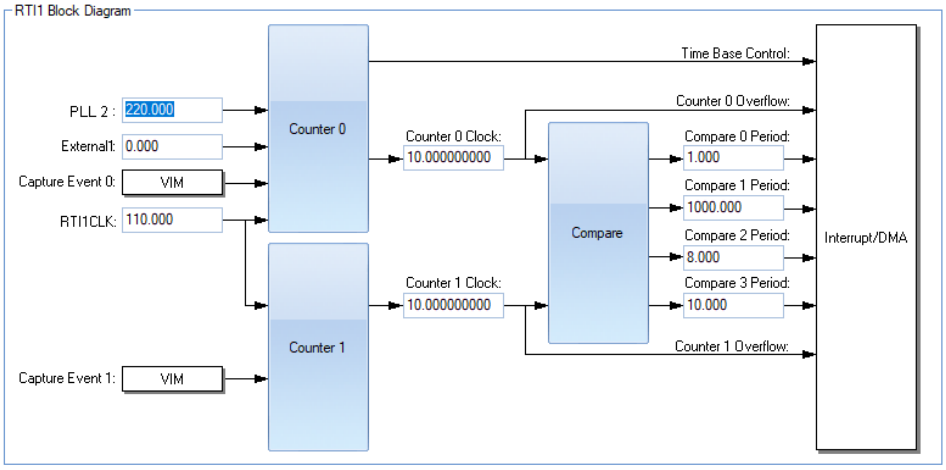


Figura 60: Diagrama de bloque RTI general

Fuente: Propia

Diagrama de comparadores

A continuación, se describe los comparadores a utilizar, el periodo ya viene pre configurado. En la tabla 18 se muestra los comparadores utilizados.

Tabla 18

Comparadores utilizados

Comparador	Periodo(ms)	Actividad
3	10	En este comparador se encarga de mantener la comunicación de la memoria SD, lo realiza mediante un temporizador.
1	1000	En este comparador se encarga de refrescar la pantalla LCD TFT cada segundo, para mantener la percepción de que esta en tiempo real

Fuente: Propia

3.4.2.7 Módulo interfaz de comunicación serial (SCI)

En la tabla 19 se muestra el Módulo SCI utilizado

Tabla 19

Modulo SCI utilizado

Modulo	Actividad	Baudrate (Hz)	Otros
SCI 2	Establece la comunicación con la computadora y la administración de la memoria SD.	115200	1 Bit stop sin paridad

Fuente: Propia

El módulo SCI 2 está configurado para establecer la comunicación con el hyperterminal de la computadora y también se puede administrar datos de la memoria SD. El valor de la velocidad de transmisión de datos ya lo recomienda el fabricante que se debe usar en esa velocidad.

En la figura 61 se muestra el diagrama de bloques SCI2.

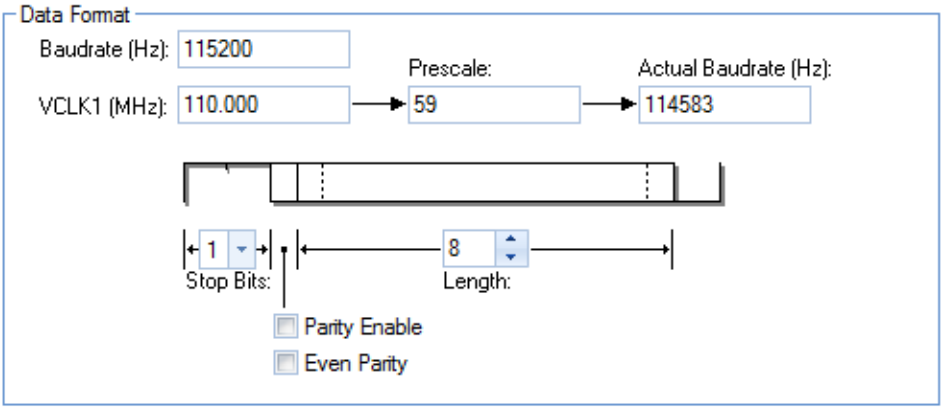


Figura 61: Diagrama de bloques SCI2

Fuente: Propia

3.4.2.8 Módulo interfaz serial periféricos (SPI)

En la tabla 20 se muestra los periféricos utilizados por el equipo registrador de vibraciones. Estos datos son los máximos que puede generar el microcontrolador.

Tabla 20

Asignación de tareas por módulo SPI

Módulo SPI	**Baudrate (MHz)**	**Módulos conectados**
SPI3	30	Este módulo está configurado para controlar la memoria RAM 23LC1024 y la memoria FLASH W25Q128
SPI5	10	Este módulo está configurado para controlar la memoria SD
SPI1	22	Este módulo está configurado para la comunicación con el módulo que controla la pantalla Touch SSD1963.

Fuente: Propia

3.4.3 DIAGRAMA DE FLUJO DEL SISTEMA OPERATIVO

3.4.3.1 Diagrama de flujo del sistema operativo principal

A continuación, se muestran en las figuras 62, 63, 64 y 65 los diagramas de flujo del sistema operativo principal.

Nota: (*) Solo se visualiza en programa, no se realizó diagrama de flujo

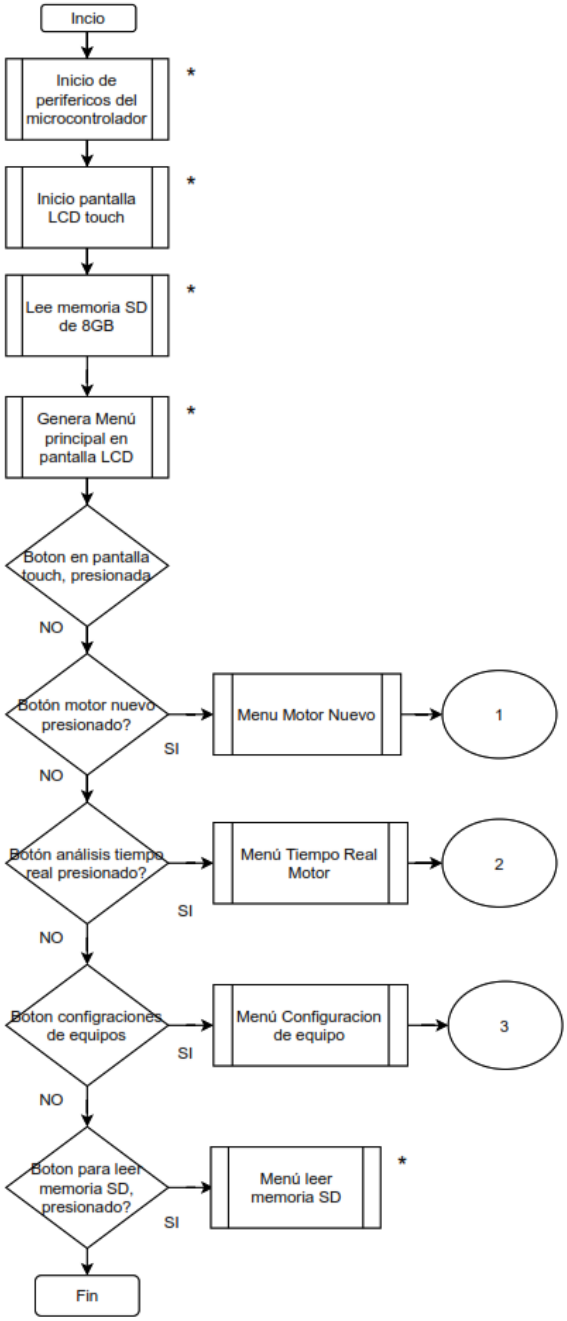


Figura 62: Diagrama de flujo del sistema operativo principal

Fuente**:** Propia

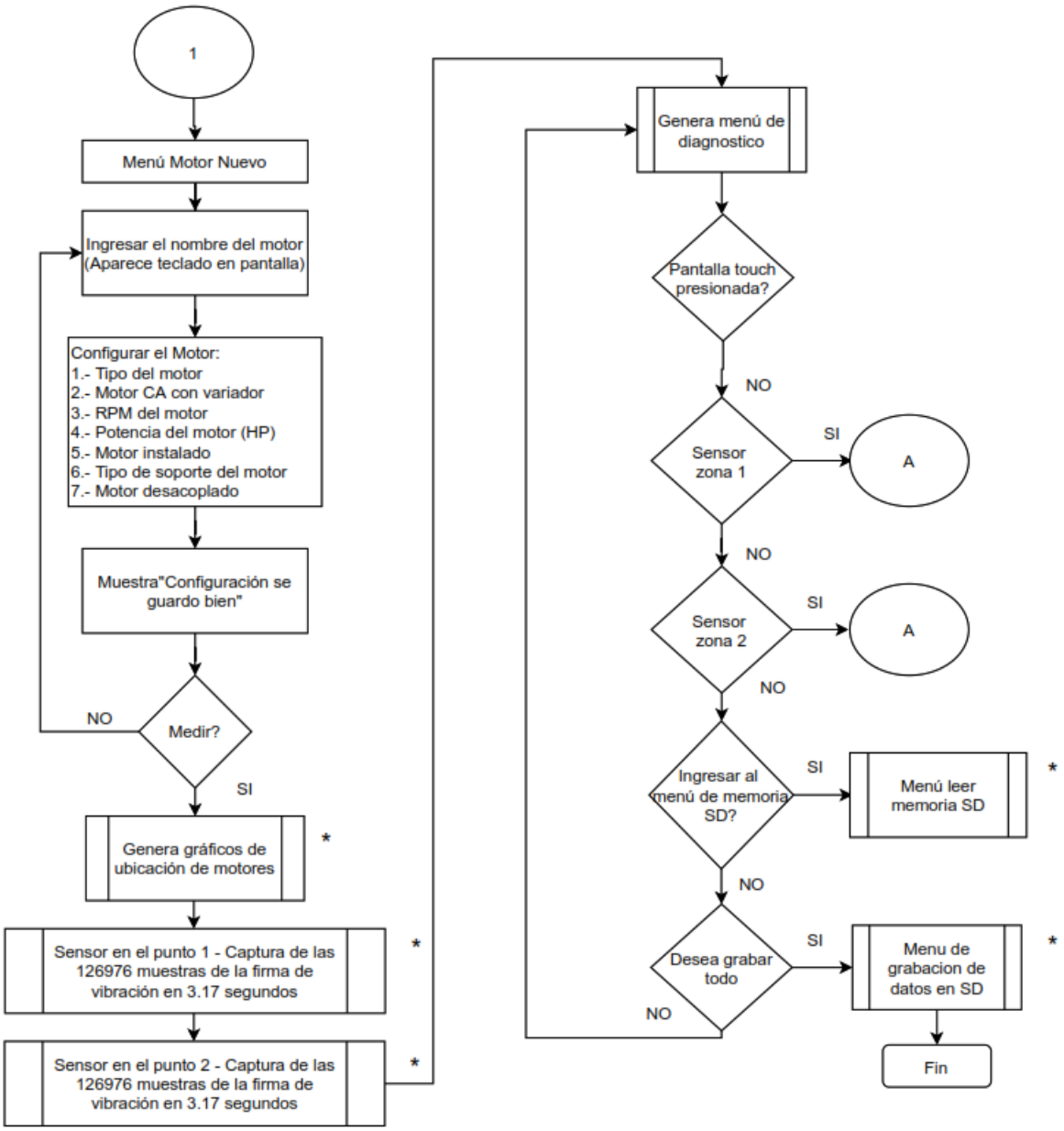


Figura 63: Diagrama de flujo del Menú Motor Nuevo

Fuente: Propia

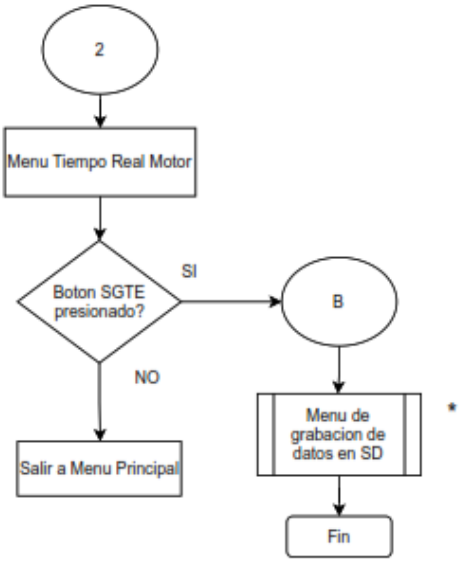


Figura 64: Diagrama de flujo del Menú tiempo Real Motor

Fuente: Propia

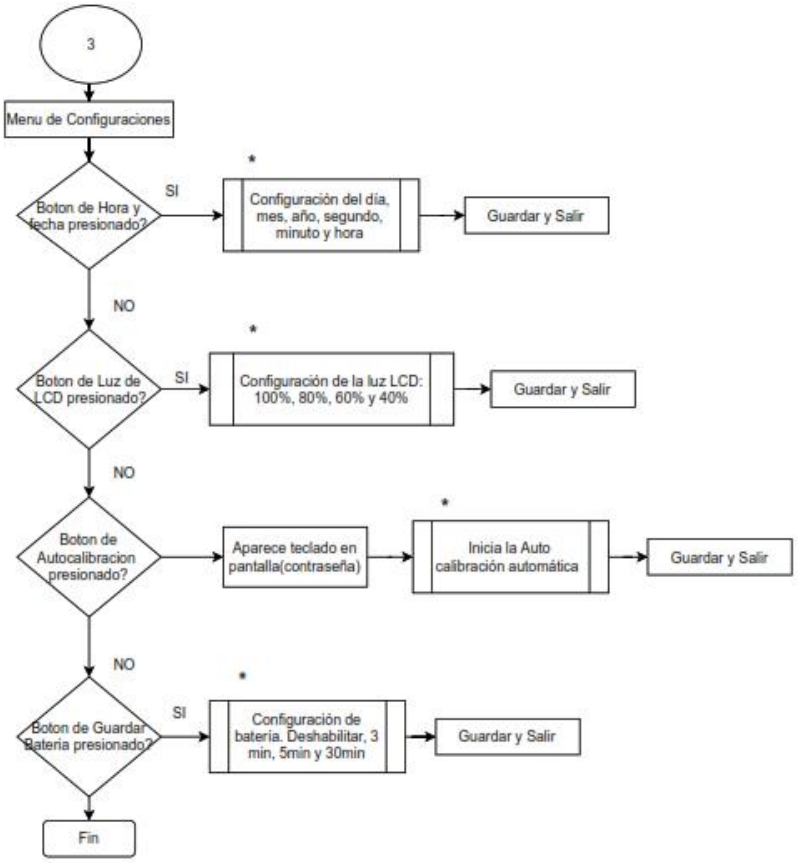


Figura 65: Diagrama de flujo de Menú de Configuraciones

Fuente: Propia

Funcionamiento del diagrama de flujo del sistema operativo principal

Como se muestra en la figura 66 y 67, para dar inicio al sistema operativo del equipo, se tiene que inicializar los módulos de internos o periféricos, de igual forma las librerías aplicadas de Texas instruments, luego inicia la pantalla LCD, luego lee la memoria SD, luego se genera el menú principal, donde se puede visualizar los submenú de "motor nuevo", "tiempo real motor", "configuración ", "SD-Leer", "Apagar", también muestra la cantidad de memoria libre y visualiza la cantidad de batería que queda en el equipo.

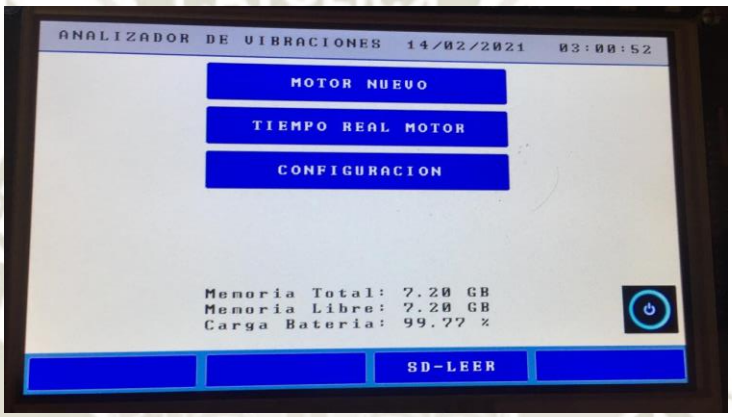


Figura 66: Menú principal del equipo

Fuente: Propia

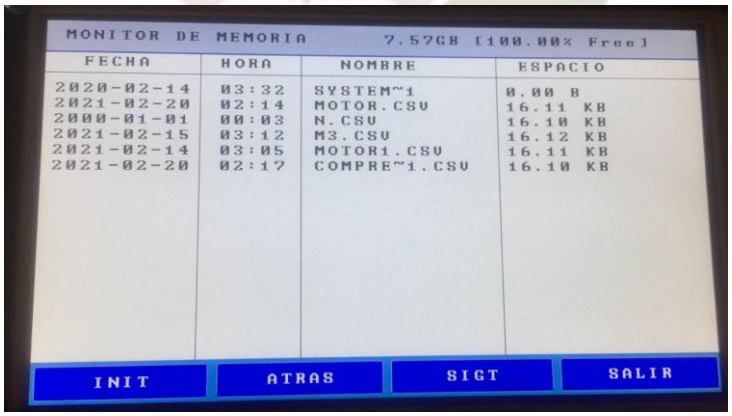


Figura 67: Menú de memoria SD

Fuente: Propia

Funcionamiento del diagrama de flujo del Menú Motor Nuevo

Como se muestra en la figuras 68, 69, 70, 71, 72, 73, 74 y 75, se describe el funcionamiento del submenú A, primero se tiene que escribir el nombre del archivo donde se guardara los datos, luego se muestra las opciones para describir los datos del motor, luego se mostrara la opción de medir, se da la opción de ubicar el sensor en las zonas donde están ubicados los rodamientos, luego se ejecuta el motor de cálculo en análisis profundo, se muestra el espectro en la pantalla LCD, donde se podrá observar el espectro de velocidad y aceleración, para luego guardarlo en la memoria SD. A continuación, se hará una demostración.

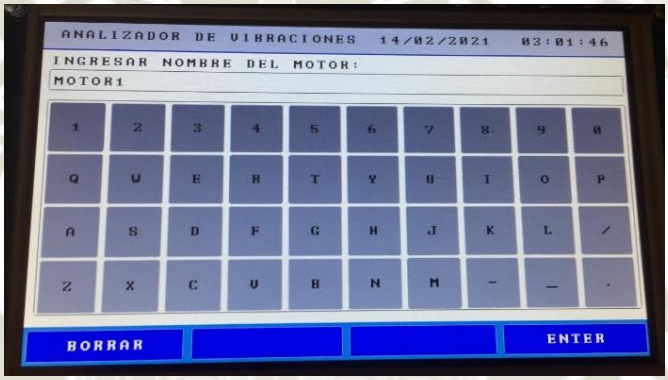


Figura 68: Introducción del nombre del archivo

Fuente: Propia

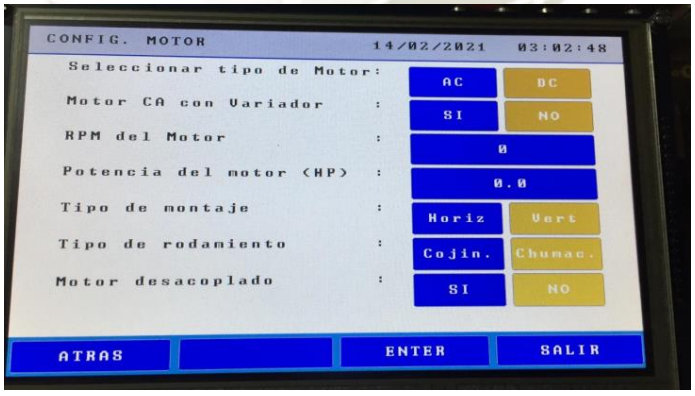


Figura 69: Selección de los datos del motor

Fuente: Propia

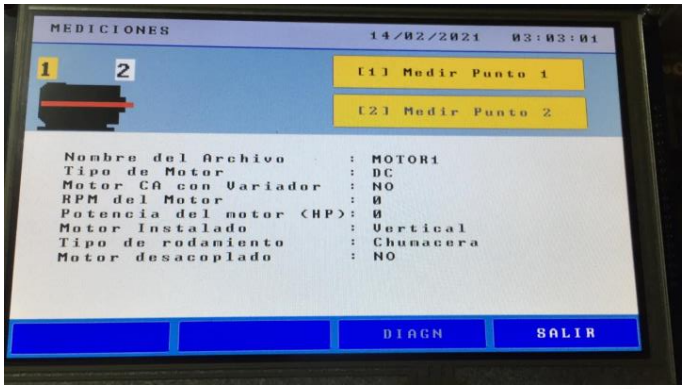


Figura 70: Selección del punto de medición

Fuente: Propia

Figura 71: Selección de la ubicación para ver los resultados

Fuente: Propia

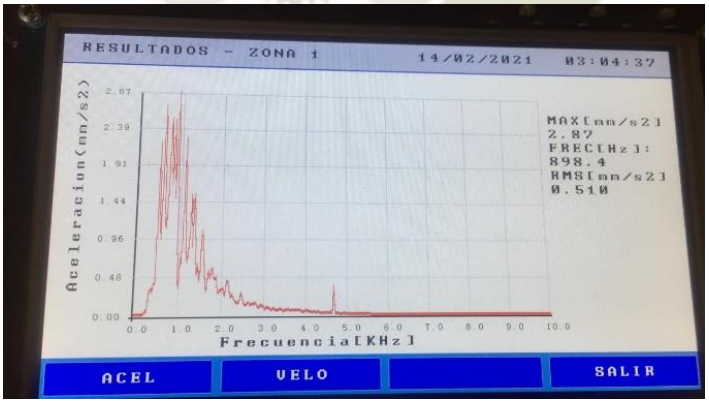


Figura 72: Resultados en la zona 1-aceleracion

Fuente: Propia

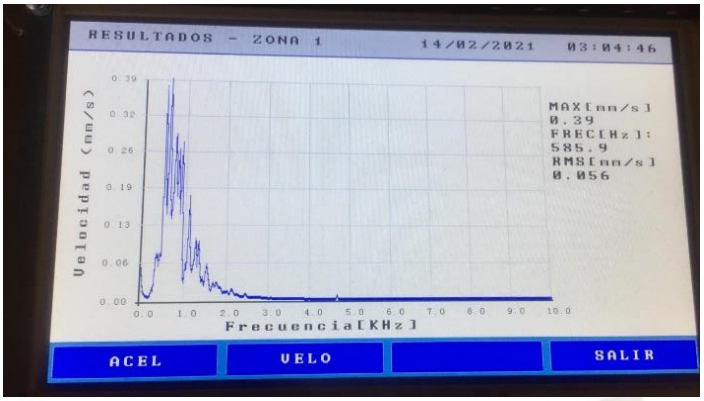


Figura 73: Resultados en la zona 1 - Velocidad

Fuente: Propia

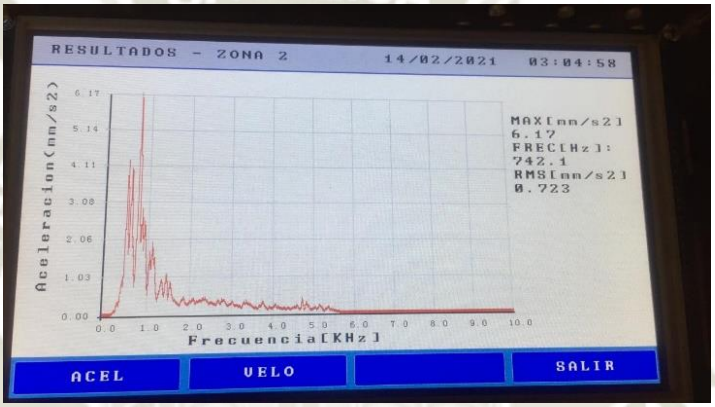


Figura 74: Resultados en la zona 2- aceleración

Fuente: Propia

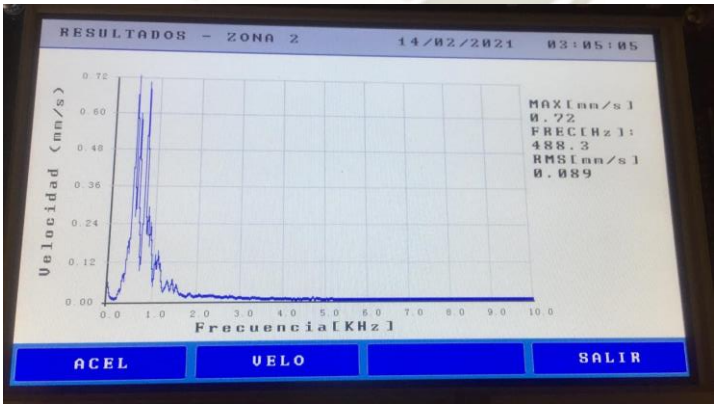


Figura 75: Resultados en la zona 2 - velocidad

Fuente: Propia

Funcionamiento del diagrama de Menú Tiempo Real

Como se muestra en la figura 76 y 77, se describe el funcionamiento del submenú B, primero se muestra el nombre del archivo donde se guardará los datos de vibración, también el voltaje del sensor de vibración, luego se muestra la pantalla LCD, donde se ejecutará el motor de cálculo de análisis en modo de tiempo real, el sensor estará activo y muestra los resultados de cualquier vibración que capte dicho sensor en un segundo.

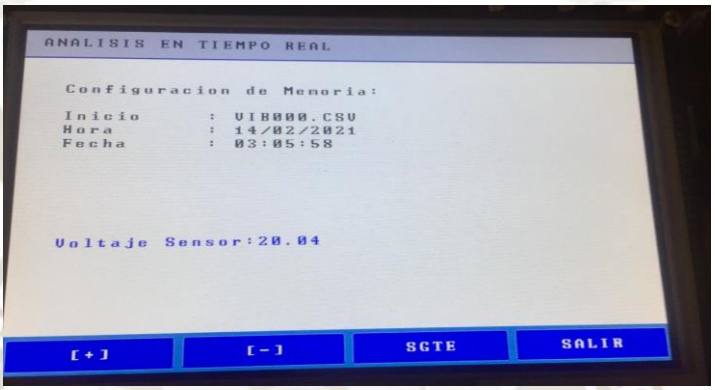


Figura 76: Selección del archivo en modo tiempo real

Fuente: Propia

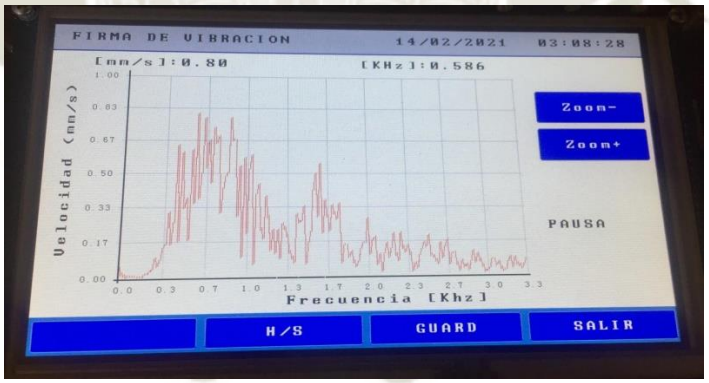


Figura 77: Espectro en modo tiempo real

Fuente: Propia

Funcionamiento del diagrama de Menú de Configuraciones

En esta parte del se describe el funcionamiento del submenú C, primero se muestra un menú donde están las siguientes opciones, “Hora y fecha”, “Luz de fondo”, “Auto calibración” y “Guardar batería”, donde se podrá modificar todo lo referente al equipo, a continuación, se explicará en que consta la auto calibración. La auto calibración consta de una medición en vacío, es decir que el sensor de vibración capta la vibración de nuestro entorno para luego restarlo con la señal capturada del motor a medir, esto se hace una sola vez, por lo que se tiene con contraseña, con el fin de que las firmas de vibración no tengan vibraciones del entorno que nos rodea, ya que pueden interferir en el análisis del espectro.

Como se muestra en la figura 78, 79, 80, 81 y 82 se muestra las opciones de configuración del equipo implementado.

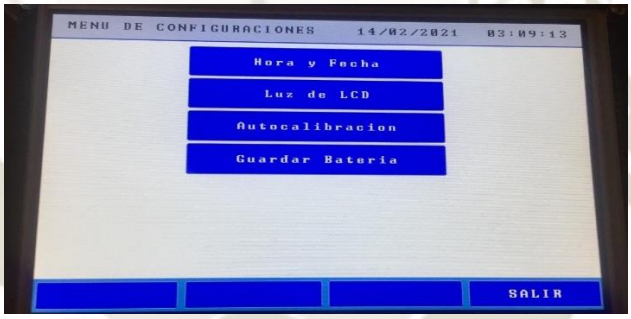


Figura 78: Selección del sub menú C

Fuente: Propia

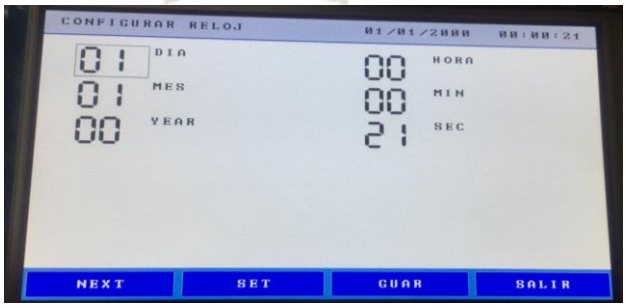


Figura 79: Configuración de la hora y fecha

Fuente: Propia

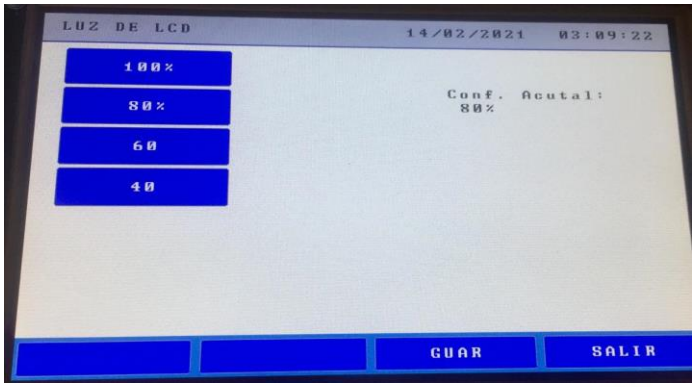


Figura 80: Selección de la luz de fondo

Fuente: Propia

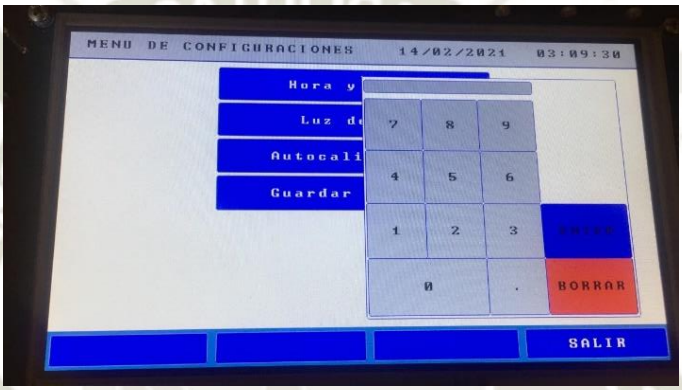


Figura 81: Ingreso de contraseña para la auto calibración

Fuente: Propia

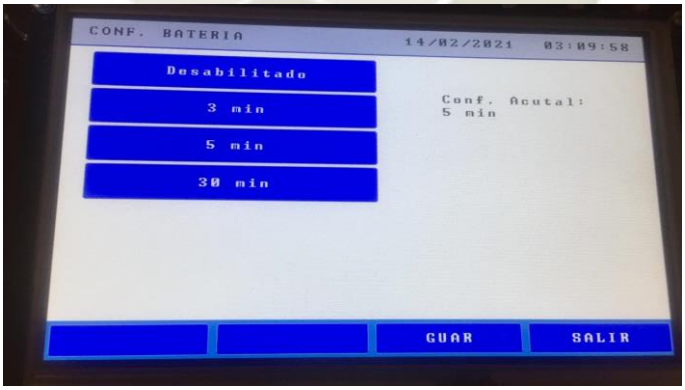


Figura 82: Selección del tiempo de espera del equipo

Fuente: Propia

3.4.4 DIAGRAMA DE BLOQUES DEL PROCESAMIENTO DE LA SEÑAL DE VIBRACION

En esta parte se describe los dos modos de análisis de vibración, que son el modo análisis profundo y el modo análisis en tiempo real, se explicará el procesamiento de la señal en ambos modos de análisis. También se explicará el motor de cálculo, ya que el procesamiento de la señal está en el motor de cálculo, y los dos modos de análisis solo dependen de la cantidad de muestras tomadas.

En la figura 83 se muestra el diagrama de bloques del procesamiento de la señal de vibracion.

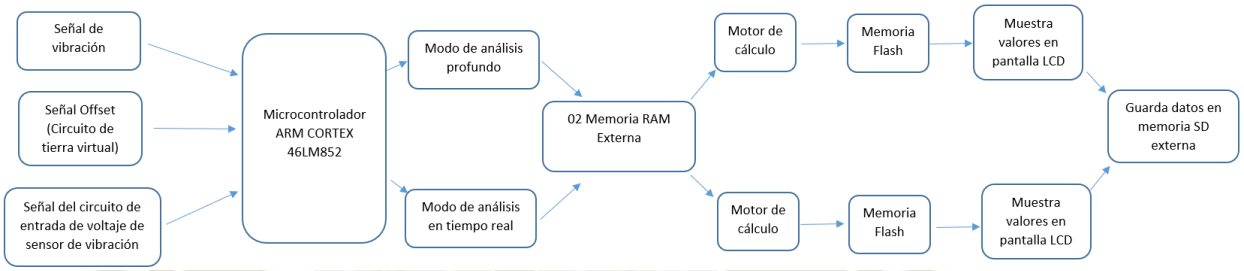


Figura 83: Diagrama de bloques del procesamiento de la señal de vibración

Fuente: Propia

3.4.4.1 Motor de cálculo

Cálculo de la resolución de frecuencia

Para el cálculo de la resolución de frecuencia se requiere el número de líneas y el rango de frecuencia que se van a mostrar en la pantalla LCD. El rango de frecuencia es de 10KHz este dato es de la hoja de datos del sensor de vibración y el número de líneas es de 512 este dato lo da el microcontrolador en la librería de DSP.

$$\Delta F = \frac{Rango\ de\ frecuencia}{Numero\ de\ lineas}$$

$$\Delta F = \frac{10KHz}{512}$$

$$\Delta F = 19.52Hz$$

En conclusión, esta respuesta nos da a entender que entre cada línea hay una separación de 19.52Hz.

Conversión de datos de voltaje a valores de aceleración

Como se mencionó anteriormente se está usando un acelerómetro con sensibilidad de 100mV/g o 10.2mV/m/s2, por lo que objetivo principal es obtener estos datos lo más preciso con el microcontrolador, por lo que se tuvo que armar el siguiente diagrama de bloques de la Figura 62, para tener una mejor visión de la conversión de unidades.

A continuación, se hace referencia a las variables:

A(t) = Valores de aceleración en m/s2.
Acexl= Valores obtenidos por el sensor.
Vacexl= Valores de aceleración en voltaje.
Vanlog= Valores de aceleración en voltaje amplificado por 10.
VD= Valores de aceleración en m/s2 después de pasar por el ADC.

En la figura 84 se muestra el diagrama de bloques de la señal de vibración.

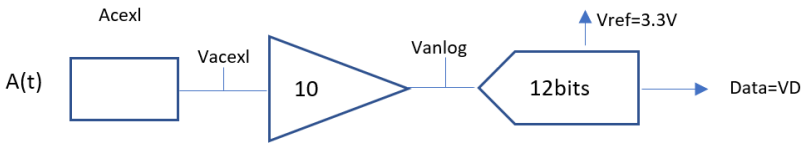


Figura 84: Diagrama de bloques de la señal de vibración

Fuente: Propia

Primero se tiene los valores que emite el acelerómetro

$$A(t) = Acxel(m/s2) = Vacexl/sensibilidad$$

$$A(t) = Acxel(m/s2) = \frac{Vacexl\ [V]}{sensibilidad\ \frac{[mV]}{m/s2}}$$

$$A(t) = Acxel(m/s2) = \frac{Vaccel * 10^3}{Sensibilidad}\left(\frac{m}{s^2}\right) \ldots\ldots (1)$$

Luego se tiene el voltaje analógico (Vanlog) que se calcula de la siguiente manera:

$$\text{Vanalog} = \text{VD} * \frac{3.3}{2^{12} - 1}$$

$$\text{Vanalog} = \text{VD} * \frac{3.3}{4095}$$

Ahora el voltaje de aceleración antes de realizar la amplificación.

$$\text{Vacexl} = \frac{\text{Vanalog}}{10}$$

$$\text{Vacexl} = \frac{3.3}{4095} * \frac{\text{VD}}{10} \ldots\ldots\ldots\ldots (2)$$

Ahora reemplazamos (2) en (1)

$$\text{A(t)} = \text{Acxel(m/s2)} = \frac{\text{Vacexl} * 10^3}{\text{Sensibilidad}} \left(\frac{\text{m}}{\text{s}^2}\right)$$

$$\text{A(t)} = \frac{3.3}{4095} * \frac{\text{VD}}{10} * \frac{10^3}{10.2} * \left(\frac{\text{m}}{\text{s2}}\right)$$

$$\text{A(t)} = \frac{3.3}{4095} * \text{VD} * 9.8 * \left(\frac{\text{m}}{\text{s2}}\right) \text{ en aceleracion}$$

$$\text{A(t)} = \text{A[K]} * [9.8 \frac{\text{m}}{\text{s2}}]$$

Diseño de filtro FIR

Este filtro debe tener como frecuencia de corte 10KHz debido al sensor de vibración y debe de ser de orden 200, este orden lo recomienda en las librerías de DSP de Texas instruments para mantener la relación señal ruido de 40dB que son para señales de velocidad rápida.

En la tabla 21 y la figura 85 muestra el código y la gráfica de la señal en Matlab para el filtro FIR.

Tabla 21

Código para el filtro FIR en MATLAB

Programa en Matlab

Fs=1/25e-6;
Fc=10000;
Filter_level=200;
h = fir1(Filter_level,Fc/Fs,'low');
freqz(h,1,512);
h_10khz=fliplr(h);
fid=fopen('archivo.txt','w');
fprintf(fid,'const float32_t firCoeffs32[] = {\n');
fprintf(fid,'%0.10ff, \n',h);

Fuente: Propia

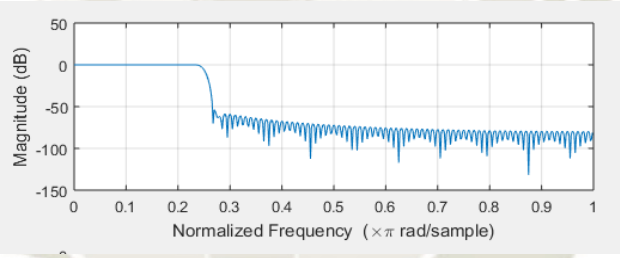


Figura 85: Filtro FIR en MATLAB

Fuente: Propia

Aplicación de la ventana hanning

A continuación, se muestra la ecuación de la ventana hanning, la aplicación de la ventana hanning lo recomienda [6].

Dónde:

N=Numero de muestras

$$Vn = 0.5 - 0.5 * \cos(\frac{2\pi n}{N - 1})$$

Aplicación de la transformada de fourier en valores de aceleración mediante el algoritmo radix 4 FFT.

La aplicación del algoritmo radix 4, requiere de varias características, primero es de tener valores de muestreo de 16, 64, 256, 1024, 2048 muestras, para que nos devuelva 512 muestras, como esta implementado con radix 4 a la cantidad de muestras se le divide 4.

En el caso de la frecuencia de muestreo se multiplica por 4, para llegar a una frecuencia optima de muestreo, tambien para que pueda aplicar la transformada de fourier los valores en tiempo deben empezar en 0 y terminar en 0, eso quiere decir que las 2048 muestras tomadas de la señal de vibracion se tiene que obligar a que empiece y termine en 0, esto se logra aplicando la ventana hanning. Este algortimo radix4 ya devuelve los valores en amplitud.

Escritura en la memoria FLASH.

Para escribir una memoria flash se debe borrar primero el sector que se desea escribir, cada sector equivale a 4KBytes, y para escribir se usa 2KBytes con la FFT radix4 en valores float. Entonces primero se debe borrar un espacio de 4KBytes y luego escribir a 2Kbytes hasta llegar a 512 muestras, ya que se está usando el algoritmo radix 4.

