DESARROLLO E IMPLEMENTACIÓN DE ALGORITMO PARA EL PROCESAMIENTO Y ANÁLISIS DE DATOS DE SISTEMAS BIOSENSORES BASADOS EN EL INTERFERÓMETRO MACH ZEHNDER

ADRIANA CAROLINA SANABRIA BORBÓN

UNIVERSIDAD SANTO TOMÁS
DIVISIÓN DE INGENIERÍAS
FACULTAD DE INGENIERÍA ELECTRÓNICA
BOGOTÁ
2012

DESARROLLO E IMPLEMENTACIÓN DE ALGORITMO PARA EL PROCESAMIENTO Y ANÁLISIS DE DATOS DE SISTEMAS BIOSENSORES BASADOS EN EL INTERFERÓMETRO MACH ZEHNDER

ADRIANA CAROLINA SANABRIA BORBÓN

Trabajo de grado para optar al título de Ingeniero Electrónico

Director
Ing. Jaime Vitola Oyaga

Codirector
Ing. David Fariña

UNIVERSIDAD SANTO TOMÁS
DIVISIÓN DE INGENIERÍAS
FACULTAD DE INGENIERÍA ELECTRÓNICA
BOGOTÁ
2012

NOTA DE ACEPTACIÓN

El presente trabajo de grado de grado titulado "DESARROLLO E IMPLEMENTACIÓN DE ALGORITMO PARA EL PROCESAMIENTO Y ANÁLISIS DE DATOS DE SISTEMAS BIOSENSORES BASADOS EN EL INTERFERÓMETRO MACH ZEHNDER" ha sido aprobado por el director del mismo, como requisito parcial para optar al titulo de Ingeniero Electrónico de acuerdo con lo mandado por la facultad de Ingeniería Electrónica de la Universidad Santo Tomás.

Ing. Jaime Vitola Oyaga

Director

NOTA DE ACEPTACIÓN

El presente trabajo de grado de grado titulado "DESARROLLO E IMPLEMENTACIÓN DE ALGORITMO PARA EL PROCESAMIENTO Y ANÁLISIS DE DATOS DE SISTEMAS BIOSENSORES BASADOS EN EL INTERFERÓMETRO MACH ZEHNDER" ha sido aprobado por el Codirector del mismo, quien recibió el software desarrollado y manifiesta que cumple con las especificaciones.

Ing. David Fariña

Codirector

Nota de aceptación

Firma
Nombre:
Jurado

Firma
Nombre:
Jurado

Firma
Nombre:
Jurado

Bogotá, Abril 22 de 2012

A mis padres Nancy y Pedro, a mi ahijada Valentina y toda mi familia que siempre me apoya y me acompaña.

AGRADECIMIENTOS

Al los ingenieros Jaime Vitola y David Fariña los directores de este trabajo de grado por orientarme en el desarrollo de este proyecto, instruirme en el uso de nuevas herramientas y de forma general por contribuir en mi formación profesional y personal.

A la Dra. Laura Lechuga y su grupo de investigación NanoB2A por acogerme y permitirme trabajar con ellos en sus proyectos de investigación.

Al Consejo Superior de Investigaciones Científicas CSIC por brindarme la oportunidad de realizar una estancia corta de introducción a la investigación en el 2009.

A la Universidad Santo Tomás por velar por la formación integral de sus estudiantes y favorecer en ellos el desarrollo técnico, humanístico e investigativo mediante los procesos de enseñanza-aprendizaje.

AUTORIDADES DE LA UNIVERSIDAD SANTO TOMÁS

Facultad de Ingeniería Electrónica

P. CARLOS MARIO ALZATE MONTES, O.P.

Rector General

P. LUIS FRACISCO SASTOQUE POVEDA, O.P.

Vicerrector Administrativo

P. EDUARDO GONZÁLEZ GIL, O.P.

Vicerrector Académico

HÉCTOR FABIO JARAMILLO SANTAMARIA

Secretario General

P. PEDRO JOSÉ DÍAZ CAMCHO, O.P .

Decano de División

ING. ADRIANA CECILIA PAEZ PINO

Decano Facultad de Ingeniería Electrónica

MYRIAM GÓMEZ COLMENARES

Secretario de División

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

GLOSARIO

ADQUISICIÓN DE DATOS: Lectura o captura de datos empleando un conversor analógico a digital (ADC) para su digitalización y posterior transmisión, visualización, procesamiento o almacenamiento.

AGILISMO: Filosofía que propone una nueva forma de desarrollar software basada en los principios del Manifiesto ágil.

ALGORITMO: En general puede definirse como una secuencia de pasos ordenados para el cumplimiento de un objetivo. En el campo de tratamiento digital de señales es un método de cálculo matemático formado por una serie de ecuaciones para el desarrollo de una tarea particular.

AMPLITUD: Es el valor o magnitud de una señal en un instante de tiempo determinado.

ANALITO: Componente de interés en el análisis de una muestra química.

BIOSENSOR: Dispositivo compacto compuesto por un elemento de reconocimiento biológico y un sistema de transducción empleado para la detección de sustancias biológicas y su transformación a una señal que pueda ser almacenada, transmitida o procesada.

BUFFER: Sustancia que fluye normalmente por las celdas de microfluídica del biosensor MZI y que corresponde a una disolución con características adecuadas para birndar un medio natural a las proteinas, ADN o demas sustancias biológicas.

BUS: Conjunto de cables empleados para transmitir información digital de un punto a otro.

DERIVADA: En términos matemáticos es la pendiente de la recta tangente a una curva en un punto. Se emplea para el estudio de la rapidez con la cual cambia una función.

FASE(Φ): Es una medida de diferencia de tiempo en una señal senoidal que se expresa en términos de ángulo, en grados o radianes. Se puede referir al ángulo inicial de la señal, a la fracción de ciclo de onda que ha transcurrido en un instante determinado con respecto al origen o a la diferencia de tiempo entre dos ondas senoidales, aunque este último es más conocido como desfase.

FRECUENCIA DE MUESTREO: Es el número de muestras por segundo tomadas de una señal analógica.

GIT: Sistema de control de versiones de tipo distribuido diseñado pro Linus Torvalds, el cual facilita la gestión de archivos y sus modificaciones.

INSTRUMENTO VIRTUAL: Término que designa un instrumento de medida implementado en un PC, con controles e indicadores que se asemejan a los de un instrumento físico.

LABVIEW: Es un paquete de software de la compañía *National Instruments* de programación gráfica que permite entre otras múltiples aplicaciones, el desarrollo de potentes tareas de procesamiento digital de señales en un PC. Este software introdujo el término Instrumento Virtual.

MZI: (Interferómetro Mach-Zehnder) Es un dispositivo empleado para determinar el desplazamiento de fase relativo de dos haces de luz proveniente de una misma fuente de luz. Una de sus aplicaciones es como biosensor de tipo óptico de alta sensibilidad.

NÚMERO COMPLEJO: Es un número formado por una parte real ($\Re$) y una parte imaginaria ($\Im$). La parte compleja esta multiplicada por el número $i(\sqrt{-1})$.

REFACTORIZACIÓN: Técnica en ingeniería de software que permite modificar el diseño del código fuente con el fin de eliminar duplicidad o hacerlo mas claro y fácil de mantener. En general consiste "limpiar" el código sin alterar su funcionamiento.

SHA(Algoritmo de Hash seguro): Es un sistema de funciones hash criptograficas desarrollado por la NSA en 1993. Es un algoritmo muy seguro empleado para la encripción de mensajes.

SPRINT: Periodo comprendido entre una y cuatro semanas en el cual se desarrolla un *Sping Backlog* o listado de tareas o funciones del software. Al final del Sprint se entrega una versión ejecutable.

TDD:(Desarrollo Guiado por Pruebas) Es una metodología o técnica para el desarrollo de software que propone construir primero las pruebas o ejemplos de lo que se espera que el software haga, se verifica que fallen, luego se programa el código que las satisfaga y finalmente se refactoriza o limpia el código; este proceso se repite hasta cumplir todos los requisitos del usuario final.

TEOREMA DE NYQUIST: Establece la frecuencia mínima a la que debe ser muestreada una señal analógica con el fin de obtener información suficiente. Esta frecuencia de muestreo debe ser superior a dos veces la frecuencia máxima de la señal.

TRANSFORMADA DE HILBERT: Es una transformación que introdce un desplazamiento de fase de 90 grados en todas las frecuencias de una señal dada.

UML: (Lenguaje Unificado de Modelado) Es un lenguaje de modelado de sistemas muy usado, que ofrece una notación estándar para la construcción de modelos que permitan visualizar, especificar y documentar un sistema orientado a objetos.

RESUMEN

En este documento se describe el proceso de diseño e implementación de un software para la adquisición y procesamiento de señales provenientes del biosensor óptico MZI (Interferómetro Mach-Zenhder) que permite medir el desfase de la señal el cual está asociado con la detección del analito.

Este software se desarrolló a partir del estudio del funcionamiento del MZI, de las características del montaje experimental del mismo y de la señal típica que entrega.

Para el diseño del software se partió del estudio del lenguaje de modelado de sistemas UML, metodologías ágiles como TDD (Desarrollo Guiado por pruebas), la programación orientado a objetos, el uso del software de programación gráfica LabVIEW y el funcionamiento del software Git como sistema de control de versiones.