Conversión de datos de aceleración a datos de velocidad

Esta ecuación lo recomienda [6], donde son unidades estándares inglesas. Con esta ecuación nos facilita el cálculo, ya que si no se tendría que integrar la señal de aceleración y le daríamos más trabajo al microcontrolador.

$$Velocidad = 86.75 * \frac{Aceleracion}{frecuencia}$$

Promedio de espectros

Se está aplicando el promedio a cada ventana hanning para reducir el ruido espectral suavizándolo y hará que los picos de frecuencia se noten más. Cabe mencionar que el uso de promedios lo recomienda [6].

$$Promedio = \frac{suma\ de\ muestras(FFT)}{cantidad\ de\ muestras}$$

Conversión de datos FFT a pixeles

Después de tener los valores de la transformada de Fourier es necesario pasar estos valores a pixeles para poder graficar en la pantalla, cabe mencionar que nuestra pantalla es de 864 x 480 pixeles, por lo que el grafico está comprendido hasta los 300 pixeles, por lo que la ecuación de ploteo será la siguiente.

$$Ploteo = \frac{Valores\ de\ la\ FFT[K]}{Valor\ Maximo\ de\ la\ FFT} * 300\ pixeles$$

3.4.4.2 Modo análisis modo profundo

A continuación, se describe los cálculos de las muestras que requiere el análisis modo profundo

Cálculo de muestras para el análisis en modo análisis profundo

Se tiene 02 memorias RAM (23LC1024) de 128Kbyte por lo que en total se tiene 256Kbyte, para la programación se necesita saber el tipo de dato que se usara, como necesitamos números enteros se usara el INT16, por lo que se tomara datos cada 2Byte.

$$MemRam = \frac{256Kbyte}{2Byte} = 128K\ int16$$

Ahora calcularemos el conjunto de datos o el número de promedios que se puede tomar con las memorias RAM, se tiene 02 memorias RAM de 128Kbyte. La FFT en radix 4 requiere de 2048 muestras por lo que el número de promedios es:

$$Numero\ de\ promedios = \frac{128K\ int16}{2048}$$

$$Numero\ de\ promedios = 62.5$$

Se tiene que tomar un número entero, por lo que quedaría en 62 promedios, entonces el total de número de muestras seria.

$$Total\ de\ muestras = 62 * 2048$$

$$Total\ de\ muestras = 126976\ muestras\ en\ int16$$

Cálculo del tiempo de la toma de muestras

$$Tiempo\ total = Total\ de\ muestras * periodo\ de\ muestreo$$

$$Tiempo\ total = 126976 * 25us$$

$$Tiempo\ total = 3.174\ segundos$$

En la figura 86 se muestra el tiempo total de muestreo en el modo análisis profundo.

Figura 86: Tiempo total de muestreo en el modo análisis profundo

Fuente: Propia

Funcionamiento del modo análisis profundo

Inicialmente se activan todos los periféricos y librerías, luego empieza la lectura del ADC de la señal de vibración, para posteriormente realizar la conversión a valores de aceleración m/s2, luego se aplica e filtro digital pasa bajos FIR con una frecuencia de corte de 10KHz y de orden 200, luego de tener la señal en el rango de frecuencias hasta los 10KHz, estos datos se envían al grupo de memorias RAM para almacenarlas hasta un límite de 126976 muestras, que son divididas en 62 porciones de 2048 muestras, a cada grupo de muestra de 2048 se aplica la ventana hanning para luego calcular la transformada de Fourier y posteriormente calcular el promedio, estos datos se guardan en un vector hasta que se llene la memoria RAM es decir hasta alcanzar las 126976 muestras, luego se integra a datos de velocidad, para plotear el espectro es necesario pasar los datos de velocidad a pixeles, luego se calcula el valor RMS y la amplitud máxima con su frecuencia. Cabe mencionar que antes de plotear el espectro los datos se tienen que escribir en la memoria FLASH.

Diagrama de flujo del modo análisis profundo

En la figura 87 se muestra el diagrama de flujo del modo análisis profundo.

Nota: (*) Solo se visualiza en programa, no se realizó diagrama de flujo.

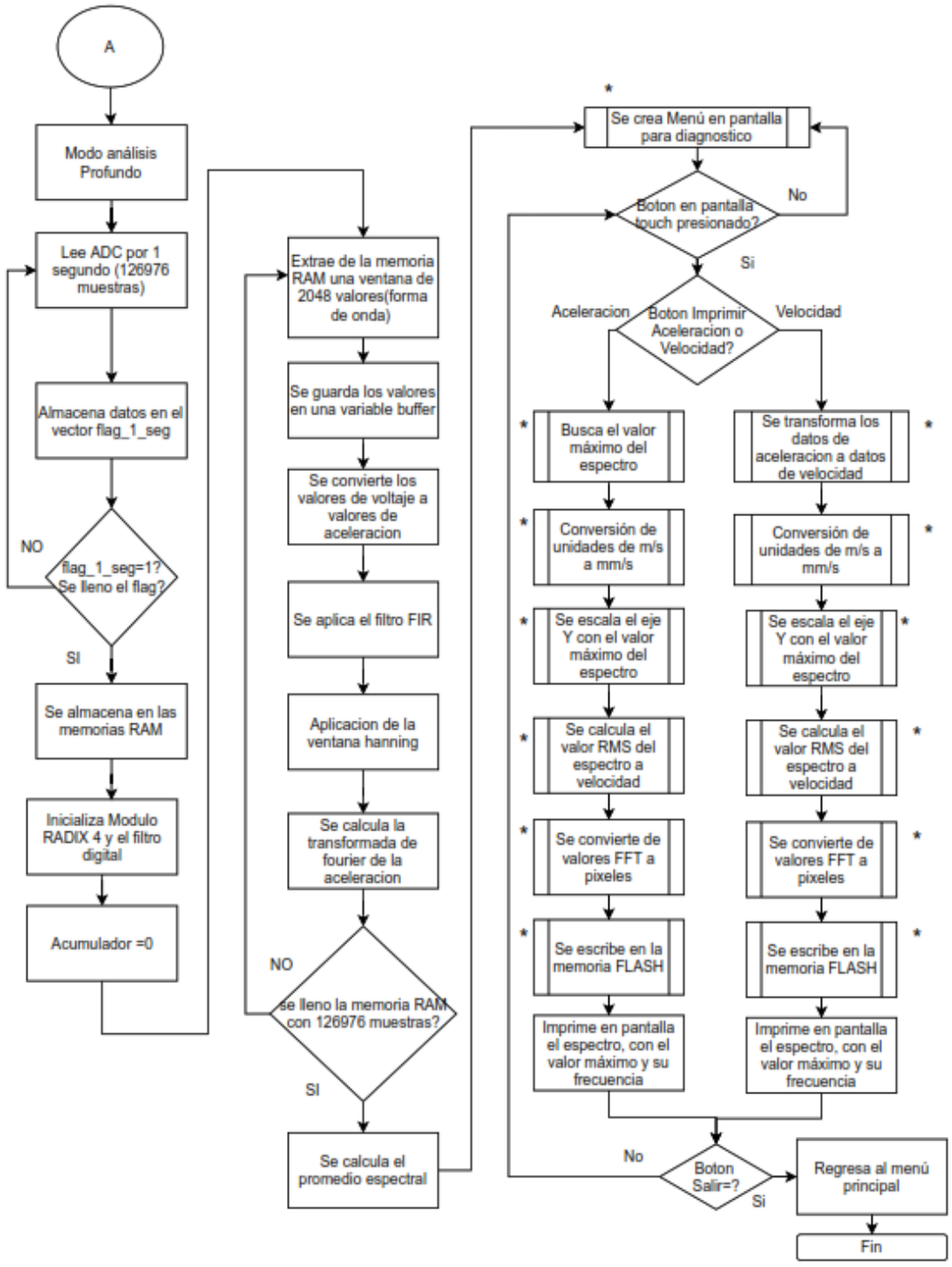


Figura 87: Diagrama de flujo del modo análisis profundo

Fuente: Propia

3.4.4.3 Modo análisis en tiempo real

A continuación, se describe los cálculos de las muestras que requiere el análisis modo tiempo real.

Cálculo de muestras para el análisis en tiempo real

Se tiene que considerar la cantidad de muestras en 1 segundo y el tiempo el periodo de muestreo que es de 25us.

$$Totalmuestras = \frac{Tiempo\ de\ muestreo}{Periodo\ de\ muestro}$$

$$Totalmuestras = \frac{1}{25us}$$

$$Totalmuestras = 40000$$

Las muestras tienen que ser múltiplo de 2048, ya que son las muestras que requiere el radix4. Y además tiene que ser las más cercanas a 1 segundo, por lo que la cantidad de muestras a escoger es de 40960, ahora verificamos si se aproxima a 1 segundo.

$$TiempoTotal = Totalmuestras * periodomuestreo$$

$$TiempoTotal = 40960 * 25us$$

$$TiempoTotal = 1.024\ segundos$$

Cálculo del número de promedios

$$Total\ de\ promedios = \frac{Total\ de\ muestras}{Muestras\ en\ 25us}$$

$$Total\ de\ promedios = \frac{40960}{2048}$$

$$TiempoTotal = 20\ promedios$$

En la figura 88 se muestra el tiempo total del análisis tiempo real.

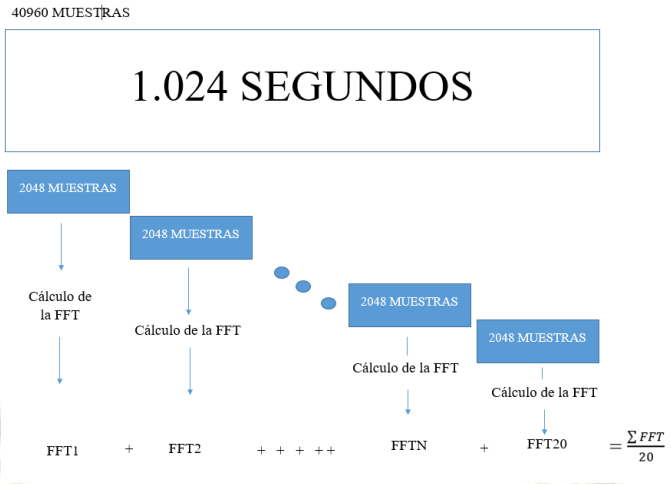


Figura 88: Tiempo total del análisis tiempo real

Fuente: Propia

Funcionamiento del modo análisis tiempo real

En este modo de análisis en tiempo real, lo primero que se realiza es inicializar los periféricos y las librerías, luego empieza la lectura del ADC de la señal de vibración, para posteriormente realizar la conversión a unidades de aceleración m/s2, luego se aplica el filtro digital pasa bajos FIR con una frecuencia de corte de 10KHz y de orden 200, luego de tener la señal en el rango de frecuencias hasta los 10KHz, estos datos se envían al grupo de memorias RAM para almacenarlas hasta un límite de 40960 muestras, que son divididas en 20 porciones de 2048 muestras, a cada grupo de muestra de 2048 se aplica la ventana hanning para luego calcular la transformada de Fourier y posteriormente calcular el promedio, estos datos se guardan en un vector hasta que se llene la memoria RAM, es decir hasta alcanzar las 126976 muestras, luego se integra a datos de velocidad, para plotear el espectro es necesario pasar los datos de velocidad a pixeles, luego se calcula el valor RMS y la amplitud máxima con su frecuencia. Cabe mencionar que antes de plotear el espectro los datos se tienen que escribir en la memoria FLASH.

Diagrama de flujo del modo análisis en tiempo real

En la figura 89 se muestra el diagrama de flujo del modo análisis en tiempo real.

Nota: (*) Solo se visualiza en programa, no se realizó diagrama de flujo.

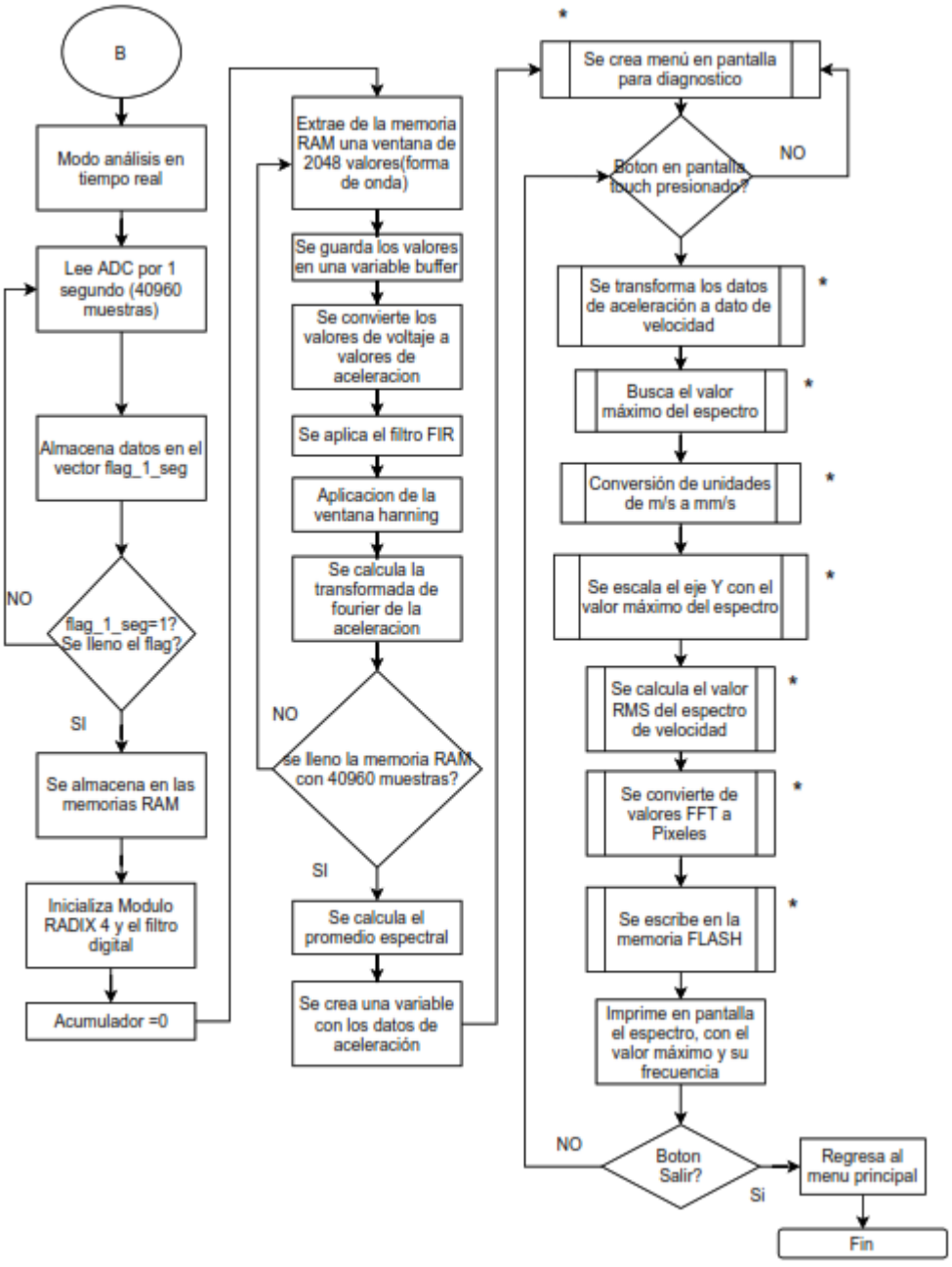


Figura 89: Diagrama de flujo del modo análisis profundo

Fuente: Propia

CAPITULO IV

4. PRUEBAS Y RESULTADOS DEL EQUIPO PROTOTIPO

4.1 Presentación del prototipo equipo registrador de vibraciones

Como se detalló en los anteriores capítulos este equipo consta de una pantalla donde se visualiza el espectro de vibración, una batería y el sensor de vibración. Además, los resultados están acordes a la norma ISO 10816.

Como se muestra en la figura 90 se muestra una foto del equipo implementado.

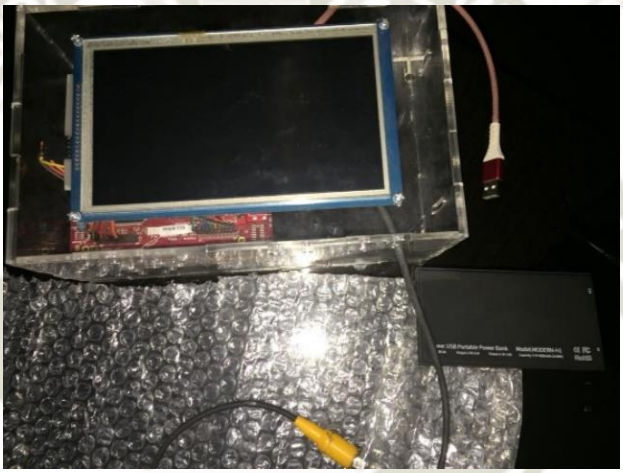


Figura 90: Equipo prototipo del registrador de vibraciones

Fuente: Propia

4.2 ¿Cómo realizar la medición con el equipo registrador de vibraciones?

Ahora se muestra cómo debemos medir con el equipo, primero al iniciar el equipo nos muestra tres opciones. Escogeremos la opción de motor nuevo para tener una mejor precisión de las mediciones. El uso en tiempo real, lo más frecuente es usarla para observar el comportamiento del motor a lo largo del tiempo. Y en la opción de configuración se puede cambiar la hora, fecha, la luz de fondo y el tiempo de espera del equipo. El sistema es bastante amigable con el operador ya que es muy intuitivo.

En la figura 91 se muestra la primera opción a escoger del equipo implementado.

Figura 91: Primero escoger la opción de motor nuevo

Fuente: Propia

En la figura 92 y 93 se muestra la colocación del nombre del motor o el tag con la que está identificado y las características del motor.

Figura 92: Segundo Colocar el nombre o tag del motor

Fuente: Propia

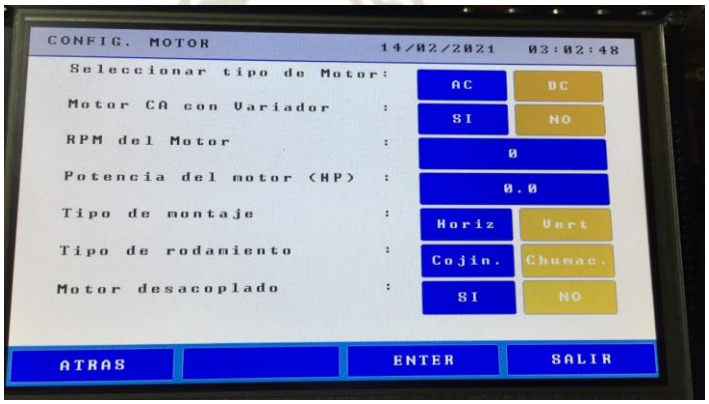


Figura 93: Tercero ingresar datos del motor

Fuente: Propia

Ahora solo toca medir en las dos zonas que marca el equipo para luego analizarlo con la norma ISO 10816.

En la figura 94 se muestra el menú de medición de la zona 1 y 2.

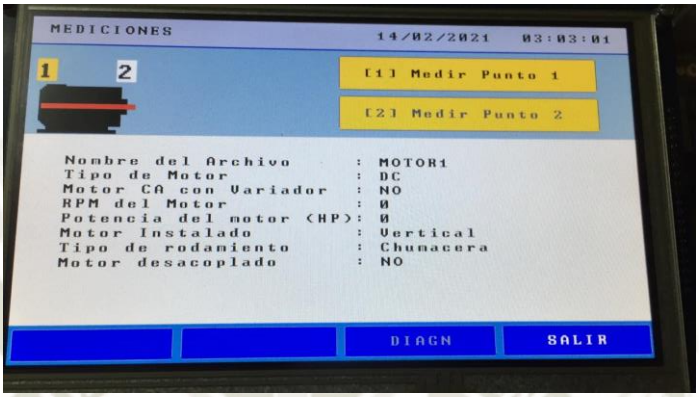


Figura 94: Menú de medición en la zona 1 y zona 2

Fuente: Propia

4.3 Prueba con un motor monofásico de 0.5 HP

El motor es pequeño de 0.5HP monofásico, es de una fábrica de calzado deportivo, el cual tiene uso para el raspado de plantas y cuero en horma. Se realizará la medición en los lados de los rodamientos, al cual denominaremos zona 1 y zona 2. La tabla de valores de frecuencia se colocará en anexos.

En la figura 95 y 96 se muestra las características del motor de 0.5HP

Figura 95: Motor de 0.5HP

Fuente: Propia

Figura 96: Placa de características del motor 0.5HP

Fuente: Propia

4.4.3.1 Datos a considerar del motor

En la tabla 22 se muestra los datos que se debe tener en consideración del motor para poder verificar con la norma ISO 10816.

Tabla 22
Datos a considerar en un motor monofásico de 0.5HP

Datos a considerar	**Descripción**
Revoluciones por minuto del motor	3500RPM
Frecuencia de giro del motor	58.33Hz
Tipo de montaje del motor	El montaje del motor está en una zona rígida
Potencia del motor	0.5 HP equivalente a: 372.85watts

Fuente: Propia

4.4.3.2 Ubicación del sensor de vibración

Hay muchas formas de tomar las mediciones del motor, para este caso solo mediremos de forma vertical en los lados de los cojinetes que denominaremos zona 1 y zona 2 respectivamente. Cabe mencionar que el operador que realice la medición debe tener como mínimo la calificación de categoría 1 en análisis de vibraciones de cualquier norma vigente. La forma de montaje del sensor es manual, pero se puede agregar un accesorio para el montaje, ya sea imantado, con tornillo, algún adhesivo o con una punta de prueba. Todo va depender del tipo de motor que se tiene. Es por eso que se deja a una

libre opción para el operador, en las figuras 97 y 98 se muestra la medición en la zona 1 y 2.

Figura 97: Medición de la zona 1 del motor de 0.5HP

Fuente: Propia

Figura 98: Medición de la zona 2 del motor de 0.5HP

Fuente: Propia

4.4.3.3 Resultados de la medición

El equipo nos muestra la frecuencia donde está el valor máximo y el valor RMS de toda la medición, en valores de aceleración y velocidad, con el fin de que el operador pueda comparar con la norma ISO 10816 de vibraciones. También muestra el espectro característico de la vibración del motor, para que pueda comparar con los espectros característicos de falla de motor.

En las figuras 99, 100, 101 y 102 se muestra los resultados de la medición.

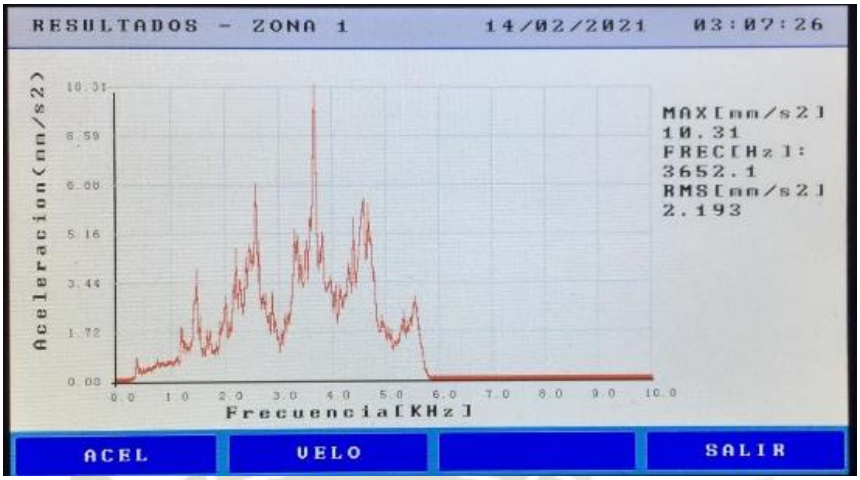


Figura 99: Espectro de la Zona 1 motor 0.5 HP -Valores de aceleración

Fuente: Propia

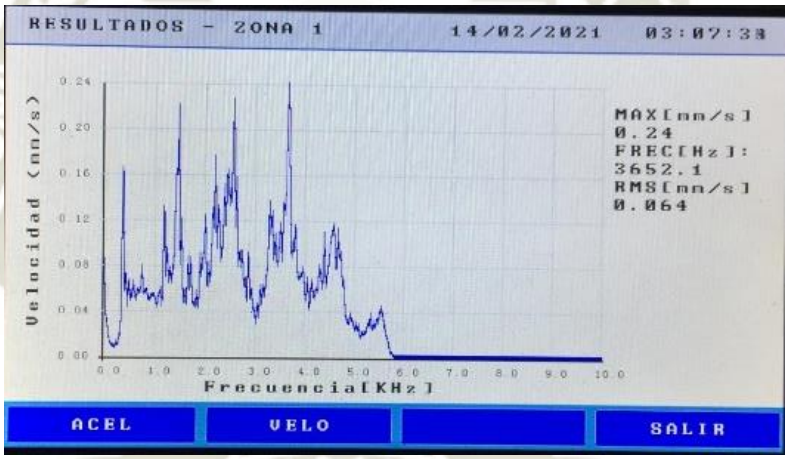


Figura 100: Espectro de la Zona 1 motor 0.5 HP -Valores de velocidad

Fuente: Propia

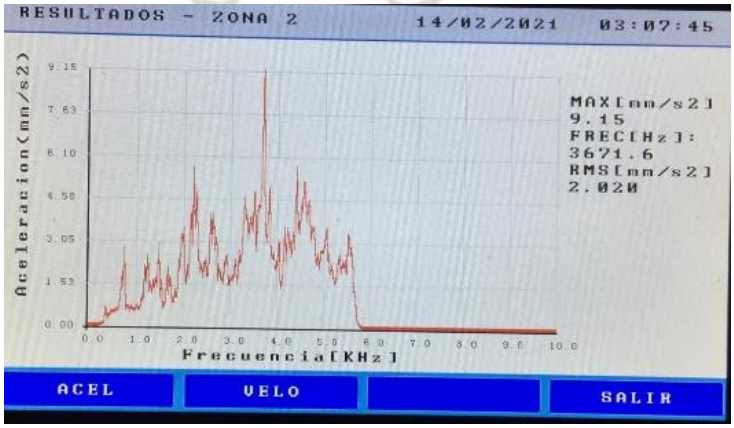


Figura 101: Espectro de la Zona 2 motor 0.5 HP -Valores de aceleración

Fuente: Propia

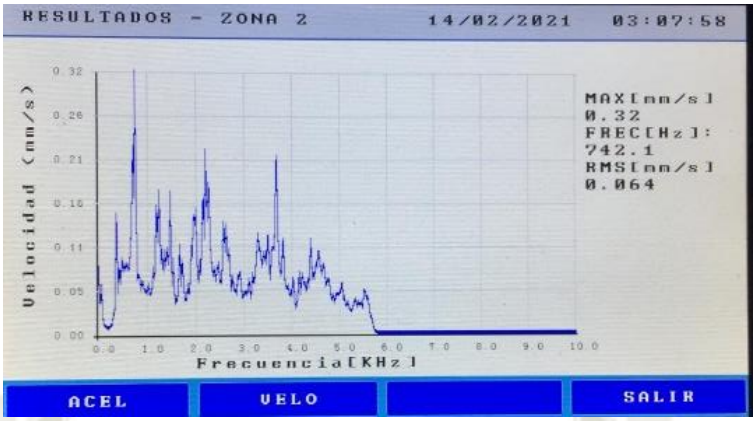


Figura 102: Espectro de la Zona 2 motor 0.5 HP -Valores de velocidad

Fuente: Propia

Para tener un análisis confiable, el equipo entrega todos los datos que se usó para graficar el espectro y las características del motor ingresadas en un archivo Excel, para que luego el operador pueda realizar un buen análisis de vibraciones. Cabe mencionar que el operador debe estar calificado como analista de vibraciones categoría 2 de cualquier norma vigente.

4.4.3.4 Confiabilidad de los resultados

Primero debemos evaluar los resultados de severidad con la norma 10816.

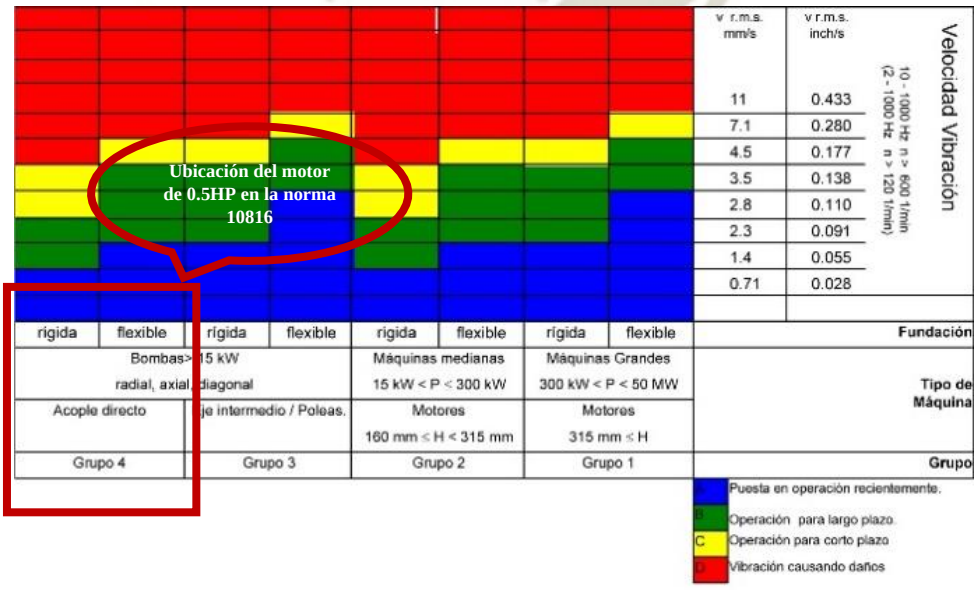


Figura 103: Confiabilidad de los resultados del motor 0.5HP"

Fuente: https://pruftechnik.wordpress.com/2011/10/13/la-vibracion-confirma-la-norma/, (2011)

Como se muestra en la figura 103 los resultados están RMS y en valores de velocidad, antes de evaluar se debe identificar qué tipo de base tiene el motor (rígida o flexible) y la potencia del motor. Esta identificación ya se realizó en la tabla 22 solo nos corresponde verificar si cumple con la norma ISO 10816. Como se puede apreciar en la figura 100 y 102, los valores RMS son 0.064 mm/s en ambas mediciones, nuestro tope para nuestra medición es de 1.4mm/s. Esto indica que está en valores aceptables según la norma ISO 10816 de análisis de vibraciones en valores de severidad.

4.4 Prueba con una compresora de aire

El motor es de 2HP monofásico, pertenece a una compresora de aire, la cual alimenta a una pegadora de plantas de calzado y a una repujadora. Se realizará la medición en los rodamientos del motor. La tabla de valores de frecuencia se colocará en anexos.

Figura 104: Compresora de aire de 2HP

Fuente: Propia

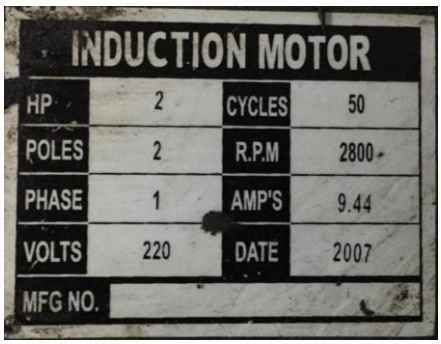


Figura 105: Placa del motor 2HP de la compresora de aire

Fuente: Propia

4.4.4.1 Datos a considerar

Para poder verificar con la norma ISO 10816 se debe considerar lo siguiente:

Tabla 23
Datos a considerar en un motor de 2HP

Datos a considerar	Descripción
Revoluciones por minuto del motor	2800RPM
Frecuencia de giro del motor	46.67Hz
Tipo de montaje del motor	El montaje del motor está en una zona rígida
Potencia del motor	2 HP equivalente a: 1491.4watts

Fuente: propia

4.4.4.2 Ubicación del sensor de vibración

Hay muchas formas de tomar las mediciones del motor, para este caso solo vamos a medir de forma vertical y horizontal en el lado libre del motor, ya que en el otro lado no se puede realizar la medición ya que el motor está en pleno movimiento y podríamos ocasionar un accidente, por lo que se optó en solo medir en un lado del motor que denominaremos zona 1 a la medición vertical y zona 2 como medición horizontal. Cabe mencionar que el operador que realice la medición debe tener como mínimo la calificación de categoría 1 en análisis de vibraciones de cualquier norma vigente. La forma de montaje del sensor es manual, pero se puede agregar un accesorio para el montaje, ya sea imantado, con tornillo, algún adhesivo o con una punta de prueba. Todo va depender del tipo de motor que se tiene. Es por eso que se deja a una libre opción para el operador.

Figura 106: Medición en posición horizontal del motor 2HP

Fuente: Propia

Figura 107: Medición en posición vertical del motor 2HP

Fuente: Propia

4.4.4.3 Resultados de medición

El equipo nos muestra la frecuencia donde está el valor máximo y el valor RMS de toda la medición, en valores de aceleración y velocidad, con el fin de que el operador pueda comparar con la norma ISO 10816 de vibraciones. También muestra el espectro característico de la vibración del motor, para que pueda comparar con los espectros característicos de falla de motor.

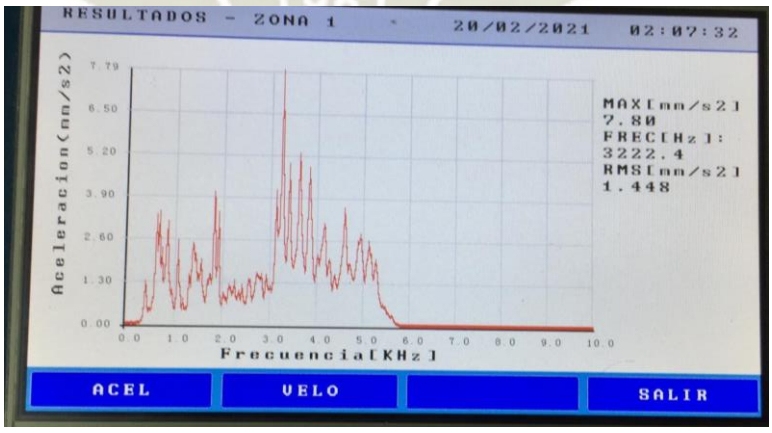


Figura 108: Espectro de la zona 1 motor 2HP-Valores de aceleración

Fuente: Propia

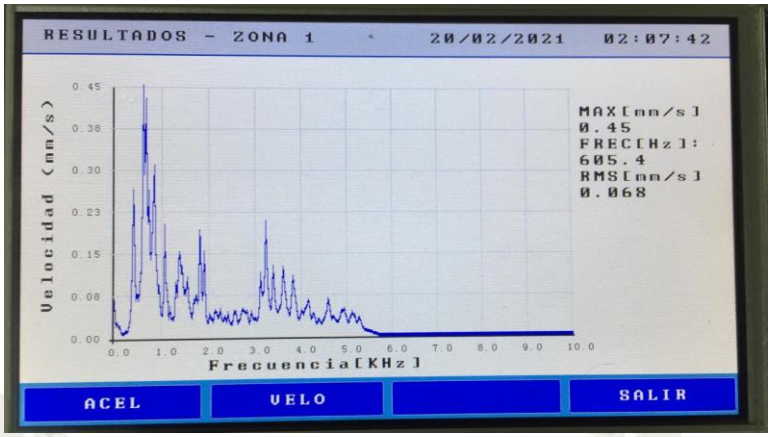


Figura 109: Espectro de la zona 1 motor 2HP-Valores de velocidad

Fuente: Propia

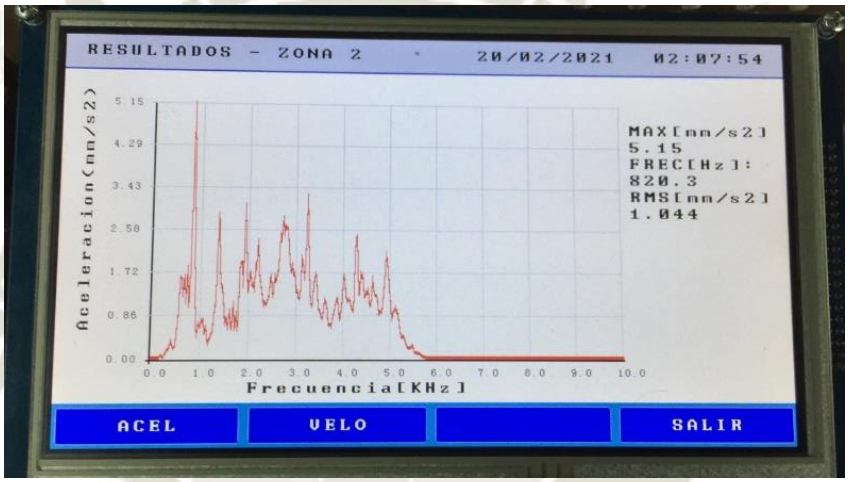


Figura 110: Espectro de la zona 2 motor 2HP -Valores de aceleración

Fuente: Propia

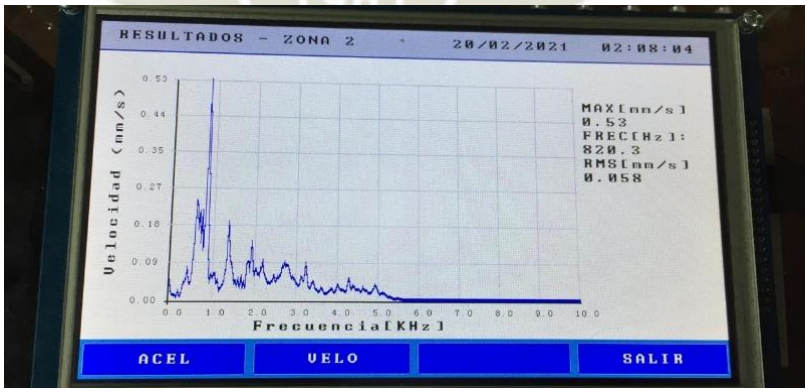


Figura 111: Espectro de la zona 2 motor 2HP-Valores de velocidad

Fuente: Propia

Para tener un análisis confiable, el equipo entrega todos los datos que se usaron para graficar el espectro y las características del motor ingresadas en un archivo Excel,

para que luego el operador pueda realizar un buen análisis de vibraciones. Cabe mencionar que el operador debe estar calificado como analista de vibraciones categoría 2 de cualquier norma vigente.

4.4.4.4 Confiabilidad de resultados

Primero debemos evaluar los resultados de severidad con la norma 10816.

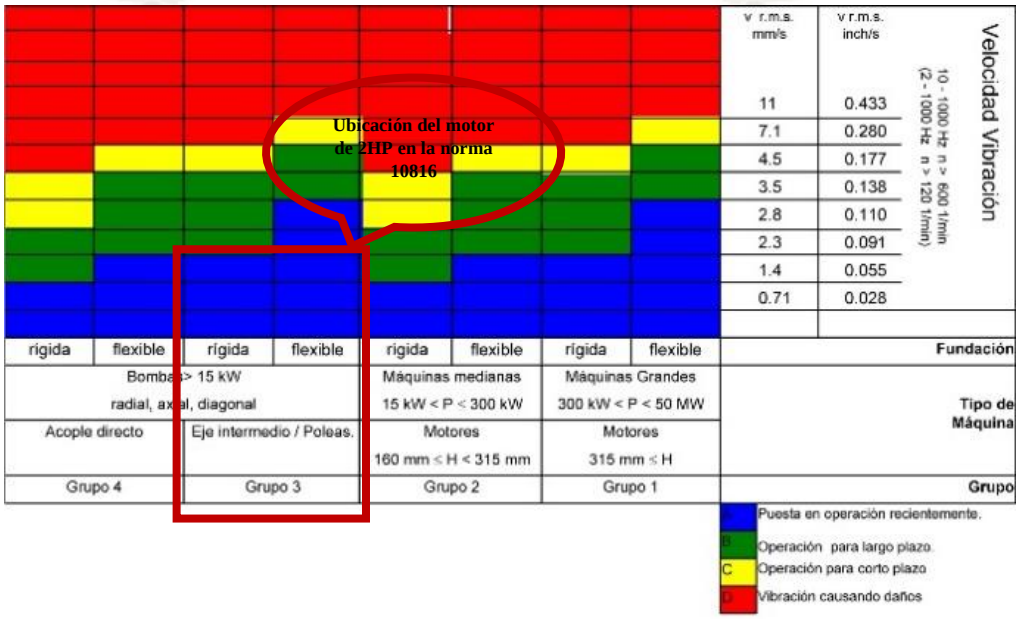


Figura 112: Confiabilidad de resultados en motor de 2HP

Fuente: https://pruftechnik.wordpress.com/2011/10/13/la-vibracion-confirma-la-norma/, (2011)

Como se muestra en la figura 112 los resultados están RMS y en valores de velocidad, antes de evaluar se debe identificar qué tipo de base tiene el motor (rígida o flexible) y la potencia del motor. Esta identificación ya se realizó en la tabla 23 solo nos corresponde verificar si cumple con la norma ISO 10816. Como se puede apreciar en las figuras 109 y 111 los valores RMS son 0.068 mm/s y de 0.058 mm/s respectivamente, nuestro tope para nuestra medición es de 1.4mm/s. Esto nos indica que está en valores aceptables según la norma ISO 10816 de análisis de vibraciones en valores de severidad.