Se empleó una técnica para el cálculo del desfase basado en la transformada de Hilbert, y se complementó con otro algoritmo basado en la acumulación sucesiva de diferencias de fase punto a punto.

Finalmente se presentan las pruebas experimentales que se realizaron para probar el funcionamiento del software y su desempeño.

PALABRAS CLAVE:

BIOSENSOR MZI; ADQUISICIÓN DE SEÑALES; PROCESAMIENTO DE SEÑALES; SOFTWARE, TDD, GIT, POO.

ANTECEDENTES

Existen muchos procesos y fenómenos que son objeto de estudio en la actualidad y sobre los cuales es de vital importancia la recolección y procesamiento de información en tiempo real. Este es el caso de entornos como el estudio de tejidos infectados con alguna enfermedad, ecosistemas contaminados con agentes tóxicos, o la producción de medicamentos, en los cuales el objetivo principal es la identificación de una sustancia determinada de forma precisa e inmediata. Como respuesta a las necesidades en este tipo de mediciones se han desarrollado e implementando sistemas denominados biosensores.

Como lo presentan Sánchez y Lechuga[1] los biosensores son dispositivos compactos compuestos por un elemento de reconocimiento biológico destinado a la identificación de una sustancia química específica y un sistema de transducción el cual produce una señal cuantificable asociada a la información reconocida por el receptor para ser transmitida y procesada. Estos dispositivos pueden detectar sustancias con una sensibilidad muy alta, basada en un reconocimiento molecular específico y selectivo de forma muy rápida como lo enuncia la red *Bio-Sens*[2].

El grupo de investigación *NanoBiosensors and Bioanalytical Applications Group (nanoB2A)* ha desarrollado un dispositivo interferométrico Mach-Zehnder (MZI) como biosensor óptico de alta sensibilidad[3].

Como lo describen Lechuga el al.[4] en este dispositivo, un haz de luz se propaga por una guía de onda, la cual tiene una unión en Y que divide el haz propagado en dos ramas (referencia y sensora). La rama sensora dispone de una ventana que permite el contacto directo entre el núcleo de la guía y el medio a medir, de manera que la interacción biológica modifica las condiciones de propagación del haz de luz. Las dos ramas vuelven a unirse mediante otra unión Y donde se produce la interferencia de los dos haces. A la salida de la guía se mide la intensidad de luz mediante fotodetectores, la cual es función del desfase entre ambos haces.

En consecuencia, este dispositivo permite determinar la presencia de un analito dentro de una muestra, tras analizar el cambio de fase de una señal de potencia contra tiempo adquirida a la salida del biosensor[5].

---

[1]SANCHEZ, José y LECHUGA, Laura. Desarrollo de un biosensor fotónico de alta sensibilidad basado en interferómetros Mach-Zehnder integrados en tecnología de silicio. Universidad Autónoma de Madrid. Facultad de Ciencias. Dpto. de Física de Materiales, 2007.

[2]BIOSENS, [citado en 10 de octubre de 2011], Disponible en: http://www.imm.cnm.csic.es /RedBiosensores/tecnologia-de-biosensores.html.

[3]NANO BIOSENSORS AND BIOANALYTICAL APPLICATIONS, [citado en 17 de septiembre de 2011], Disponible en Internet: http://www.cin2.es/ biosensores/

[4]LECHUGA, Laura; ZINOVIEV, Kirill; CARRASCOSA, Laura y MORENO, M. Nanodevices for Biosensing: Design, Fabrication and Applications. En: Nanotechnologies for the Life Sciences, Septiembre, 2007, vol. 10, p. 317-344.

[5]LECHUGA, Laura; ZINOVIEV, Kirill; HIDALGO, Orlando; DOMINGUEZ, Carlos y ELIZALDE J. Biosensing microsystems: fast, label-free, real-time clinical testing. Diciembre 2008, SPIE Newsroom, 2p.

PLANTEAMIENTO DEL PROBLEMA

El biosensor MZI se caracteriza por su gran sensibilidad, sin embargo tiene dificultades asociadas con la periodicidad de la señal por lo que no es posible determinar de forma precisa el cambio de fase causado por la interacción biomolecular observando la señal y aproximando el valor empíricamente. Por esto se hizo necesario desarrollar un software que realice la adquisición de la señal proveniente del montaje experimental del MZI, la almacene y le aplique técnicas de procesamiento que permitan entregar numérica y gráficamente el desfase de la medida. El valor del desfase obtenido permitirá determinar el nivel de adsorción o disociación de las reacciones producidas en el experimento.