CAPITULO V

5. PRESUPUESTO DEL EQUIPO PROTOTIPO

A continuación, se en la tabla 24 se detalla el costo del equipo prototipo implementado, incluyendo mano de obra y materiales

Tabla 24
Costo del prototipo en Nuevos Soles

ITEM	CODIGO	DESCRIPCION	CANT	Unidad	Precio Unit. (S./)	Total (S./)
1	74HC14	Compuerta lógica Smitch trigger Inversora	4.00	und	2.00	8.00
2	25LC1024	Circuito integrado / Memoria EEPROM de comunicación SPI	2.00	und	11.00	22.00
3	23LC1024	Circuito integrado / Memoria SRAM de comunicación SPI	8.00	und	8.00	64.00
4	MCP6022	Circuito Integrado Amplificador de Instrumentación Marca Microchip	20.00	und	5.00	100.00
5	DS1307	Circuito integrado Reloj en tiempo Real comunicación I2C	2.00	und	15.00	30.00
6	LAUNCHXL2-RM57L	Microcontrolador Hércules RM57	1.00	und	310.00	310.00
7	SSD13963	Pantalla LCD TouchScreen de 7 pulgadas	1.00	und	300.00	300.00
8	CR2032	Batería para Reloj de 3,3V	1.00	und	10.00	10.00
9	74HC595	Circuito Integrado desplazamiento de señal serial a paralelo	4.00	und	3.00	12.00
10	MAX232	Transceiver de comunicación serial	4.00	und	3.00	12.00
11	MAX485	Transceiver de comunicación RS485	4.00	und	3.00	12.00
12	CD4050	Circuito Integrado Buffer de Corriente	4.00	und	3.00	12.00
13		Conectores tipo peine Macho	4.00	und	4.00	16.00
14		Conectores tipo peine Hembra	4.00	und	4.00	16.00
15		Conectores tipo pin torneado	4.00	und	4.00	16.00
16		Borneras de 2 entradas	10.00	und	0.50	5.00
17		Borneras de 3 entradas	10.00	und	0.50	5.00
18		Conectores Molex de 2 entradas	10.00	und	2.00	20.00
19		Conectores Molex de 3 entradas	10.00	und	2.00	20.00
20		Conectores tipo Jamper Hembra - Hembra	3.00	und	5.00	15.00
21		Conectores tipo Jamper Hembra - Macho	3.00	und	5.00	15.00

22	10uF/16V	Condensador de 10uF con voltaje de trabajo de 16 Voltios	10.00	und	1.00	10.00
23	100uF/16V	Condensador de 100uF con voltaje de trabajo de 16 Voltios	10.00	und	1.00	10.00
24	4,7uF/16V	Condensador de 4,7uF, con voltaje de trabajo de 16 Voltios	5.00	und	1.00	5.00
25	100nF	Condensador tipo lenteja de 100nF	20.00	und	0.50	10.00
26	1nF	Condensador tipo lenteja de 1nF	20.00	und	0.50	10.00
27	10nF	Condensador tipo lenteja de 10nF	20.00	und	0.50	10.00
28	22pF	Condensadores de 22 picofaradios	10.00	und	0.50	5.00
29	32KHz/Cristal	Cristal Oscilador de 32Khz para RTC	2.00	und	3.00	6.00
30		Porta baterías para CR2032	2.00	und	5.00	10.00
31	BC547	Transistor NPN	5.00	und	0.50	2.50
32	2N3906	Transistor PNP	5.00	und	0.50	2.50
33	1MΩ	Resistencias de 1/4 de watt	40.00	und	0.10	4.00
34	220k	Resistencias de 1/4 de watt	40.00	und	0.10	4.00
35	22k	Resistencias de 1/4 de watt	40.00	und	0.10	4.00
36	10k	Resistencias de 1/4 de watt	40.00	und	0.10	4.00
37	50K	Trimpot color azul	10.00	und	0.10	1.00
38	100	Resistencia de 100 ohm de alta precisión <1%	20.00	und	0.50	10.00
39		Memoria SD de 16GB Clase 10	1.00	und	50.00	50.00
40		Servicio de fabricación de placa impresa	1.00	glb	200.00	200.00
41		Programación y diseño de circuito del prototipo	1.00	glb	2000.00	2000.00
42	A0120LF EMERSON	Sensores de vibración de 500mV/g + Cable de Instrumentación y conexionado	2.00	und	700.00	1400.00
					Total	**4,778.00**

En la figura 113 se muestra el precio de un equipo parecido a nuestro equipo implementado.

ANALIZADOR DE VIBRACIONES FLUKE 810

Nº Artículo 3542635

Precio disponible bajo cotización, póngase en contacto con nostros para obtener su mejor precio. Características y ventajas La identificación y localización conjunta de las averías mecánicas más comunes (rodamientos, alineación incorrecta, desequilibrio, holguras) concentra los trabajos de

› Leer descripción completa › Vista previa de impresión

Entrega: Disponible en 7/10 días.

Cantidad: - 1 +

Precio : € 9.966,00 IVA excl.

Figura 113: Precio del analizador de vibraciones Fluke 810

Fuente: https://www.fluitronic.es/analizador-de-vibraciones-fluke-810-3542635, (2021)

Como se muestra en la tabla 24 y la figura 113, hay un gran ahorro de nuestro parte implementado el equipo registrador de vibraciones.

CONCLUSIONES

1. Se diseñó e implemento un equipo portátil que permite registrar y visualizar los espectros de frecuencia que genera las vibraciones de un motor eléctrico. Además este equipo no requiere de una fuente de alimentación ya que es alimentado por una batería recargable y para tener una mejor visualización del espectro, exporta los datos del gráfico a una tabla Excel mediante una memoria micro SD extraíble, para que el operador pueda analizarlo en su computador.

2. A partir de la revisión de las hojas de datos y las recomendaciones de los diferentes fabricantes se pudo seleccionar los componentes adecuados para la implementación del equipo registrador de vibraciones.

3. Se logró la implementación de la placa impresa para la etapa de acondicionamiento de señal cumpliendo con la norma ANSI IPC 2221,

4. Se logró aplicar la transformada rápida de Fourier y filtros con el fin de convertir todas estas señales de las vibraciones mecánicas al dominio de la frecuencia, permitiéndonos visualizar de manera óptima las amplitudes y las frecuencias que corresponde a cada elemento rotativo del motor.

5. Se logró implementar con un microcontrolador RM 46L852 de la marca Texas instruments, este microcontrolador es de alta calidad, diseñado para aplicaciones industriales, el cual permite una alta precisión en la medición.

6. El análisis que puede realizar el operador con el equipo registrador de vibraciones puede formar parte del mantenimiento predictivo, ya que con el análisis el operador puede determinar el estado de los componentes rotativos y con esto puede planificar una parada del motor para su respectivo mantenimiento, evitando contratiempos y gastos innecesarios.

7. Con los resultados que se obtuvo de las pruebas, se pudo realizar una comparación con la norma ISO 10816 y determinar si el daño del motor es severo y pueda generar algún paro en el futuro.

RECOMENDACIONES

1. Este equipo puede ser también elaborado con dos acelerómetros, para agilizar las mediciones, teniendo en cuenta que se pudieran analizar muchos motores eléctricos.

2. Con un microcontrolador con más capacidad se puede ajustar la resolución de frecuencia a lo más mínimo, para tener datos más exactos, y poder detectar muchas más fallas del motor eléctrico esto generalmente se utiliza en motores de baja frecuencia ya que tienden a ser utilizados en molinos de bolas en minería.

REFERENCIAS

1. Chapman, S. J. (2012). *Maquinas electricas.* MCGRAWHILL.

2. Cheng, Y.-T. (Marzo de 2000). Auto escala radix-4 FFT. *Autoscaling Radix-4 FFT for TMS320C6000*. Taiwan Semiconductor Sales & Marketing.

3. Gomez Gil, M. d. (2017). *Procesamiento digital de señales.* INAOE.

4. Holek, R. S. (2005). Aspectos clave para un exitoso programa de monitoreo de vibraciones y la norma ISO 13374-1. Mexico: Reliability world.

5. Martinez, J. M. (2015). Tecnologia de circuitos impresos.

6. Power MI Cloud Condition Monitoring. (2018). *Manual de analisis de vibraciones.* Power MI Cloud Condition Monitoring.

7. Technical Associates of Charlotte. (1996). Lista ilustrada de diagnostico de vibraciones.

8. Texas Instruments. (Noviembre de 2000). A single Supply Op-Amp Circuit Collection. *Op-Amp Applications, High Performance Linear Products*. Texas Instruments.

9. Texas Instruments. (2018). *Manual de referencia tecnica Microcontrolador RM46x.* Texas Instruments.

10. Vega, C. P. (2019). *Ruido.* Universdiad de cantabria .

11. White, G. (1990). *Introduccion al analisis de vibraciones.* USA: Azima DL.

ANEXO 1

Programa principal: Sys_main.c

```
#include "sys_common.h"
/* USER CODE BEGIN (1) */
#include "system.h"
#include "type_defs.h"
#include "arm_math.h"
#include "spi.h"
#include "het.h"
#include "gio.h"
#include "sci.h"
#include "stdio.h"
#include "adc.h"
#include "etpwm.h"
#include "sys_core.h"
#include "Driver_Ram.h"
#include "LCD_MainMenu.h"
#include "DS1302.h"
#include "SPIFlash.h"
#include "TFT_LCD.h"
#include "rti.h"
#include <stdlib.h>     /* strtol */
#include "Keyboard.h"
#include "OS_Settings.h"    /*Kernel Configuraciones de Equipo*/
#include "OS_Measure.h"    /*Kernel Mediciones*/
#include "Kernel_MARIAN.h" /*Kernel de Motor de calculo Electrico*/
/* USER CODE END */

/** @fn void main(void)
 *   @brief Application main function
 *   @note This function is empty by default.
 *
 *   This function is called after startup.
```

```
 *   The user can use this function to implement the application.
 */

/* USER CODE BEGIN (2) */
uint8 Touch;

extern btnMensajes BtnMensaje;

bool Flag_Timer_Medicion = 0;
/* ---------------------------------------------------------------
 * Global variables for Digital Analyser Harmonic
 * --------------------------------------------------------------- */
uint32 in1 = 0, in2 = 0;
bool Buffer1_Full = 0;              // Flag para muestreos largos
bool Buffer2_RealTime = 0;          //Flag para muestreos de 1 segundo

extern bool Start_get_adc;
bool LCD_Update;
/* ---------------------------------------------------------------
 * Botones en la pantalla
 * --------------------------------------------------------------- */
int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int   Btn_NEW_MOTOR,   Btn_MEM_MOTOR,   Btn_CONFIG,   Btn_Fluc,   Btn_Tran,
Btn_Osc,
      BTN_OFF;
extern uint8_t BigFont[];
/* ---------------------------------------------------------------
 * Imagenes del Programa
 * --------------------------------------------------------------- */
extern const unsigned short Apagar[]; // Dimensions    : 70x64 pixels

char Kermotor[20] = "";
Signal_wave VIBRATION;
float32_t Voltaje_DC;
```

```
bool Iflash_Status = 0;

uint8 IndeRAM = 1;       //Memoria RAM, por Default

void Test_Supply_Sensor();
void Apagar_Analizador();

bool LCD_Backlight = 1;   //Pantalla Encendido, al iniciar el equipo
uint16 POWER_BACKLIGHT = 0;
float TEMP_BACKLIGHT = 120; //120--->2 MINUTO
/* USER CODE END */

int main(void)
{
   /* USER CODE BEGIN (3) */

   spiInit();
   gioInit();
   hetInit();
   sciInit();
   tftInit();
   rtiInit();
   adcInit();         //Inicializo las configuraciones del ADC
   etpwmInit();       //Inicializo las configuraciones del PWM
   SRAM_Init();
   SPIFlash_Init();
   rtcInit();

   UTFT_Buttons_Init();
   UTFT_Buttons_setTextFont(BigFont);

   /* Enable RTI Compare 0 interrupt notification */
   rtiEnableNotification(rtiNOTIFICATION_COMPARE0);
```

```
    rtiEnableNotification(rtiNOTIFICATION_COMPARE1);

    adcEnableNotification(adcREG1, adcGROUP0); //Habilito las interrupciones por ADC
y PWM Trigger
    adcStartConversion(adcREG1, adcGROUP0); //Inicializo la conversión del ADC

    _enable_IRQ();
    _enable_interrupt_();         //Habilito las interrupciones

    /* Start RTI Counter Block 0 */
    rtiStartCounter(rtiCOUNTER_BLOCK0);
    // WRITE_FLOAT_EXT_EEPROM(ADDR_BACKLIGHT, 80);
   //                                TEMP_BACKLIGHT                                =
READ_FLOAT_EXT_EEPROM(ADDRS_TEMP_BACKLIGHT);

    Main_Menu_Principal: /*Flag de retorno de las variables de salida*/
    MG_TaskScren();

    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("ANALIZADOR DE VIBRACIONES", 20, 10, 0);

    UTFT_Buttons_deleteAllButtons();

    BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
    BUTTON_DISABLED,
                          0);
    BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "",
    BUTTON_DISABLED,
                          1);
    BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "SD-LEER",
    BUTTON_DISABLED,
                          2);
    BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "",
```

```
  BUTTON_DISABLED,
                        3);

  Btn_NEW_MOTOR = UTFT_Buttons_addButton(200, 50, 380, 50, "MOTOR NUEVO",
  BUTTON_DISABLED,
                              10);
  Btn_MEM_MOTOR = UTFT_Buttons_addButton(200, 110, 380, 50,
                              "TIEMPO REAL MOTOR",
                              BUTTON_DISABLED,
                              11);
  Btn_CONFIG = UTFT_Buttons_addButton(200, 170, 380, 50, "CONFIGURACION",
  BUTTON_DISABLED,
                            12);
  BTN_OFF = UTFT_Buttons_addButton(720, 350, 70, 64, "",
  BUTTON_DISABLED,
                          16);
  /* ---------------------------------------------------------------
   **Generación de botones en pantalla
   ** ---------------------------------------------------------------- */
  UTFT_Buttons_disableButton(BTN1, false);
  UTFT_Buttons_disableButton(BTN2, false);
  UTFT_Buttons_enableButton(BTN3, false);
  UTFT_Buttons_disableButton(BTN4, false);

  UTFT_Buttons_enableButton(Btn_NEW_MOTOR, false);
  UTFT_Buttons_enableButton(Btn_MEM_MOTOR, false);
  UTFT_Buttons_enableButton(Btn_CONFIG, false);
  UTFT_Buttons_enableButton(BTN_OFF, false);
  UTFT_Buttons_drawButtons();
  drawBitmap(720, 350, 70, 64, Apagar, 1);       //Dibuja el botón de apagado

  Status_Backlight_eeprom();
```

```
SD_GetSpace();

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BTN3)
      {
         OS_MonitorSD(); // Realiza el cambio en el lugar no requiere regresar
         goto Main_Menu_Principal;
      }
      else if (PUSH_BUTTON == BTN_OFF)
      {
         Apagar_Analizador();
         goto Main_Menu_Principal;
      }
      else if (PUSH_BUTTON == Btn_NEW_MOTOR)
      {

         full_Keyboard("INGRESAR NOMBRE DEL MOTOR:",
                  Analizis_Motor.Nombre);

         OS_Conf_Motor();
         goto Main_Menu_Principal;
      }
      else if (PUSH_BUTTON == Btn_MEM_MOTOR)
      {
         adcStartConversion(adcREG1, adcGROUP0); //Inicializo la conversión del ADC
         OS_Armonicos_Analyer();
         goto Main_Menu_Principal;
      }
      else if (PUSH_BUTTON == Btn_CONFIG)
```

```
            {
                OS_Init(); // Realiza el cambio en el lugar no requiere regresar
                goto Main_Menu_Principal;
            }
        }

        Test_Supply_Sensor();
        screen_clock();
    }

    /* USER CODE END */

    return 0;
}

/* USER CODE BEGIN (4) */

void Apagar_Analizador()
{

    MG_TaskScren_blue();        // pantalla principal de pruebas
    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("HASTA LUEGO", 20, 10, 0);
    setColor(0x00, 0x00, 0x00);
    setBackColor(0x77, 0x99, 0xdd);
    TFT_print("APAGANDO EL SISTEMA", 250, 200, 0);
    drawBitmap(330, 236, 70, 64, Apagar, 2);        //Dibuja el boton de apagado

    delay(4);
    Set_Buzzer(1);

    delay(1);
```

```
    /* ---------------------------------------------------------------------
     **Apago el ADC
     ** --------------------------------------------------------------------- */
    adcStopConversion(adcREG1, adcGROUP0);
    /* ---------------------------------------------------------------------
     **Apago los temporizadores
     ** --------------------------------------------------------------------- */

    rtiDisableNotification(rtiNOTIFICATION_COMPARE0); // sirve para capturar la frecuencia de línea
    rtiDisableNotification(rtiNOTIFICATION_COMPARE1);
    rtiDisableNotification(rtiNOTIFICATION_COMPARE2); // sirve para Actualizar la pantalla LCD
    rtiDisableNotification(rtiNOTIFICATION_COMPARE3);
    rtiStopCounter(rtiCOUNTER_BLOCK0);
    rtiStopCounter(rtiCOUNTER_BLOCK1);
    /* ---------------------------------------------------------------------
     **Apago las Interrupciones
     ** --------------------------------------------------------------------- */
    _disable_IRQ_interrupt_();
    digitalWrite(LCD_CS, LOW);
    /* ---------------------------------------------------------------------
     **Apago la pantalla LCD
     ** --------------------------------------------------------------------- */
    pwmSetDuty(hetRAM1, pwm3, 0);
    Set_Buzzer(0);
    while (1)
        ;

}

void Test_Supply_Sensor()
{
```

```
    setColor(0x00, 0x00, 0x00);
    setBackColor(0xff, 0xff, 0xff);

    if (Voltaje_DC < 18)    //menor a 18Voltios
    {
       sprintf(BtnMensaje.Btn1, "Alarma!: Revisar Alimentación Sensor:%3.2f",
             Voltaje_DC);
       Set_Buzzer(true);
       delay(200e-3);
       Set_Buzzer(false);
       delay(200e-3);
       setFont(BigFont);
       TFT_print(BtnMensaje.Btn1, 10, 300, 0);
    }
    else
    {

       if (LCD_Update == 1)
       {

          LCD_Update = 0;
          float Bat_Pot = (Voltaje_DC / 20.1) * 100;
          sprintf(BtnMensaje.Btn1, "Carga Bateria: %3.2f %%  ", Bat_Pot);
          TFT_print(BtnMensaje.Btn1, 200, 390, 0);
       }
    }

}

void rtiNotification(uint32 notification)
{
    /*  enter user code between the USER CODE BEGIN and USER CODE END. */
    /* USER CODE BEGIN (9) */
    if (notification == rtiNOTIFICATION_COMPARE0)
```

```
    {
        disk_timerproc();
        if(POWER_BACKLIGHT==0){
          // Status_Backlight_eeprom();

        }
    }
    else if (notification == rtiNOTIFICATION_COMPARE1)
    {  // 1000ms
        Flag_Timer_Medicion = 1;
        LCD_Update = 1;

        if (POWER_BACKLIGHT > TEMP_BACKLIGHT)
        {
            TEMP_BACKLIGHT = TEMP_BACKLIGHT;
            pwmSetDuty(hetRAM1, pwm3, 0);
        }
        POWER_BACKLIGHT++;
    }
    /* USER CODE END */
}

void adcNotification(adcBASE_t *adc, uint32 group)
{
    adcData_t data[3];
    int value;
    count = adcGetData(adcREG1, adcGROUP0, data);

    if (Start_get_adc == 1)
    {

        value = (data[0].value - data[1].value);
        /* ----------------Almacenamiento en memoria RAM EXTERNA-------------------------
- */
```

```
        Write_int16_SRAM(in1, value);
        /* ------------------------------------------------------------------------- */
        in1++;

        if (in1 > RAM_SAMPLES_SIGNAL)
        {
           adcStopConversion(adcREG1, adcGROUP0); //Inicializo la conversion del ADC
           // adcDisableNotification(adcREG1, adcGROUP0); //Habilito las interruciones por
ADC y PWM Trigger
           Buffer1_Full = 1;
           in1 = 0;
        }
     }
     else
     {
        Voltaje_DC = data[2].value * ADC_VOLT / 0.1525423;
        value = (data[0].value - data[1].value);
        // VIBRATION.WaveSampling[in2] = (data[0].value - data[1].value);
        /* ----------------Almacenamiento en memoria RAM EXTERNA-------------------------
- */
        Write_int16_SRAM(in2, value);
        /* ------------------------------------------------------------------------- */
        in2++;
        if (in2 > RAM_SAMPLES_1_SEG)
        {
           adcStopConversion(adcREG1, adcGROUP0); //Inicializo la conversión del ADC
           in2 = 0;
           Buffer2_RealTime = 1;
           //adcDisableNotification(adcREG1, adcGROUP0);

           /* arm_copy_f32(VDD_20.WaveSampling, VDD_20.WaveSampling_Buffer,
            TEST_LENGTH_SAMPLES);*/
           //                                    arm_copy_f32(VIBRATION.WaveSampling,
VIBRATION.WaveSampling_Buffer,
```

```
            // TEST_LENGTH_SAMPLES);
            //adcEnableNotification(adcREG1, adcGROUP0);
        }
    }
}

/* USER CODE END */
```

ANEXO 2

Motor de cálculo: Kernel_Marian.c

```
/*
 * Kernel_MARIAN.c
*/
#include "Kernel_MARIAN.h" /*Kernel de Motor de cálculo Eléctrico*/
#include <SPIFlash.h>
#include "Driver_Ram.h"
#include "sci.h"
#include "stdio.h"
#include "math.h"
/* ----------------------------------------------------------------
 * Global variables for FFT Bin Example
 * ---------------------------------------------------------------- */
uint32_t fftSize = LENGTH_SAMPLES / 2;
uint32_t ifftFlag = 0;
uint32_t doBitReverse = 1;
float32_t scaleFFt = 2 / ((float32_t) LENGTH_SAMPLES); //(2 * FFT_MATLAB) /
((float32_t) LENGTH_SAMPLES);
/* ----------------------------------------------------------------
 * Variables del Filtro Digital
 * ---------------------------------------------------------------- */
extern const float32_t firCoeffs32[NUM_TAPS];
/* ----------------------------------------------------------------
 * Declare State buffer of size (numTaps + blockSize - 1)
 * ---------------------------------------------------------------- */
static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
/* ----------------------------------------------------------------
 * Global variables for FIR LPF Example
 * ---------------------------------------------------------------- */
extern bool Flag_Timer_Medicion;
uint32_t blockSize = BLOCK_SIZE;
uint32_t numBlocks = TEST_LENGTH_SAMPLES / BLOCK_SIZE;
extern bool Buffer1_Full;
```

```
extern const float32_t example[2048];

bool Start_get_adc = 0;
float32_t FFT_Deled[512];

/* ----------------------------------------------------------------
 * Global variables for FFT Bin Example
 * ---------------------------------------------------------------- */

/* ----------------------------------------------------------------
 * Estructura de datos para el motor de calculo
 * ---------------------------------------------------------------- */
extern Signal_wave VIBRATION;
/* ----------------------------------------------------------------
 ** Inicializa el Cálculo de Espectro en general
 ** ---------------------------------------------------------------- */

void Kernel_FFT_Init()  //Operativo
{

   /* Initialize the CFFT/CIFFT module */
   arm_status status = arm_cfft_radix4_init_f32(&FFT, fftSize, ifftFlag,
                                        doBitReverse);
   /* ----------------------------------------------------------------
    ** Inicialización del Filtro Digital
    ** ---------------------------------------------------------------- */
   arm_fir_init_f32(&S, NUM_TAPS, (float32_t*) &firCoeffs32[0],
                    &firStateF32[0], blockSize);
}
void Kernel_FFT_test(Signal_wave *Vibra_data)
{

   /* ----------------------------------------------------------------
```

```
 ** conversión de voltaje a m/s2
 ** ---------------------------------------------------------------- */
int i = 0;

for (i = 0; i < LENGTH_SAMPLES; i++)
   Vibra_data->Buffer_A[i] = example[i];

/* ----------------------------------------------------------------
 ** Calcula la transformada de Fourier / Vibra_data
 ** ---------------------------------------------------------------- */
arm_cfft_radix4_f32(&FFT, Vibra_data->Buffer_A);/* Process the data through the
CFFT/CIFFT module */
/* ----------------------------------------------------------------
 ** Calcula la transformada de Fourier / Vibra_data
 ** ---------------------------------------------------------------- */
arm_scale_f32(Vibra_data->Buffer_A, scaleFFt, Vibra_data->Buffer_A,
LENGTH_SAMPLES);
/* ----------------------------------------------------------------
 ** Calculo de la magnitud / Vibra_data
 ** ---------------------------------------------------------------- */
arm_cmplx_mag_f32(Vibra_data->Buffer_A, Vibra_data->fft_MAG, fftSize);
/* ----------------------------------------------------------------
 ** Calcula la transformada de Fourier / Vibra_data
 ** ---------------------------------------------------------------- */
arm_cfft_swag_f32(Vibra_data->fft_MAG, Vibra_data->Buffer_B, fftSize);

/* ----------------------------------------------------------------
 ** Búsqueda del Máximo, valor del espectro / Vibra_data
 ** ---------------------------------------------------------------- */
arm_max_f32(Vibra_data->Buffer_B, fftSize, &Vibra_data->valor_MAX,
        &Vibra_data->max_marks1);

}
```

```
void arm_cfft_swag_f32(float32_t *pSrcA, float32_t *pDst, uint32_t blockSize)
{
   /* ----------------------------------------------------------------------
    ** Calcula la transformada de Fourier / Vibra_data
    ** ---------------------------------------------------------------------- */
   int jk = (blockSize - 1), i;
   for (i = 0; i < (blockSize / 2); i++)
   {
      pDst[i] = pSrcA[jk];
      jk--;
   }
}

void arm_20log10_f32(float32_t *pSrcA, float32_t *pDst, uint32_t blockSize)
{
   /* ----------------------------------------------------------------------
    ** Calcula la transformada de Fourier / Vibra_data
    ** ---------------------------------------------------------------------- */
   int i;
   float32_t K_gr = 1e-9; // 1e-9 m/s -----> Referencia
   float32_t _temp;
   for (i = 0; i < (blockSize / 2); i++)
   {

      _temp = 20 * log10f(pSrcA[i] / K_gr);
      if (_temp < 60) // la potencia más baja es 60dB
         _temp = 60;
      pDst[i] = _temp;

   }
}

/* ----------------------------------------------------------------------
 ** Kernel_FFT_RAM()
```

```
 ** Retorna la señal de vibración
 ** ---------------------------------------------------------------- */
void Kernel_FFT_RAM()
{
   int x, ij;
   float32_t V_an;
   // float32_t temp;
   uint32 BLockSize = 0;
   float32_t Scale_promfft = 0;

   Kernel_FFT_Init();

   /* ----------------------------------------------------------------
    ** La variable acumuladora, siempre debe iniciar con valores de cero
    ** ---------------------------------------------------------------- */
   arm_fill_f32(0.0, VIBRATION.fft_MAG, LENGTH_SAMPLES / 4);

   /* ----------------------------------------------------------------
    ** El bucle se encarga de realizar la lectura de datos de la memoria RAM
    ** y calcula al transformada de fourier por ventana
    ** ---------------------------------------------------------------- */
   for (ij = 0; ij < SDRAM_SIZE_1_SEG; ij++)
   {
      BLockSize = LENGTH_SAMPLES * ij;
      /* ----------------------------------------------------------------
       ** Extrae de la memoria RAM una ventana de 2048 valores y los almacena
       ** en una variable Buffer
       ** ---------------------------------------------------------------- */
      for (x = 0; x < LENGTH_SAMPLES; x++)
      {
         V_an = ((float32_t) Read_int16_SRAM(x + BLockSize) * VOLT_ACELL);
//Voltios a Aceleracion
         VIBRATION.WaveSampling_Buffer[x] = V_an;
      }
```

```
        /* ------------------------------------------------------------------
         ** Kernel_FFT_Accel: Utiliza la ventana de hanning y la transformada de Fourier
         ** y el escalamiento de la señal, no esta integrando la señal de fourier
         ** ------------------------------------------------------------------ */
        Kernel_FFT_Accel(VIBRATION.WaveSampling_Buffer,
VIBRATION.Acceleration);
        VIBRATION.Acceleration[0] = 0;
        /* ------------------------------------------------------------------
         ** Agrega el promedio a la transformada al acumulador
         ** ------------------------------------------------------------------ */
        arm_add_f32(VIBRATION.fft_MAG, VIBRATION.Acceleration,
                VIBRATION.fft_MAG, LENGTH_SAMPLES / 4);
    }

    Scale_promfft = (float32_t) 1 / SDRAM_SIZE_1_SEG;
    arm_scale_f32(VIBRATION.fft_MAG, Scale_promfft, VIBRATION.Acceleration,
    LENGTH_SAMPLES / 4);

}

void Kernel_FFT_Calculo(Signal_wave *Vibra_data)
{

    int i;
    /* float32_t Vn, omega;
     uint32_t N = LENGTH_SAMPLES - 1, n;
     omega = (float) ((2 * pi) / N);
     float32_t *inputF32, *outputF32;*/

    Kernel_FFT_RAM();
    /* ------------------------------------------------------------------
     ** Realiza el intercambio del espectro
     ** ------------------------------------------------------------------ */
```

```
arm_cfft_swag_f32(VIBRATION.Acceleration, VIBRATION.fft_MAG, fftSize);
//FFT_MAG--> firma de aceleracion

/* ----------------------------------------------------------------
 ** Calculo de Espectros de Velocidad
 ** ---------------------------------------------------------------- */
int jj = 0;

for (jj = 0; jj < fftSize / 2; jj++)
{
   VIBRATION.Buffer_B[jj] = VIBRATION.fft_MAG[jj] * 86.75
         / ((jj + 1) * RES_FREC);
}

//  arm_20log10_f32(Vibra_data->Buffer_B, Vibra_data->fft_MAG, fftSize);
/* ----------------------------------------------------------------
 ** Búsqueda del Máximo, valor del espectro  / Vibra_data
 ** ---------------------------------------------------------------- */
arm_max_f32(VIBRATION.Buffer_B, fftSize / 2, &VIBRATION.valor_MAX,
         &VIBRATION.max_marks1); //m/s
arm_min_f32(VIBRATION.Buffer_B, fftSize / 2, &VIBRATION.valor_MIN,
         &VIBRATION.min_marks1);

/* ----------------------------------------------------------------
 ** Conversion de unidades de [m/s] a  [mm/s]
 ** ---------------------------------------------------------------- */
VIBRATION.Velocidad_max = VIBRATION.valor_MAX * 1000;    //[mm/s]

/* ----------------------------------------------------------------
 ** Escalamiento del eje Y, para el ploteo en pantalla
 ** ---------------------------------------------------------------- */
if ((VIBRATION.Velocidad_max > 0) && (VIBRATION.Velocidad_max <= 1))
   VIBRATION.tmep_MAX = 1;
else if ((VIBRATION.Velocidad_max > 1) && (VIBRATION.Velocidad_max <= 3))
```

```
    VIBRATION.tmep_MAX = 2;
  else if ((VIBRATION.Velocidad_max > 3) && (VIBRATION.Velocidad_max <= 5))
    VIBRATION.tmep_MAX = 5;
  else if ((VIBRATION.Velocidad_max > 5) && (VIBRATION.Velocidad_max <= 10))
    VIBRATION.tmep_MAX = 10;
  else if ((VIBRATION.Velocidad_max > 10) && (VIBRATION.Velocidad_max <=
15))
    VIBRATION.tmep_MAX = 15;
  else if ((VIBRATION.Velocidad_max > 15) && (VIBRATION.Velocidad_max <=
20))
    VIBRATION.tmep_MAX = 20;
  else
    VIBRATION.tmep_MAX = VIBRATION.valor_MAX;

  float32_t _tempixel = 0;
  uint16 _tempint16 = 0;
  for (i = 0; i < fftSize / 2; i++)
  {

    _tempixel = (VIBRATION.Buffer_B[i] * 1e3 / VIBRATION.tmep_MAX) * 300;
    _tempint16 = (uint16) _tempixel;
    if (_tempint16 > 300)
      _tempint16 = 300;
    VIBRATION.XY_plot[i] = _tempint16;
  }

}
/* ----------------------------------------------------------------------
 ** Inicializa el Cálculo de Espectro en general
 ** ---------------------------------------------------------------------- */

void Kernel_Prom_Firm_Calibration(int Size)
{
  float32_t Scale_promfft = 0;
```

```
    /* ---------------------------------------------------------------
     ** Calculo del Promedio de las firmas de vibración
     ** --------------------------------------------------------------- */
    Scale_promfft = (float32_t) 1 / Size;
    arm_scale_f32(Data_Vibration.FFTp_Acceleration, Scale_promfft,
                  Data_Vibration.FFTp_Acceleration, LENGTH_SAMPLES / 4);

    /* ---------------------------------------------------------------
     ** Se generan los Cluster de datos en la memoria Flash
     ** --------------------------------------------------------------- */

    SPIFlash_blockErase32K(ADDR_FFT_Accel_offset);

    Cluster_FWrite(ADDR_FFT_Accel_offset, Data_Vibration.FFTp_Acceleration,
    LENGTH_SAMPLES / 4);
}
/* ---------------------------------------------------------------
 ** Genera el retardo de Aproximadamente
 ** 1.536 segundos, por cada 61440 datos del ADC
 ** del tipo int16.
 ** --------------------------------------------------------------- */
void Kernel_WRITE_Buffer()
{
    /* ---------------------------------------------------------------
     ** Inicializa la captura del ADC
     ** --------------------------------------------------------------- */
    Start_get_adc = 1;

    adcStartConversion(adcREG1, adcGROUP0); //Inicializo la conversion del ADC
    /* ---------------------------------------------------------------
     ** Bucle de espera
     ** --------------------------------------------------------------- */
    while (Buffer1_Full != 1)
        ;
```

```
    /* ---------------------------------------------------------------
     ** Detiene la captura de Datos
     ** --------------------------------------------------------------- */
    Start_get_adc = 0;
    Buffer1_Full = 0;   // Resetear Buffer para iniciar de nuevo.

}
/* ---------------------------------------------------------------
 ** Kernel_READ_Buffer():
 **
 ** --------------------------------------------------------------- */
void Kernel_READ_Buffer()
{
    int x, ij;
    float32_t V_an;
    // float32_t temp;
    uint32 BLockSize = 0;
    float32_t Scale_promfft = 0;
    Kernel_FFT_Init();

    /* ---------------------------------------------------------------
     ** La variable acumuladora, siempre debe iniciar con valores de cero
     ** --------------------------------------------------------------- */
    arm_fill_f32(0.0, VIBRATION.fft_MAG, LENGTH_SAMPLES / 4);

    /* ---------------------------------------------------------------
     ** El bucle se encarga de realizar la lectura de datos de la memoria RAM
     ** y calcula al transformada de fourier por ventana
     ** --------------------------------------------------------------- */
    for (ij = 0; ij < SDRAM_SIZE; ij++)
    {
        BLockSize = LENGTH_SAMPLES * ij;
        /* ---------------------------------------------------------------
         ** Extrae de la memoria RAM una ventana de 2048 valores y los almacena
```

```
        ** en una variable Buffer
        ** ---------------------------------------------------------------- */
       for (x = 0; x < LENGTH_SAMPLES; x++)
       {
          V_an = ((float32_t) Read_int16_SRAM(x + BLockSize) * VOLT_ACELL);
//Voltios a Aceleracion
          VIBRATION.WaveSampling_Buffer[x] = V_an;
       }
       /* ----------------------------------------------------------------
        ** Kernel_FFT_Accel: Utiliza la ventana de hanning y la transformadora de Fourier
        ** y el escalamiento de la señal, no está integrando la señal de fourier
        ** ---------------------------------------------------------------- */
       Kernel_FFT_Accel(VIBRATION.WaveSampling_Buffer,
VIBRATION.Acceleration);
       /* ----------------------------------------------------------------
        ** Agrega el promedio a la transformada al acumulador
        ** ---------------------------------------------------------------- */
       arm_add_f32(VIBRATION.fft_MAG, VIBRATION.Acceleration,
                VIBRATION.fft_MAG, LENGTH_SAMPLES / 4);
   }

   Scale_promfft = (float32_t) 1 / SDRAM_SIZE;
   arm_scale_f32(VIBRATION.fft_MAG, Scale_promfft, VIBRATION.fft_MAG,
   LENGTH_SAMPLES / 4);

}

void Kernel_READ_Buffer_to_SCI()
{
   uint32 x, ij;
   int V_an;
   float32_t temp;
   // uint32 BLockSize = 0;
   char MSG_SCI[40] = "";
```

```
    /* ----------------------------------------------------------------
     ** Lectura de la Memoria RAM Externa y acumulación de
     ** las TRF de las firmas de vibración
     ** ---------------------------------------------------------------- */
    for (ij = 0; ij < RAM_SAMPLES_SIGNAL; ij++)
    {
        V_an = Read_int16_SRAM(ij);

        sprintf(MSG_SCI, "%d     \n\r", V_an);
        sciDisplayText(scilinREG, MSG_SCI, 40);

    }

}

/* ----------------------------------------------------------------
 ** La función cuenta con Filtrado de señal y ventana Hannig con
 ** transformada de fourier
 ** ---------------------------------------------------------------- */
void Kernel_FFT_Accel(float32_t *pSrcA, float32_t *pDst)
{
    int i;
    float Vn, omega;
    uint32_t N = LENGTH_SAMPLES - 1, n;
    omega = (float) ((2 * pi) / N);
    float32_t *inputF32, *outputF32;

    inputF32 = &pSrcA[0];
    outputF32 = &pDst[0];

    for (i = 0; i < numBlocks; i++)
    {
        arm_fir_f32(&S, inputF32 + (i * blockSize), outputF32 + (i * blockSize),
                blockSize);
```

```
    }
    /* ----------------------------------------------------------------
     ** Aplicación de la ventana de Hanning para el análisis de la TRF
     ** ---------------------------------------------------------------- */

    for (n = 0; n < LENGTH_SAMPLES; n++)
    {
       Vn = 0.54 - 0.54 * cos(omega * n);
       pSrcA[n] = 2 * pDst[n] * Vn;
    }

    /* ----------------------------------------------------------------
     ** Calculo de la TRF de la Aceleración
     ** ---------------------------------------------------------------- */
    arm_cfft_radix4_init_f32(&FFT, fftSize, 0, doBitReverse);
    arm_cfft_radix4_f32(&FFT, pSrcA);
    /* ----------------------------------------------------------------
     ** Multiplica la transformada
     ** ---------------------------------------------------------------- */
    arm_scale_f32(pSrcA, scaleFFt, pSrcA, LENGTH_SAMPLES);
    arm_cmplx_mag_f32(pSrcA, pDst, fftSize);

}

    /* ----------------------------------------------------------------
     ** Aplicación de la ventana de Hanning para el análisis de la TRF
     ** ---------------------------------------------------------------- */

    for (n = 0; n < LENGTH_SAMPLES; n++)
    {
       Vn = 0.54 - 0.46 * cos(omega * n);
       Wave_InOut->Buffer_A[n] = 2 * Wave_InOut->Acceleration[n] * Vn;
    }
    /* ----------------------------------------------------------------
```

```
 ** Calculo de la TRF de la Aceleración
 ** ---------------------------------------------------------------- */
arm_cfft_radix4_init_f32(&FFT, fftSize, 0, doBitReverse);
arm_cfft_radix4_f32(&FFT, Wave_InOut->Buffer_A);
/* ----------------------------------------------------------------
 ** Multiplica la trasnformada
 ** ---------------------------------------------------------------- */
arm_scale_f32(Wave_InOut->Buffer_A, scaleFFt, Wave_InOut->Buffer_A,
LENGTH_SAMPLES);
arm_cmplx_mag_f32(Wave_InOut->Buffer_A, Wave_InOut->FFT_Acceleration,
                fftSize);

/* ----------------------------------------------------------------
 ** Realiza el intercambio del espectro
 ** ---------------------------------------------------------------- */
/*  arm_cfft_swag_f32(Wave_InOut->FFT_Displacement,
 Wave_InOut->FFT_Acceleration, fftSize);*/
/* ----------------------------------------------------------------
 ** Eliminando componentes de DC
 ** ---------------------------------------------------------------- */
Wave_InOut->FFT_Acceleration[0] = 1e-9;
/* ----------------------------------------------------------------
 ** Calculo de Espectros de Velocidad y Desplazamiento
 ** ---------------------------------------------------------------- */
int jj = 0;

for (jj = 0; jj < 512; jj++)
{
   Wave_InOut->FFT_Velocity[jj] = Wave_InOut->FFT_Acceleration[jj] * 86.75
         / ((jj + 1) * 19.53125);

}

}
```

```
/* ---------------------------------------------------------------------
 ** Esta función solo almacena en la memoria Flash, la FFT
 ** de aceleración, ya promediada
 ** --------------------------------------------------------------------- */
void Kernel_Write_Flash_Punto_1()
{
   SPIFlash_blockErase4K(ADDR_FFTP1_Accel);
   /* ---------------------------------------------------------------------
    ** Se generan los Cluster de datos en la memoria Flash
    ** --------------------------------------------------------------------- */
   Cluster_FWrite(ADDR_FFTP1_Accel, VIBRATION.fft_MAG,
   LENGTH_SAMPLES / 4);
}
/* ---------------------------------------------------------------------
 ** Esta función solo almacena en la memoria Flash, la FFT
 ** de aceleración, ya promediada
 ** --------------------------------------------------------------------- */
void Kernel_Write_Flash_Punto_2()
{
   SPIFlash_blockErase4K(ADDR_FFTP2_Accel);
   /* ---------------------------------------------------------------------
    ** Se generan los Cluster de datos en la memoria Flash
    ** --------------------------------------------------------------------- */
   Cluster_FWrite(ADDR_FFTP2_Accel, VIBRATION.fft_MAG,
   LENGTH_SAMPLES / 4);
}

void Kernel_Plot_FFT(float32_t *pSrcA)
{
   float32_t ScrTemp[512];
   float32_t pSrcA_max_value, pSrcA_max_point;
   /* ---------------------------------------------------------------------
    ** Búsqueda del Máximo, valor del espectro  / Vibra_data
```

```
 ** ---------------------------------------------------------------- */
arm_max_f32(pSrcA, LENGTH_SAMPLES / 4, &pSrcA_max_value,
&pSrcA_max_point);
/* ----------------------------------------------------------------
 ** Modificación del Espectro
 ** ---------------------------------------------------------------- */
arm_scale_f32(pSrcA, FFT_MAX_HIGH / pSrcA_max_value, ScrTemp,
LENGTH_SAMPLES / 4);

/* ----------------------------------------------------------------
 ** Etapa donde se Dibuja el espectro de frecuencia armónica en pantalla
 ** ---------------------------------------------------------------- */
int FFT_M;
int FFT_axis_zero_X = 50;
int FFT_axis_zero_Y = 50;
int x;

for (x = 0; x < LENGTH_SAMPLES / 4; x = x++)
{
   /* ----------------------------------------------------------------
    ** Imprime en pantalla las barras de potencia armónica.
    ** ---------------------------------------------------------------- */
   setColor(0x00, 0x00, 0x00);
   drawPixel(FFT_axis_zero_X, FFT_axis_zero_Y - ScrTemp[x]);
   FFT_M = abs(ScrTemp[x + 1] - ScrTemp[x]);
   if (FFT_M > 0)
      drawLine(FFT_axis_zero_X, FFT_axis_zero_Y - ScrTemp[x],
            FFT_axis_zero_X + 1, FFT_axis_zero_Y - ScrTemp[x + 1]);
   /* ----------------------------------------------------------------
    ** Configuración de Incremento entre barras y cambio de estados.
    ** ---------------------------------------------------------------- */
   FFT_axis_zero_X = FFT_axis_zero_X + 1;
}
```

```
}
void Kernel_FFT_Velocidad(Signal_wave *Vibra_data, Vibration_Firma *FFT_data)
{
   int i;
   /* ----------------------------------------------------------------
    ** Calculo de Espectros de Velocidad
    ** ---------------------------------------------------------------- */
   int jj = 0;

   for (jj = 0; jj < fftSize / 2; jj++)
   {
      FFT_data->FFTp_Velocidad[jj] = FFT_data->FFTp_Acceleration[jj] * 86.75
            / ((jj + 1) * RES_FREC);
   }

   //  arm_20log10_f32(Vibra_data->Buffer_B, Vibra_data->fft_MAG, fftSize);
   /* ----------------------------------------------------------------
    ** Búsqueda del Máximo, valor del espectro  / Vibra_data
    ** ---------------------------------------------------------------- */
   arm_max_f32(FFT_data->FFTp_Velocidad, fftSize / 2, &Vibra_data->valor_MAX,
            &Vibra_data->max_marks1); //m/s
   arm_rms_f32(FFT_data->FFTp_Velocidad, fftSize / 2, &Vibra_data->RMS); //mm/s
   /* ----------------------------------------------------------------
    ** Conversión de unidades
    ** ---------------------------------------------------------------- */
   Vibra_data->Velocidad_max = Vibra_data->valor_MAX * 1000;    //m/s

   /* ----------------------------------------------------------------
    ** Utiliza el valor máximo para escalar la función de Ploteo
    ** ---------------------------------------------------------------- */
   Vibra_data->tmep_MAX = Vibra_data->valor_MAX;
  /*  if (Vibra_data->tmep_MAX < 1e-3)
      Vibra_data->tmep_MAX = 1e3;*/
```

```
    float32_t _tempixel = 0;
    uint16 _tempint16 = 0;
    for (i = 0; i < fftSize / 2; i++)
    {

       _tempixel = (FFT_data->FFTp_Velocidad[i] / Vibra_data->tmep_MAX) * 300;
       _tempint16 = (uint16) _tempixel;
       // _tempixel = tempixel;
       Vibra_data->XY_plot[i] = _tempint16;
    }

}

void Kernel_FFT_Aceleracion(Signal_wave *Vibra_data,
                            Vibration_Firma *FFT_data)
{
    int i;

    /* ----------------------------------------------------------------
     ** Búsqueda del Máximo, valor del espectro  / Vibra_data
     ** ---------------------------------------------------------------- */
    arm_max_f32(FFT_data->FFTp_Acceleration, fftSize / 2,
              &Vibra_data->valor_MAX, &Vibra_data->max_marks1); //m/s2
    arm_rms_f32(FFT_data->FFTp_Acceleration, fftSize / 2, &Vibra_data->RMS); //m/s2
    /* ----------------------------------------------------------------
     ** Conversión de unidades
     ** ---------------------------------------------------------------- */
    Vibra_data->Aceleracion_max = Vibra_data->valor_MAX;    //m/s2

    /* ----------------------------------------------------------------
     ** Utiliza el valor máximo para escalar la función de Ploteo
     ** ---------------------------------------------------------------- */
    Vibra_data->tmep_MAX = Vibra_data->Aceleracion_max;
    /*if (Vibra_data->tmep_MAX < 1e-3)
```

```
        Vibra_data->tmep_MAX = 1e3;*/

    float32_t _tempixel = 0;
    uint16 _tempint16 = 0;
    for (i = 0; i < fftSize / 2; i++)
    {

        _tempixel = (FFT_data->FFTp_Acceleration[i] / Vibra_data->tmep_MAX)
                * 300;
        _tempint16 = (uint16) _tempixel;
        // _tempixel = tempixel;
        Vibra_data->XY_plot[i] = _tempint16;
    }
}
```

ANEXO 3

Análisis en tiempo real: OS_Armonicos.c

```
/*
 * OS_Armonicos.c

#include "OS_Armonicos.h"
//#include "KERNEL_MCE.h"        /*Kernel de Motor de calculo Electrico*/
//#include "KERNEL_GPU.h"        /*Kernel de Unidad de Procesamiento Grafico*/
#include "stdio.h"
#include "Kernel_MARIAN.h" /*Kernel de Motor de calculo Electrico*/
#include "LCD_MainMenu.h"
#include "TFT_LCD.h"
#include "rti.h"
#include "etpwm.h"
#include "gio.h"
#include "DS1302.h"
#include "SPIFlash.h"
#include "ff.h"
#include "sd_defs.h"
#include <stdlib.h>     /* strtol */

/* ----------------------------------------------------------------
 **Defines
 ** ---------------------------------------------------------------- */
#define X_point         100
#define Y_point         380

/* ----------------------------------------------------------------
 **Variables del tamaño de letra y touch screen
 ** ---------------------------------------------------------------- */
extern uint8_t BigFont[];
extern uint8_t SmallFont[];
extern uint8 Touch;
```

```
btnMensajes LCD_Mensaje;
extern RTC_DS1302 Calendario;
extern btnMensajes BtnMensaje;

uint8 flag_btn1 = 0; //Sirve para cambiar el texto del botón 1 en el inferior de la pantalla

uint8 flag_btn2 = 0, Flag_HOLD = 1; //Sirve para cambiar el texto del botón 1 en el inferior de la pantalla
extern bool LCD_Update;
extern bool Buffer2_RealTime;         //Flag para muestreos de 1 segundo
//extern Harmonics_data Voltaje, Corriente, Potencia;
//extern MotorCalculo Wave_Info[];
RTC_DS1302 Hora_Actual, Hora_Inicio;

extern int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int BTN5, BTN6, BTN7, BTN8;
extern Signal_wave VDD_20, VIBRATION;
float BW_fft = 0, tempF;
/* ----------------------------------------------------------------------
 **Variables de la memoria SD
 ** ---------------------------------------------------------------------- */
FRESULT iFResult;
FIL F_XLS; /* File objects */

int touch_x, Cursor_X;
int touch_y, Cursor_Y;
float frec_val, Ampl_val;
extern float32_t Voltaje_DC;

void OS_Armonicos_Analyer()
{
   /*Habilito el Menú de Configuraciones*/
   /* ----------------------------------------------------------------------
    ** Iniciando el Muestreo de la Señal Eléctrica
```

```
     ** ---------------------------------------------------------------- */

    int x, a = X_point;
    float FFT_M;
    int x_base = Y_point;
    int Resol = 1, ih;
    float yval = 0; //imprime los valores del eje y
    float xval = 0; //imprime los valores del eje x
    int X_Resol = 3; // Tipo de paso para ampliar o reducir el espectro de frecuencia
    int Limit_x = 512;  //Límite máximo del ancho de grafico
    int Num_lin = (int) (Limit_x) / 10; //Numero de divisiones que tendrá el eje X, en este
caso 7

    uint16 Y_buffer = 0, X_buffer = 0;
//         InitSamplingADC();
    rtiSetPeriod(rtiCOMPARE1, 5000000U); // Periodo de actualización de 1 Segundo
    /* ----------------------------------------------------------------
     **Inicializo el Kernel matemático
     ** ---------------------------------------------------------------- */
    //Kernel_Init();
    Kernel_FFT_Init();

    /* Kernel_Init();*/
    Init_harmonic:
    /* ----------------------------------------------------------------
     ** Iniciando graficas en pantalla
     ** ---------------------------------------------------------------- */
    MG_TaskScren();             // pantalla principal de pruebas
    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("ARMONICOS", 20, 10, 0);
    /* ----------------------------------------------------------------
     ** Iniciando graficas en pantalla
     ** ---------------------------------------------------------------- */
```

```
setColor(0x00, 0x00, 0x00);
setBackColor(0xd2, 0xd2, 0xd2);
TFT_print("ANALISIS EN TIEMPO REAL", 20, 10, 0);

UTFT_Buttons_deleteAllButtons();

BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "[+]",
BUTTON_DISABLED,
                    0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "[-]",
BUTTON_DISABLED,
                    1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "SGTE",
BUTTON_DISABLED,
                    2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                    3);

/* ---------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** --------------------------------------------------------------- */
UTFT_Buttons_enableButton(BTN1, true);
UTFT_Buttons_enableButton(BTN2, true);
UTFT_Buttons_enableButton(BTN3, false);
UTFT_Buttons_enableButton(BTN4, false);
UTFT_Buttons_drawButtons();

/*Inicializo los valores de los mensajes en botones*/
/*sprintf(BtnMensaje.Btn1, "      ");
 sprintf(BtnMensaje.Btn2, "      ");
 sprintf(BtnMensaje.Btn3, "      ");
 sprintf(BtnMensaje.Btn4, "INICIAR");
 sprintf(BtnMensaje.Btn5, "SALIR");
```

```
    BtnMensaje.GetValue = 0;          */                  // Ningún Botón Seleccionado
//        MG_Btn_TouchAction(&BtnMensaje);
   /* ---------------------------------------------------------------------
    ** Preparando memoria SD
    ** --------------------------------------------------------------------- */

   /*Se realiza una revisión de si memoria SD, está conectada al equipo*/
   iFResult = f_mount(&g_sFatFs, "", 1);

   if (iFResult != FR_OK)
   {
      setColor(0xff, 0xff, 0xff);
      setBackColor(0x77, 0x99, 0xdd);
      sprintf(LCD_Mensaje.Btn1, "Error en la SD: %s",
            StringFromFResult(iFResult));
      TFT_print(LCD_Mensaje.Btn1, 10, 50, 0);

      /*Agregar aquí el botón de salida de la pantalla inferior de la LCD*/
      if (!URTouch_dataAvailable() == true)
      {
         PUSH_BUTTON = UTFT_Buttons_checkButtons();

         if (PUSH_BUTTON == BTN4)
         {
            goto final_harmonic;
         }
      }
   }

   /* ---------------------------------------------------------------------
    **Inicializo los valores de los mensajes en botones
    ** --------------------------------------------------------------------- */
   //Cmd_lcd  Cmd_lcd();                                  //Imprime    el    tamaño    en
porcentaje de la memoria
```

```
setColor(0X00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
TFT_print("Configuracion de Memoria:", 50, 80, 0);
TFT_print("Inicio     :", 50, 120, 0);
TFT_print("Hora       :", 50, 140, 0);
TFT_print("Fecha      :", 50, 160, 0);

uint16 NUMBER_FILE = (uint16) READ_FLOAT_EXT_EEPROM(ADDRS_REAL_TIME_MOTOR);

while (1)
{

   sprintf(LCD_Mensaje.Btn4, "Voltaje Sensor:%3.2f  ", Voltaje_DC);
   if (Voltaje_DC < 18)    //menor a 18Voltios
   {
      sprintf(LCD_Mensaje.Btn4,
            "Alarma!: Revisar Alimentación Sensor:%3.2f", Voltaje_DC);
      Set_Buzzer(true);
      delay(100e-3);
      Set_Buzzer(false);
      delay(100e-3);
   }
   setFont(BigFont);
   setColor(0x00, 0x00, 0xff);
   TFT_print(LCD_Mensaje.Btn4, 50, 300, 0);

   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BTN1)
      {
         if (NUMBER_FILE < 1000)
```

```
            NUMBER_FILE++;
        WRITE_FLOAT_EXT_EEPROM(ADDRS_REAL_TIME_MOTOR,
                        (float) NUMBER_FILE);
    }
    else if (PUSH_BUTTON == BTN2)
    {

        if (NUMBER_FILE > 0)
            NUMBER_FILE--;
        WRITE_FLOAT_EXT_EEPROM(ADDRS_REAL_TIME_MOTOR,
                        (float) NUMBER_FILE);
    }
    else if (PUSH_BUTTON == BTN3)
    {
        goto Iniciar_Analisis_Armonico;
    }

    else if (PUSH_BUTTON == BTN4)
    {
        goto final_harmonic;
    }
}

sprintf(BtnMensaje.Btn10, "VIB%03u.CSV", NUMBER_FILE);

Calendario = rtc_read();
sprintf(Calendario.MSG_Fecha, "%02u/%02u/%04u", Calendario.day,
        Calendario.mth, Calendario.year);
sprintf(Calendario.MSG_Hora, "%02u:%02u:%02u", Calendario.hr,
        Calendario.min, Calendario.sec);

setColor(0x00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
TFT_print(BtnMensaje.Btn10, 260, 120, 0);
```

```
    TFT_print(Calendario.MSG_Fecha, 260, 140, 0);
    TFT_print(Calendario.MSG_Hora, 260, 160, 0);

}

Iniciar_Analisis_Armonico:

MG_TaskScren();  // pantalla principal de pruebas
setColor(0X00, 0x00, 0x00);
setBackColor(0xd2, 0xd2, 0xd2);
TFT_print("FIRMA DE VIBRACION", 20, 10, 0);
/* ---------------------------------------------------------------
 ** Inicializo las botoneras inferiores
 ** --------------------------------------------------------------- */

UTFT_Buttons_deleteAllButtons();

BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
BUTTON_DISABLED,
                        0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "H/S",
BUTTON_DISABLED,
                        1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "GUARD",
BUTTON_DISABLED,
                        2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

BTN5 = UTFT_Buttons_addButton(640, 100, 150, 50, "Zoom-",
BUTTON_DISABLED,
                        4);
BTN6 = UTFT_Buttons_addButton(640, 160, 150, 50, "Zoom+",
```

```
BUTTON_DISABLED,
                        5);

/*BTN7 = UTFT_Buttons_addButton(640, 200, 150, 45, "[+]",
 BUTTON_DISABLED,
 6);
 BTN8 = UTFT_Buttons_addButton(640, 260, 150, 45, "[-]",
 BUTTON_DISABLED,
 7);*/

/* ------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** ------------------------------------------------------------ */
UTFT_Buttons_disableButton(BTN1, true);
UTFT_Buttons_enableButton(BTN2, true);
UTFT_Buttons_enableButton(BTN3, true);
UTFT_Buttons_enableButton(BTN4, true);
UTFT_Buttons_enableButton(BTN5, true);
UTFT_Buttons_enableButton(BTN6, true);

/*Etapa blanca de la pantalla para generación de gráficas y cuadros de texto*/

setColor(0x00, 0x00, 0x00);

int Xinit = a - 6, Xend = a + 512;
int Yend = x_base + 6, Yinit = x_base - FFT_MAX_HIGH;

/*Genera el eje X*/
drawLine(a - 6, x_base + 1, a + 512, x_base + 1);
drawLine(a - 6, x_base + 2, a + 512, x_base + 2);
/*Genera el eje Y*/
drawLine(a - 1, x_base - FFT_MAX_HIGH, a - 1, x_base + 6);
drawLine(a - 2, x_base - FFT_MAX_HIGH, a - 2, x_base + 6);
```

```
/*Configuración de propiedades del texto*/
setFont(BigFont);
setColor(0, 0, 0);
setBackColor(0xff, 0xff, 0xff);

setColor(0, 0, 0);
TFT_print("Velocidad (mm/s)", 20, 350, -90);
TFT_print("Frecuencia [Khz]", 300, x_base + 25, 0);

while (1)
{

   screen_clock();

   if (URTouch_dataAvailable() == true)
   {

      PUSH_BUTTON = UTFT_Buttons_checkButtons();
      //touch_x = URTouch_getX();
      // touch_y = URTouch_getY();

      if (PUSH_BUTTON == BTN1)
      {
      }
      else if (PUSH_BUTTON == BTN2)
      {
         if (Flag_HOLD == 1)
         {
            Flag_HOLD = 0;
            setBackColor(0xff, 0xff, 0xff);
            setColor(0x00, 0x00, 0x00);
            TFT_print("PAUSA  ", 640, 300, 0);

         }
```

```
    else
    {
       Flag_HOLD = 1;
       setBackColor(0xff, 0xff, 0xff);
       setColor(0x00, 0x00, 0x00);
       TFT_print("PLAY   ", 640, 300, 0);
    }

}
else if (PUSH_BUTTON == BTN3)
{
    OS_Harmonic_SD();

    sprintf(BtnMensaje.Btn10, "VIB%03u.CSV", NUMBER_FILE);

    setBackColor(0xff, 0xff, 0xff);
    setColor(0x00, 0x00, 0x00);
    TFT_print("GUARDADO", 640, 320, 0);
    TFT_print(BtnMensaje.Btn10, 640, 340, 0);

    NUMBER_FILE++;
    WRITE_FLOAT_EXT_EEPROM(ADDRS_REAL_TIME_MOTOR,
                    (float) NUMBER_FILE);
}

else if (PUSH_BUTTON == BTN4)
{
    goto final_harmonic;
}
else if (PUSH_BUTTON == BTN5)   // Aumenta el ancho de banda
{

    X_Resol--;
    if (X_Resol < 1)
```

```
        X_Resol = 1;
}
else if (PUSH_BUTTON == BTN6)   // reduce el ancho de banda
{
   X_Resol++;
   if (X_Resol > 5)
        X_Resol = 5;

}
/* ------------------------------------------------------------------
 ** Cursor en pantalla
 ** ------------------------------------------------------------------ */

/* if ((touch_x >= Xinit && touch_x <= Xend) && (touch_y >= Yinit)
 && (touch_y <= Yend) && (Flag_HOLD == 1))
 {

 setColor_word(VGA_PURPLE);
 drawLine(touch_x, Yinit, touch_x, Yend);
 drawLine(Xinit, touch_y, Xend, touch_y);

 if (X_buffer != touch_x)
 {
 setColor(0xff, 0xff, 0xff);
 drawLine(X_buffer, Yinit, X_buffer, Yend);
 }
 X_buffer = touch_x;

 if (Y_buffer != touch_y)
 {
 setColor(0xff, 0xff, 0xff);
 drawLine(Xinit, Y_buffer, Xend, Y_buffer);
 }
 Y_buffer = touch_y;
```

```
    Cursor_X = touch_x - Xinit;
    Cursor_Y = Yend - touch_y;

    }*/
}

if ((Buffer2_RealTime == 1) && (Flag_HOLD == 1))
{
   Buffer2_RealTime = 0;

   Kernel_FFT_Calculo(&VIBRATION);
   /* ---------------------------------------------------------------
    ** Inicializo la conversión del ADC
    ** --------------------------------------------------------------- */
   adcStartConversion(adcREG1, adcGROUP0);

   /* ---------------------------------------------------------------
    ** Cursor en pantalla
    ** --------------------------------------------------------------- */
   a = X_point;

   /*Configuración de propiedades del texto*/
   setFont(SmallFont);

   setBackColor(0xff, 0xff, 0xff);
   float mul = 0;
   for (ih = 0; ih <= 30; ih = ih + 5)
   {

      yval = VIBRATION.tmep_MAX * mul;
      sprintf(LCD_Mensaje.Btn1, "%2.2f ", yval);
      mul = mul + 0.16666;    //---->5/30
```

```
    setColor(0, 0, 0);
    TFT_print(LCD_Mensaje.Btn1, a - 45, x_base - ih * 10 - 6, 0);
    if (ih != 0)
    {
       setColor(0xd2, 0xd2, 0xd2);
       drawLine(a - 6, x_base - ih * 10, a + 512,
             x_base - ih * 10);
    }
}
int xh = 0;

//Eje x

xval = 0;

for (ih = 0; ih <= 10; ih++)
{

    tempF = (xval / X_Resol);
    int temF_int = (int) tempF;
    // xval = VIBRATION.Velocidad_max * mul;
    sprintf(LCD_Mensaje.Btn1, "%2.1f ", tempF);
    // mul = mul + 0.16666;

    setColor(0, 0, 0);
    TFT_print(LCD_Mensaje.Btn1, a + xh - 5, x_base + 10, 0);
    if (ih != 0)
    {
       setColor(0xd2, 0xd2, 0xd2);
       drawLine(a + xh, x_base - 300, a + xh, x_base);
    }
    xval++;
    xh = xh + Num_lin;
}
```

```
BW_fft = tempF;

/*   Ampl_val = (float) Cursor_Y / FFT_MAX_HIGH
 * VIBRATION.Velocidad_max;
 frec_val = (float) Cursor_X / Limit_x * BW_fft;*/

setColor(0x00, 0x00, 0x00);
/*Genera el eje X*/
drawLine(a - 6, x_base + 1, a + 512, x_base + 1);
drawLine(a - 6, x_base + 2, a + 512, x_base + 2);
/*Genera el eje Y*/
drawLine(a - 1, x_base - 308, a - 1, x_base + 6);
drawLine(a - 2, x_base - 308, a - 2, x_base + 6);

//   sprintf(LCD_Mensaje.Btn4, "Vsupply:%3.2f  ", Voltaje_DC);

sprintf(LCD_Mensaje.Btn5, "[mm/s]:%3.2f  ",
      VIBRATION.Velocidad_max);
sprintf(LCD_Mensaje.Btn6, "[KHz]:%2.3f ",
      VIBRATION.max_marks1 * RES_FREC / 1000);

setFont(BigFont);
setColor(0x00, 0x00, 0x00);
TFT_print(LCD_Mensaje.Btn5, 50, 50, 0);
TFT_print(LCD_Mensaje.Btn6, 400, 50, 0);
/*    setColor(0x00, 0x00, 0xff);
 TFT_print(LCD_Mensaje.Btn4, 550, 50, 0);*/

for (x = 0; x < Limit_x / X_Resol; x++)
{

   /* --------------------------------------------------------------
    ** Borra la gráfica anterior para una nueva grafica
    ** -------------------------------------------------------------- */
```

```
setColor(0xff, 0xff, 0xff);
drawPixel(a, x_base - VIBRATION.XY_buff[x]);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 2);

FFT_M = abs(VIBRATION.XY_buff[x + 1] - VIBRATION.XY_buff[x]);

if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_buff[x], a + 1,
         x_base - VIBRATION.XY_buff[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 1, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] + 1);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 2, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] + 2);
}
/* ---------------------------------------------------------------
 ** Imprime en pantalla las barras de potencia armónica.
 ** --------------------------------------------------------------- */
setColor(0xff, 0x00, 0x00);
drawPixel(a, x_base - VIBRATION.XY_plot[x]);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 2);
FFT_M = abs(VIBRATION.XY_plot[x + 1] - VIBRATION.XY_plot[x]);
if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_plot[x], a + 1,
         x_base - VIBRATION.XY_plot[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_plot[x] - 1, a + 1,
         x_base - VIBRATION.XY_plot[x + 1] + 1);
   drawLine(a, x_base - VIBRATION.XY_plot[x] - 2, a + 1,
         x_base - VIBRATION.XY_plot[x + 1] + 2);
}
```

```
            a = a + X_Resol;
        }

        /* ------------------------------------------------------------------
         ** Vuelve a cargar el buffer de la muestra anterior del XY.
         ** ------------------------------------------------------------------ */
        int i;
        for (i = 0; i < Limit_x; i++)
        {
            VIBRATION.XY_buff[i] = VIBRATION.XY_plot[i];

        }
    }

    //Clock_Button_scrn();

  }
  final_harmonic: return 0;
}

void OS_Harmonic_SD()
{ //30-05-2018

  /* ------------------------------------------------------------------
   ** Creación de archivo EXCEL
   ** ------------------------------------------------------------------ */

  const char *NAMEFILE = BtnMensaje.Btn10;

  /* Open  the file for append */
  iFResult = open_append(&F_XLS, NAMEFILE);

  setBackColor(0xff, 0xff, 0xff);
  setColor(0xff, 0x00, 0x00);
```

```
if (iFResult != FR_OK)
{
   sprintf(LCD_Mensaje.Btn1, "NO SE PUEDE CREAR ARCHIVO");
   TFT_print(LCD_Mensaje.Btn1, 10, 50, 0);
   /* Error. Cannot create the file */
   while (1)
   {
      if (URTouch_dataAvailable() == true)
      {

         PUSH_BUTTON = UTFT_Buttons_checkButtons();

         if (PUSH_BUTTON == BTN4)
         {
            goto Salir_No_GUARD;
         }
      }
   }
}
/* ---------------------------------------------------------------
 ** CABEZERA DEL ARCHIVO EXCEL
 ** --------------------------------------------------------------- */
f_printf(&F_XLS, "Medicion de 1 Segundo de Vibracion\n ");
f_printf(&F_XLS, "Frec(Hz),Aceleracion(mm/s2),Velocidad(mm/s)\n ");

/* ---------------------------------------------------------------
 ** Almacenamiento hasta el armónico 51
 ** --------------------------------------------------------------- */
int x;
float32_t Accel_SD, veloc_SD;
float32_t Frec;
for (x = 0; x < 512; x++)
{
```

```
        Frec = x * 19.53;
        Accel_SD = VIBRATION.fft_MAG[x] * 1000;     //Conversion de m/s2 a mm/s2
        veloc_SD = VIBRATION.Buffer_B[x] * 1000;    //Conversion de m/s a mm/s

        sprintf(LCD_Mensaje.Btn2, "%3.1f,%3.3f,%3.3f\n", Frec, Accel_SD,
              veloc_SD);

        f_printf(&F_XLS, "%s", LCD_Mensaje.Btn2);

    }
    f_printf(&F_XLS, "Datos de Captura de Informacion\n ");
    f_printf(&F_XLS, "Fecha,Hora\n");
    Hora_Actual = rtc_read();

    sprintf(Hora_Actual.MSG_Fecha, "%02u/%02u/%04u,", Hora_Actual.day,
          Hora_Actual.mth, Hora_Actual.year);
    sprintf(Hora_Actual.MSG_Hora, "%02u:%02u:%02u \n", Hora_Actual.hr,
          Hora_Actual.min, Hora_Actual.sec);

    f_printf(&F_XLS, "%s", Hora_Actual.MSG_Fecha);
    f_printf(&F_XLS, "%s", Hora_Actual.MSG_Hora);

    iFResult = f_close(&F_XLS);
    Salir_No_GUARD: return 0;
}
```

ANEXO 4

Menú de análisis en tiempo real : OS_Measure

```
/*
 * OS_Measure.c
 *
#include "OS_Measure.h"    /*Kernel de Unidad de Procesamiento Grafico*/
#include "LCD_MainMenu.h"
#include "stdio.h"
#include "gio.h"
#include "rti.h"
#include "adc.h"
#include "Kernel_MARIAN.h" /*Kernel de Motor de calculo Electrico*/
#include "SPIFlash.h"
#include "TFT_LCD.h"

lcd_msg Msg_InfoWave = { "", "", "", "", "", "" };
lcd_msg Lcd_InfoWave = { "", "", "", "", "", "" };
lcd_msg Lcd_XY_Axis = { "", "", "", "", "", "" };

extern btnMensajes BtnMensaje;

extern const unsigned short Motor1[];
extern const unsigned short Motor2[];
extern const unsigned short Motor3[];
extern bool Flag_Timer_Medicion;
extern Info_Motor Analizis_Motor;
/* ----------------------------------------------------------------
 * Botones en la pantalla
 * ---------------------------------------------------------------- */
extern int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int MOTOR_NUEVO, MOTOR_COPIA;
char Motor_Name[40] = "";

void OS_Conf_Motor()
```

```
{
   rtiEnableNotification(rtiNOTIFICATION_COMPARE1);

   char Mensaje1[20] = "";
   char Mensaje2[20] = "";

   int btn1, btn2, btn3, btn4, btn_RPM, btn_HP, btn_Instalado_A,
         btn_Instalado_B;
   int btn_MontaA, btn_MontaB, btn_acopleA, btn_acopleB;
   MG_TaskScren();

   /*Configuracion de Menu de Inicio*/
   setColor(0x00, 0x00, 0x00);
   setBackColor(0xd2, 0xd2, 0xd2);
   TFT_print("CONFIG. MOTOR", 20, 10, 0);

   Init_Program:

   UTFT_Buttons_deleteAllButtons();

   BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "ATRAS", BUTTON_DISABLED,
0);
   BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "", BUTTON_DISABLED, 1);
   BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "ENTER",
BUTTON_DISABLED,
                        2);
   BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

   btn1 = UTFT_Buttons_addButton(500, 50, 120, 45, "AC", BUTTON_DISABLED, 10);
   btn2 = UTFT_Buttons_addButton(630, 50, 120, 45, "DC", BUTTON_DISABLED, 11);

   btn3 = UTFT_Buttons_addButton(500, 100, 120, 45, "SI", BUTTON_DISABLED, 12);
```

```
btn4 = UTFT_Buttons_addButton(630, 100, 120, 45, "NO", BUTTON_DISABLED,
13);

btn_Instalado_A = UTFT_Buttons_addButton(500, 250, 120, 45, "Horiz",
BUTTON_DISABLED,
                                16);
btn_Instalado_B = UTFT_Buttons_addButton(630, 250, 120, 45, "Vert",
BUTTON_DISABLED,
                                17);

btn_MontaA = UTFT_Buttons_addButton(500, 300, 120, 45, "Cojin.",
BUTTON_DISABLED,
                              18);
btn_MontaB = UTFT_Buttons_addButton(630, 300, 120, 45, "Chumac.",
BUTTON_DISABLED,
                              19);

btn_acopleA = UTFT_Buttons_addButton(500, 350, 120, 45, "SI",
BUTTON_DISABLED,
                               20);
btn_acopleB = UTFT_Buttons_addButton(630, 350, 120, 45, "NO",
BUTTON_DISABLED,
                               21);

sprintf(Mensaje1, "%d", (int) Analizis_Motor.RPM);
btn_RPM = UTFT_Buttons_addButton(500, 150, 250, 45, Mensaje1,
BUTTON_DISABLED,
                             14);

sprintf(Mensaje2, "%4.1f", Analizis_Motor.HP);
btn_HP = UTFT_Buttons_addButton(500, 200, 250, 45, Mensaje2,
BUTTON_DISABLED,
                            15);
```

```
UTFT_Buttons_enableButton(BTN1, true);
UTFT_Buttons_disableButton(BTN2, true);
UTFT_Buttons_enableButton(BTN3, true);
UTFT_Buttons_enableButton(BTN4, true);

setColor(0x00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
TFT_print("Seleccionar tipo de Motor:", 50, 50, 0);
TFT_print("Motor CA con Variador    :", 50, 100, 0);
TFT_print("RPM del Motor           :", 50, 150, 0);
TFT_print("Potencia del motor (HP)  :", 50, 200, 0);
TFT_print("Tipo de montaje         :", 50, 250, 0);
TFT_print("Tipo de rodamiento       :", 50, 300, 0);
TFT_print("Motor desacoplado       :", 50, 350, 0);

if (Analizis_Motor.TypeMotor == true)  //
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn1, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn2, true);
}
else
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn2, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn1, true);
}

if (Analizis_Motor.VFD == true)  //
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn3, true);
```

```
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn4, true);

}
else
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn4, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn3, true);
}

UTFT_Buttons_setButtonColor(VGA_BLUE);
UTFT_Buttons_enableButton(btn_RPM, true);
UTFT_Buttons_enableButton(btn_HP, true);

if (Analizis_Motor.Montaje_Motor == true)  //
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_Instalado_A, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_Instalado_B, true);

}
else
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_Instalado_B, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_Instalado_A, true);
}

if (Analizis_Motor.TypeSoporte == true)  //
{
```

```
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_MontaA, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_MontaB, true);

}
else
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_MontaB, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_MontaA, true);

}

if (Analizis_Motor.Motor_Acoplado == true)  //
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_acopleA, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_acopleB, true);

}
else
{
    UTFT_Buttons_setButtonColor(VGA_GOLDEN);
    UTFT_Buttons_enableButton(btn_acopleB, true);
    UTFT_Buttons_setButtonColor(VGA_BLUE);
    UTFT_Buttons_enableButton(btn_acopleA, true);
}
/*Etiqueta de color azul para subtitulos*/
/* setColor(181, 200, 200);
 fillRect(0, 45, 799, 172);*/
```

```
/*setColor(0xff, 0xff, 0xff);
 fillRect(200, 350, 600, 390);*/

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BTN1)
      {
         full_Keyboard("INGRESAR NOMBRE DEL MOTOR:",
                  Analizis_Motor.Nombre);
         OS_Conf_Motor();
         goto Init_Program;
      }
      else if (PUSH_BUTTON == BTN3) // Guardar los datos e iniciar las mediciones
      {
         OS_Motor_Guardado();

      }
      else if (PUSH_BUTTON == BTN4)
      {
         break;
      }
      else if (PUSH_BUTTON == btn_RPM)
      {
         Analizis_Motor.RPM = UTFT_Keyboard(0);
         goto Init_Program;
      }
      else if (PUSH_BUTTON == btn_HP)
      {
         Analizis_Motor.HP = UTFT_Keyboard(0);
         goto Init_Program;
```

```
}
else if (PUSH_BUTTON == btn1)   //tipo de motor
{
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn2, true);
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn1, true);
   Analizis_Motor.TypeMotor = true;  //motor en modo AC
}
else if (PUSH_BUTTON == btn2)
{
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn1, true);
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn2, true);

   Analizis_Motor.TypeMotor = false;  //motor en modo DC
}
else if (PUSH_BUTTON == btn3)  //variador conectado
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn3, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn4, true);

   Analizis_Motor.VFD = true;
}
else if (PUSH_BUTTON == btn4)
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn4, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn3, true);
   Analizis_Motor.VFD = false;
```

```
}
else if (PUSH_BUTTON == btn_Instalado_A)    //horientacion del motor
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn_Instalado_A, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn_Instalado_B, true);
   Analizis_Motor.Montaje_Motor = true;

}
else if (PUSH_BUTTON == btn_Instalado_B)
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn_Instalado_B, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn_Instalado_A, true);
   Analizis_Motor.Montaje_Motor = false;
}
else if (PUSH_BUTTON == btn_MontaA)
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn_MontaA, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn_MontaB, true);
   Analizis_Motor.TypeSoporte = true;
}
else if (PUSH_BUTTON == btn_MontaB)
{
   UTFT_Buttons_setButtonColor(VGA_GOLDEN);
   UTFT_Buttons_enableButton(btn_MontaB, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   UTFT_Buttons_enableButton(btn_MontaA, true);
   Analizis_Motor.TypeSoporte = false;
}
```

```
        else if (PUSH_BUTTON == btn_acopleA)
        {
           UTFT_Buttons_setButtonColor(VGA_GOLDEN);
           UTFT_Buttons_enableButton(btn_acopleA, true);
           UTFT_Buttons_setButtonColor(VGA_BLUE);
           UTFT_Buttons_enableButton(btn_acopleB, true);
           Analizis_Motor.Motor_Acoplado = true;
        }
        else if (PUSH_BUTTON == btn_acopleB)
        {
           UTFT_Buttons_setButtonColor(VGA_GOLDEN);
           UTFT_Buttons_enableButton(btn_acopleB, true);
           UTFT_Buttons_setButtonColor(VGA_BLUE);
           UTFT_Buttons_enableButton(btn_acopleA, true);
           Analizis_Motor.Motor_Acoplado = false;
        }

      }
      screen_clock();
   }
}

void OS_Motor_Guardado()
{
   MG_TaskScren_blue();       // pantalla principal de pruebas

   /*Configuración de Menú de Inicio*/
   setColor(0x00, 0x00, 0x00);
   setBackColor(0xd2, 0xd2, 0xd2);

   UTFT_Buttons_deleteAllButtons();

   BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "NUEVO ",
   BUTTON_DISABLED,
```

```
                    0);
  BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "", BUTTON_DISABLED, 1);
  BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "MEDIR",
BUTTON_DISABLED,
                    2);
  BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                    3);

  UTFT_Buttons_enableButton(BTN1, true);
  UTFT_Buttons_disableButton(BTN2, true);
  UTFT_Buttons_enableButton(BTN3, true);
  UTFT_Buttons_enableButton(BTN4, true);

  setColor(0x00, 0x00, 0x00);    // Amarillo
  setBackColor(0x77, 0x99, 0xdd);    // Azul
  TFT_print("Conf. se guardó bien", 200, 160, 0);
  //OS_Print_Motor_Data();

  while (1)
  {
    if (URTouch_dataAvailable() == true)
    {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BTN1)
      {
        full_Keyboard("INGRESAR NOMBRE DEL MOTOR:",
                Analizis_Motor.Nombre);
        OS_Conf_Motor();
      }
      else if (PUSH_BUTTON == BTN3)    //Iniciar Mediciones
      {
        OS_Medicion_Vibracion();
```

```
            }
            else if (PUSH_BUTTON == BTN4)    // boton de salir del programa
            {
               break;
            }
         }
         screen_clock();
      }
   }
```