Por esto es necesario inicialmente documentar el funcionamiento de este biosensor y las características de la señal que entrega para proponer el algoritmo de procesamiento y elegir una plataforma adecuada para la implementación del mismo, buscando además de un adecuado funcionamiento, que el proceso sea rápido y tenga una carga computacional moderada.

Por otra parte es importante considerar que al tratarse de un desarrollo de software se debe tratar como tal, y en este sentido se deben explorar y emplear las metodologías de diseño e ingeniería de software que sean necesarias para lograr un sistema que además de funcional, sea estable y esté bien estructurado y documentado para que sea fácil de mantener y de modificar.

Finalmente es fundamental definir un medio, estrategia o plataforma para comunicar al desarrollador con el dueño del software y permitir compartir los archivos necesarios para la realización de las pruebas con los sistemas biosensores teniendo en cuenta las limitaciones de la distancia.

JUSTIFICACIÓN

La detección de sustancias biológicas por medio de biosensores tiene múltiples aplicaciones por ejemplo en la medicina y en el estudio del medio ambiente, por lo que es necesario tener parámetros que permitan comparar medidas para determinar de forma precisa el grado de presencia de una determinada sustancia en una muestra bajo estudio. El MZI es uno de los biosensores empleados en estas biodetecciones, por lo cual el parámetro asociado con la detección de una sustancia es el cambio de fase entre las dos señales en la unión Y. Es por esto que resulta importante el desarrollo de un sistema de adquisición y procesamiento de datos robusto que aporte la precisión necesaria.

Por otra parte el software además de ser funcional, desde el punto de vista del usuario debe tener un entorno intuitivo e informar su funcionamiento. En lo que respecta al dueño del software, este debe estar muy bien estructurado, diseñado y documentado de forma que sea sencillo de entender y depurar por otro desarrollador.

OBJETIVOS

Objetivo General

Desarrollar un software para la adquisición y procesamiento de las señales de un nanobiosensor tipo MZI con el fin de extraer información que permita calcular el desfase y con él estimar la biodetección de una sustancia determinada.

Objetivos Específicos

1. Consultar la bibliografía referente al tema para documentar el problema y las respectivas alternativas de solución.
2. Desarrollar el software de adquisición y almacenamiento de datos con interfaz gráfica.
3. Identificar las características de las señales a procesar y los requerimientos del procesamiento.
4. Seleccionar las mejores estrategias y procedimientos que permitan extraer la información del desfase entre las señales de forma más confiable y precisa.
5. Planear un algoritmo con etapas bien definidas, que permita extraer de las señales información que indique la existencia y el grado de interacción con una sustancia determinada.
6. Seleccionar una o varias plataformas de hardware y software que permita la implementación del algoritmo, realizando pruebas para determinar los tiempos ejecución, la robustez del algoritmo, la precisión y confiabilidad de los resultados en cada caso.
7. Evaluar el rendimiento del algoritmo implementado con un banco de señales con el fin de establecer su tolerancia a variación de parámetros, manteniendo un buen rendimiento.
8. Escribir un documento de soporte que describa el desarrollo del proyecto y los resultados obtenidos.

1. MARCO TEÓRICO

1.1. SENSÓRICA

Normalmente en el mercado se encuentran dispositivos capaces de detectar propiedades físicas como temperatura, presión, luz, campos eléctricos o magnéticos, o propiedades químicas como concentración de gases e interacciones molecuares. Estos desarrollos sin duda fueron inspirados en el cuerpo humano el cual está conformado por sistemas complejos que detectan las condiciones del entorno, las traducen en señales electro-químicas y las envían al cerebro para realizar alguna acción con base en esa información. No obstante, dichos sentidos tienen limitaciones frente a diversas magnitudes que no son capaces de detectar (ejemplo el ultrasonido o la luz ultravioleta) para lo cual se han desarrollado dispositivos específicos que brindan mayor información sobre el entorno.

Fue así como surgieron los sensores, dispositivos que ante las propiedades del entorno reaccionan o presentan un cambio el cual es función de la magnitud detectada y por medio de un transductor se puede convertir a otra forma de energía para ser procesada, almacenada o transmitida[6]. De forma más concreta Pérez[7] define un sensor como un dispositivo capaz de registrar de forma directa, continua y reversible un parámetro físico o la concentración de una sustancia química.

Teniendo en cuenta que los sensores se implementan para recolectar información y con base en ella tomar decisiones y ejecutar acciones, es importante que estos funcionen de forma continua (en algunos casos en tiempo real), que tengan buena sensibilidad, es decir que los cambios a la entrada sean notables en la salida, que tengan baja interferencia, alta sensibilidad, sean lineales y dependiendo la aplicación, que tengan bajo costo, tamaño reducido y alta capacidad de integración.