ANEXO 5

Menús de mediciones para pantalla LCD : OS_Measure_2

```
#include "OS_Measure.h"    /*Kernel de Unidad de Procesamiento Grafico*/
#include "LCD_MainMenu.h"
#include "stdio.h"
#include "gio.h"
#include "rti.h"
#include "adc.h"
#include "Kernel_MARIAN.h" /*Kernel de Motor de cálculo Eléctrico*/
#include "SPIFlash.h"
#include "TFT_LCD.h"

btnMensajes Msg_Analizis;

extern const unsigned short Motor1[];
extern const unsigned short Motor2[];
extern const unsigned short Motor3[];
extern bool Flag_Timer_Medicion;
extern Info_Motor Analizis_Motor;
/* ----------------------------------------------------------------
 * Botones en la pantalla
 * ---------------------------------------------------------------- */
extern int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int btn_POS1, btn_POS2;
extern uint32_t _color_text;

uint8 Flag_task = 0;

void OS_Medicion_Vibracion()
{

   int Touch;
   rtiEnableNotification(rtiNOTIFICATION_COMPARE1);
```

```
/* ---------------------------------------------------------------------
 ** Variable que se utiliza para modificar el control de los botones
 ** en pantalla e iniciarlo secuencialmente de acuerdo
 ** a la tarea que se va a realizar
 ** --------------------------------------------------------------------- */
Flag_task = 0;          //siempre que se ingresa se inicia en cero el task
/* ---------------------------------------------------------------------
 ** Boora la pantalla
 ** --------------------------------------------------------------------- */
MG_TaskScren();
/*Etiqueta de color azul para subtítulos*/

setColor(181, 200, 200);
fillRect(0, 42, 799, 172);

/* ---------------------------------------------------------------------
 ** Inicia el programa Principal
 ** --------------------------------------------------------------------- */

drawBitmap(10, 62, 128, 109, Motor1, 1);
Init_Measure:

setColor(0x00, 0x00, 0x00);
setBackColor(0xd2, 0xd2, 0xd2);
TFT_print("MEDICIONES", 20, 10, 0);
UTFT_Buttons_deleteAllButtons();

BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, " ", BUTTON_DISABLED, 0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "", BUTTON_DISABLED, 1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "DIAGN",
BUTTON_DISABLED,
                    2);
```

```
  BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

  btn_POS1 = UTFT_Buttons_addButton(400, 50, 350, 50, "[1] Medir Punto 1 ",
  BUTTON_DISABLED,
                          10);

  btn_POS2 = UTFT_Buttons_addButton(400, 110, 350, 50, "[2] Medir Punto 2 ",
  BUTTON_DISABLED,
                          11);

  /* ------------------------------------------------------------
   ** Dibujo botones en pantalla
   ** ------------------------------------------------------------ */
  UTFT_Buttons_setButtonColor(VGA_BLUE);
  UTFT_Buttons_disableButton(BTN1, true);
  UTFT_Buttons_disableButton(BTN2, true);

  UTFT_Buttons_enableButton(BTN4, true);

  UTFT_Buttons_setButtonColor(VGA_YELLOW);
  _color_text = VGA_BLACK;

  if (Flag_task == 0)
  {
     UTFT_Buttons_enableButton(btn_POS1, true);
     UTFT_Buttons_disableButton(btn_POS2, true);
     UTFT_Buttons_setButtonColor(VGA_BLUE);
     _color_text = VGA_WHITE;
     UTFT_Buttons_disableButton(BTN3, true);

  }
  else if (Flag_task == 1)
```

```
{
   UTFT_Buttons_disableButton(btn_POS1, true);
   UTFT_Buttons_enableButton(btn_POS2, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   _color_text = VGA_WHITE;
   UTFT_Buttons_disableButton(BTN3, true);
}
else if (Flag_task == 2)
{
   UTFT_Buttons_disableButton(btn_POS1, true);
   UTFT_Buttons_disableButton(btn_POS2, true);
   UTFT_Buttons_setButtonColor(VGA_BLUE);
   _color_text = VGA_WHITE;
   UTFT_Buttons_enableButton(BTN3, true);
}

sprintf(Msg_Analizis.Btn1, "%s", Analizis_Motor.Nombre);
sprintf(Msg_Analizis.Btn2, "%d", (int) Analizis_Motor.RPM);
sprintf(Msg_Analizis.Btn3, "%d", (int) Analizis_Motor.HP);

setColor(0x00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
OS_Print_Motor_Data();
while (1)
{

   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == btn_POS1)
      {
         //adcEnableNotification(adcREG1, adcGROUP0);
         OS_Medicion_Punto_1();
```

```
            Flag_task = 1;
            Menu_after_mesure();
            goto Init_Measure;
        }
        else if (PUSH_BUTTON == btn_POS2)
        {

            OS_Medicion_Punto_2();
            Flag_task = 2;
            Menu_after_mesure();
            goto Init_Measure;
        }
        else if (PUSH_BUTTON == BTN3)
        {
            OS_Diagnostico();
            goto Salida;
        }
        else if (PUSH_BUTTON == BTN4)
        {
            goto Salida;
        }
    }
    screen_clock();
}

Salida: asm("nop");
}
void Menu_after_mesure()
{
    MG_TaskScren();
    /*Etiqueta de color azul para subtitulos*/
    setColor(181, 200, 200);
    fillRect(0, 42, 799, 172);
    setColor(0xff, 0xff, 0xff);
```

```
    fillRect(200, 350, 600, 390);
    drawBitmap(10, 62, 128, 109, Motor1, 1);
    drawBitmap(200, 62, 128, 109, Motor2, 1);

}

/* ---------------------------------------------------------------------
 ** Esta función realiza la captura de la señal y
 ** calcula la transformada de fourier
 ** de 126976 MUESTRAS COMO MAXIMO int16
 ** --------------------------------------------------------------------- */
void OS_Medicion_Punto_1()
{
    MG_TaskScren_blue();
    UTFT_Buttons_deleteAllButtons();

    setColor(254, 204, 16);    // Amarillo
    setBackColor(0x77, 0x99, 0xdd);    // Azul
    TFT_print("Midiendo...", 280, 200, 0);

    /* ---------------------------------------------------------------------
     ** Inicia el Proceso de Medición de la firma de vibración del Motor
     ** en la posicion 1
     ** --------------------------------------------------------------------- */
    //Kernel_Init_Firm_Vibration();
    Kernel_WRITE_Buffer();

    Kernel_READ_Buffer();
    // Kernel_Prom_Firm_Vibration(SIZE_FFT_PROM);
    /* ---------------------------------------------------------------------
     ** Fin del Proceso de Medición 30 Segundos después
     ** --------------------------------------------------------------------- */
    setColor(254, 204, 16);         // Amarillo
    setBackColor(0x77, 0x99, 0xdd);         // Azul
```

```
    TFT_print("Guardando en", 280, 200, 0);
    TFT_print("Memoria Interna...", 280, 220, 0);
    Kernel_Write_Flash_Punto_1();

}

void OS_Medicion_Punto_2()
{
    MG_TaskScren_blue();
    UTFT_Buttons_deleteAllButtons();

    setColor(254, 204, 16);    // Amarillo
    setBackColor(0x77, 0x99, 0xdd);    // Azul
    TFT_print("Midiendo...", 280, 200, 0);

    /* ------------------------------------------------------------
     ** Graba en la memoria interna
     ** ------------------------------------------------------------ */
    Kernel_WRITE_Buffer();

    Kernel_READ_Buffer();
    /* ------------------------------------------------------------
     ** Fin del Proceso de Medición 30 Segundos después
     ** ------------------------------------------------------------ */
    setColor(254, 204, 16);         // Amarillo
    setBackColor(0x77, 0x99, 0xdd);         // Azul
    TFT_print("Guardando en", 280, 200, 0);
    TFT_print("Memoria Interna...", 280, 220, 0);

    Kernel_Write_Flash_Punto_2();

}
     **Genera Botones en la pantalla
     ** ------------------------------------------------------------ */
```

```
UTFT_Buttons_deleteAllButtons();
UTFT_Buttons_setButtonColor(VGA_BLUE);

BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "INIT",
BUTTON_DISABLED,
                        0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "BACK",
BUTTON_DISABLED,
                        1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "NEXT",
BUTTON_DISABLED,
                        2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

/* ---------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** --------------------------------------------------------------- */
UTFT_Buttons_disableButton(BTN1, false);
UTFT_Buttons_disableButton(BTN2, false);
UTFT_Buttons_disableButton(BTN3, false);
UTFT_Buttons_enableButton(BTN4, false);
UTFT_Buttons_drawButtons();

setColor(254, 204, 16);    // Amarillo
setBackColor(0x77, 0x99, 0xdd);    // Azul
TFT_print("Calibrando...", 280, 200, 0);

setColor(0xff, 0xff, 0xff);
fillRect(218, 230, 548, 260);

setColor(254, 204, 16);
```

```
    /* ------------------------------------------------------------------
     ** Inicia el Proceso de Medición de la firma de vibración del Motor
     ** en la posicion 1
     ** ------------------------------------------------------------------ */
    Kernel_Init_Firm_Vibration();
    int ik = 0;
    for (ik = 0; ik < SIZE_FFT_PROM; ik++)
    {
       Kernel_WRITE_Buffer();
       Kernel_READ_Buffer();
       fillRect(220 + 10 * ik, 232, 230 + 10 * ik, 258);
    }
// Kernel_Prom_Firm_Vibration(SIZE_FFT_PROM);
    Kernel_Prom_Firm_Calibration(SIZE_FFT_PROM);
    /* ------------------------------------------------------------------
     ** Fin del Proceso de Medición 30 Segundos después
     ** ------------------------------------------------------------------ */
    setColor(254, 204, 16);        // Amarillo
    setBackColor(0x77, 0x99, 0xdd);        // Azul
    TFT_print("Calibrado...", 280, 200, 0);

    setColor(0x77, 0x99, 0xdd);        // Azul
    fillRect(218, 230, 522, 260);

}

void OS_Print_Motor_Data()
{
    TFT_print("Nombre del Archivo     :", 50, 200, 0);
    TFT_print("Tipo de Motor          :", 50, 220, 0);
    TFT_print("Motor CA con Variador  :", 50, 240, 0);
    TFT_print("RPM del Motor          :", 50, 260, 0);
    TFT_print("Potencia del motor (HP):", 50, 280, 0);
```

```
TFT_print("Motor Instalado        :", 50, 300, 0);
TFT_print("Tipo de rodamiento     :", 50, 320, 0);
TFT_print("Motor desacoplado      :", 50, 340, 0);

TFT_print(Msg_Analizis.Btn1, 450, 200, 0);

if (Analizis_Motor.TypeMotor == true)
{
   TFT_print("AC", 450, 220, 0);
}
else
{
   TFT_print("DC", 450, 220, 0);
}

if (Analizis_Motor.VFD == true)
{
   TFT_print("SI", 450, 240, 0);
}
else
{
   TFT_print("NO", 450, 240, 0);

}

TFT_print(Msg_Analizis.Btn2, 450, 260, 0);
TFT_print(Msg_Analizis.Btn3, 450, 280, 0);

if (Analizis_Motor.Montaje_Motor == true)
{
   TFT_print("Horizontal", 450, 300, 0);
}
else
{
```

```
        TFT_print("Vertical", 450, 300, 0);
    }

    if (Analizis_Motor.TypeSoporte == true)
    {
        TFT_print("Cojin", 450, 320, 0);
    }
    else
    {
        TFT_print("Chumacera", 450, 320, 0);
    }

    if (Analizis_Motor.Motor_Acoplado == true)
    {

        TFT_print("SI", 450, 340, 0);
    }
    else
    {
        TFT_print("NO", 450, 340, 0);
    }

}
```

ANEXO 6

Resultados del analisis : OS_Diagnosis

```
#include "OS_Measure.h"    /*Kernel de Unidad de Procesamiento Grafico*/
#include "OS_Diagnosis.h"
#include "LCD_MainMenu.h"
#include "stdio.h"
#include "rti.h"
#include "Kernel_MARIAN.h" /*Kernel de Motor de cálculo Eléctrico*/
#include "SPIFlash.h"
#include "TFT_LCD.h"
#include "DS1302.h"
#include "ff.h"
#include "sd_defs.h"
#include <stdlib.h>    /* strtol */
/* ----------------------------------------------------------------
 **Defines
 ** ---------------------------------------------------------------- */
#define X_point        100
#define Y_point        380
/* ----------------------------------------------------------------
 **Variables del tamaño de letra y touch screen
 ** ---------------------------------------------------------------- */
extern uint8_t BigFont[];
extern uint8_t SmallFont[];

extern btnMensajes BtnMensaje;
extern btnMensajes Msg_Analizis;

extern const unsigned short Motor1[];
extern const unsigned short Motor2[];
extern const unsigned short Motor3[];
extern bool Flag_Timer_Medicion;
extern Info_Motor Analizis_Motor;
/* ----------------------------------------------------------------
```

```
 * Botones en la pantalla
 * ---------------------------------------------------------------- */
extern int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int BTN_ZONA1, BTN_ZONA2;
extern uint32_t _color_text;
extern Signal_wave VIBRATION;

Vibration_Firma Zona1, Zona2;
extern btnMensajes LCD_Mensaje;

void OS_Diagnostico()
{
   /* ----------------------------------------------------------------
    **Creación del Menú Principal
    ** ---------------------------------------------------------------- */
   Diagnosis_Main: MG_TaskScren();
   setColor(0x00, 0x00, 0x00);
   setBackColor(0xd2, 0xd2, 0xd2);
   TFT_print("RESULTADOS", 20, 10, 0);

   UTFT_Buttons_deleteAllButtons();

   /* ----------------------------------------------------------------
    **Inicialización de botones en pantalla
    ** ---------------------------------------------------------------- */
   BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
   BUTTON_DISABLED,
                        0);
   BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "VER SD",
   BUTTON_DISABLED,
                        1);
   BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "GRABAR",
   BUTTON_DISABLED,
                        2);
```

```
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

BTN_ZONA1 = UTFT_Buttons_addButton(200, 100, 380, 80, "SENSOR ZONA 1",
BUTTON_DISABLED,
                            10);
BTN_ZONA2 = UTFT_Buttons_addButton(200, 200, 380, 80, "SENSOR ZONA 2",
BUTTON_DISABLED,
                            11);
/* ---------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** --------------------------------------------------------------- */
UTFT_Buttons_disableButton(BTN1, true);
UTFT_Buttons_enableButton(BTN2, true);
UTFT_Buttons_enableButton(BTN3, true);
UTFT_Buttons_enableButton(BTN4, true);
UTFT_Buttons_enableButton(BTN_ZONA1, true);
UTFT_Buttons_enableButton(BTN_ZONA2, true);

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();
      if (PUSH_BUTTON == BTN_ZONA1)
      {
         OS_Diagnostico_Zona1();
         goto Diagnosis_Main;
      }
      else if (PUSH_BUTTON == BTN_ZONA2)
      {
         OS_Diagnostico_Zona2();
         goto Diagnosis_Main;
```

```
        }
        else if (PUSH_BUTTON == BTN2)
        {
           OS_MonitorSD(); // Realiza el cambio en el lugar no requiere regresar
           goto Diagnosis_Main;
        }
        else if (PUSH_BUTTON == BTN3)
        {
           OS_Almacenar_SD();
           goto Diagnosis_Main;
        }
        else if (PUSH_BUTTON == BTN4)
        {
           break;
        }
      }
      screen_clock();
   }

}

void OS_Almacenar_SD()

{/* ---------------------------------------------------------------
 **Variables de la memoria SD
 ** ---------------------------------------------------------------- */
   FRESULT iFResult;
   FIL F_XLS; /* File objects */
   btnMensajes LCD_Mensaje, titulo;
   RTC_DS1302 Hora_Actual;

   /* ---------------------------------------------------------------
    **Creación del Menú Principal
    ** ---------------------------------------------------------------- */
```

```
MG_TaskScren();
setColor(0x00, 0x00, 0x00);
setBackColor(0xd2, 0xd2, 0xd2);
TFT_print("ALMACENANDO DATOS", 20, 10, 0);

UTFT_Buttons_deleteAllButtons();
/* ---------------------------------------------------------------------
 **Inicialización de botones en pantalla
 ** --------------------------------------------------------------------- */
BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
BUTTON_DISABLED,
                    0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "",
BUTTON_DISABLED,
                    1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "",
BUTTON_DISABLED,
                    2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                    3);
/* ---------------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** --------------------------------------------------------------------- */
UTFT_Buttons_disableButton(BTN1, true);
UTFT_Buttons_disableButton(BTN2, true);
UTFT_Buttons_disableButton(BTN3, true);
UTFT_Buttons_enableButton(BTN4, true);

setColor(0x00, 0x00, 0x00);
setBackColor(0Xff, 0Xff, 0xff);
TFT_print("Lectura de Datos Memoria Interna", 10, 50, 0);

delay(1);
```

```
TFT_print("Lectura de Datos Zona 1", 10, 70, 0);
delay(1);
/* ---------------------------------------------------------------------
 ** Lectura de la memoria Interna del Microcontrolador
 ** --------------------------------------------------------------------- */
Cluster_FRead(ADDR_FFTP1_Accel, Zona1.FFTp_Acceleration,
LENGTH_SAMPLES / 4);

TFT_print("Lectura de Datos Zona 2", 10, 90, 0);
delay(1);
Cluster_FRead(ADDR_FFTP2_Accel, Zona2.FFTp_Acceleration,
LENGTH_SAMPLES / 4);

TFT_print("Iniciando Kernel", 10, 110, 0);
Kernel_FFT_Velocidad(&VIBRATION, &Zona1);
Kernel_FFT_Velocidad(&VIBRATION, &Zona2);
delay(1);
TFT_print("Creando Archivo en memoria SD:", 10, 130, 0);
/* ---------------------------------------------------------------------
 ** Creación de archivo EXCEL
 ** --------------------------------------------------------------------- */
sprintf(titulo.Btn1, "%s.CSV", Analizis_Motor.Nombre);
delay(1);

const char *NAMEFILE = titulo.Btn1;

/* Open  the file for append */
iFResult = open_append(&F_XLS, NAMEFILE);

setBackColor(0xff, 0xff, 0xff);
setColor(0xff, 0x00, 0x00);
if (iFResult != FR_OK)
{
   sprintf(LCD_Mensaje.Btn1, "NO SE PUEDE CREAR ARCHIVO");
```

```
    TFT_print(LCD_Mensaje.Btn1, 10, 150, 0);
    /* Error. Cannot create the file */
    while (1)
    {
       if (URTouch_dataAvailable() == true)
       {

          PUSH_BUTTON = UTFT_Buttons_checkButtons();

          if (PUSH_BUTTON == BTN4)
          {
             goto Salir_No_save;
          }
       }
    }
  }
  else
  {
    //  sprintf(LCD_Mensaje.Btn1, "NO SE PUEDE CREAR ARCHIVO");
    TFT_print(titulo.Btn1, 10, 150, 0);
  }
  /* ------------------------------------------------------------------
   ** CABEZERA DEL ARCHIVO EXCEL
   ** ------------------------------------------------------------------ */
  f_printf(
        &F_XLS,
        "Frec(Hz),Aceleracion 1(mm/s2),Velocidad 1(mm/s),Aceleracion
2(mm/s2),Velocidad 2(mm/s)\n ");
  /* ------------------------------------------------------------------
   ** Almacenamiento hasta el armónico 51
   ** ------------------------------------------------------------------ */
  int x;
  float Accel_SD_Zona1, veloc_SD_Zona1;
  float Accel_SD_Zona2, veloc_SD_Zona2;
```

```
float Frec;

for (x = 0; x < 512; x++)
{

   Frec = x * 19.53;
   Accel_SD_Zona1 = Zona1.FFTp_Acceleration[x] * 1000;
   veloc_SD_Zona1 = Zona1.FFTp_Velocidad[x] * 1000;

   Accel_SD_Zona2 = Zona2.FFTp_Acceleration[x] * 1000;
   veloc_SD_Zona2 = Zona2.FFTp_Velocidad[x] * 1000;

   sprintf(LCD_Mensaje.Btn2, "%3.1f,%3.3f,%3.3f,%3.3f,%3.3f\n", Frec,
         Accel_SD_Zona1, veloc_SD_Zona1, Accel_SD_Zona2, veloc_SD_Zona2);

   f_printf(&F_XLS, "%s", LCD_Mensaje.Btn2);

}

Hora_Actual = rtc_read();

sprintf(Hora_Actual.MSG_Fecha, "%02u/%02u/%04u,", Hora_Actual.day,
      Hora_Actual.mth, Hora_Actual.year);
sprintf(Hora_Actual.MSG_Hora, "%02u:%02u:%02u \n", Hora_Actual.hr,
      Hora_Actual.min, Hora_Actual.sec);

f_printf(&F_XLS, "Fecha,");
f_printf(&F_XLS, "%s\n", Hora_Actual.MSG_Fecha);
f_printf(&F_XLS, "Hora,");
f_printf(&F_XLS, "%s\n", Hora_Actual.MSG_Hora);

f_printf(&F_XLS, "Datos del Motor:,%s\n", Analizis_Motor.Nombre);

f_printf(&F_XLS, "RPM:,%d\n", (int) Analizis_Motor.RPM);
```

```
sprintf(LCD_Mensaje.Btn2, "%5.1f", Analizis_Motor.HP);
f_printf(&F_XLS, "Potencia(HP):,%s\n", LCD_Mensaje.Btn2);

if (Analizis_Motor.TypeMotor == true)
{
   f_printf(&F_XLS, "Tipo de Motor:, AC\n");
}
else
{
   f_printf(&F_XLS, "Tipo de Motor:, DC\n");
}

if (Analizis_Motor.VFD == true)
{
   f_printf(&F_XLS, "Conectado a VFD:, Si\n");
}
else
{
   f_printf(&F_XLS, "Conectado a VFD:, No\n");
}

if (Analizis_Motor.Montaje_Motor == true)
{
   f_printf(&F_XLS, "Tipo de Montaje:, Horizontal\n");
}
else
{
   f_printf(&F_XLS, "Tipo de Montaje:, Vertical\n");
}

if (Analizis_Motor.TypeSoporte == true)
{
   f_printf(&F_XLS, "Tipo de Rodamiento:, Cojin\n");
```

```
}
else
{
   f_printf(&F_XLS, "Tipo de Rodamiento:, Chumacera\n");
}

if (Analizis_Motor.Motor_Acoplado == true)
{
   f_printf(&F_XLS, "Motor Desacoplado:, Si\n");
}
else
{
   f_printf(&F_XLS, "Motor Desacoplado:, No\n");
}
setColor(0x00, 0x00, 0x00);
setBackColor(0Xff, 0Xff, 0xff);
TFT_print("Datos Almacenados", 10, 170, 0);
TFT_print("Presionar [SALIR], para salir", 10, 190, 0);
iFResult = f_close(&F_XLS);

while (1)
{
   if (URTouch_dataAvailable() == true)
   {

      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BTN4)
      {
         goto Salir_No_save;
      }
   }
}
```

```
    Salir_No_save: return 0;
}

void OS_Diagnostico_Zona1()
{

    MG_TaskScren();
    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("RESULTADOS - ZONA 1", 20, 10, 0);
    Init_Buttons_zone();

    Cluster_FRead(ADDR_FFTP1_Accel, Data_Vibration.FFTp_Acceleration,
    LENGTH_SAMPLES / 4);

    /*   Kernel_FFT_Velocidad(&VIBRATION, &Data_Vibration);
     Plot_XY_VELOCIDAD();*/

    Kernel_FFT_Aceleracion(&VIBRATION, &Data_Vibration);
    Plot_XY_ACELERACION();

    setFont(BigFont);
    setColor(0, 0, 0);
    setBackColor(0xff, 0xff, 0xff);

    float Frecuencia = 19.53 * VIBRATION.max_marks1;
    float Max_Aceel = VIBRATION.Aceleracion_max * 1000; // convertir de m/s2 a
mm/s2
    float RMS_Accel = VIBRATION.RMS * 1000;     // convertir de m/s2 a mm/s2
    sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
    sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
    sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

    setBackColor(0xff, 0xff, 0xff);
```

```
setColor(0x00, 0x00, 0x00);
TFT_print("MAX[mm/s2]", 620, 100, 0);
TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
TFT_print("FREC[Hz]:", 620, 140, 0);
TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
TFT_print("RMS[mm/s2]", 620, 180, 0);
TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();
      if (PUSH_BUTTON == BTN1)
      {
         Kernel_FFT_Aceleracion(&VIBRATION, &Data_Vibration);
         Plot_XY_ACELERACION();
         setFont(BigFont);
         setColor(0, 0, 0);
         setBackColor(0xff, 0xff, 0xff);

         Frecuencia = 19.53 * VIBRATION.max_marks1;
         Max_Aceel = VIBRATION.Aceleracion_max * 1000; // convertir de m/s2 a
mm/s2
         RMS_Accel = VIBRATION.RMS * 1000;   // convertir de m/s2 a mm/s2
         sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
         sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
         sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

         setBackColor(0xff, 0xff, 0xff);
         setColor(0x00, 0x00, 0x00);
         TFT_print("MAX[mm/s2]", 620, 100, 0);
         TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
         TFT_print("FREC[Hz]:", 620, 140, 0);
```

```
    TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
    TFT_print("RMS[mm/s2]", 620, 180, 0);
    TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);
}
else if (PUSH_BUTTON == BTN2)
{
    Kernel_FFT_Velocidad(&VIBRATION, &Data_Vibration);
    Plot_XY_VELOCIDAD();
    setFont(BigFont);
    setColor(0, 0, 0);
    setBackColor(0xff, 0xff, 0xff);

    Frecuencia = 19.53 * VIBRATION.max_marks1;
    Max_Aceel = VIBRATION.valor_MAX * 1000; // convertir de m/s a mm/s
    RMS_Accel = VIBRATION.RMS * 1000;   // convertir de m/s a mm/s
    sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
    sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
    sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

    setBackColor(0xff, 0xff, 0xff);
    setColor(0x00, 0x00, 0x00);
    TFT_print("MAX[mm/s]", 620, 100, 0);
    TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
    TFT_print("FREC[Hz]:", 620, 140, 0);
    TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
    TFT_print("RMS[mm/s]", 620, 180, 0);
    TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);
}
else if (PUSH_BUTTON == BTN3)
{
}
else if (PUSH_BUTTON == BTN4)
{
    break;
```

```
            }
        }
        screen_clock();
    }
}
void OS_Diagnostico_Zona2()
{
    MG_TaskScren();
    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("RESULTADOS - ZONA 2", 20, 10, 0);
    Init_Buttons_zone();

    Cluster_FRead(ADDR_FFTP2_Accel, Data_Vibration.FFTp_Acceleration,
    LENGTH_SAMPLES / 4);
    Kernel_FFT_Aceleracion(&VIBRATION, &Data_Vibration);
    Plot_XY_ACELERACION();

    float Frecuencia = 19.53 * VIBRATION.max_marks1;
    float Max_Aceel = VIBRATION.Aceleracion_max * 1000; // convertir de m/s2 a
mm/s2
    float RMS_Accel = VIBRATION.RMS * 1000;     // convertir de m/s2 a mm/s2
    sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
    sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
    sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

    setFont(BigFont);
    setColor(0, 0, 0);
    setBackColor(0xff, 0xff, 0xff);
    setBackColor(0xff, 0xff, 0xff);
    setColor(0x00, 0x00, 0x00);
    TFT_print("MAX[mm/s2]", 620, 100, 0);
    TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
    TFT_print("FREC[Hz]:", 620, 140, 0);
```

```
TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
TFT_print("RMS[mm/s2]", 620, 180, 0);
TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);
/*Kernel_FFT_Velocidad(&VIBRATION, &Data_Vibration);
 Plot_XY_VELOCIDAD();*/

/* Kernel_FFT_Aceleracion(&VIBRATION, &Data_Vibration);
 Plot_XY_ACELERACION();*/

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();
      if (PUSH_BUTTON == BTN1)
      {
         Kernel_FFT_Aceleracion(&VIBRATION, &Data_Vibration);
         Plot_XY_ACELERACION();

         Frecuencia = 19.53 * VIBRATION.max_marks1;
         Max_Aceel = VIBRATION.Aceleracion_max * 1000; // convertir de m/s2 a
mm/s2
         RMS_Accel = VIBRATION.RMS * 1000;   // convertir de m/s2 a mm/s2
         sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
         sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
         sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

         setFont(BigFont);
         setColor(0, 0, 0);
         setBackColor(0xff, 0xff, 0xff);
         setBackColor(0xff, 0xff, 0xff);
         setColor(0x00, 0x00, 0x00);
         TFT_print("MAX[mm/s2]", 620, 100, 0);
         TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
```

```
    TFT_print("FREC[Hz]:", 620, 140, 0);
    TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
    TFT_print("RMS[mm/s2]", 620, 180, 0);
    TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);
}
else if (PUSH_BUTTON == BTN2)
{
    Kernel_FFT_Velocidad(&VIBRATION, &Data_Vibration);
    Plot_XY_VELOCIDAD();

    Frecuencia = 19.53 * VIBRATION.max_marks1;
    Max_Aceel = VIBRATION.valor_MAX * 1000; // convertir de m/s a mm/s
    RMS_Accel = VIBRATION.RMS * 1000;   // convertir de m/s a mm/s
    sprintf(LCD_Mensaje.Btn5, "%3.2f  ", Max_Aceel);
    sprintf(LCD_Mensaje.Btn6, "%2.1f", Frecuencia);
    sprintf(LCD_Mensaje.Btn7, "%2.3f", RMS_Accel);

    setFont(BigFont);
    setColor(0, 0, 0);
    setBackColor(0xff, 0xff, 0xff);
    setBackColor(0xff, 0xff, 0xff);
    setColor(0x00, 0x00, 0x00);
    TFT_print("MAX[mm/s]", 620, 100, 0);
    TFT_print(LCD_Mensaje.Btn5, 620, 120, 0);
    TFT_print("FREC[Hz]:", 620, 140, 0);
    TFT_print(LCD_Mensaje.Btn6, 620, 160, 0);
    TFT_print("RMS[mm/s]", 620, 180, 0);
    TFT_print(LCD_Mensaje.Btn7, 620, 200, 0);
}
else if (PUSH_BUTTON == BTN3)
{
}
else if (PUSH_BUTTON == BTN4)
{
```

```
            break;
          }
        }
        screen_clock();
    }

}

void Plot_XY_VELOCIDAD()
{
    btnMensajes LCD_Mensaje;
    /*Etapa blanca de la pantalla para generación de gráficas y cuadros de texto*/
    int x, a = X_point;
    float FFT_M;
    int x_base = Y_point;
    int Resol = 1, ih;
    float yval = 0; //imprime los valores del eje y
    float xval = 0; //imprime los valores del eje x
    int X_Resol = 1; // Tipo de paso para ampliar o reducir el espectro de frecuencia
    int Limit_x = 512;  //Límite máximo del ancho de grafico
    int Num_lin = (int) (Limit_x) / 10; //Numero de divisiones que tendrá el eje X, en este
caso 7
    float tempF;

    /*Etapa blanca de la pantalla para generación de gráficas y cuadros de texto*/
    setColor(0xff, 0xff, 0xff);
    fillRect(0, 42, 799, 430);

    setColor(0x00, 0x00, 0x00);

    /*Genera el eje X*/
    drawLine(a - 6, x_base + 1, a + 512, x_base + 1);
    drawLine(a - 6, x_base + 2, a + 512, x_base + 2);
    /*Genera el eje Y*/
```

```
drawLine(a - 1, x_base - FFT_MAX_HIGH, a - 1, x_base + 6);
drawLine(a - 2, x_base - FFT_MAX_HIGH, a - 2, x_base + 6);

/*Configuración de propiedades del texto*/
setFont(BigFont);
setColor(0, 0, 0);
setBackColor(0xff, 0xff, 0xff);

setColor(0, 0, 0);
TFT_print("Velocidad (mm/s)", 20, 350, -90);
TFT_print("Frecuencia[KHz]", 200, x_base + 25, 0);

a = X_point;

/*Configuracion de propiedades del texto*/
setFont(SmallFont);

setBackColor(0xff, 0xff, 0xff);
float mul = 0;
for (ih = 0; ih <= 30; ih = ih + 5)
{

   yval = VIBRATION.Velocidad_max * mul;
   sprintf(LCD_Mensaje.Btn1, "%2.2f ", yval);
   mul = mul + 0.16666;    //---->5/30

   setColor(0, 0, 0);
   TFT_print(LCD_Mensaje.Btn1, a - 45, x_base - ih * 10 - 6, 0);
   if (ih != 0)
   {
      setColor(0xd2, 0xd2, 0xd2);
      drawLine(a - 6, x_base - ih * 10, a + 512, x_base - ih * 10);
   }
}
```

```
int xh = 0;

//Eje x

xval = 0;

for (ih = 0; ih <= 10; ih++)
{

   tempF = (xval / X_Resol);
   int temF_int = (int) tempF;
   // xval = VIBRATION.Velocidad_max * mul;
   sprintf(LCD_Mensaje.Btn1, "%2.1f ", tempF);

   setColor(0, 0, 0);
   TFT_print(LCD_Mensaje.Btn1, a + xh - 5, x_base + 10, 0);
   if (ih != 0)
   {
      setColor(0xd2, 0xd2, 0xd2);
      drawLine(a + xh, x_base - 300, a + xh, x_base);
   }
   xval++;
   xh = xh + Num_lin;
}

for (x = 0; x < Limit_x / X_Resol; x++)
{

   /* ----------------------------------------------------------------
    ** Borra la gráfica anterior para una nueva grafica
    ** ---------------------------------------------------------------- */

   setColor(0xff, 0xff, 0xff);
   drawPixel(a, x_base - VIBRATION.XY_buff[x]);
```

```
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 2);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 3);

FFT_M = abs(VIBRATION.XY_buff[x + 1] - VIBRATION.XY_buff[x]);

if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_buff[x], a + 1,
         x_base - VIBRATION.XY_buff[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 1, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 1);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 2, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 2);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 3, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 3);
}
/* ---------------------------------------------------------------
 ** Imprime en pantalla las barras de potencia armónica.
 ** --------------------------------------------------------------- */
setColor(0x00, 0x00, 0xff);
drawPixel(a, x_base - VIBRATION.XY_plot[x]);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 2);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 3);
FFT_M = abs(VIBRATION.XY_plot[x + 1] - VIBRATION.XY_plot[x]);
if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_plot[x], a + 1,
         x_base - VIBRATION.XY_plot[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_plot[x] - 1, a + 1,
         x_base - VIBRATION.XY_plot[x + 1] - 1);

   drawLine(a, x_base - VIBRATION.XY_plot[x] - 2, a + 1,
```

```
                x_base - VIBRATION.XY_plot[x + 1] - 2);

            drawLine(a, x_base - VIBRATION.XY_plot[x] - 2, a + 1,
                x_base - VIBRATION.XY_plot[x + 1] - 2);
        }
        a = a + X_Resol;
    }
}

void Plot_XY_ACELERACION()
{
    btnMensajes LCD_Mensaje;
    /*Etapa blanca de la pantalla para generación de gráficas y cuadros de texto*/
    int x, a = X_point;
    float FFT_M;
    int x_base = Y_point;
    int Resol = 1, ih;
    float yval = 0; //imprime los valores del eje y
    float xval = 0; //imprime los valores del eje x
    int X_Resol = 1; // Tipo de paso para ampliar o reducir el espectro de frecuencia
    int Limit_x = 512;  //Limite maximo de l ancho de grafico
    int Num_lin = (int) (Limit_x) / 10; //Numero de divisiones que tendra el eje X, en este
caso 7
    float tempF;

    /*Etapa blanca de la pantalla para generación de gráficas y cuadros de texto*/
    setColor(0xff, 0xff, 0xff);
    fillRect(0, 42, 799, 430);

    setColor(0x00, 0x00, 0x00);

    /*Genera el eje X*/
    drawLine(a - 6, x_base + 1, a + 512, x_base + 1);
    drawLine(a - 6, x_base + 2, a + 512, x_base + 2);
```

```
/*Genera el eje Y*/
drawLine(a - 1, x_base - FFT_MAX_HIGH, a - 1, x_base + 6);
drawLine(a - 2, x_base - FFT_MAX_HIGH, a - 2, x_base + 6);

/*Configuración de propiedades del texto*/
setFont(BigFont);
setColor(0, 0, 0);
setBackColor(0xff, 0xff, 0xff);

setColor(0, 0, 0);
TFT_print("Aceleracion(mm/s2)", 20, 350, -90);
TFT_print("Frecuencia[KHz]", 200, x_base + 25, 0);

a = X_point;

/*Configuración de propiedades del texto*/
setFont(SmallFont);

setBackColor(0xff, 0xff, 0xff);
float mul = 0;
for (ih = 0; ih <= 30; ih = ih + 5)
{

   yval = VIBRATION.Aceleracion_max * mul * 1000;
   sprintf(LCD_Mensaje.Btn1, "%2.2f ", yval);
   mul = mul + 0.16666;    //---->5/30

   setColor(0, 0, 0);
   TFT_print(LCD_Mensaje.Btn1, a - 45, x_base - ih * 10 - 6, 0);
   if (ih != 0)
   {
      setColor(0xd2, 0xd2, 0xd2);
      drawLine(a - 6, x_base - ih * 10, a + 512, x_base - ih * 10);
   }
```

```
}
int xh = 0;

//Eje x

xval = 0;

for (ih = 0; ih <= 10; ih++)
{

   tempF = (xval / X_Resol);
   int temF_int = (int) tempF;
   // xval = VIBRATION.Velocidad_max * mul;
   sprintf(LCD_Mensaje.Btn1, "%2.1f ", tempF);

   setColor(0, 0, 0);
   TFT_print(LCD_Mensaje.Btn1, a + xh - 5, x_base + 10, 0);
   if (ih != 0)
   {
      setColor(0xd2, 0xd2, 0xd2);
      drawLine(a + xh, x_base - 300, a + xh, x_base);
   }
   xval++;
   xh = xh + Num_lin;
}

for (x = 0; x < Limit_x / X_Resol; x++)
{

   /* ----------------------------------------------------------------
    ** Borra la gráfica anterior para una nueva grafica
    ** ---------------------------------------------------------------- */

   setColor(0xff, 0xff, 0xff);
```

```
drawPixel(a, x_base - VIBRATION.XY_buff[x]);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 2);
drawPixel(a, x_base - VIBRATION.XY_buff[x] - 3);

FFT_M = abs(VIBRATION.XY_buff[x + 1] - VIBRATION.XY_buff[x]);

if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_buff[x], a + 1,
         x_base - VIBRATION.XY_buff[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 1, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 1);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 2, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 2);
   drawLine(a, x_base - VIBRATION.XY_buff[x] - 3, a + 1,
         x_base - VIBRATION.XY_buff[x + 1] - 3);
}
/* ---------------------------------------------------------------------
 ** Imprime en pantalla las barras de potencia armonica.
 ** --------------------------------------------------------------------- */
setColor(0xff, 0x00, 0x00);
drawPixel(a, x_base - VIBRATION.XY_plot[x]);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 1);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 2);
drawPixel(a, x_base - VIBRATION.XY_plot[x] - 3);
FFT_M = abs(VIBRATION.XY_plot[x + 1] - VIBRATION.XY_plot[x]);
if (FFT_M > Resol)
{
   drawLine(a, x_base - VIBRATION.XY_plot[x], a + 1,
         x_base - VIBRATION.XY_plot[x + 1]);
   drawLine(a, x_base - VIBRATION.XY_plot[x] - 1, a + 1,
         x_base - VIBRATION.XY_plot[x + 1] - 1);
```

```
            drawLine(a, x_base - VIBRATION.XY_plot[x] - 2, a + 1,
                    x_base - VIBRATION.XY_plot[x + 1] - 2);

            drawLine(a, x_base - VIBRATION.XY_plot[x] - 2, a + 1,
                    x_base - VIBRATION.XY_plot[x + 1] - 2);
        }
        a = a + X_Resol;
    }
}

void Init_Buttons_zone()
{
    UTFT_Buttons_deleteAllButtons();
    /* ------------------------------------------------------------------
     **Inicializacion de botones en pantalla
     ** ------------------------------------------------------------------ */
    BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "ACEL",
    BUTTON_DISABLED,
                            0);
    BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "VELO",
    BUTTON_DISABLED,
                            1);
    BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "",
    BUTTON_DISABLED,
                            2);
    BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
    BUTTON_DISABLED,
                            3);
    /* ------------------------------------------------------------------
     ** Impresión de Botones en pantalla
     ** ------------------------------------------------------------------ */
    UTFT_Buttons_enableButton(BTN1, true);
    UTFT_Buttons_enableButton(BTN2, true);
    UTFT_Buttons_disableButton(BTN3, true);
```

```
    UTFT_Buttons_enableButton(BTN4, true);
}
```