1.2. SENSANDO A NANOESCALA

Con el propósito de lograr dispositivos cada vez más funcionales, efectivos y portables la tecnología ha avanzado hacia la exploración del desarrollo de componentes a menor escala que permitan mayor integración, menor consumo y mayor facilidad de uso. En este sentido es ahora posible pensar que en un futuro cercano las personas no deberán ir al laboratorio a que les tomen las muestras y esperar tiempo para obtener los resultados sino que podrán llevar un pequeño laboratorio con ellos que incluso realice pruebas y monitoreo constante para alertar de forma inmediata frente a alguna anomalía o situación de riesgo.

Conforme al nivel de integración y desarrollo del siglo XXI se puede hablar de la nanoescala haciendo referencia a las partículas con un tamaño entre 1 y 100 nm. Es así como el término nanosensor, se usa para referirse a dispositvos sensores capaces de recolectar información a nivel de las nanoparticulas que conforman el universo macroscópico.

Hay investigadores[8] que sugieren que a esa escala no se debe hablar de nanociencia sino de una

---

[6]PALLÁS, Ramón. *Sensores y acondicionadores de señal*, 4 ed., Barcelona: Ed. Marcombo. 2008. 217 p.

[7]PÉREZ, Concepción *Sensores ópticos*, Universidad de Valencia, 1996.

[8]GIRALDO, Jairo; GONZÁLEZ, Edgar y GÓMEZ Fernando. *Nanotecnociencia: Nociones preliminares sobre el universo nanoscópico*, Bogotá D.C.: Ediciones Buinaima, 2007. 254 p.

física con fundamentos mecánico cuánticos, ya que la importancia que en la macroescala tienen la inercia y la fuerza de la gravedad propiedades que dependen de la masa en la nanoescala no la tienen, por el contrario resultan fundamentales propiedades como la maleabilidad, la fricción, la resistencia al corte, la adhesión, y las fuerzas de interacción entre los átomos. Necesariamente este cambio de importancia de las fuerzas y propiedades que actúan en las estructuras a nivel molecular, y las diferencias de percepción de las condiciones como intensidad de luz y temperatura da una idea de lo sofisticado que debe ser un sensor a esa escala.

Hay otro factor que introduce mayor complejidad en el desarrollo de estos dispositivos y es el tipo de material con el que construyen los nanosistemas o las nanoestructuras, el cual si es de tipo inorgánico se trabajará en un entorno seco, pero si es orgánico es decir un sistema biológico debe ser tratado en solución o en un medio acuoso.

1.3. BIOSENSORES

El prefijo bio hace referencia a los seres vivos, por lo que un biosensor no es más que un dispositivo que detecte sustancias provenientes de estos seres. Las principales aplicaciones que se han dado a estos dispositivos pertenecen al campo de la medicina donde son empleados para encontrar sustancias que sean causales de enfermedades porque estén en cantidades fuera de los rangos normales o porque resulten atípicas en algún órgano como pueden ser las células cancerígenas. Otro ejemplo de las aplicaciones de los biosensores es en la determinación de la calidad del medio ambiente, es decir la composición del aire, el suelo y el agua, y la medida en que estas condiciones afectan o favorecen la vida de las personas que allí habitan. Sin embargo también se emplean en procesamiento de comida, en la industria química, y en la defensa contra ataques terroristas frente a lo que se conoce como armas biológicas.

Los biosensores son dispositivos compactos compuestos por un elemento de reconocimiento biológico y un sistema de transducción. El primer componente está destinado a la identificación de una sustancia química específica, para esto el analito objetivo interactúa con una sustancia inmovilizada en el receptor generando cambios físicos o químicos convertidos en un efecto medible, preferiblemente óptico o eléctrico. La segunda etapa por su parte, produce una señal cuantificable proporcional a la información reconocida por el receptor para ser transmitida y procesada por un sistema electrónico como lo enuncian Sánchez[9]y Zhang[10].

Estos dispositivos pueden detectar sustancias con una sensibilidad muy alta, basada en un reconocimiento molecular específico y selectivo de forma muy rápida como lo describe la red *Bio-Sens*[11]. El éxito de estos dispositivos esta en el mecanismo de reconocimiento y en la integración del repector y el transductor de forma que se comportan muy bien en conjunto. Al ser estos dipositivos sensores de tipo químico donde su detección está basada en reacciones químicas que normalmete no son reversibles, estos dispositivos pueden ser desechables, tener reserva de sustancias químicas o puede regenerarse la superficie del receptor.

Con esto claro, es difícil imaginarse un biosensor de gran tamaño, teniendo en cuenta que las sustancias biológicas a analizar son conjuntos de moléculas y a lo sumo tejidos que deben ser

[9]SANCHEZ, José y LECHUGA, Laura. Desarrollo de un biosensor fotónico de alta sensibilidad basado en interferómetros Mach-Zehnder integrados en tecnología de silicio. Universidad Autónoma de Madrid. Facultad de Ciencias. Dpto. de Física de Materiales, 2007.