ANEXO 7

Menu de configuraciones de equipo : OS_Settings

```
#include "OS_Settings.h"    /*Kernel de Unidad de Procesamiento Grafico*/
#include "LCD_MainMenu.h"
#include "stdio.h"
#include "gio.h"
#include "het.h"
#include "DS1302.h"
#include "SPIFlash.h"
/* ------------------------------------------------------------------
 **Tipos de Texto
 ** ------------------------------------------------------------------ */
extern uint8_t BigFont[];
extern uint8_t SmallFont[];
extern uint8_t SevenSegNumFont[];
/* ------------------------------------------------------------------
 **Imagenes
 ** ------------------------------------------------------------------ */
extern const unsigned short logo_bluetooth[];
/* ------------------------------------------------------------------
 **Variables y Estructuras
 ** ------------------------------------------------------------------ */
//btnMensajes BtnMensaje = { 0, "", "", "", "", "", "", "", "" };
extern int BTN1, BTN2, BTN3, BTN4, PUSH_BUTTON;
int BNT_TIME, BTN_LCD, BTN_AUTO, BTN_BLUT, BTN_ELECTRIC, BTN_BATT;
extern uint32_t _color_text, _color_text_inactive, _color_background,
       _color_border, _color_hilite;
RTC_DS1302 Calendario;

Seleccion OS_Select;
uint8 Touch, Menu_CSelect;
```

```
extern btnMensajes BtnMensaje;
extern RTC_DS1302 Calendario;

#define NumberConfi 8          // Numero de configuraciones que se tiene en el Menu

#define PASSWD_EQ                  1234        //Almacena el KVARH
extern float TEMP_BACKLIGHT; //120--->2 MINUTO

void OS_Init()
{
   OS_Menu_Principal: //Status_Bluetooth_eeprom();
   /* ------------------------------------------------------------------
    **Limpia la pantalla solo el recuadro superior y el inferior
    ** ------------------------------------------------------------------ */
   _color_text = VGA_WHITE;
   _color_text_inactive = VGA_GRAY;
   _color_background = VGA_BLUE;
   _color_border = VGA_GRAY;
   _color_hilite = VGA_RED;
   Status_Backlight_eeprom();
   MG_TaskScren();        // pantalla principal de pruebas

   setColor(0x00, 0x00, 0x00);
   setBackColor(0xd2, 0xd2, 0xd2);
   TFT_print("MENU DE CONFIGURACIONES", 20, 10, 0);
   /* ------------------------------------------------------------------
    **Genera Botones en la pantalla
    ** ------------------------------------------------------------------ */

   UTFT_Buttons_deleteAllButtons();
   UTFT_Buttons_setButtonColor(VGA_BLUE);

   BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
   BUTTON_DISABLED,
```

```
                    0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "",
BUTTON_DISABLED,
                    1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "",
BUTTON_DISABLED,
                    2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                    3);
BNT_TIME = UTFT_Buttons_addButton(200, 50, 380, 50, "Hora y Fecha",
BUTTON_DISABLED,
                      10);
BTN_LCD = UTFT_Buttons_addButton(200, 105, 380, 50, "Luz de LCD",
BUTTON_DISABLED,
                      11);
BTN_AUTO = UTFT_Buttons_addButton(200, 160, 380, 50, "Autocalibracion",
BUTTON_DISABLED,
                      12);
/*  BTN_BLUT = UTFT_Buttons_addButton(200, 215, 380, 50, "Bluetooth",
 BUTTON_DISABLED,
 13);*/
/* BTN_ELECTRIC = UTFT_Buttons_addButton(200, 215, 380, 50, "Conf. Electricas",
 BUTTON_DISABLED,
 13);*/
BTN_BATT = UTFT_Buttons_addButton(200, 215, 380, 50, "Guardar Bateria",
BUTTON_DISABLED,
                      14);
/* ------------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** ------------------------------------------------------------------ */
UTFT_Buttons_disableButton(BTN1, false);
UTFT_Buttons_disableButton(BTN2, false);
UTFT_Buttons_disableButton(BTN3, false);
```

```
UTFT_Buttons_enableButton(BTN4, false);

UTFT_Buttons_enableButton(BNT_TIME, false);
UTFT_Buttons_enableButton(BTN_LCD, false);
UTFT_Buttons_enableButton(BTN_AUTO, false);
//UTFT_Buttons_disableButton(BTN_BLUT, false);
//   UTFT_Buttons_enableButton(BTN_ELECTRIC, false);
UTFT_Buttons_enableButton(BTN_BATT, false);

UTFT_Buttons_drawButtons();

/* ---------------------------------------------------------------
 **Inicia la ejecucion del programa
 ** --------------------------------------------------------------- */
setColor(0x00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
//* TFT_print("Cop. 11.09.20", 20, 400, 0);

int PSSWRD;

while (1)
{

   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();

      if (PUSH_BUTTON == BNT_TIME)
      {
         Set_date_time();
         goto OS_Menu_Principal;
      }
      else if (PUSH_BUTTON == BTN_LCD)
      {
```

```
    OS_Set_Lcd_BackLight();
    goto OS_Menu_Principal;

  }
  else if (PUSH_BUTTON == BTN_AUTO)
  {

    PSSWRD = (int) UTFT_Keyboard(0);
    if (PSSWRD == PASSWD_EQ)
    {       //Ingresar Contraseña

      OS_Calibracion_FFT(); // Realiza el cambio en el lugar no requiere regresar
    }

    goto OS_Menu_Principal;
  }
  else if (PUSH_BUTTON == BTN_BLUT)
  {

  }
  else if (PUSH_BUTTON == BTN_ELECTRIC)
  {
    //  OS_Set__Voltmax_Corrmax();
    goto OS_Menu_Principal;
  }
  else if (PUSH_BUTTON == BTN_BATT)
  {
    OS_Set_GUAR_Bateria();
    goto OS_Menu_Principal;
  }
  else if (PUSH_BUTTON == BTN4)
    break;

}
```

```
        screen_clock();
    }
}

void Set_date_time()
{
    /* ---------------------------------------------------------------------
     **Limpia la pantalla solo el recuadro superior y el inferior
     ** --------------------------------------------------------------------- */
    MG_TaskScren();        // pantalla principal de pruebas

    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("CONFIGURAR RELOJ", 20, 10, 0);

    /* ---------------------------------------------------------------------
     **Genera Botones en la pantalla
     ** --------------------------------------------------------------------- */

    UTFT_Buttons_deleteAllButtons();

    BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "NEXT",
    BUTTON_DISABLED,
                                  0);
    BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "SET",
    BUTTON_DISABLED,
                                  1);
    BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "GUAR",
    BUTTON_DISABLED,
                                  2);
    BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
    BUTTON_DISABLED,
                                  3);
```

```
/* ------------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** ------------------------------------------------------------------ */
UTFT_Buttons_enableButton(BTN1, false);
UTFT_Buttons_enableButton(BTN2, false);
UTFT_Buttons_enableButton(BTN3, false);
UTFT_Buttons_enableButton(BTN4, false);

UTFT_Buttons_drawButtons();
setColor(0, 0, 0);
setBackColor(0xff, 0xff, 0xff);
TFT_print("DIA", 150, 60, 0);
TFT_print("MES", 150, 120, 0);
TFT_print("YEAR", 150, 180, 0);

TFT_print("HORA", 550, 60, 0);
TFT_print("MIN", 550, 120, 0);
TFT_print("SEC", 550, 180, 0);

setFont(SevenSegNumFont);

int Buscar_clk = 0;
int conf_day = bcdToDec(read_ds1302(0x87));
int conf_mth = bcdToDec(read_ds1302(0x89));
int conf_year = bcdToDec(read_ds1302(0x8d));
int conf_hr = bcdToDec(read_ds1302(0x85));
int conf_min = bcdToDec(read_ds1302(0x83));

while (1)
{
   if (URTouch_dataAvailable() == true)
   {
      PUSH_BUTTON = UTFT_Buttons_checkButtons();
```

```
if (PUSH_BUTTON == BTN1)
{
   Buscar_clk++;
   if (Buscar_clk > 4)
      Buscar_clk = 0;
}
else if (PUSH_BUTTON == BTN2)
{
   if (Buscar_clk == 0)    // Dias
   {
      conf_day++;
      if (conf_day > 31)
         conf_day = 1;
   }
   else if (Buscar_clk == 1)   //Meses
   {
      conf_mth++;
      if (conf_mth > 12)
         conf_mth = 1;
   }
   else if (Buscar_clk == 2)   //Años
   {
      conf_year++;
      if (conf_year > 99)
         conf_year = 0;
   }
   else if (Buscar_clk == 3)   //Horas
   {
      conf_hr++;
      if (conf_hr > 24)
         conf_hr = 0;
   }
   else if (Buscar_clk == 4)   //Minutos
   {
```

```
            conf_min++;
            if (conf_min > 59)
                conf_min = 0;
        }

    }
    else if (PUSH_BUTTON == BTN3)
    {
        rtc_set_datetime(conf_day, conf_mth, conf_year, 0, conf_hr,
                    conf_min); //Comando para la configuracion de la hora en el RTC
    }
    else if (PUSH_BUTTON == BTN4)
        break;
}

setFont(SevenSegNumFont);       // Tipo de Letra gigante
Config_clock_time(Buscar_clk);

int conf_sec = bcdToDec(read_ds1302(0x81));

sprintf(BtnMensaje.Btn8, "%02u", conf_day);
sprintf(BtnMensaje.Btn9, "%02u", conf_mth);
sprintf(BtnMensaje.Btn10, "%02u", conf_year);

sprintf(BtnMensaje.Btn11, "%02u", conf_hr);
sprintf(BtnMensaje.Btn12, "%02u", conf_min);
sprintf(BtnMensaje.Btn13, "%02u", conf_sec);

setColor(0, 0, 0);
setBackColor(0xff, 0xff, 0xff);

TFT_print(BtnMensaje.Btn8, 50, 60, 0);
TFT_print(BtnMensaje.Btn9, 50, 120, 0);
TFT_print(BtnMensaje.Btn10, 50, 180, 0);
```

```
        TFT_print(BtnMensaje.Btn11, 450, 60, 0);
        TFT_print(BtnMensaje.Btn12, 450, 120, 0);
        TFT_print(BtnMensaje.Btn13, 450, 180, 0);
        setFont(BigFont);
        screen_clock();

        //delay(200e-3);
    }

}

void OS_Set_GUAR_Bateria()
{
    float Set_GUAR_Bateria = 0;
    /* ------------------------------------------------------------------
     **Limpia la pantalla solo el recuadro superior y el inferior
     ** ------------------------------------------------------------- */
    _color_text = VGA_WHITE;
    _color_text_inactive = VGA_GRAY;
    _color_background = VGA_BLUE;
    _color_border = VGA_GRAY;
    _color_hilite = VGA_RED;

    MG_TaskScren();       // pantalla principal de pruebas

    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("CONF. BATERIA", 20, 10, 0);
    /* ------------------------------------------------------------------
     **Genera Botones en la pantalla
     ** ------------------------------------------------------------- */

    UTFT_Buttons_deleteAllButtons();
    UTFT_Buttons_setButtonColor(VGA_BLUE);
```

```
BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
BUTTON_DISABLED,
                        0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "",
BUTTON_DISABLED,
                        1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "GUAR",
BUTTON_DISABLED,
                        2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                        3);

BNT_TIME = UTFT_Buttons_addButton(30, 50, 380, 50, "Desabilitado",
BUTTON_DISABLED,
                          10);
BTN_LCD = UTFT_Buttons_addButton(30, 105, 380, 50, "3 min",
BUTTON_DISABLED,
                          11);
BTN_AUTO = UTFT_Buttons_addButton(30, 160, 380, 50, "5 min",
BUTTON_DISABLED,
                          12);
BTN_BLUT = UTFT_Buttons_addButton(30, 215, 380, 50, "30 min",
BUTTON_DISABLED,
                          13);
/* ---------------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** --------------------------------------------------------------------- */
UTFT_Buttons_disableButton(BTN1, false);
UTFT_Buttons_disableButton(BTN2, false);
UTFT_Buttons_enableButton(BTN3, false);
UTFT_Buttons_enableButton(BTN4, false);

UTFT_Buttons_enableButton(BNT_TIME, false);
```

```
    UTFT_Buttons_enableButton(BTN_LCD, false);
    UTFT_Buttons_enableButton(BTN_AUTO, false);
    UTFT_Buttons_enableButton(BTN_BLUT, false);
    UTFT_Buttons_drawButtons();

    Set_GUAR_Bateria                                                    =
READ_FLOAT_EXT_EEPROM(ADDRS_TEMP_BACKLIGHT);

    // Set_GUAR_Bateria = 120; //cambiar

    if (Set_GUAR_Bateria == 120)
       sprintf(BtnMensaje.Btn1, "3 min  ");
    else if (Set_GUAR_Bateria == 300)
       sprintf(BtnMensaje.Btn1, "5 min  ");
    else if (Set_GUAR_Bateria == 1800)
       sprintf(BtnMensaje.Btn1, "30 min ");

    OS_Menu_Bateria: setColor(0x00, 0x00, 0x00);
    setBackColor(0xff, 0xff, 0xff);
    TFT_print("Conf. Acutal:", 500, 100, 0);
    TFT_print(BtnMensaje.Btn1, 500, 120, 0);
    TFT_print("         ", 500, 200, 0);
    while (1)
    {

       if (URTouch_dataAvailable() == true)
       {
          PUSH_BUTTON = UTFT_Buttons_checkButtons();

          if (PUSH_BUTTON == BNT_TIME)
          {

             sprintf(BtnMensaje.Btn1, "Apagado ");
             goto OS_Menu_Bateria;
```

```
        }
        else if (PUSH_BUTTON == BTN_LCD)
        {
           Set_GUAR_Bateria = 180;   //120Segundos
           sprintf(BtnMensaje.Btn1, "3 min   ");
           goto OS_Menu_Bateria;

        }
        else if (PUSH_BUTTON == BTN_AUTO)
        {
           Set_GUAR_Bateria = 300;
           sprintf(BtnMensaje.Btn1, "5 min   ");
           goto OS_Menu_Bateria;

        }
        else if (PUSH_BUTTON == BTN_BLUT)
        {
           Set_GUAR_Bateria = 600;
           sprintf(BtnMensaje.Btn1, "100 min ");
           goto OS_Menu_Bateria;

        }

        else if (PUSH_BUTTON == BTN3)
        {
           WRITE_FLOAT_EXT_EEPROM(ADDRS_TEMP_BACKLIGHT,
Set_GUAR_Bateria);
           setColor(0x00, 0x00, 0x00);
           setBackColor(0xff, 0xff, 0xff);
           TFT_print("Guardado", 500, 200, 0);
           TEMP_BACKLIGHT                                          =
READ_FLOAT_EXT_EEPROM(ADDRS_TEMP_BACKLIGHT);
           //    goto OS_Menu_Bateria;
        }
```

```
            else if (PUSH_BUTTON == BTN4)
                break;

        }
        screen_clock();
    }
}

void Status_Backlight_eeprom()
{
    int bcklight_lcd = (int) READ_FLOAT_EXT_EEPROM(ADDR_BACKLIGHT);
    pwmSetDuty(hetRAM1, pwm3, bcklight_lcd); // Pantalla Inicializa apagada para no mostrar efectos de actualizacion
// pwmSetDuty(hetRAM2, pwm0, bcklight_lcd); // POTENCIA DEL BACKIGHT LCD
}

void OS_Set_Lcd_BackLight()
{
    float Set_lcd_backlight = 0;
    /* ---------------------------------------------------------------------
     **Limpia la pantalla solo el recuadro superior y el inferior
     ** --------------------------------------------------------------- */
    _color_text = VGA_WHITE;
    _color_text_inactive = VGA_GRAY;
    _color_background = VGA_BLUE;
    _color_border = VGA_GRAY;
    _color_hilite = VGA_RED;

    MG_TaskScren();        // pantalla principal de pruebas

    setColor(0x00, 0x00, 0x00);
    setBackColor(0xd2, 0xd2, 0xd2);
    TFT_print("LUZ DE LCD", 20, 10, 0);
    /* ---------------------------------------------------------------------
```

```
 **Genera Botones en la pantalla
 ** ---------------------------------------------------------------- */

UTFT_Buttons_deleteAllButtons();
UTFT_Buttons_setButtonColor(VGA_BLUE);
BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "",
BUTTON_DISABLED,
                    0);
BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "",
BUTTON_DISABLED,
                    1);
BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "GUAR",
BUTTON_DISABLED,
                    2);
BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
BUTTON_DISABLED,
                    3);

BNT_TIME = UTFT_Buttons_addButton(30, 50, 200, 50, "100%",
BUTTON_DISABLED,
                     10);
BTN_LCD = UTFT_Buttons_addButton(30, 105, 200, 50, "80%",
BUTTON_DISABLED,
                     11);
BTN_AUTO = UTFT_Buttons_addButton(30, 160, 200, 50, "60",
BUTTON_DISABLED,
                     12);
BTN_BLUT = UTFT_Buttons_addButton(30, 215, 200, 50, "40",
BUTTON_DISABLED,
                     13);
/* ---------------------------------------------------------------
 ** Impresión de Botones en pantalla
 ** ---------------------------------------------------------------- */
UTFT_Buttons_disableButton(BTN1, false);
```

```
UTFT_Buttons_disableButton(BTN2, false);
UTFT_Buttons_enableButton(BTN3, false);
UTFT_Buttons_enableButton(BTN4, false);

UTFT_Buttons_enableButton(BNT_TIME, false);
UTFT_Buttons_enableButton(BTN_LCD, false);
UTFT_Buttons_enableButton(BTN_AUTO, false);
UTFT_Buttons_enableButton(BTN_BLUT, false);
UTFT_Buttons_drawButtons();

Set_lcd_backlight = READ_FLOAT_EXT_EEPROM(ADDR_BACKLIGHT);

Status_Backlight_eeprom();

OS_blacklight_lcd: sprintf(BtnMensaje.Btn1, "%3.0f%%   ",
                   Set_lcd_backlight);

setColor(0x00, 0x00, 0x00);
setBackColor(0xff, 0xff, 0xff);
TFT_print("Conf. Acutal:", 500, 100, 0);
TFT_print(BtnMensaje.Btn1, 500, 120, 0);
TFT_print("        ", 500, 200, 0);
while (1)
{

  if (URTouch_dataAvailable() == true)
  {
    PUSH_BUTTON = UTFT_Buttons_checkButtons();

    if (PUSH_BUTTON == BNT_TIME)
    {

      Set_lcd_backlight = 100;
      goto OS_blacklight_lcd;
```

```
    }
    else if (PUSH_BUTTON == BTN_LCD)
    {
       Set_lcd_backlight = 80;
       goto OS_blacklight_lcd;

    }
    else if (PUSH_BUTTON == BTN_AUTO)
    {
       Set_lcd_backlight = 60;
       goto OS_blacklight_lcd;

    }
    else if (PUSH_BUTTON == BTN_BLUT)
    {
       Set_lcd_backlight = 40;
       goto OS_blacklight_lcd;

    }

    else if (PUSH_BUTTON == BTN3)
    {
       WRITE_FLOAT_EXT_EEPROM(ADDR_BACKLIGHT, Set_lcd_backlight);
       setColor(0x00, 0x00, 0x00);
       setBackColor(0xff, 0xff, 0xff);
       TFT_print("Guardado", 500, 200, 0);
       Status_Backlight_eeprom();
       //    goto OS_Menu_Bateria;
    }
    else if (PUSH_BUTTON == BTN4)
       break;

}
screen_clock();
```

```
    }
}

void OS_MonitorSD()
{

   /* ------------------------------------------------------------------
    **Limpia la pantalla solo el recuadro superior y el inferior
    ** ------------------------------------------------------------------ */
   MG_TaskScren();      // Limpia la pantalla
   // MG_TaskScren_blue();        // pantalla principal de pruebas

   setColor(0x00, 0x00, 0x00);
   setBackColor(0xd2, 0xd2, 0xd2);
   TFT_print("MONITOR DE MEMORIA", 20, 10, 0);
   /* ------------------------------------------------------------------
    **Genera Botones en la pantalla
    ** ------------------------------------------------------------------ */

   UTFT_Buttons_deleteAllButtons();
   UTFT_Buttons_setButtonColor(VGA_BLUE);

   BTN1 = UTFT_Buttons_addButton(4, 435, 190, 40, "INIT",
   BUTTON_DISABLED,
                           0);
   BTN2 = UTFT_Buttons_addButton(205, 435, 190, 40, "ATRAS",
   BUTTON_DISABLED,
                           1);
   BTN3 = UTFT_Buttons_addButton(405, 435, 190, 40, "SIGT",
   BUTTON_DISABLED,
                           2);
   BTN4 = UTFT_Buttons_addButton(605, 435, 190, 40, "SALIR",
   BUTTON_DISABLED,
                           3);
```

```
    /* ---------------------------------------------------------------------
     ** Impresión de Botones en pantalla
     ** --------------------------------------------------------------------- */
    UTFT_Buttons_enableButton(BTN1, false);
    UTFT_Buttons_enableButton(BTN2, false);
    UTFT_Buttons_enableButton(BTN3, false);
    UTFT_Buttons_enableButton(BTN4, false);
    UTFT_Buttons_drawButtons();
    /* ---------------------------------------------------------------------
     **Inicializo los valores de los mensajes en botones
     ** --------------------------------------------------------------------- */
    Cmd_Monitor();
    //setFont(BigFont);

}
```