[10]ZHANG, Yan. *Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor*,Trabajo de grado Doctor of Philosophy in Electrica Engineering, Virginia Polytechnic Institute and State University, Diciembre 2005

[11]BIOSENS, [citado en 10 de octubre de 2011], Disponible en:http://www.imm.cnm.csic.es /RedBiosensores/tecnologia-de-biosensores.html

analizados como tales para la detección de alguna sustancia específica en ellos.

Es así como en la actualidad se puede hablar de bionanosensores[12] donde las nanopartículas con ciertas prestaciones o propiedades particularmente convenientes se han aplicado para la detección y transmisión de información de efectos y situaciones de tipo electroquímico, óptico, acústico incluso magnético. Estos dispositivos recolectan señales biológicas específicas generalmente mediante la producción de una señal digital asociada a un determinado compuesto químico o biológico.

Williams y Wade[13] destacan las características más importantes de los bionanosensores que son:

- ⋆ Capacidad de aislar sustancias biológicas específicas con poca interferencia.
- ⋆ Corto tiempo de respuesta
- ⋆ Compatibilidad con sustancias biológicas
- ⋆ Tamaño reducido permitiendo gran integración.
- ⋆ Alta sensibilidad y precisión
- ⋆ Alta Resistencia
- ⋆ Bajo costo en relación con la cantidad de pruebas que se pueden hacer con cada dispositivo.

1.4. BIOSENSORES ÓPTICOS: CARACTERÍSTICAS Y PRINCIPIOS DE FUNCIONAMIENTO

Los sensores ópticos se pueden definir como dispositivos que convierten en señales eléctricas la información transmitida por medio de luz visible o radiaciones electromagnéticas.

Este tipo de sensores se caracterizan porque el estímulo físico o químico produce cambios en las propiedades ópticas de la zona de reconocimiento y da lugar a una señal de tipo óptico[14]. Esta clase de sensores se clasifican a su vez en:

- ⋆ Sensores de absorbancia
- ⋆ Sensores de reflectancia
- ⋆ Sensores de luminiscencia
- ⋆ Sensores de dipersión Raman
- ⋆ Sensores de onda evanescente
- ⋆ Sensores de índice de refracción

Un biosensor óptico esta compuesto por un bioreceptor, un transductor óptico, una fuente de luz y un detector como lo enuncia Zhang[15]. A su vez estos dispositivos ópticos se pueden clasificar según utilicen o no compuestos para marcar. Los que no emplean marcadores son superiores ya que no son invasivos y además presentan inmunidad a los campos electromagnéticos[16]. Otra de las ventajas más importantes es la posibilidad de realizar monitoreo en tiempo real de diferentes

---

[12]GIRALDO, Jairo; GONZÁLEZ, Edgar y GÓMEZ Fernando. *Nanotecnociencia: Nociones preliminares sobre el universo nanoscópico*, Bogotá D.C.: Ediciones Buinaima, 2007. 254 p.

[13]WILLIAMS, Linda y WADE, Adam *Nanotechnology Demystified*, Ed. McGraw-Hill, 2007.

[14]PÉREZ, Concepción *Sensores ópticos*, Universidad de Valencia, 1996

[15]ZHANG, Yan. *Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor*,Trabajo de grado Doctor of Philosophy in Electrica Engineering, Virginia Polytechnic Institute and State University, Diciembre 2005

[16]KITSARA, María; MISIAKOS, Konstatinos; RAPTIS, Ioannis y MAKARONA, Eleni. *Integrated optical frequency-resolved Mach-Zehnder interferometers for label-free affinity sensing.* En: Optics express, 2010, vol. 18 no.8, p. 8193-8206. Disponible en Internet: http://www.ncbi.nlm.nih.gov/pubmed/20588665