ANEXO 8

Librerías para el control de la LCD: TFT_LCD.c

```
#include "sys_common.h"
#include "gio.h"                    // Necesaria para controlar los gioPORTA
#include "het.h"
#include "spi.h"
#include "math.h"
#include "stdio.h"
#include "string.h"
#include "TFT_LCD.h"

bool _transparent;
extern uint8_t BigFont[];
extern const uint8_t Dingbats1_XL[];
uint8 orient = 1;
long disp_y_size = 799;

int TP_X, TP_Y;
long _default_orientation;
uint8 prec;
uint8 display_model;
long disp_x_size, default_orientation;
long touch_x_left, touch_x_right, touch_y_top, touch_y_bottom;

uint8 fcolorr, fcolorg, fcolorb;
uint8 bcolorr, bcolorg, bcolorb;
int HIGH = 1, LOW = 0;
spiDAT1_t HC595;

button_type buttons[MAX_BUTTONS];
uint32_t _color_text, _color_text_inactive, _color_background, _color_border,
```

```
        _color_hilite;
uint8_t *_font_text, *_font_symbol;
extern uint16 POWER_BACKLIGHT;

void tftInit()
{
   lcdPin_t cs = { hetPORT1, 20 };
   lcdPin_t rst = { hetPORT1, 18 };
   lcdPin_t wr = { hetPORT1, 16 };
   lcdPin_t rs = { hetPORT1, 30 };

   lcdPin_t T_clock = { hetPORT1, 12 };
   lcdPin_t T_cs = { gioPORTA, 5 };
   lcdPin_t T_DIN = { gioPORTA, 2 };
   lcdPin_t T_DOUT = { gioPORTA, 1 };
   lcdPin_t T_IRQ = { gioPORTA, 0 };      //Agregar Interrupción al programa

   lcdPin_t T_Buzzer = { gioPORTB, 2 };
   lcdPin_t T_LED_RECORD = { hetPORT1, 4 };

   lcdPin_t PWM0_LCD_BACKLIGHT = { hetPORT1, 14 };

   HC595.CS_HOLD = FALSE;
   HC595.WDEL = TRUE;
   HC595.DFSEL = SPI_FMT_0;          //No require
   HC595.CSNR = SPI_PIN_CS1;

   lcdSetPinDirection(&rs, pinWrite);
   lcdSetPinDirection(&wr, pinWrite);
   lcdSetPinDirection(&cs, pinWrite);
   lcdSetPinDirection(&rst, pinWrite);

   lcdSetPinDirection(&T_clock, pinWrite);
   lcdSetPinDirection(&T_cs, pinWrite);
```

```
    lcdSetPinDirection(&T_DIN, pinWrite);
    lcdSetPinDirection(&T_DOUT, pinRead);
    lcdSetPinDirection(&T_IRQ, pinRead);
    lcdSetPinDirection(&PWM0_LCD_BACKLIGHT, pinWrite);

    lcdSetPinDirection(&T_LED_RECORD, pinWrite);
    lcdSetPinDirection(&T_Buzzer, pinWrite);
    InitLCD(LANDSCAPE);
    /*Configuracion Iniciales de la pantalla LCD*/
    setFont(BigFont);
    fillScr(0xff, 0xff, 0xff);
    pwmSetDuty(hetRAM2, pwm2, 0);                    // POTENCIA DEL BACKIGHT LCD
    URTouch_InitTouch(LANDSCAPE);
    URTouch_setPrecision(PREC_MEDIUM);
}

void Sound_Buzzer()
{
    Set_Buzzer(true);
    delay(10e-2);
    Set_Buzzer(false);
    // delay(100e-3);
}

void Set_Buzzer(bool flag_buzzer)
{
    if (flag_buzzer == true)
    {
        gioSetBit(gioPORTB, 2, 1);      //Buzzer Pin control

    }
    else
    {
        gioSetBit(gioPORTB, 2, 0);
```

```
    }
}

void HC595_SPI(uint16 write_data)
{
    uint16 buffer_write[2]; // Initial Instruction

    /*Load Instruction and address and data*/
    buffer_write[0] = (uint16) write_data;
    buffer_write[1] = 0;

    /*Low CS0 and Transmit Data SPI to 25LC256 and High CS0*/
    //  gioSetBit(spiPORT1, 1, LOW);
    spiTransmitData(spiREG1, &HC595, 1, buffer_write);
    //   gioSetBit(spiPORT1, 1, HIGH);
}

uint8 digitalRead(unsigned int pin)
{
    uint8 value;
    value = gioGetBit(gioPORTA, pin);
    return value;
}
void digitalWrite(unsigned int pin, unsigned int value)
{
    if (pin == CS || pin == TOUCH_DIN)
        gioSetBit(gioPORTA, pin, value);
    else
        gioSetBit(hetPORT1, pin, value);
}

void Lcd_Writ_Bus(char VH, char VL)
{
    //unsigned char data;
```

```
    uint16 data;
    data = VL + (VH << 8);

    HC595_SPI(data);

    digitalWrite(LCD_WR, LOW);
    //delay(1e-3);
    digitalWrite(LCD_WR, HIGH);
    //delay(1e-3);
}

void Lcd_Write_Com(char VH, char VL)
{
    digitalWrite(LCD_RS, LOW);
    Lcd_Writ_Bus(VH, VL);
}

void Lcd_Write_Data(char VH, char VL)
{
    digitalWrite(LCD_RS, HIGH);
    Lcd_Writ_Bus(VH, VL);
}

void Lcd_Write_Com_Data(int com, int dat)
{
    Lcd_Write_Com(com >> 8, com);
    Lcd_Write_Data(dat >> 8, dat);
}

void Address_set(unsigned int x1, unsigned int y1, unsigned int x2,
                 unsigned int y2)
{
    Lcd_Write_Com_Data(0x0044, (x2 << 8) + x1);
    Lcd_Write_Com_Data(0x0045, y1);
```

```
    Lcd_Write_Com_Data(0x0046, y2);
    Lcd_Write_Com_Data(0x004e, x1);
    Lcd_Write_Com_Data(0x004f, y1);
    Lcd_Write_Com(0x00, 0x22);
}

void InitLCD(uint8 orientation)
{

    orient = orientation;

    digitalWrite(LCD_REST, HIGH);
    delay(5e-3);
    digitalWrite(LCD_REST, LOW);
    delay(15e-3);
    digitalWrite(LCD_REST, HIGH);
    delay(15e-3);

    digitalWrite(LCD_CS, LOW);

    /*Inicializacion de la nueva pantalla LCD*/
    LCD_Write_COM(0xE2);                //PLL multiplier, set PLL clock to 120M
    LCD_Write_DATA(0x00, 0x23);    //N=0x36 for 6.5M, 0x23 for 10M crystal
    LCD_Write_DATA(0x00, 0x02);
    LCD_Write_DATA(0x00, 0x04);
    LCD_Write_COM(0xE0);                // PLL enable
    LCD_Write_DATA(0x00, 0x01);
    delay(10e-3);
    LCD_Write_COM(0xE0);
    LCD_Write_DATA(0x00, 0x03);
    delay(10e-3);
    LCD_Write_COM(0x01);                // software reset
    delay(100e-3);
    LCD_Write_COM(0xE6);          //PLL setting for PCLK, depends on resolution
```

```
LCD_Write_DATA(0x00, 0x04);
LCD_Write_DATA(0x00, 0x93);
LCD_Write_DATA(0x00, 0xE0);

LCD_Write_COM(0xB0);              //LCD SPECIFICATION
LCD_Write_DATA(0x00, 0x00);       // 0x24
LCD_Write_DATA(0x00, 0x00);
LCD_Write_DATA(0x00, 0x03);       //Set HDP      799
LCD_Write_DATA(0x00, 0x1F);
LCD_Write_DATA(0x00, 0x01);       //Set VDP      479
LCD_Write_DATA(0x00, 0xDF);
LCD_Write_DATA(0x00, 0x00);

LCD_Write_COM(0xB4);              //HSYNC
LCD_Write_DATA(0x00, 0x03);       //Set HT       928
LCD_Write_DATA(0x00, 0xA0);
LCD_Write_DATA(0x00, 0x00);       //Set HPS      46
LCD_Write_DATA(0x00, 0x2E);
LCD_Write_DATA(0x00, 0x30);       //Set HPW      48
LCD_Write_DATA(0x00, 0x00);       //Set LPS      15
LCD_Write_DATA(0x00, 0x0F);
LCD_Write_DATA(0x00, 0x00);

LCD_Write_COM(0xB6);              //VSYNC
LCD_Write_DATA(0x00, 0x02);       //Set VT       525
LCD_Write_DATA(0x00, 0x0D);
LCD_Write_DATA(0x00, 0x00);       //Set VPS      16
LCD_Write_DATA(0x00, 0x10);
LCD_Write_DATA(0x00, 0x10);       //Set VPW      16

/*IMPORTANTE REVISAR*/
LCD_Write_DATA(0x00, 0x00);       //Set FPS      8

LCD_Write_DATA(0x00, 0x08);
```

```
LCD_Write_COM(0xBA);
LCD_Write_DATA(0x00, 0x05);          //GPIO[3:0] out 1

LCD_Write_COM(0xB8);
LCD_Write_DATA(0x00, 0x07);    //GPIO3=input, GPIO[2:0]=output
LCD_Write_DATA(0x00, 0x01);          //GPIO0 normal

LCD_Write_COM(0x36);                 //rotation
LCD_Write_DATA(0x00, 0x22);          // -- Set to 0x21 to rotate 180 degrees

LCD_Write_COM(0xF0);                 //pixel data interface
LCD_Write_DATA(0x00, 0x03);

delay(10e-3);

setXY(0, 0, 799, 479);
LCD_Write_COM(0x29);                 //display on

LCD_Write_COM(0xBE);                 //set PWM for B/L
LCD_Write_DATA(0x00, 0x06);
LCD_Write_DATA(0x00, 0xF0);
LCD_Write_DATA(0x00, 0x01);
LCD_Write_DATA(0x00, 0xF0);
LCD_Write_DATA(0x00, 0x00);
LCD_Write_DATA(0x00, 0x00);

LCD_Write_COM(0xD0);
LCD_Write_DATA(0x00, 0x0D);

LCD_Write_COM(0x2C);

/***************************************/
```

```
    digitalWrite(LCD_CS, HIGH);
    setColor(255, 255, 255);
    setBackColor(0, 0, 0);
    cfont.font = 0;

}

void setXY(word x1, word y1, word x2, word y2)
{
    if (orient == LANDSCAPE)
    {
        swap(word, x1, y1);
        swap(word, x2, y2)
        y1 = disp_y_size - y1;
        y2 = disp_y_size - y2;
        swap(word, y1, y2)
    }

    swap(word, x1, y1);
    swap(word, x2, y2);

    LCD_Write_COM(0x2a);
    LCD_Write_DATA(0x00, x1 >> 8);
    LCD_Write_DATA(0x00, x1);
    LCD_Write_DATA(0x00, x2 >> 8);
    LCD_Write_DATA(0x00, x2);
    LCD_Write_COM(0x2b);
    LCD_Write_DATA(0x00, y1 >> 8);
    LCD_Write_DATA(0x00, y1);
    LCD_Write_DATA(0x00, y2 >> 8);
    LCD_Write_DATA(0x00, y2);
    LCD_Write_COM(0x2c);

}
```

```
void setColor_word(uint32_t color)
{

   fcolorr = (color >> 16) & 0xFF; // Extract the RR byte
   fcolorg = (color >> 8) & 0xFF; // Extract the GG byte
   fcolorb = (color) & 0xFF; // Extract the BB byte

}
void setBackColor_word(uint32_t color)
{
   if (color == VGA_TRANSPARENT)
      _transparent = true;
   else
   {
      bcolorr = (color >> 16) & 0xFF; // Extract the RR byte
      bcolorg = (color >> 8) & 0xFF; // Extract the GG byte
      bcolorb = (color) & 0xFF; // Extract the BB byte
      _transparent = false;
   }
}
uint32_t getColor_word()
{

   return (fcolorr << 16 | fcolorg << 8 | fcolorb);
}

uint32_t getBackColor()
{
   return (bcolorr << 16 | bcolorg << 8 | bcolorb);
}

void setColor(uint8 r, uint8 g, uint8 b)
{
```

```
    fcolorr = r;
    fcolorg = g;
    fcolorb = b;
}

void setBackColor(uint8 r, uint8 g, uint8 b)
{
    bcolorr = r;
    bcolorg = g;
    bcolorb = b;
}

void setFont(uint8_t *font)
{
    cfont.font = font;
    cfont.x_size = pgm_read_byte(&cfont.font[0]);
    cfont.y_size = pgm_read_byte(&cfont.font[1]);
    cfont.offset = pgm_read_byte(&cfont.font[2]);
    cfont.numchars = pgm_read_byte(&cfont.font[3]);
}

uint8_t* getFont()
{
    return cfont.font;
}

uint8_t getFontXsize()
{
    return cfont.x_size;
}

uint8_t getFontYsize()
{
    return cfont.y_size;
```

```
}

void TFT_print(char *st, int x, int y, int deg)
{
  int stl, i;

  stl = strlen(st);

  if (orient == PORTRAIT)
  {
    if (x == RIGHT)
      x = 480 - (stl * cfont.x_size);
    if (x == CENTER)
      x = (480 - (stl * cfont.x_size)) / 2;
  }
  else
  {
    if (x == RIGHT)
      x = (disp_y_size + 1) - (stl * cfont.x_size);
    if (x == CENTER)
      x = ((disp_y_size + 1) - (stl * cfont.x_size)) / 2;
  }

  for (i = 0; i < stl; i++)
    if (deg == 0)
      TFT_printChar(*st++, x + (i * (cfont.x_size)), y);
    else
      rotateChar(*st++, x, y, i, deg);
}

void TFT_printChar(uint8 c, int x, int y)
{
  int i, j, temp;
  uint8 ch;
```

```
//digitalWrite(LCD_CS, LOW);
digitalWrite(LCD_CS, LOW);
if (orient == PORTRAIT)
{
   setXY(x, y, x + cfont.x_size - 1, y + cfont.y_size - 1);

   temp = ((c - cfont.offset) * ((cfont.x_size / 8) * cfont.y_size)) + 4;
   for (j = 0; j < ((cfont.x_size / 8) * cfont.y_size); j++)
   {
      ch = pgm_read_byte(&cfont.font[temp]);
      for (i = 0; i < 8; i++)
      {
         if ((ch & (1 << (7 - i))) != 0)
         {
            setPixel(fcolorr, fcolorg, fcolorb);
         }
         else
         {
            setPixel(bcolorr, bcolorg, bcolorb);
         }
      }
      temp++;
   }
}
else
{
   temp = ((c - cfont.offset) * ((cfont.x_size / 8) * cfont.y_size)) + 4;

   for (j = 0; j < ((cfont.x_size / 8) * cfont.y_size);
         j += (cfont.x_size / 8))
   {
      setXY(x, y + (j / (cfont.x_size / 8)), x + cfont.x_size - 1,
           y + (j / (cfont.x_size / 8)));
```

```
            int zz = 0;
            for (zz = (cfont.x_size / 8) - 1; zz >= 0; zz--)
            {
               ch = pgm_read_byte(&cfont.font[temp + zz]);
               for (i = 0; i < 8; i++)
               {
                  if ((ch & (1 << i)) != 0)
                  {
                     setPixel(fcolorr, fcolorg, fcolorb);
                  }
                  else
                  {
                     setPixel(bcolorr, bcolorg, bcolorb);
                  }
               }
            }
            temp += (cfont.x_size / 8);
         }
      }
      //digitalWrite(LCD_CS, HIGH);
      digitalWrite(LCD_CS, HIGH);
}
void setPixel(uint8 r, uint8 g, uint8 b)
{
   LCD_Write_DATA(((r & 248) | g >> 5), ((g & 28) << 3 | b >> 3));
}

void LCD_Write_DATA(char VH, char VL)
{
   //*P_RS |= B_RS;
   digitalWrite(LCD_RS, HIGH);
   Lcd_Writ_Bus(VH, VL);
}
```

```
void LCD_Write_COM(char VL)
{
   //*P_RS &= ~B_RS;
   digitalWrite(LCD_RS, LOW);
   Lcd_Writ_Bus(0x00, VL);
}
void main_W_com_data(char com1, int dat1)
{
   LCD_Write_COM(com1);
   LCD_Write_DATA(dat1 >> 8, dat1);
}

void rotateChar(uint8 c, int x, int y, int pos, int deg)
{
   uint8 i, j, ch;
   word temp;
   int newx, newy;
   double radian;
   radian = deg * 0.0175;

   digitalWrite(LCD_CS, LOW);

   temp = ((c - cfont.offset) * ((cfont.x_size / 8) * cfont.y_size)) + 4;
   for (j = 0; j < cfont.y_size; j++)
   {
      int zz = 0;
      for (zz = 0; zz < (cfont.x_size / 8); zz++)
      {
         ch = pgm_read_byte(&cfont.font[temp + zz]);
         for (i = 0; i < 8; i++)
         {
            newx = x
                  + (((i + (zz * 8) + (pos * cfont.x_size)) * cos(radian))
                        - ((j) * sin(radian)));
```

```
            newy = y
                  + (((j) * cos(radian))
                        + ((i + (zz * 8) + (pos * cfont.x_size))
                              * sin(radian)));
            setXY(newx, newy, newx + 1, newy + 1);

            if ((ch & (1 << (7 - i))) != 0)
            {
               setPixel(fcolorr, fcolorg, fcolorb);
            }
            else
            {
               setPixel(bcolorr, bcolorg, bcolorb);
            }
         }
      }
      temp += (cfont.x_size / 8);
   }
   digitalWrite(LCD_CS, HIGH);
}

void drawRect(int x1, int y1, int x2, int y2)
{
   int tmp;

   if (x1 > x2)
   {
      tmp = x1;
      x1 = x2;
      x2 = tmp;
   }
   if (y1 > y2)
   {
      tmp = y1;
```

```
        y1 = y2;
        y2 = tmp;
    }

    drawHLine(x1, y1, x2 - x1);
    drawHLine(x1, y2, x2 - x1);
    drawVLine(x1, y1, y2 - y1);
    drawVLine(x2, y1, y2 - y1);
}

void drawLine(int x1, int y1, int x2, int y2)
{
    int tmp, i;
    double delta, tx, ty;
    //double m, b, dx, dy;
    char ch, cl;

    ch = ((fcolorr & 248) | fcolorg >> 5);
    cl = ((fcolorg & 28) << 3 | fcolorb >> 3);

    if (((x2 - x1) < 0))
    {
        tmp = x1;
        x1 = x2;
        x2 = tmp;
        tmp = y1;
        y1 = y2;
        y2 = tmp;
    }
    if (((y2 - y1) < 0))
    {
        tmp = x1;
        x1 = x2;
        x2 = tmp;
```

```
    tmp = y1;
    y1 = y2;
    y2 = tmp;
}

if (y1 == y2)
{
    if (x1 > x2)
    {
        tmp = x1;
        x1 = x2;
        x2 = tmp;
    }
    drawHLine(x1, y1, x2 - x1);
}
else if (x1 == x2)
{
    if (y1 > y2)
    {
        tmp = y1;
        y1 = y2;
        y2 = tmp;
    }
    drawVLine(x1, y1, y2 - y1);
}
else if (abs(x2 - x1) > abs(y2 - y1))
{
    digitalWrite(LCD_CS, LOW);
    delta = ((double) (y2 - y1) / (double) (x2 - x1));
    ty = (double) (y1);
    if (x1 > x2)
    {
        for (i = x1; i >= x2; i--)
        {
```

```
            setXY(i, (int) (ty + 0.5), i, (int) (ty + 0.5));
            LCD_Write_DATA(ch, cl);
            ty = ty - delta;
        }
    }
    else
    {
        for (i = x1; i <= x2; i++)
        {
            setXY(i, (int) (ty + 0.5), i, (int) (ty + 0.5));
            LCD_Write_DATA(ch, cl);
            ty = ty + delta;
        }
    }
    digitalWrite(LCD_CS, HIGH);
}
else
{
    //*P_CS &= ~B_CS;
    digitalWrite(LCD_CS, LOW);
    delta = ((float) (x2 - x1) / (float) (y2 - y1));
    tx = (float) x1;
    if (y1 > y2)
    {
        for (i = y2 + 1; i > y1; i--)
        {
            setXY((int) (tx + 0.5), i, (int) (tx + 0.5), i);
            LCD_Write_DATA(ch, cl);
            tx = tx + delta;
        }
    }
    else
    {
        for (i = y1; i < y2 + 1; i++)
```

```
            {
                setXY((int) (tx + 0.5), i, (int) (tx + 0.5), i);
                LCD_Write_DATA(ch, cl);
                tx = tx + delta;
            }
        }
        //*P_CS |= B_CS;
        digitalWrite(LCD_CS, HIGH);
    }

    if (orient == PORTRAIT)
        setXY(0, 0, 479, disp_y_size);
    else
        setXY(0, 0, disp_y_size, 479);
}

void drawHLine(int x, int y, int l)
{
    char ch, cl;
    int i;
    ch = ((fcolorr & 248) | fcolorg >> 5);
    cl = ((fcolorg & 28) << 3 | fcolorb >> 3);

    //digitalWrite(LCD_CS, LOW);
    digitalWrite(LCD_CS, LOW);
    setXY(x, y, x + l, y);
    for (i = 0; i < l + 1; i++)
    {
        LCD_Write_DATA(ch, cl);
    }
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);
}
```

```
void drawVLine(int x, int y, int l)
{
  char ch, cl;
  int i;

  ch = ((fcolorr & 248) | fcolorg >> 5);
  cl = ((fcolorg & 28) << 3 | fcolorb >> 3);

  //digitalWrite(LCD_CS, LOW);
  digitalWrite(LCD_CS, LOW);
  setXY(x, y, x, y + l);
  for (i = 0; i < l; i++)
  {
    LCD_Write_DATA(ch, cl);
  }
  //digitalWrite(LCD_CS, HIGH);
  digitalWrite(LCD_CS, HIGH);
}

void drawRoundRect(int x1, int y1, int x2, int y2)
{
  int tmp;

  if (x1 > x2)
  {
    tmp = x1;
    x1 = x2;
    x2 = tmp;
  }
  if (y1 > y2)
  {
    tmp = y1;
    y1 = y2;
    y2 = tmp;
```

```
    }
    if ((x2 - x1) > 4 && (y2 - y1) > 4)
    {
        drawPixel(x1 + 1, y1 + 1);
        drawPixel(x2 - 1, y1 + 1);
        drawPixel(x1 + 1, y2 - 1);
        drawPixel(x2 - 1, y2 - 1);
        drawHLine(x1 + 2, y1, x2 - x1 - 4);
        drawHLine(x1 + 2, y2, x2 - x1 - 4);
        drawVLine(x1, y1 + 2, y2 - y1 - 4);
        drawVLine(x2, y1 + 2, y2 - y1 - 4);
    }
}
void drawPixel(int x, int y)
{

    //digitalWrite(LCD_CS, LOW);
    digitalWrite(LCD_CS, LOW);
    setXY(x, y, x, y);
    setPixel(fcolorr, fcolorg, fcolorb);
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);

    if (orient == PORTRAIT)
        setXY(0, 0, 479, disp_y_size);
    else
        setXY(0, 0, disp_y_size, 479);
}
void fillRoundRect(int x1, int y1, int x2, int y2)
{
    int tmp, i;

    if (x1 > x2)
    {
```

```
        tmp = x1;
        x1 = x2;
        x2 = tmp;
    }
    if (y1 > y2)
    {
        tmp = y1;
        y1 = y2;
        y2 = tmp;
    }

    if ((x2 - x1) > 4 && (y2 - y1) > 4)
    {
        for (i = 0; i < ((y2 - y1) / 2) + 1; i++)
        {
            switch (i)
            {
            case 0:
                drawHLine(x1 + 2, y1 + i, x2 - x1 - 4);
                drawHLine(x1 + 2, y2 - i, x2 - x1 - 4);
                break;
            case 1:
                drawHLine(x1 + 1, y1 + i, x2 - x1 - 2);
                drawHLine(x1 + 1, y2 - i, x2 - x1 - 2);
                break;
            default:
                drawHLine(x1, y1 + i, x2 - x1);
                drawHLine(x1, y2 - i, x2 - x1);
            }
        }
    }
}
void drawCircle(int x, int y, int radius)
{
```

```
int f = 1 - radius;
int ddF_x = 1;
int ddF_y = -2 * radius;
int x1 = 0;
int y1 = radius;
char ch, cl;

ch = ((fcolorr & 248) | fcolorg >> 5);
cl = ((fcolorg & 28) << 3 | fcolorb >> 3);

//digitalWrite(LCD_CS, LOW);
digitalWrite(LCD_CS, LOW);
setXY(x, y + radius, x, y + radius);
LCD_Write_DATA(ch, cl);
setXY(x, y - radius, x, y - radius);
LCD_Write_DATA(ch, cl);
setXY(x + radius, y, x + radius, y);
LCD_Write_DATA(ch, cl);
setXY(x - radius, y, x - radius, y);
LCD_Write_DATA(ch, cl);

while (x1 < y1)
{
   if (f >= 0)
   {
      y1--;
      ddF_y += 2;
      f += ddF_y;
   }
   x1++;
   ddF_x += 2;
   f += ddF_x;
   setXY(x + x1, y + y1, x + x1, y + y1);
   LCD_Write_DATA(ch, cl);
```

```
        setXY(x - x1, y + y1, x - x1, y + y1);
        LCD_Write_DATA(ch, cl);
        setXY(x + x1, y - y1, x + x1, y - y1);
        LCD_Write_DATA(ch, cl);
        setXY(x - x1, y - y1, x - x1, y - y1);
        LCD_Write_DATA(ch, cl);
        setXY(x + y1, y + x1, x + y1, y + x1);
        LCD_Write_DATA(ch, cl);
        setXY(x - y1, y + x1, x - y1, y + x1);
        LCD_Write_DATA(ch, cl);
        setXY(x + y1, y - x1, x + y1, y - x1);
        LCD_Write_DATA(ch, cl);
        setXY(x - y1, y - x1, x - y1, y - x1);
        LCD_Write_DATA(ch, cl);
    }
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);
    if (orient == PORTRAIT)
        setXY(0, 0, 479, disp_y_size);
    else
        setXY(0, 0, disp_y_size, 479);
}

void fillCircle(int x, int y, int radius)
{
    //digitalWrite(LCD_CS, LOW);
    int y1, x1;
    digitalWrite(LCD_CS, LOW);
    for (y1 = -radius; y1 <= radius; y1++)
        for (x1 = -radius; x1 <= radius; x1++)
            if (x1 * x1 + y1 * y1 <= radius * radius)
            {
                setXY(x + x1, y + y1, x + x1, y + y1);
                LCD_Write_DATA(((fcolorr & 248) | fcolorg >> 5),
```

```
                    ((fcolorg & 28) << 3 | fcolorb >> 3));
            }
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);
    if (orient == PORTRAIT)
        setXY(0, 0, 479, disp_y_size);
    else
        setXY(0, 0, disp_y_size, 479);
}

void clrScr()
{
    long i;

    //digitalWrite(LCD_CS, LOW);
    digitalWrite(LCD_CS, LOW);
    if (orient == PORTRAIT)
        setXY(0, 0, 479, disp_y_size);
    else
        setXY(0, 0, disp_y_size, 479);
    //*P_RS |= B_RS;
    digitalWrite(LCD_RS, HIGH);

    for (i = 0; i < (480 * (disp_y_size + 1)); i++)
    {
        Lcd_Writ_Bus(0, 0);
    }
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);
}

void fillScr(uint8 r, uint8 g, uint8 b)
{
    long i;
```

```
    char ch, cl;

    ch = ((r & 248) | g >> 5);
    cl = ((g & 28) << 3 | b >> 3);

    //digitalWrite(LCD_CS, LOW);
    digitalWrite(LCD_CS, LOW);
    if (orient == PORTRAIT)
      setXY(0, 0, 479, disp_y_size);
    else
      setXY(0, 0, disp_y_size, 479);
    //*P_RS |= B_RS;
    digitalWrite(LCD_RS, HIGH);

    for (i = 0; i < (480 * (disp_y_size + 1)); i++)
    {
      Lcd_Writ_Bus(ch, cl);
    }
    //digitalWrite(LCD_CS, HIGH);
    digitalWrite(LCD_CS, HIGH);
}
void fillRect(int x1, int y1, int x2, int y2)
{
    int tmp, i;

    if (x1 > x2)
    {
      tmp = x1;
      x1 = x2;
      x2 = tmp;
    }
    if (y1 > y2)
    {
      tmp = y1;
```

```
      y1 = y2;
      y2 = tmp;
    }

    if (orient == PORTRAIT)
    {
      for (i = 0; i < ((y2 - y1) / 2) + 1; i++)
      {
        drawHLine(x1, y1 + i, x2 - x1);
        drawHLine(x1, y2 - i, x2 - x1);
      }
    }
    else
    {
      for (i = 0; i < ((x2 - x1) / 2) + 1; i++)
      {
        drawVLine(x1 + i, y1, y2 - y1);
        drawVLine(x2 - i, y1, y2 - y1);
      }
    }
}
/*
 void drawFlashBitmap(int x, int y, int sx, int sy, uint32 addrs_data, int scale)
 {
 unsigned int col, data_col;
 int tx, ty;
 _disable_IRQ();        //Disable to interruption to write the image
 digitalWrite(LCD_CS, LOW);
 for (ty = 0; ty < sy; ty++)
 {
 setXY(x, y + ty, x + sx - 1, y + ty);
 for (tx = sx; tx >= 0; tx--)
 {
```

```
 col = SPIFlash_Read_uint16((ty * sx) + tx + addrs_data);
 LCD_Write_DATA(col >> 8, col & 0xff);
 }
 }
 digitalWrite(LCD_CS, HIGH);
 _enable_IRQ();       //Enable to interruption to write the image
 }
 */
void drawBitmap(int x, int y, int sx, int sy, const unsigned short *data,
              int scale)
{
   unsigned int col;
   int tx, ty, tc, tsx, tsy;
   //uint8 r, g, b;

   if (scale == 1)
   {
      if (orient == PORTRAIT)
      {
         digitalWrite(LCD_CS, LOW);
         setXY(x, y, x + sx - 1, y + sy - 1);
         for (tc = 0; tc < (sx * sy); tc++)
         {
            col = pgm_read_word(&data[tc]);
            LCD_Write_DATA(col >> 8, col & 0xff);
         }
         digitalWrite(LCD_CS, HIGH);
      }
      else
      {
         digitalWrite(LCD_CS, LOW);
         for (ty = 0; ty < sy; ty++)
         {
            setXY(x, y + ty, x + sx - 1, y + ty);
```

```
            for (tx = sx; tx >= 0; tx--)
            {
               col = pgm_read_word(&data[(ty * sx) + tx]);
               LCD_Write_DATA(col >> 8, col & 0xff);
            }
         }
         digitalWrite(LCD_CS, HIGH);
      }
   }
   else
   {
      if (orient == PORTRAIT)
      {
         digitalWrite(LCD_CS, LOW);
         for (ty = 0; ty < sy; ty++)
         {
            setXY(x, y + (ty * scale), x + ((sx * scale) - 1),
                  y + (ty * scale) + scale);
            for (tsy = 0; tsy < scale; tsy++)
               for (tx = 0; tx < sx; tx++)
               {
                  col = pgm_read_word(&data[(ty * sx) + tx]);
                  for (tsx = 0; tsx < scale; tsx++)
                     LCD_Write_DATA(col >> 8, col & 0xff);
               }
         }
         digitalWrite(LCD_CS, HIGH);
      }
      else
      {
         digitalWrite(LCD_CS, LOW);
         for (ty = 0; ty < sy; ty++)
         {
            for (tsy = 0; tsy < scale; tsy++)
```

```
            {
                setXY(x, y + (ty * scale) + tsy, x + ((sx * scale) - 1),
                    y + (ty * scale) + tsy);
                for (tx = sx; tx >= 0; tx--)
                {
                    col = pgm_read_word(&data[(ty * sx) + tx]);
                    for (tsx = 0; tsx < scale; tsx++)
                        LCD_Write_DATA(col >> 8, col & 0xff);
                }
            }
        }
        digitalWrite(LCD_CS, HIGH);
    }
  }
}

void lcdSetPinDirection(lcdPin_t *pin, t_pinDir dir)
{
  uint32 uDir = pin->port->DIR;
  if (dir == pinWrite)
  {
    uDir |= (1U << pin->bit);
  }
  else
  {
    uDir &= ~(1U << pin->bit);
  }
  gioSetDirection(pin->port, uDir);
}

void spistar()                          //SPI Start
{
  digitalWrite(CS, HIGH);
  digitalWrite(DCLK, HIGH);
```

```
    digitalWrite(TOUCH_DIN, HIGH);
    digitalWrite(DCLK, HIGH);

}
//**********************************************************
void WriteCharTo7843(unsigned char num)        //SPI Write Data
{
    unsigned char count = 0;
    unsigned char temp;

    temp = num;
    digitalWrite(DCLK, LOW);
    for (count = 0; count < 8; count++)
    {
        if (temp & 0x80)
            digitalWrite(TOUCH_DIN, HIGH);
        else
            digitalWrite(TOUCH_DIN, LOW);

        temp = temp << 1;

        digitalWrite(DCLK, LOW);
        delay(0.5e-3);
        digitalWrite(DCLK, HIGH);
        delay(0.5e-3);
    }
}

//**********************************************************
unsigned int ReadFromCharFrom7843()          //SPI Read Data
{
    // unsigned nop;
    unsigned char count = 0;
    unsigned int Num = 0;
```

```
    for (count = 0; count < 12; count++)
    {
        Num <<= 1;
        digitalWrite(DCLK, HIGH); //DCLK=1; _nop_();_nop_();_nop_();
        delay(0.5e-3);
        digitalWrite(DCLK, LOW); //DCLK=0; _nop_();_nop_();_nop_();
        delay(0.5e-3);
        if (digitalRead(TOUCH_DOUT))
            Num++;
    }
    return (Num);
}

void read_axis_touch(panel_touch *panel1)
{
    int TP_X, TP_Y;

    digitalWrite(CS, LOW);

    WriteCharTo7843(0x90);
    digitalWrite(DCLK, HIGH);
    digitalWrite(DCLK, LOW);
    panel1->ejex = ReadFromCharFrom7843();

    WriteCharTo7843(0xD0);
    digitalWrite(DCLK, HIGH);
    digitalWrite(DCLK, LOW);
    panel1->ejey = ReadFromCharFrom7843();

    digitalWrite(CS, HIGH);

    /*   float x_temp = ((float) TP_X / 4095) * 799;
     float y_temp = ((float) TP_Y / 4095) * 479;
     TP_X = (int) x_temp;
```

```
    TP_Y = (int) y_temp;
    //TP_X = ((TP_X - 340) * 10 / 144);
    //TP_Y = ((TP_Y - 320) / 11);

    panel1->ejey = TP_Y;
    panel1->ejex = TP_X;*/

}
/*
 int Touch_Key()
 {

 char Mensaje[50] = "";
 panel_touch xy_coordenadas;
 uint32_t lx, ly;
 int touch_comd = 0;

 read_axis_touch(&xy_coordenadas);

 lx = xy_coordenadas.ejex;
 ly = xy_coordenadas.ejey;

 setColor(0xff, 0x00, 0x00);
 setBackColor(0xff, 0xff, 0xff);
 sprintf(Mensaje, "lx=%3d ly=%3d   ", lx, ly);
 TFT_print(Mensaje, 20, 200, 0);

 return touch_comd;

 }*/
void Pant(char VH, char VL)
{
   int i, j;
   digitalWrite(LCD_CS, LOW);
```

```
    Address_set(0, 0, 479, 799);
    for (i = 0; i < 320; i++)
    {
       for (j = 0; j < 480; j++)
       {
          Lcd_Write_Data(VH, VL);
       }

    }
    digitalWrite(LCD_CS, HIGH);
}

void delay(float32 time)
{

    uint32 k_delay;

    k_delay = (uint32) (-3.6932 + 8.33333e6 * time) * 176 / 100; //Ecuación para el
Retardo, calculo físico

    while (k_delay)
    {
       k_delay--;
    };
}

void URTouch_InitTouch(uint8 orientation)
{
    orient = orientation;
    _default_orientation = CAL_S >> 31;

    touch_x_left = (CAL_X >> 14) & 0x3FFF;
    touch_x_right = CAL_X & 0x3FFF;
    touch_y_top = (CAL_Y >> 14) & 0x3FFF;
```

```
    touch_y_bottom = CAL_Y & 0x3FFF;

    disp_x_size = 479;
    // disp_y_size = CAL_S & 0x0FFF;
    prec = 10;
}

void URTouch_read()
{
    panel_touch xy_coordenadas;
    unsigned long tx = 0, temp_x = 0;
    unsigned long ty = 0, temp_y = 0;
    unsigned long minx = 99999, maxx = 0;
    unsigned long miny = 99999, maxy = 0;
    int datacount = 0;

    //cbi(P_CS, B_CS);
    int i;
    //pinMode(T_IRQ,  INPUT);
    for (i = 0; i < prec; i++)
    {
        read_axis_touch(&xy_coordenadas);

        temp_x = xy_coordenadas.ejex;
        temp_y = xy_coordenadas.ejey;

        if ((temp_x > 0) && (temp_x < 4096) && (temp_y > 0) && (temp_y < 4096))
        {
            tx += temp_x;
            ty += temp_y;
            if (prec > 5)
            {
                if (temp_x < minx)
                    minx = temp_x;
```

```
            if (temp_x > maxx)
                maxx = temp_x;
            if (temp_y < miny)
                miny = temp_y;
            if (temp_y > maxy)
                maxy = temp_y;
        }
        datacount++;
    }

}
//  pinMode(T_IRQ,  OUTPUT);

  if (prec > 5)
  {
    tx = tx - (minx + maxx);
    ty = ty - (miny + maxy);
    datacount -= 2;
  }

//  sbi(P_CS, B_CS);
  if ((datacount == (prec - 2)) || (datacount == PREC_LOW))
  {
    if (orient == _default_orientation)
    {
      TP_X = ty / datacount;
      TP_Y = tx / datacount;
    }
    else
    {
      TP_X = tx / datacount;
      TP_Y = ty / datacount;
    }
  }
```

```
    else
    {
       TP_X = -1;
       TP_Y = -1;
    }
}

uint32 URTouch_dataAvailable()
{

    uint32 avail;
    avail = !gioGetBit(gioPORTA, 0);

    return avail;
}

int URTouch_getX()
{
    int c;

    if ((TP_X == -1) || (TP_Y == -1))
       return -1;
    /*    if (orient == _default_orientation)
     {
     c = ((TP_X - touch_x_left) * (disp_x_size))
     / (touch_x_right - touch_x_left);
     if (c < 0)
     c = 0;
     if (c > disp_x_size)
     c = disp_x_size;
     }
     else
     {
     if (_default_orientation == PORTRAIT)
```

```
    c = ((TP_X - touch_y_top) * (-disp_y_size))
    / (touch_y_bottom - touch_y_top) + (disp_y_size);
    else*/

   c = ((TP_X - touch_y_top) * (disp_y_size)) / (touch_y_bottom - touch_y_top);

   c = 799 - c;
   if (c < 0)
      c = 0;
   if (c > disp_y_size)
      c = disp_y_size;
   //}
   return c;
}

int URTouch_getY()
{
   int c;

   if ((TP_X == -1) || (TP_Y == -1))
      return -1;
   /*  if (orient == _default_orientation)
    {
    c = ((TP_Y - touch_y_top) * (disp_y_size))
    / (touch_y_bottom - touch_y_top);
    if (c < 0)
    c = 0;
    if (c > disp_y_size)
    c = disp_y_size;
    }
    else
    {
    if (_default_orientation == PORTRAIT)
    c = ((TP_Y - touch_x_left) * (disp_x_size))
```

```
 / (touch_x_right - touch_x_left);
 else*/

 float y_temp = ((float) TP_Y / 4095) * 480;
 c = (int) y_temp;

 /*  c = ((TP_Y - touch_x_left) * (-disp_x_size))
 / (touch_x_right - touch_x_left) + (disp_x_size);
 c = 479 - c;*/
 if (c < 0)
    c = 0;
 if (c > disp_x_size)
    c = disp_x_size;
 //}
 return c;
}

void URTouch_setPrecision(uint8 precision)
{
 switch (precision)
 {
 case PREC_LOW:
    prec = 1;       // DO NOT CHANGE!
    break;
 case PREC_MEDIUM:
    prec = 12;  // Iterations + 2
    break;
 case PREC_HI:
    prec = 27;  // Iterations + 2
    break;
 case PREC_EXTREME:
    prec = 102; // Iterations + 2
    break;
 default:
```

```
        prec = 12;  // Iterations + 2
        break;
    }
}

void URTouch_calibrateRead()
{
   panel_touch xy_coordenadas;
   /*unsigned long tx = 0;
    unsigned long ty = 0;

    cbi(P_CS, B_CS);    //clear bite

    touch_WriteData(0x90);
    pulse_high(P_CLK, B_CLK);
    tx = touch_ReadData();

    touch_WriteData(0xD0);
    pulse_high(P_CLK, B_CLK);
    ty = touch_ReadData();

    sbi(P_CS, B_CS);    // set bite
    */
   read_axis_touch(&xy_coordenadas);

   TP_X = xy_coordenadas.ejex;
   TP_Y = xy_coordenadas.ejey;

   /*    TP_X = ty;
    TP_Y = tx;*/
}

void UTFT_Buttons_Init() //UTFT *ptrUTFT, URTouch *ptrURTouch)
{
```

```
    UTFT_Buttons_deleteAllButtons();
    _color_text = VGA_WHITE;
    _color_text_inactive = VGA_GRAY;
    _color_background = VGA_BLUE;
    _color_border = VGA_GRAY;
    _color_hilite = VGA_RED;
    _font_text = NULL;
    _font_symbol = NULL;
}

int UTFT_Buttons_addButton(uint16 x, uint16 y, uint16 width, uint16 height,
                           char *label, uint16 flags, int btcnt)
{

    /*   while (((buttons[btcnt].flags && BUTTON_UNUSED) == 0)
     && (btcnt < MAX_BUTTONS))
     btcnt++;*/

    if (btcnt == MAX_BUTTONS)
        return -1;
    else
    {
        buttons[btcnt].pos_x = x;
        buttons[btcnt].pos_y = y;
        buttons[btcnt].width = width;
        buttons[btcnt].height = height;
        buttons[btcnt].flags = flags;
        buttons[btcnt].label = label;
        buttons[btcnt].data = NULL;
        return btcnt;
    }
}
```

```
int UTFT_Buttons_addButton_img(uint16 x, uint16 y, uint16 width, uint16 height,
bitmapdatatype data,
                        uint16 flags)
{
   int btcnt = 0;

   while (((buttons[btcnt].flags & BUTTON_UNUSED) == 0)
         && (btcnt < MAX_BUTTONS))
      btcnt++;

   if (btcnt == MAX_BUTTONS)
      return -1;
   else
   {
      buttons[btcnt].pos_x = x;
      buttons[btcnt].pos_y = y;
      buttons[btcnt].width = width;
      buttons[btcnt].height = height;
      buttons[btcnt].flags = flags | BUTTON_BITMAP;
      buttons[btcnt].label = NULL;
      buttons[btcnt].data = data;
      return btcnt;
   }
}

void UTFT_Buttons_drawButtons()
{
   int i;
   for (i = 0; i < MAX_BUTTONS; i++)
   {
      if ((buttons[i].flags & BUTTON_UNUSED) == 0)
         UTFT_Buttons_drawButton(i);
   }
}
```

```
void UTFT_Buttons_drawButton(int buttonID)
{
  int text_x, text_y;
  uint8_t *_font_current = getFont();
  uint32_t _current_color = getColor_word();
  uint32_t _current_back = getBackColor();

  if (buttons[buttonID].flags & BUTTON_BITMAP)
  {
    drawBitmap(buttons[buttonID].pos_x, buttons[buttonID].pos_y,
          buttons[buttonID].width, buttons[buttonID].height,
          buttons[buttonID].data, 1);
    if (!(buttons[buttonID].flags & BUTTON_NO_BORDER))
    {
      if ((buttons[buttonID].flags & BUTTON_DISABLED))
        setColor_word(_color_text_inactive);
      else
        setColor_word(_color_border);
      drawRect(buttons[buttonID].pos_x, buttons[buttonID].pos_y,
            buttons[buttonID].pos_x + buttons[buttonID].width,
            buttons[buttonID].pos_y + buttons[buttonID].height);
    }
  }
  else
  {
    setColor_word(_color_background);
    fillRoundRect(buttons[buttonID].pos_x, buttons[buttonID].pos_y,
            buttons[buttonID].pos_x + buttons[buttonID].width,
            buttons[buttonID].pos_y + buttons[buttonID].height);
    setColor_word(_color_border);
    drawRoundRect(buttons[buttonID].pos_x, buttons[buttonID].pos_y,
            buttons[buttonID].pos_x + buttons[buttonID].width,
            buttons[buttonID].pos_y + buttons[buttonID].height);
```

```
if (buttons[buttonID].flags & BUTTON_DISABLED)
   setColor_word(_color_text_inactive);
else
   setColor_word(_color_text);

if (buttons[buttonID].flags & BUTTON_SYMBOL)
{
   setFont(_font_symbol);
   text_x = (buttons[buttonID].width / 2) - (getFontXsize() / 2)
         + buttons[buttonID].pos_x;

   text_y = (buttons[buttonID].height / 2) - (getFontYsize() / 2);
}
else
{
   setFont(_font_text);
   text_x = ((buttons[buttonID].width / 2)
         - ((strlen(buttons[buttonID].label) * getFontXsize()) / 2))
         + buttons[buttonID].pos_x;
   text_y = (buttons[buttonID].height / 2) - (getFontYsize() / 2)
         + buttons[buttonID].pos_y;
}

setBackColor_word(_color_background);

TFT_print(buttons[buttonID].label, text_x, text_y, 0);

if ((buttons[buttonID].flags & BUTTON_SYMBOL)
      && (buttons[buttonID].flags & BUTTON_SYMBOL_REP_3X))
{
   TFT_print(buttons[buttonID].label, text_x - getFontXsize(), text_y,
         0);
   TFT_print(buttons[buttonID].label, text_x + getFontXsize(), text_y,
```

```
                0);
        }
    }
    setFont(_font_current);
    setColor_word(_current_color);
    setBackColor_word(_current_back);
}

void UTFT_Buttons_enableButton(int buttonID, bool redraw)
{
    if (!(buttons[buttonID].flags & BUTTON_UNUSED))
    {
        buttons[buttonID].flags = buttons[buttonID].flags & ~BUTTON_DISABLED;
        if (redraw)
            UTFT_Buttons_drawButton(buttonID);
    }
}

void UTFT_Buttons_disableButton(int buttonID, bool redraw)
{
    if (!(buttons[buttonID].flags & BUTTON_UNUSED))
    {
        buttons[buttonID].flags = buttons[buttonID].flags | BUTTON_DISABLED;
        if (redraw)
            UTFT_Buttons_drawButton(buttonID);
    }
}

void UTFT_Buttons_relabelButton(int buttonID, char *label, bool redraw)
{
    if (!(buttons[buttonID].flags & BUTTON_UNUSED))
    {
        buttons[buttonID].label = label;
        if (redraw)
```

```
            UTFT_Buttons_drawButton(buttonID);
    }
}

bool UTFT_Buttons_buttonEnabled(int buttonID)
{
    return !(buttons[buttonID].flags & BUTTON_DISABLED);
}

void UTFT_Buttons_deleteButton(int buttonID)
{
    if (!(buttons[buttonID].flags & BUTTON_UNUSED))
        buttons[buttonID].flags = BUTTON_UNUSED;
}

void UTFT_Buttons_deleteAllButtons()
{
    int i;
    for (i = 0; i < MAX_BUTTONS; i++)
    {
        buttons[i].pos_x = 0;
        buttons[i].pos_y = 0;
        buttons[i].width = 0;
        buttons[i].height = 0;
        buttons[i].flags = BUTTON_UNUSED;
        buttons[i].label = "";
    }
}

int UTFT_Buttons_checkButtons()
{
    /* char Mensaje1[40] = "";
     char Mensaje2[40] = "";*/
    int i;
```

```
if (URTouch_dataAvailable() == true)
{
   POWER_BACKLIGHT = 0;
  // Status_Backlight_eeprom();
   pwmSetDuty(hetRAM1, pwm3, 80);
   URTouch_read();

   Sound_Buzzer();
   int result = -1;
   int touch_x = URTouch_getX();
   int touch_y = URTouch_getY();
   uint32_t _current_color = getColor_word();

   /* sprintf(Mensaje1, "x = %d  ", touch_x);
    sprintf(Mensaje2, "y = %d   ", touch_y);

    setColor(0x00, 0x00, 0x00);
    setBackColor(0Xff, 0xff, 0xff);
    TFT_print(Mensaje1, 500, 10, 0);
    TFT_print(Mensaje2, 500, 30, 0);*/

   for (i = 0; i < MAX_BUTTONS; i++)
   {
      if (((buttons[i].flags & BUTTON_UNUSED) == 0)
            && ((buttons[i].flags & BUTTON_DISABLED) == 0)
            && (result == -1))
      {
         if ((touch_x >= buttons[i].pos_x)
               && (touch_x <= (buttons[i].pos_x + buttons[i].width))
               && (touch_y >= buttons[i].pos_y)
               && (touch_y <= (buttons[i].pos_y + buttons[i].height)))
         {
            result = i;
            break;
```

```
        }
    }
}
if (result != -1)
{
    if (!(buttons[result].flags & BUTTON_NO_BORDER))
    {
        setColor_word(_color_hilite);
        if (buttons[result].flags & BUTTON_BITMAP)
            drawRect(buttons[result].pos_x, buttons[result].pos_y,
                     buttons[result].pos_x + buttons[result].width,
                     buttons[result].pos_y + buttons[result].height);
        else
            drawRoundRect(
                    buttons[result].pos_x, buttons[result].pos_y,
                    buttons[result].pos_x + buttons[result].width,
                    buttons[result].pos_y + buttons[result].height);
    }
}

while (URTouch_dataAvailable() == true)
{
};
if (result != -1)
{
    if (!(buttons[result].flags & BUTTON_NO_BORDER))
    {
        setColor_word(_color_border);
        if (buttons[result].flags & BUTTON_BITMAP)
            drawRect(buttons[result].pos_x, buttons[result].pos_y,
                     buttons[result].pos_x + buttons[result].width,
                     buttons[result].pos_y + buttons[result].height);
        else
            drawRoundRect(
```

```
                    buttons[result].pos_x, buttons[result].pos_y,
                    buttons[result].pos_x + buttons[result].width,
                    buttons[result].pos_y + buttons[result].height);
            }
        }
        setColor_word(_current_color);
        return result;
    }
    else
        return -1;
}

void UTFT_Buttons_setTextFont(uint8_t *font)
{
    _font_text = font;
}

void UTFT_Buttons_setSymbolFont(uint8_t *font)
{
    _font_symbol = font;
}
void UTFT_Buttons_setButtonColors(uint32 atxt, uint32 iatxt, uint32 brd,
                                  uint32 brdhi, uint32 back)
{
    _color_text = atxt;
    _color_text_inactive = iatxt;
    _color_background = back;
    _color_border = brd;
    _color_hilite = brdhi;
}
void UTFT_Buttons_setButtonColor(uint32 back)
{
    _color_background = back;
}
```

ANEXO 9

Valores de frecuencia del motor monofasico 0.5HP

Frec(Hz)	Aceleracion 1(mm/s2)	Velocidad 1(mm/s)	Aceleracion 2(mm/s2)	Velocidad 2(mm/s)
0	0.019	0.086	0.018	0.081
19.5	0.019	0.042	0.018	0.04
39.1	0.021	0.031	0.025	0.038
58.6	0.029	0.032	0.052	0.058
78.1	0.021	0.019	0.028	0.024
97.7	0.019	0.014	0.017	0.013
117.2	0.019	0.012	0.018	0.011
136.7	0.019	0.011	0.018	0.01
156.2	0.019	0.009	0.017	0.009
175.8	0.021	0.009	0.019	0.008
195.3	0.02	0.008	0.019	0.008
214.8	0.024	0.009	0.022	0.008
234.4	0.036	0.012	0.033	0.011
253.9	0.032	0.01	0.034	0.011
273.4	0.039	0.012	0.043	0.013
293	0.07	0.019	0.091	0.025
312.5	0.069	0.018	0.09	0.024
332	0.172	0.043	0.148	0.037
351.5	0.716	0.167	0.622	0.146
371.1	0.724	0.161	0.561	0.125
390.6	0.28	0.059	0.238	0.05
410.1	0.354	0.072	0.389	0.079
429.7	0.382	0.074	0.392	0.076
449.2	0.237	0.044	0.333	0.062
468.7	0.365	0.065	0.517	0.092
488.3	0.394	0.067	0.505	0.086
507.8	0.302	0.05	0.468	0.077
527.3	0.34	0.054	0.53	0.084
546.8	0.349	0.053	0.517	0.079
566.4	0.445	0.066	0.571	0.085
585.9	0.411	0.059	0.579	0.083
605.4	0.39	0.054	0.561	0.078
625	0.373	0.05	0.603	0.081
644.5	0.443	0.058	0.803	0.105
664	0.471	0.06	0.995	0.126
683.5	0.455	0.056	1.547	0.191
703.1	0.486	0.058	1.889	0.227
722.6	0.553	0.065	1.415	0.165
742.1	0.702	0.08	2.785	0.317

761.7	0.534	0.059	1.612	0.179
781.2	0.512	0.055	0.609	0.066
800.7	0.553	0.058	0.677	0.072
820.3	0.573	0.059	0.685	0.071
839.8	0.585	0.059	0.667	0.067
859.3	0.518	0.051	0.544	0.054
878.8	0.539	0.052	0.653	0.063
898.4	0.527	0.05	0.665	0.063
917.9	0.586	0.054	0.608	0.056
937.4	0.598	0.054	0.58	0.053
957	0.616	0.055	0.577	0.051
976.5	0.627	0.055	0.559	0.049
996	0.605	0.052	0.665	0.057
1015.6	0.592	0.05	0.708	0.059
1035.1	0.537	0.044	0.597	0.049
1054.6	0.626	0.051	0.593	0.048
1074.2	0.699	0.055	0.643	0.051
1093.7	0.733	0.057	0.772	0.06
1113.2	0.788	0.06	1.039	0.08
1132.7	0.596	0.045	0.89	0.067
1152.3	0.747	0.055	1.088	0.081
1171.8	1.823	0.133	2.124	0.155
1191.3	1.716	0.123	1.923	0.138
1210.9	1.018	0.072	1.58	0.111
1230.4	1.37	0.095	2.496	0.173
1249.9	1.266	0.086	1.895	0.13
1269.4	0.98	0.066	1.274	0.086
1289	1.184	0.078	1.518	0.101
1308.5	1.107	0.072	1.245	0.081
1328	1.024	0.066	1.191	0.077
1347.6	1.265	0.08	1.588	0.101
1367.1	1.248	0.078	1.322	0.083
1386.6	1.626	0.1	1.553	0.096
1406.2	2.59	0.158	1.7	0.103
1425.7	2.761	0.166	1.413	0.085
1445.2	2.752	0.163	1.759	0.104
1464.8	3.83	0.224	2.939	0.172
1484.3	1.873	0.108	1.575	0.091
1503.8	1.446	0.082	1.052	0.06
1523.3	2.175	0.122	1.615	0.091
1542.9	1.298	0.072	1.034	0.057
1562.4	0.88	0.048	0.663	0.036
1581.9	1.074	0.058	0.776	0.042
1601.5	0.916	0.049	0.752	0.04
1621	0.884	0.047	0.962	0.051
1640.5	1.44	0.075	1.563	0.082
1660.1	1.706	0.088	2.075	0.107

1679.6	1.167	0.06	1.173	0.06
1699.1	1.729	0.087	1.74	0.088
1718.6	1.517	0.076	1.636	0.082
1738.2	0.987	0.049	1.209	0.06
1757.7	0.976	0.048	0.856	0.042
1777.2	0.971	0.047	0.856	0.041
1796.8	0.924	0.044	0.866	0.041
1816.3	1.225	0.058	1.264	0.06
1835.8	1.011	0.047	1.079	0.05
1855.3	0.931	0.043	1.063	0.049
1874.9	1.295	0.059	1.606	0.074
1894.4	2.013	0.091	2.351	0.107
1913.9	1.599	0.072	2.105	0.094
1933.5	1.918	0.085	3.079	0.137
1953	2.042	0.09	3.269	0.144
1972.5	2.179	0.095	3.036	0.132
1992.1	2.906	0.125	3.495	0.151
2011.6	2.4	0.102	2.204	0.094
2031.1	1.5	0.063	1.65	0.07
2050.6	1.495	0.063	1.51	0.063
2070.2	2.018	0.084	2.133	0.089
2089.7	1.623	0.067	1.762	0.072
2109.2	1.884	0.077	2.65	0.108
2128.8	3.017	0.122	3.949	0.159
2148.3	3.298	0.132	4.053	0.162
2167.8	2.903	0.115	2.922	0.116
2187.4	4.545	0.179	5.653	0.222
2206.9	3.191	0.124	4.426	0.172
2226.4	2.52	0.097	3.562	0.138
2245.9	3.516	0.135	4.758	0.182
2265.5	3.384	0.128	4.484	0.17
2285	2.953	0.111	2.981	0.112
2304.5	2.304	0.086	2.461	0.092
2324.1	2.442	0.09	2.222	0.082
2343.6	2.805	0.103	1.927	0.071
2363.1	3.613	0.132	2.184	0.079
2382.7	4.238	0.153	2.077	0.075
2402.2	3.285	0.118	1.785	0.064
2421.7	4.437	0.158	2.23	0.079
2441.3	4.701	0.166	2.249	0.079
2460.8	4.39	0.154	2.462	0.086
2480.3	3.904	0.135	1.812	0.063
2499.8	4.34	0.149	1.866	0.064
2519.4	4.062	0.139	1.805	0.062
2538.9	5.588	0.189	2.714	0.092
2558.4	6.816	0.229	4.015	0.135
2578	5.368	0.179	3.611	0.121
2597.5	3.489	0.116	3.017	0.1

2617	4.85	0.16	4	0.132
2636.6	3.167	0.103	3.114	0.102
2656.1	2.497	0.081	2.74	0.089
2675.6	2.87	0.092	3.167	0.102
2695.1	2.924	0.093	2.77	0.089
2714.7	2.513	0.08	1.654	0.052
2734.2	2.982	0.094	1.975	0.062
2753.7	2.377	0.074	2.222	0.069
2773.3	2.053	0.064	1.755	0.055
2792.8	2.261	0.07	1.693	0.052
2812.3	1.828	0.056	1.697	0.052
2831.9	1.647	0.05	1.57	0.048
2851.4	3.048	0.092	2.276	0.069
2870.9	2.839	0.085	2.228	0.067
2890.4	1.765	0.053	2.477	0.074
2910	1.977	0.059	2.529	0.075
2929.5	2.029	0.06	1.992	0.059
2949	1.46	0.043	1.5	0.044
2968.6	1.461	0.042	1.576	0.046
2988.1	1.356	0.039	1.73	0.05
3007.6	1.03	0.03	1.703	0.049
3027.1	1.501	0.043	1.802	0.051
3046.7	1.659	0.047	1.668	0.047
3066.2	1.248	0.035	1.94	0.055
3085.7	2.005	0.056	2.359	0.066
3105.3	2.267	0.063	2.082	0.058
3124.8	1.728	0.048	1.561	0.043
3144.3	2.153	0.059	1.965	0.054
3163.9	2.295	0.063	2.702	0.074
3183.4	1.984	0.054	2.309	0.063
3202.9	2.585	0.07	2.348	0.063
3222.4	2.822	0.076	2.786	0.075
3242	3.016	0.08	3.495	0.093
3261.5	4.305	0.114	4.011	0.106
3281	5.254	0.138	4.6	0.121
3300.6	4.216	0.11	3.884	0.101
3320.1	4.692	0.122	3.965	0.103
3339.6	5.021	0.13	3.765	0.097
3359.2	3.408	0.088	3.202	0.082
3378.7	3.728	0.095	3.531	0.09
3398.2	4.305	0.109	3.886	0.099
3417.8	3.478	0.088	3.532	0.089
3437.3	3.269	0.082	3.912	0.098
3456.8	3.56	0.089	3.401	0.085
3476.3	4.285	0.106	4.771	0.118
3495.9	4.836	0.119	4.762	0.118
3515.4	4.135	0.101	3.192	0.078
3534.9	3.511	0.086	2.729	0.067

3554.5	5.654	0.137	4.042	0.098
3574	5.603	0.135	4.318	0.104
3593.5	5.289	0.127	4.321	0.104
3613.1	6.842	0.163	4.288	0.102
3632.6	8.7	0.207	5.629	0.134
3652.1	10.314	0.244	8.646	0.204
3671.6	8.515	0.2	9.155	0.215
3691.2	5.996	0.14	5.654	0.132
3710.7	4.156	0.097	3.874	0.09
3730.2	3.853	0.089	3.496	0.081
3749.8	4.447	0.102	3.065	0.071
3769.3	4.149	0.095	3.367	0.077
3788.8	5.145	0.117	4.765	0.109
3808.4	4.965	0.113	5.055	0.115
3827.9	3.525	0.079	3.096	0.07
3847.4	3.237	0.073	2.698	0.061
3866.9	3.095	0.069	2.49	0.056
3886.5	3.206	0.071	2.476	0.055
3906	3.313	0.073	2.633	0.058
3925.5	3.302	0.073	2.226	0.049
3945.1	3.602	0.079	2.085	0.046
3964.6	3.315	0.072	2.338	0.051
3984.1	2.811	0.061	2.099	0.045
4003.6	2.36	0.051	3.094	0.067
4023.2	2.815	0.06	2.903	0.062
4042.7	3.286	0.07	1.85	0.04
4062.2	2.216	0.047	1.458	0.031
4081.8	2.016	0.043	1.722	0.036
4101.3	2.608	0.055	1.963	0.041
4120.8	3.271	0.069	2.835	0.059
4140.4	3.042	0.063	3.567	0.074
4159.9	2.627	0.055	2.332	0.048
4179.4	2.533	0.052	3.097	0.064
4199	2.878	0.059	3.537	0.073
4218.5	2.978	0.061	3.189	0.065
4238	2.85	0.058	2.919	0.059
4257.5	3.056	0.062	3.073	0.062
4277.1	4.04	0.082	2.558	0.052
4296.6	3.754	0.075	2.793	0.056
4316.1	3.712	0.074	3.348	0.067
4335.7	3.061	0.061	2.82	0.056
4355.2	3.58	0.071	3.77	0.075
4374.7	5.637	0.111	5.733	0.113
4394.3	4.141	0.081	4.778	0.094
4413.8	3.236	0.063	3.441	0.067
4433.3	3.599	0.07	3.743	0.073
4452.8	4.043	0.078	3.389	0.066
4472.4	4.435	0.086	4.212	0.081

4491.9	5.065	0.097	4.424	0.085
4511.4	5.596	0.107	4.178	0.08
4531	5.814	0.111	5.171	0.099
4550.5	6.109	0.116	5.181	0.098
4570	6.293	0.119	4.528	0.086
4589.5	5.291	0.1	4.255	0.08
4609.1	4.236	0.079	3.818	0.072
4628.6	4.305	0.08	4.389	0.082
4648.1	6.132	0.114	4.528	0.084
4667.7	4.918	0.091	4.168	0.077
4687.2	4.815	0.089	3.141	0.058
4706.7	5.133	0.094	3.341	0.061
4726.3	4.334	0.079	3.587	0.066
4745.8	3.536	0.064	3.516	0.064
4765.3	3.733	0.068	3.589	0.065
4784.9	3.188	0.058	2.824	0.051
4804.4	2.069	0.037	2.382	0.043
4823.9	2	0.036	2.262	0.041
4843.4	1.774	0.032	2.648	0.047
4863	1.651	0.029	2.39	0.042
4882.5	1.703	0.03	2.13	0.038
4902	2.176	0.038	2.693	0.047
4921.6	1.825	0.032	2.696	0.047
4941.1	1.905	0.033	2.724	0.048
4960.6	1.801	0.031	2.697	0.047
4980.1	1.576	0.027	2.923	0.051
4999.7	1.398	0.024	2.91	0.05
5019.2	1.779	0.031	3.454	0.059
5038.7	1.474	0.025	3.534	0.061
5058.3	1.392	0.024	2.464	0.042
5077.8	1.699	0.029	2.626	0.045
5097.3	1.145	0.019	2.095	0.036
5116.9	1.033	0.017	1.965	0.033
5136.4	1.26	0.021	2.153	0.036
5155.9	1.3	0.022	2.274	0.038
5175.5	1.212	0.02	1.859	0.031
5195	1.351	0.022	1.747	0.029
5214.5	1.276	0.021	1.602	0.027
5234	1.433	0.024	1.685	0.028
5253.6	1.729	0.028	1.892	0.031
5273.1	1.844	0.03	2.207	0.036
5292.6	1.767	0.029	2.419	0.04
5312.2	2.378	0.039	2.572	0.042
5331.7	1.771	0.029	2.251	0.036
5351.2	1.524	0.025	1.998	0.032
5370.8	1.987	0.032	2.413	0.039
5390.3	1.656	0.027	2.182	0.035
5409.8	1.687	0.027	2.217	0.035