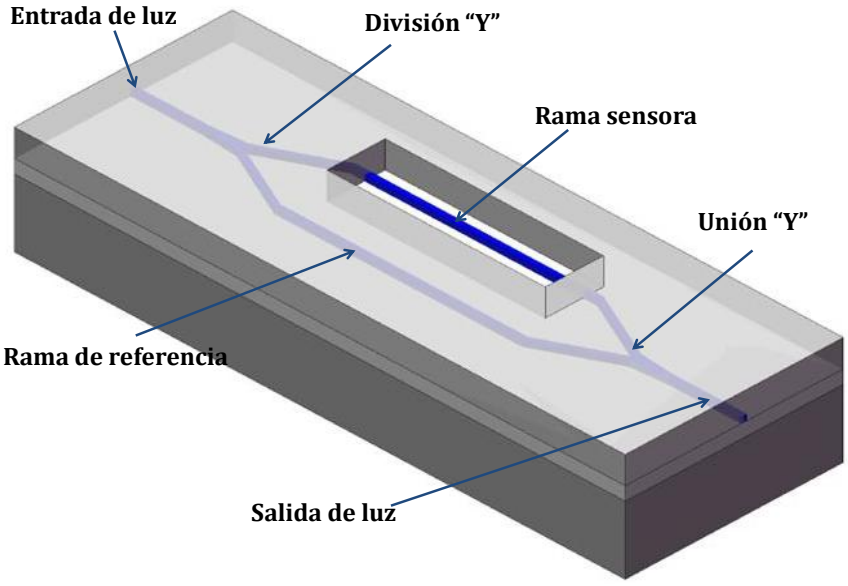
muestras de forma simultánea multiplexando varios dispositivos.

1.5. MZI COMO DISPOSITIVO BIOSENSOR

Este tipo de dispositivos se caracterizan por su alta sensibilidad, tamaño reducido, alta capacidad de paralelización de medidas, y alta integración, lo que permite crear dispositivos Lab-on-a-chip (acoplamiento de fuentes, sensores, detectores, electrónica y procesamiento en un solo chip)[17]. Además, este tipo de biosensores son una de las tecnologías que permite la detección directa de interacciones moleculares[18].

Los interferómetros Mach-Zehnder están basados en guías de onda monomodo, es decir estructuras que dirigen y orientan la propagación de las ondas electromagnéticas, soportando dos modos de polarización. Físicamente, este dispositivo se compone de una guía de onda de

Figura 1. Guía de onda del nano-biosensor MZI



entrada cuyo material es nitruro de silicio sobre un sustrato de óxido de silicio, la cual se divide en dos ramas (referencia y sensora) mediante una división Y, luego vuelve a unirse mediante otra unión Y donde se produce la interferencia entre las señales.

La guía de onda que conforma el MZI es de reflexión total interna, lo que implica que la luz se transmite confinada en su interior, sin embargo el campo electromagnético asociado decae de forma exponencial en la intercara de la guía, viajando parte de la onda por el exterior del núcleo de la guía, lo que se conoce como campo evanescente. Este campo en la rama sensora interactúa con la sustancia a detectar alterando la propagación de la onda en esta rama. La rama sensora posee una ventana que permite el contacto directo entre la superficie de la guía

[17]ZINOVIEV, Kirill; CARRASCOSA, Laura; SANCHEZ, José; SEPÚLVEDA, Borja; DOMINGUEZ, Carlos, y LECHUGA, Laura. Silicon Photonic Biosensors for Lab-on-a-Chip Applications, En: Advances in Optical Technologies, Abril, 2008, pp. 1-7.

[18]PRIETO Francisco; SEPÚLVEDA, Borja; CALLE, Ana; LLOBERA, Andreu; DOMINGUEZ, Carlos; ABAD, Antonio; MOTOYA, Ángel y LECHUGA, Laura. An integrated optical interferometric nanodevice based on silicon technology for biosensor applications. Nanotechnology, vol. 14, Aug. 2003, pp. 907-912.

y el medio a ser analizado como lo ilustra la figura 1 tomada de Prieto et al.[19].

Para permitir esta interacción se emplea una celda de flujo (microfluídica) en la que se introduce una sustancia biológica; dichas celdas poseen canales microfluídicos para transportar la muestra bajo estudio (analito) sobre la superficie de la guía de onda[20]. Sobre ésta última, ha sido previamente inmovilizado el receptor biológico específico para el analito a detectar[21]. Al presentarse la interacción biológica entre el receptor inmovilizado en la superficie de la guía y la sustancia que fluye por la celda, se modifica el índice de refracción, afectando los parámetros de propagación del haz en la rama sensora[22].

Los haces provenientes de cada una de las ramas vuelven a unirse mediante otra unión Y dando lugar a la interferencia entre los mismos y produciendo una variación de la intensidad de luz a la salida de la guía, la cual es función del coseno del desfase entre ambos haces. De esta forma, el cambio de fase en la señal permite medir y estimar la cantidad de analito detectado en la sustancia. Por lo tanto, resulta indispensable tener un resultado preciso.

La interferencia detectada a la salida de la guía de onda esta descrita por la expresión:

$$I\alpha[1 + Vcos\Delta\Phi] \tag{1}$$

Siendo $\Delta\Phi$ la diferencia de fase entre el haz de luz proveniente de la rama de referencia con respecto al de la rama sensora y V el factor de visibilidad que corresponde a la diferencia entre la intensidad máxima y mínima [23]. Este desfase se puede describir en términos de los parámetros físicos de la guía de onda como sigue:

$$\Delta\Phi = \frac{2\pi}{\lambda} L\Delta N \tag{2}$$

Siendo λ la longitud de onda, L la longitud del área sensora y ΔN el cambio en el índice de refracción.