5429.3	2.079	0.033	2.387	0.038
5448.9	2.108	0.033	2.075	0.033
5468.4	1.902	0.03	1.972	0.031
5487.9	2.51	0.04	3.336	0.053
5507.5	2.584	0.041	3.268	0.051
5527	2.882	0.045	3.151	0.049
5546.5	2.416	0.038	2.617	0.041
5566	2.47	0.038	2.888	0.045
5585.6	1.768	0.027	2.46	0.038
5605.1	1.813	0.028	1.923	0.03
5624.6	1.27	0.02	1.333	0.02
5644.2	1.189	0.018	1.308	0.02
5663.7	1.023	0.016	1.002	0.015
5683.2	0.702	0.011	0.593	0.009
5702.8	0.417	0.006	0.313	0.005
5722.3	0.267	0.004	0.201	0.003
5741.8	0.25	0.004	0.133	0.002
5761.4	0.159	0.002	0.08	0.001
5780.9	0.06	0.001	0.053	0.001
5800.4	0.05	0.001	0.038	0.001
5819.9	0.027	0	0.027	0
5839.5	0.022	0	0.024	0
5859	0.021	0	0.023	0
5878.5	0.02	0	0.023	0
5898.1	0.02	0	0.022	0
5917.6	0.02	0	0.022	0
5937.1	0.02	0	0.021	0
5956.6	0.019	0	0.021	0
5976.2	0.018	0	0.02	0
5995.7	0.018	0	0.02	0
6015.2	0.018	0	0.019	0
6034.8	0.019	0	0.02	0
6054.3	0.017	0	0.019	0
6073.8	0.018	0	0.019	0
6093.4	0.017	0	0.018	0
6112.9	0.017	0	0.018	0
6132.4	0.016	0	0.018	0
6152	0.016	0	0.017	0
6171.5	0.016	0	0.017	0
6191	0.016	0	0.017	0
6210.5	0.015	0	0.017	0
6230.1	0.016	0	0.017	0
6249.6	0.016	0	0.016	0
6269.1	0.015	0	0.016	0
6288.7	0.015	0	0.016	0
6308.2	0.015	0	0.016	0
6327.7	0.015	0	0.015	0
6347.3	0.015	0	0.015	0

6366.8	0.015	0	0.015	0
6386.3	0.014	0	0.015	0
6405.8	0.014	0	0.015	0
6425.4	0.014	0	0.015	0
6444.9	0.014	0	0.015	0
6464.4	0.014	0	0.014	0
6484	0.014	0	0.014	0
6503.5	0.014	0	0.014	0
6523	0.014	0	0.014	0
6542.5	0.014	0	0.014	0
6562.1	0.014	0	0.014	0
6581.6	0.013	0	0.014	0
6601.1	0.013	0	0.014	0
6620.7	0.013	0	0.014	0
6640.2	0.013	0	0.013	0
6659.7	0.013	0	0.014	0
6679.3	0.013	0	0.013	0
6698.8	0.013	0	0.014	0
6718.3	0.013	0	0.013	0
6737.9	0.013	0	0.013	0
6757.4	0.013	0	0.013	0
6776.9	0.013	0	0.013	0
6796.4	0.013	0	0.013	0
6816	0.013	0	0.013	0
6835.5	0.012	0	0.013	0
6855	0.012	0	0.013	0
6874.6	0.012	0	0.013	0
6894.1	0.012	0	0.013	0
6913.6	0.012	0	0.013	0
6933.1	0.012	0	0.013	0
6952.7	0.012	0	0.012	0
6972.2	0.012	0	0.012	0
6991.7	0.012	0	0.012	0
7011.3	0.012	0	0.012	0
7030.8	0.012	0	0.012	0
7050.3	0.012	0	0.012	0
7069.9	0.012	0	0.012	0
7089.4	0.012	0	0.012	0
7108.9	0.012	0	0.012	0
7128.5	0.012	0	0.012	0
7148	0.012	0	0.012	0
7167.5	0.012	0	0.012	0
7187	0.012	0	0.012	0
7206.6	0.011	0	0.012	0
7226.1	0.011	0	0.012	0
7245.6	0.011	0	0.012	0
7265.2	0.011	0	0.012	0
7284.7	0.011	0	0.012	0

7304.2	0.011	0	0.012	0
7323.8	0.011	0	0.011	0
7343.3	0.011	0	0.012	0
7362.8	0.011	0	0.012	0
7382.3	0.011	0	0.012	0
7401.9	0.011	0	0.011	0
7421.4	0.011	0	0.012	0
7440.9	0.011	0	0.011	0
7460.5	0.011	0	0.011	0
7480	0.011	0	0.011	0
7499.5	0.011	0	0.011	0
7519	0.011	0	0.011	0
7538.6	0.011	0	0.011	0
7558.1	0.011	0	0.011	0
7577.6	0.011	0	0.011	0
7597.2	0.011	0	0.011	0
7616.7	0.011	0	0.011	0
7636.2	0.011	0	0.011	0
7655.8	0.011	0	0.011	0
7675.3	0.011	0	0.011	0
7694.8	0.011	0	0.011	0
7714.4	0.011	0	0.011	0
7733.9	0.011	0	0.011	0
7753.4	0.011	0	0.011	0
7772.9	0.011	0	0.011	0
7792.5	0.011	0	0.011	0
7812	0.011	0	0.011	0
7831.5	0.011	0	0.011	0
7851.1	0.011	0	0.011	0
7870.6	0.011	0	0.011	0
7890.1	0.011	0	0.011	0
7909.6	0.011	0	0.011	0
7929.2	0.011	0	0.011	0
7948.7	0.011	0	0.011	0
7968.2	0.011	0	0.011	0
7987.8	0.011	0	0.011	0
8007.3	0.011	0	0.011	0
8026.8	0.011	0	0.011	0
8046.4	0.011	0	0.011	0
8065.9	0.011	0	0.011	0
8085.4	0.011	0	0.011	0
8105	0.011	0	0.011	0
8124.5	0.011	0	0.011	0
8144	0.011	0	0.011	0
8163.5	0.011	0	0.011	0
8183.1	0.011	0	0.011	0
8202.6	0.011	0	0.011	0
8222.1	0.011	0	0.011	0

8241.7	0.011	0	0.011	0
8261.2	0.011	0	0.011	0
8280.7	0.011	0	0.011	0
8300.3	0.011	0	0.011	0
8319.8	0.011	0	0.011	0
8339.3	0.011	0	0.011	0
8358.8	0.011	0	0.011	0
8378.4	0.011	0	0.011	0
8397.9	0.011	0	0.011	0
8417.4	0.011	0	0.011	0
8437	0.011	0	0.011	0
8456.5	0.011	0	0.011	0
8476	0.011	0	0.011	0
8495.5	0.011	0	0.011	0
8515.1	0.011	0	0.011	0
8534.6	0.011	0	0.011	0
8554.1	0.011	0	0.011	0
8573.7	0.011	0	0.011	0
8593.2	0.011	0	0.011	0
8612.7	0.011	0	0.011	0
8632.3	0.011	0	0.011	0
8651.8	0.011	0	0.011	0
8671.3	0.011	0	0.011	0
8690.8	0.011	0	0.011	0
8710.4	0.011	0	0.011	0
8729.9	0.011	0	0.011	0
8749.4	0.011	0	0.011	0
8769	0.011	0	0.011	0
8788.5	0.011	0	0.011	0
8808	0.011	0	0.011	0
8827.6	0.011	0	0.011	0
8847.1	0.011	0	0.011	0
8866.6	0.011	0	0.011	0
8886.2	0.011	0	0.011	0
8905.7	0.011	0	0.011	0
8925.2	0.011	0	0.011	0
8944.7	0.011	0	0.011	0
8964.3	0.011	0	0.011	0
8983.8	0.011	0	0.011	0
9003.3	0.011	0	0.011	0
9022.9	0.011	0	0.011	0
9042.4	0.011	0	0.011	0
9061.9	0.011	0	0.011	0
9081.5	0.011	0	0.011	0
9101	0.011	0	0.011	0
9120.5	0.011	0	0.011	0
9140	0.011	0	0.011	0
9159.6	0.011	0	0.011	0

9179.1	0.011	0	0.011	0
9198.6	0.011	0	0.011	0
9218.2	0.011	0	0.011	0
9237.7	0.011	0	0.011	0
9257.2	0.011	0	0.011	0
9276.8	0.011	0	0.011	0
9296.3	0.011	0	0.011	0
9315.8	0.011	0	0.011	0
9335.3	0.011	0	0.011	0
9354.9	0.011	0	0.011	0
9374.4	0.011	0	0.011	0
9393.9	0.011	0	0.011	0
9413.5	0.011	0	0.011	0
9433	0.011	0	0.011	0
9452.5	0.011	0	0.011	0
9472	0.011	0	0.011	0
9491.6	0.011	0	0.011	0
9511.1	0.011	0	0.011	0
9530.6	0.011	0	0.011	0
9550.2	0.011	0	0.011	0
9569.7	0.011	0	0.011	0
9589.2	0.011	0	0.011	0
9608.8	0.011	0	0.011	0
9628.3	0.011	0	0.011	0
9647.8	0.011	0	0.011	0
9667.3	0.011	0	0.011	0
9686.9	0.011	0	0.011	0
9706.4	0.011	0	0.011	0
9725.9	0.011	0	0.011	0
9745.5	0.011	0	0.011	0
9765	0.011	0	0.011	0
9784.5	0.011	0	0.011	0
9804.1	0.011	0	0.011	0
9823.6	0.011	0	0.011	0
9843.1	0.011	0	0.011	0
9862.7	0.011	0	0.011	0
9882.2	0.011	0	0.011	0
9901.7	0.011	0	0.012	0
9921.2	0.011	0	0.011	0
9940.8	0.011	0	0.012	0
9960.3	0.011	0	0.012	0
9979.8	0.011	0	0.012	0

Fecha 14/02/2021
Hora 03:07:00

Datos del Motor: MJ1
RPM: 3500
Potencia(HP): 0.5

Tipo de Motor:	AC
Conectado a VFD:	No
Tipo de Montaje:	Horizontal
Tipo de Rodamiento:	Cojinete
Motor Desacoplado:	Si

ANEXO 10

Valores de frecuencia de compresora de aire de aire 2HP

Frec(Hz)	Aceleracion 1(mm/s2)	Velocidad 1(mm/s)	Aceleracion 2(mm/s2)	Velocidad 2(mm/s)
0	0.015	0.067	0.01	0.045
19.5	0.015	0.033	0.01	0.022
39.1	0.015	0.023	0.01	0.014
58.6	0.023	0.026	0.01	0.011
78.1	0.025	0.022	0.014	0.012
97.7	0.025	0.019	0.015	0.011
117.2	0.033	0.021	0.017	0.011
136.7	0.021	0.012	0.013	0.007
156.2	0.015	0.008	0.011	0.006
175.8	0.016	0.007	0.033	0.015
195.3	0.018	0.007	0.054	0.022
214.8	0.022	0.008	0.037	0.014
234.4	0.034	0.012	0.023	0.008
253.9	0.037	0.012	0.033	0.01
273.4	0.032	0.01	0.071	0.021
293	0.051	0.014	0.109	0.03
312.5	0.101	0.027	0.155	0.04
332	0.132	0.033	0.161	0.04
351.5	0.281	0.066	0.2	0.047
371.1	0.636	0.141	0.219	0.049
390.6	1.256	0.266	0.361	0.076
410.1	1.058	0.214	0.306	0.062
429.7	0.623	0.12	0.231	0.045
449.2	0.368	0.068	0.194	0.036
468.7	0.432	0.077	0.217	0.039
488.3	0.444	0.076	0.275	0.047
507.8	0.413	0.068	0.473	0.078
527.3	0.541	0.086	0.696	0.11
546.8	0.759	0.116	0.995	0.152
566.4	0.802	0.119	1.582	0.234
585.9	2.193	0.314	1.591	0.228
605.4	3.275	0.455	1.561	0.217
625	2.197	0.296	1.229	0.165
644.5	2.606	0.34	1.524	0.199
664	3.397	0.431	1.644	0.209
683.5	1.91	0.236	1.089	0.134
703.1	1.717	0.206	1.488	0.179
722.6	2.264	0.265	1.805	0.211
742.1	1.374	0.157	1.023	0.116
761.7	1.272	0.141	1.194	0.133

781.2	1.499	0.162	1.993	0.216
800.7	2.044	0.216	3.881	0.41
820.3	2.669	0.276	5.152	0.532
839.8	3.074	0.31	3.661	0.37
859.3	1.693	0.167	2.3	0.227
878.8	1.364	0.132	1.293	0.125
898.4	1.163	0.11	0.674	0.064
917.9	0.871	0.081	0.426	0.039
937.4	1.017	0.092	0.617	0.056
957	0.781	0.069	0.688	0.061
976.5	0.494	0.043	0.603	0.053
996	0.476	0.041	0.619	0.053
1015.6	0.477	0.04	0.739	0.062
1035.1	1.055	0.087	0.722	0.059
1054.6	2.503	0.202	0.766	0.062
1074.2	1.346	0.107	0.486	0.039
1093.7	1.147	0.089	0.527	0.041
1113.2	1.097	0.084	0.625	0.048
1132.7	0.586	0.044	0.451	0.034
1152.3	0.457	0.034	0.275	0.02
1171.8	0.734	0.053	0.351	0.026
1191.3	0.488	0.035	0.388	0.028
1210.9	0.463	0.033	0.457	0.032
1230.4	0.481	0.033	0.497	0.034
1249.9	0.465	0.032	0.541	0.037
1269.4	0.658	0.044	0.781	0.053
1289	1.477	0.098	1.022	0.068
1308.5	1.352	0.088	1.063	0.069
1328	1.138	0.073	1.817	0.117
1347.6	2.056	0.13	2.889	0.183
1367.1	2.431	0.152	2.661	0.166
1386.6	2.342	0.144	1.811	0.112
1406.2	1.911	0.116	1.553	0.095
1425.7	2.105	0.126	1.406	0.084
1445.2	1.744	0.103	1.414	0.084
1464.8	1.34	0.078	1.169	0.068
1484.3	1.463	0.084	0.934	0.054
1503.8	1.333	0.076	0.662	0.038
1523.3	1.685	0.095	0.79	0.044
1542.9	1.929	0.107	0.768	0.043
1562.4	1.307	0.072	0.579	0.032
1581.9	0.996	0.054	0.778	0.042
1601.5	0.946	0.051	0.957	0.051
1621	0.705	0.037	0.645	0.034
1640.5	0.659	0.034	0.573	0.03
1660.1	1.048	0.054	0.866	0.045

1679.6	1.323	0.068	1.107	0.056
1699.1	1.242	0.063	0.55	0.028
1718.6	1.493	0.075	0.695	0.035
1738.2	1.464	0.072	0.896	0.044
1757.7	1.242	0.061	0.556	0.027
1777.2	1.483	0.072	0.717	0.035
1796.8	4.014	0.192	1.719	0.082
1816.3	3.651	0.173	1.874	0.089
1835.8	1.977	0.092	1.902	0.089
1855.3	1.442	0.067	1.905	0.088
1874.9	1.836	0.084	1.413	0.065
1894.4	3.385	0.153	2.43	0.11
1913.9	2.422	0.109	3.059	0.137
1933.5	1.019	0.045	2.42	0.107
1953	0.698	0.031	1.752	0.077
1972.5	0.583	0.025	1.309	0.057
1992.1	0.664	0.029	1.402	0.06
2011.6	0.942	0.04	1.673	0.071
2031.1	0.875	0.037	1.718	0.073
2050.6	0.757	0.032	1.454	0.061
2070.2	0.791	0.033	1.441	0.06
2089.7	0.659	0.027	1.477	0.061
2109.2	0.946	0.039	1.338	0.055
2128.8	1.189	0.048	1.879	0.076
2148.3	1.103	0.044	1.758	0.07
2167.8	1.102	0.044	2.119	0.084
2187.4	0.886	0.035	2.366	0.093
2206.9	0.967	0.038	1.728	0.067
2226.4	1.023	0.039	1.556	0.06
2245.9	1.301	0.05	1.499	0.057
2265.5	0.926	0.035	1.301	0.049
2285	0.789	0.03	1.245	0.047
2304.5	0.757	0.028	1.028	0.038
2324.1	0.928	0.034	1.132	0.042
2343.6	0.987	0.036	1.065	0.039
2363.1	0.8	0.029	1.05	0.038
2382.7	0.729	0.026	1.142	0.041
2402.2	1.191	0.043	1.202	0.043
2421.7	0.976	0.035	1.23	0.044
2441.3	0.747	0.026	1.594	0.056
2460.8	0.662	0.023	1.656	0.058
2480.3	0.663	0.023	1.361	0.047
2499.8	0.651	0.022	1.217	0.042
2519.4	1.054	0.036	1.359	0.046
2538.9	1.357	0.046	1.661	0.056
2558.4	1.303	0.044	1.515	0.051

2578	1.325	0.044	1.595	0.053
2597.5	0.977	0.032	1.729	0.057
2617	0.715	0.024	1.76	0.058
2636.6	0.873	0.029	2.163	0.071
2656.1	0.927	0.03	2.488	0.081
2675.6	1.093	0.035	2.345	0.075
2695.1	1.398	0.045	2.638	0.084
2714.7	1.535	0.049	2.825	0.09
2734.2	1.501	0.047	2.439	0.077
2753.7	1.436	0.045	2.646	0.083
2773.3	1.422	0.044	2.601	0.081
2792.8	1.354	0.042	2.613	0.081
2812.3	1.527	0.047	2.542	0.078
2831.9	1.295	0.039	2.192	0.067
2851.4	0.886	0.027	1.967	0.059
2870.9	0.792	0.024	1.752	0.053
2890.4	1.157	0.034	1.694	0.05
2910	1.495	0.044	1.873	0.055
2929.5	1.281	0.038	1.655	0.049
2949	1.093	0.032	1.691	0.049
2968.6	1.073	0.031	1.572	0.046
2988.1	1.07	0.031	1.319	0.038
3007.6	1.144	0.033	1.343	0.038
3027.1	1.01	0.029	1.239	0.035
3046.7	1.158	0.033	1.168	0.033
3066.2	1.39	0.039	1.226	0.034
3085.7	2.284	0.064	1.44	0.04
3105.3	4.099	0.114	2.039	0.057
3124.8	3.083	0.085	1.903	0.052
3144.3	2.701	0.074	1.804	0.049
3163.9	2.887	0.079	1.572	0.043
3183.4	3.035	0.082	1.863	0.05
3202.9	4.906	0.132	2.482	0.067
3222.4	7.795	0.209	3.264	0.087
3242	5.324	0.142	2.655	0.071
3261.5	3.858	0.102	1.815	0.048
3281	2.696	0.071	1.32	0.035
3300.6	2.212	0.058	1.111	0.029
3320.1	1.899	0.049	1.025	0.027
3339.6	2.05	0.053	1.166	0.03
3359.2	3.252	0.083	1.496	0.038
3378.7	4.932	0.126	1.564	0.04
3398.2	3.915	0.099	1.694	0.043
3417.8	2.936	0.074	1.619	0.041
3437.3	2.157	0.054	1.259	0.032
3456.8	1.858	0.046	1.016	0.025

3476.3	1.758	0.044	0.936	0.023
3495.9	1.795	0.044	0.938	0.023
3515.4	2.103	0.052	0.847	0.021
3534.9	2.124	0.052	0.775	0.019
3554.5	2.503	0.061	0.824	0.02
3574	3.641	0.088	0.979	0.024
3593.5	5.227	0.126	1.184	0.028
3613.1	4.767	0.114	1.106	0.026
3632.6	3.484	0.083	1.07	0.025
3652.1	3.272	0.077	0.895	0.021
3671.6	2.259	0.053	0.661	0.016
3691.2	2.135	0.05	0.668	0.016
3710.7	1.991	0.046	0.659	0.015
3730.2	1.602	0.037	0.629	0.015
3749.8	1.801	0.041	0.757	0.017
3769.3	2.625	0.06	0.84	0.019
3788.8	3.846	0.088	0.913	0.021
3808.4	4.797	0.109	1.01	0.023
3827.9	4.125	0.093	1.107	0.025
3847.4	3.51	0.079	1.177	0.026
3866.9	2.939	0.066	1.181	0.026
3886.5	2.404	0.053	1.029	0.023
3906	1.668	0.037	0.762	0.017
3925.5	1.445	0.032	0.81	0.018
3945.1	1.345	0.029	1.036	0.023
3964.6	1.687	0.037	1.171	0.025
3984.1	1.964	0.043	1.567	0.034
4003.6	2.116	0.046	1.663	0.036
4023.2	1.952	0.042	1.48	0.032
4042.7	1.834	0.039	1.302	0.028
4062.2	2.222	0.047	1.148	0.024
4081.8	2.397	0.051	1.151	0.024
4101.3	2.524	0.053	1.131	0.024
4120.8	2.874	0.06	1.174	0.025
4140.4	3.076	0.064	1.102	0.023
4159.9	2.955	0.061	0.992	0.021
4179.4	2.164	0.045	0.957	0.02
4199	1.524	0.031	0.941	0.019
4218.5	1.611	0.033	1.301	0.027
4238	1.933	0.039	2.089	0.043
4257.5	1.997	0.041	2.408	0.049
4277.1	1.865	0.038	2.464	0.05
4296.6	1.413	0.028	1.836	0.037
4316.1	1.088	0.022	1.255	0.025
4335.7	1.217	0.024	1.289	0.026
4355.2	1.463	0.029	1.725	0.034

4374.7	1.515	0.03	1.612	0.032
4394.3	1.333	0.026	1.693	0.033
4413.8	1.175	0.023	1.484	0.029
4433.3	1.165	0.023	1.212	0.024
4452.8	1.329	0.026	1.248	0.024
4472.4	1.539	0.03	1.317	0.025
4491.9	1.656	0.032	1.278	0.025
4511.4	2.02	0.039	1.374	0.026
4531	2.255	0.043	1.169	0.022
4550.5	2.617	0.05	1.026	0.019
4570	3.574	0.068	1.095	0.021
4589.5	3.23	0.061	1.236	0.023
4609.1	2.712	0.051	1.619	0.03
4628.6	2.102	0.039	1.392	0.026
4648.1	1.627	0.03	1.285	0.024
4667.7	1.841	0.034	1.237	0.023
4687.2	1.858	0.034	1.246	0.023
4706.7	1.8	0.033	1.215	0.022
4726.3	1.629	0.03	1.044	0.019
4745.8	1.485	0.027	0.872	0.016
4765.3	1.426	0.026	0.926	0.017
4784.9	1.419	0.026	0.914	0.017
4804.4	1.543	0.028	0.92	0.017
4823.9	1.847	0.033	1.068	0.019
4843.4	2.082	0.037	1.322	0.024
4863	2.57	0.046	1.481	0.026
4882.5	2.515	0.045	1.786	0.032
4902	2.573	0.045	2.097	0.037
4921.6	2.763	0.049	1.896	0.033
4941.1	2.498	0.044	1.778	0.031
4960.6	2.045	0.036	1.391	0.024
4980.1	1.914	0.033	1.255	0.022
4999.7	1.437	0.025	1.016	0.018
5019.2	1.341	0.023	0.77	0.013
5038.7	1.7	0.029	0.897	0.015
5058.3	2.015	0.034	0.926	0.016
5077.8	2.078	0.035	0.975	0.017
5097.3	2.57	0.044	1.024	0.017
5116.9	2.286	0.039	0.899	0.015
5136.4	2.118	0.036	0.947	0.016
5155.9	2.178	0.037	0.839	0.014
5175.5	1.687	0.028	0.703	0.012
5195	1.412	0.023	0.534	0.009
5214.5	1.526	0.025	0.456	0.008
5234	1.848	0.031	0.407	0.007
5253.6	1.996	0.033	0.435	0.007

5273.1	1.725	0.028	0.444	0.007
5292.6	1.551	0.025	0.369	0.006
5312.2	1.115	0.018	0.339	0.006
5331.7	0.878	0.014	0.256	0.004
5351.2	0.792	0.013	0.217	0.004
5370.8	0.641	0.01	0.197	0.003
5390.3	0.588	0.009	0.209	0.003
5409.8	0.635	0.01	0.193	0.003
5429.3	0.571	0.009	0.221	0.004
5448.9	0.553	0.009	0.251	0.004
5468.4	0.557	0.009	0.257	0.004
5487.9	0.406	0.006	0.154	0.002
5507.5	0.403	0.006	0.115	0.002
5527	0.389	0.006	0.103	0.002
5546.5	0.337	0.005	0.117	0.002
5566	0.264	0.004	0.102	0.002
5585.6	0.263	0.004	0.102	0.002
5605.1	0.286	0.004	0.1	0.002
5624.6	0.262	0.004	0.071	0.001
5644.2	0.176	0.003	0.056	0.001
5663.7	0.136	0.002	0.047	0.001
5683.2	0.098	0.001	0.029	0
5702.8	0.067	0.001	0.024	0
5722.3	0.052	0.001	0.018	0
5741.8	0.035	0.001	0.013	0
5761.4	0.027	0	0.01	0
5780.9	0.018	0	0.008	0
5800.4	0.016	0	0.007	0
5819.9	0.013	0	0.007	0
5839.5	0.011	0	0.006	0
5859	0.011	0	0.006	0
5878.5	0.011	0	0.006	0
5898.1	0.011	0	0.006	0
5917.6	0.011	0	0.006	0
5937.1	0.01	0	0.006	0
5956.6	0.01	0	0.006	0
5976.2	0.01	0	0.006	0
5995.7	0.01	0	0.006	0
6015.2	0.01	0	0.006	0
6034.8	0.01	0	0.006	0
6054.3	0.009	0	0.006	0
6073.8	0.009	0	0.006	0
6093.4	0.009	0	0.006	0
6112.9	0.009	0	0.006	0
6132.4	0.009	0	0.005	0
6152	0.009	0	0.005	0

6171.5	0.009	0	0.005	0
6191	0.009	0	0.005	0
6210.5	0.009	0	0.005	0
6230.1	0.008	0	0.005	0
6249.6	0.008	0	0.005	0
6269.1	0.008	0	0.005	0
6288.7	0.008	0	0.005	0
6308.2	0.008	0	0.005	0
6327.7	0.008	0	0.005	0
6347.3	0.008	0	0.005	0
6366.8	0.008	0	0.005	0
6386.3	0.008	0	0.005	0
6405.8	0.008	0	0.005	0
6425.4	0.008	0	0.005	0
6444.9	0.008	0	0.005	0
6464.4	0.008	0	0.005	0
6484	0.008	0	0.005	0
6503.5	0.008	0	0.005	0
6523	0.007	0	0.005	0
6542.5	0.007	0	0.005	0
6562.1	0.007	0	0.005	0
6581.6	0.007	0	0.005	0
6601.1	0.007	0	0.005	0
6620.7	0.007	0	0.005	0
6640.2	0.007	0	0.005	0
6659.7	0.007	0	0.005	0
6679.3	0.007	0	0.005	0
6698.8	0.007	0	0.005	0
6718.3	0.007	0	0.005	0
6737.9	0.007	0	0.005	0
6757.4	0.007	0	0.004	0
6776.9	0.007	0	0.004	0
6796.4	0.007	0	0.004	0
6816	0.007	0	0.004	0
6835.5	0.007	0	0.004	0
6855	0.007	0	0.004	0
6874.6	0.007	0	0.004	0
6894.1	0.007	0	0.004	0
6913.6	0.007	0	0.004	0
6933.1	0.007	0	0.004	0
6952.7	0.007	0	0.004	0
6972.2	0.006	0	0.004	0
6991.7	0.006	0	0.004	0
7011.3	0.006	0	0.004	0
7030.8	0.006	0	0.004	0
7050.3	0.006	0	0.004	0

7069.9	0.006	0	0.004	0
7089.4	0.006	0	0.004	0
7108.9	0.006	0	0.004	0
7128.5	0.006	0	0.004	0
7148	0.006	0	0.004	0
7167.5	0.006	0	0.004	0
7187	0.006	0	0.004	0
7206.6	0.006	0	0.004	0
7226.1	0.006	0	0.004	0
7245.6	0.006	0	0.004	0
7265.2	0.006	0	0.004	0
7284.7	0.006	0	0.004	0
7304.2	0.006	0	0.004	0
7323.8	0.006	0	0.004	0
7343.3	0.006	0	0.004	0
7362.8	0.006	0	0.004	0
7382.3	0.006	0	0.004	0
7401.9	0.006	0	0.004	0
7421.4	0.006	0	0.004	0
7440.9	0.006	0	0.004	0
7460.5	0.006	0	0.004	0
7480	0.006	0	0.004	0
7499.5	0.006	0	0.004	0
7519	0.006	0	0.004	0
7538.6	0.006	0	0.004	0
7558.1	0.006	0	0.004	0
7577.6	0.006	0	0.004	0
7597.2	0.006	0	0.004	0
7616.7	0.006	0	0.004	0
7636.2	0.006	0	0.004	0
7655.8	0.006	0	0.004	0
7675.3	0.006	0	0.004	0
7694.8	0.006	0	0.004	0
7714.4	0.006	0	0.004	0
7733.9	0.006	0	0.004	0
7753.4	0.006	0	0.004	0
7772.9	0.006	0	0.004	0
7792.5	0.006	0	0.004	0
7812	0.006	0	0.004	0
7831.5	0.006	0	0.004	0
7851.1	0.006	0	0.004	0
7870.6	0.006	0	0.004	0
7890.1	0.006	0	0.004	0
7909.6	0.006	0	0.004	0
7929.2	0.006	0	0.004	0
7948.7	0.006	0	0.004	0

7968.2	0.006	0	0.004	0
7987.8	0.006	0	0.004	0
8007.3	0.006	0	0.004	0
8026.8	0.006	0	0.004	0
8046.4	0.006	0	0.004	0
8065.9	0.006	0	0.004	0
8085.4	0.006	0	0.004	0
8105	0.006	0	0.004	0
8124.5	0.006	0	0.004	0
8144	0.006	0	0.004	0
8163.5	0.006	0	0.004	0
8183.1	0.006	0	0.004	0
8202.6	0.006	0	0.004	0
8222.1	0.006	0	0.004	0
8241.7	0.006	0	0.004	0
8261.2	0.006	0	0.004	0
8280.7	0.006	0	0.004	0
8300.3	0.006	0	0.004	0
8319.8	0.006	0	0.004	0
8339.3	0.006	0	0.004	0
8358.8	0.006	0	0.004	0
8378.4	0.006	0	0.004	0
8397.9	0.006	0	0.004	0
8417.4	0.006	0	0.004	0
8437	0.006	0	0.004	0
8456.5	0.006	0	0.004	0
8476	0.006	0	0.004	0
8495.5	0.006	0	0.004	0
8515.1	0.006	0	0.004	0
8534.6	0.006	0	0.004	0
8554.1	0.006	0	0.004	0
8573.7	0.006	0	0.004	0
8593.2	0.006	0	0.004	0
8612.7	0.006	0	0.004	0
8632.3	0.006	0	0.004	0
8651.8	0.006	0	0.004	0
8671.3	0.006	0	0.004	0
8690.8	0.006	0	0.004	0
8710.4	0.006	0	0.004	0
8729.9	0.006	0	0.004	0
8749.4	0.006	0	0.004	0
8769	0.006	0	0.004	0
8788.5	0.006	0	0.004	0
8808	0.006	0	0.004	0
8827.6	0.006	0	0.004	0
8847.1	0.006	0	0.004	0

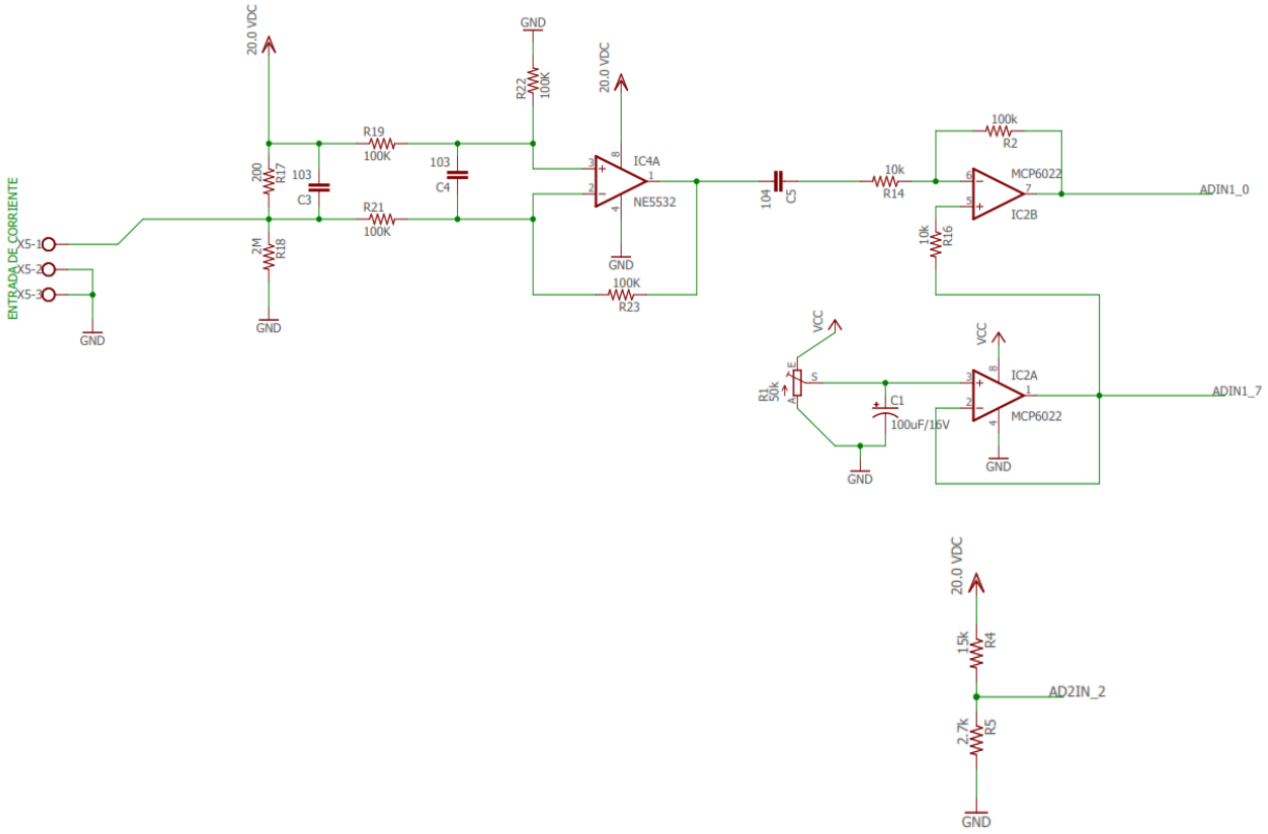
8866.6	0.006	0	0.004	0
8886.2	0.006	0	0.004	0
8905.7	0.006	0	0.004	0
8925.2	0.006	0	0.004	0
8944.7	0.006	0	0.004	0
8964.3	0.006	0	0.004	0
8983.8	0.006	0	0.004	0
9003.3	0.006	0	0.004	0
9022.9	0.006	0	0.004	0
9042.4	0.006	0	0.004	0
9061.9	0.006	0	0.004	0
9081.5	0.006	0	0.004	0
9101	0.006	0	0.004	0
9120.5	0.006	0	0.004	0
9140	0.006	0	0.004	0
9159.6	0.006	0	0.004	0
9179.1	0.006	0	0.004	0
9198.6	0.006	0	0.004	0
9218.2	0.006	0	0.004	0
9237.7	0.006	0	0.004	0
9257.2	0.006	0	0.004	0
9276.8	0.006	0	0.004	0
9296.3	0.006	0	0.004	0
9315.8	0.006	0	0.004	0
9335.3	0.006	0	0.004	0
9354.9	0.006	0	0.004	0
9374.4	0.006	0	0.004	0
9393.9	0.006	0	0.004	0
9413.5	0.006	0	0.004	0
9433	0.006	0	0.004	0
9452.5	0.006	0	0.004	0
9472	0.006	0	0.004	0
9491.6	0.006	0	0.004	0
9511.1	0.006	0	0.004	0
9530.6	0.006	0	0.004	0
9550.2	0.006	0	0.004	0
9569.7	0.006	0	0.004	0
9589.2	0.006	0	0.004	0
9608.8	0.006	0	0.004	0
9628.3	0.006	0	0.004	0
9647.8	0.006	0	0.004	0
9667.3	0.006	0	0.004	0
9686.9	0.006	0	0.004	0
9706.4	0.007	0	0.004	0
9725.9	0.007	0	0.004	0
9745.5	0.006	0	0.004	0

9765	0.007	0	0.004	0
9784.5	0.007	0	0.004	0
9804.1	0.006	0	0.004	0
9823.6	0.006	0	0.004	0
9843.1	0.006	0	0.004	0
9862.7	0.006	0	0.004	0
9882.2	0.006	0	0.004	0
9901.7	0.006	0	0.004	0
9921.2	0.006	0	0.004	0
9940.8	0.007	0	0.004	0
9960.3	0.007	0	0.004	0
9979.8	0.006	0	0.004	0

Fecha	20/02/2021
Hora	02:08:29
Datos del Motor:	MJ8
RPM:	2800
Potencia(HP):	2
Tipo de Motor:	AC
Conectado a VFD:	No
Tipo de Montaje:	Horizontal
Tipo de Rodamiento:	Cojinete
Motor Desacoplado:	Si

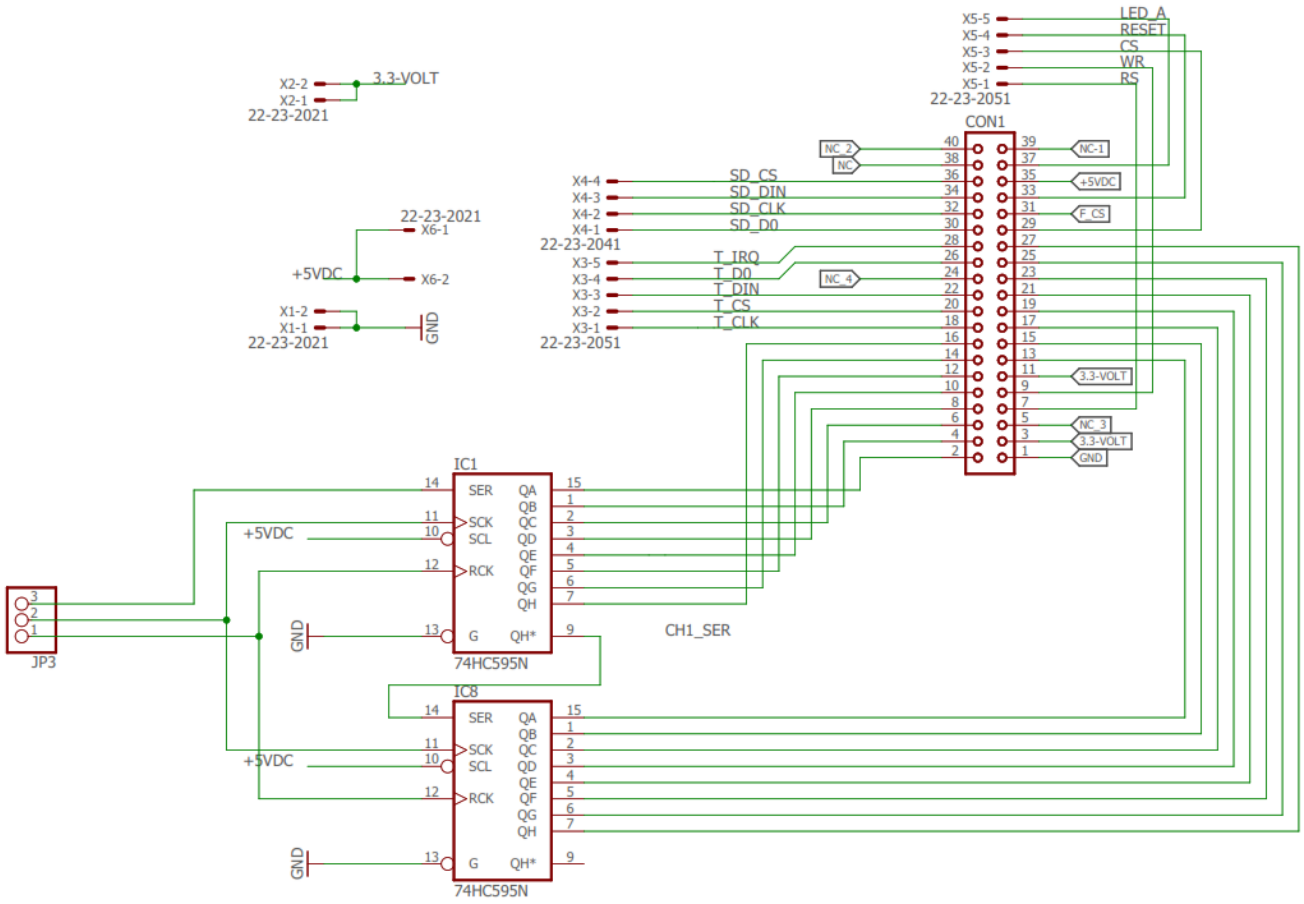
ANEXO 11

Esquemático de la etapa de acondicionador de señal



ANEXO 12

Esquemático de la etapa del circuito de visualización



ANEXO 13

Esquemático de la etapa del sistema de reloj y memoria