La salida de la guía de onda se enfrenta directamente a una fibra óptica multimodo o a un objetivo que recoge la luz y la trasmite a los fotodiodos. El fotodetector traduce los cambios de intensidad del haz sensado en variaciones de corriente entre sus terminales, esta señal luego es amplificada y transformada en voltaje. Finalmente la señal se digitaliza y se transmite a un computador para almacenar los datos de cada muestra.

La figura 2 muestra la variación de corriente a la salida del fotodetector en función del tiempo, para una prueba en la que se empleó una muestra de solución de ácido clorhídrico.

Estas señales se caracterizan por tener tres partes: entrada, central y salida. En la ventana sensora se hace fluir constantemente una sustancia denominada buffer con un índice de refracción inicial. Luego se inyecta la sustancia biológica a ser analizada provocando un cambio

---

[19]PRIETO Francisco; SEPÚLVEDA, Borja; CALLE, Ana; LLOBERA, Andreu; DOMINGUEZ, Carlos; ABAD, Antonio; MOTOYA, Ángel y LECHUGA, Laura. An integrated optical interferometric nanodevice based on silicon technology for biosensor applications. Nanotechnology, vol. 14, Aug. 2003, pp. 907-912.

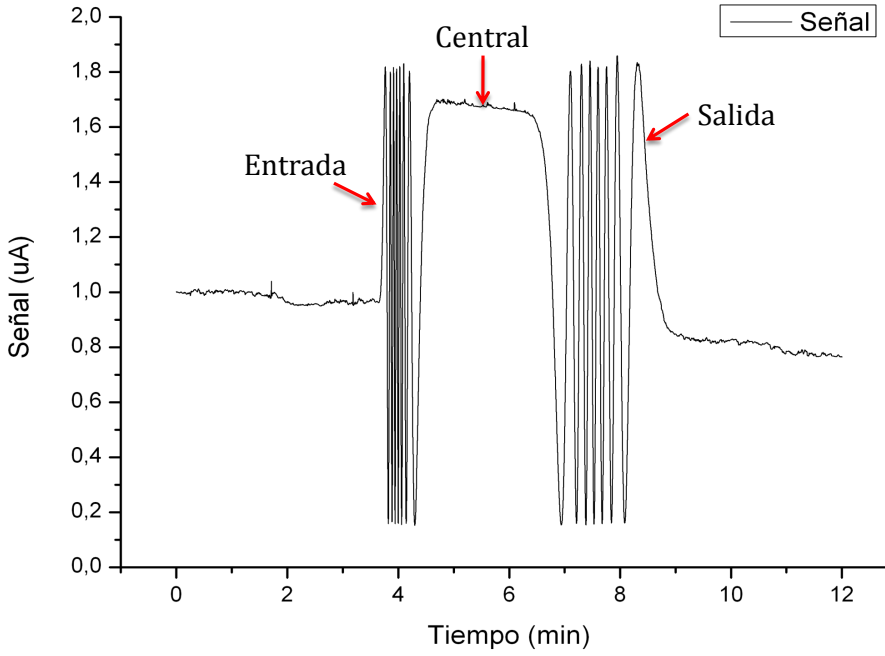
[20]BLANCO, Francisco; AGIRREGABIRIA, M.; BERGANZO, Javier; MAYORGA. K.; ELIZALDE, J.; CALLE, Ana; DOMINGUEZ, Carlos y LECHUGA, Laura *Microfluidic-optical integrated CMOS compatible devices for label-free biochemical sensing* En: Journal of Micromechanics and Microengineering, vol. 16, no. 5, Mayo 2006, pp. 1006-1016.

[21]ZINOVIEV, Kirill; CARRASCOSA, Laura; SANCHEZ, José; SEPÚLVEDA, Borja; DOMINGUEZ, Carlos, y LECHUGA, Laura. Silicon Photonic Biosensors for Lab-on-a-Chip Applications, En: Advances in Optical Technologies, Abril, 2008, pp. 1-7.

[22]Ibíd

[23]PRIETO Francisco; SEPÚLVEDA, Borja; CALLE, Ana; LLOBERA, Andreu; DOMINGUEZ, Carlos; ABAD, Antonio; MOTOYA, Ángel y LECHUGA, Laura. *An integrated optical interferometric nanodevice based on silicon technology for biosensor applications.* Nanotechnology, vol. 14, Aug. 2003, pp. 907-912.

Figura 2. Forma de señal capturada y digitalizada a la salida de la guía de onda del MZI



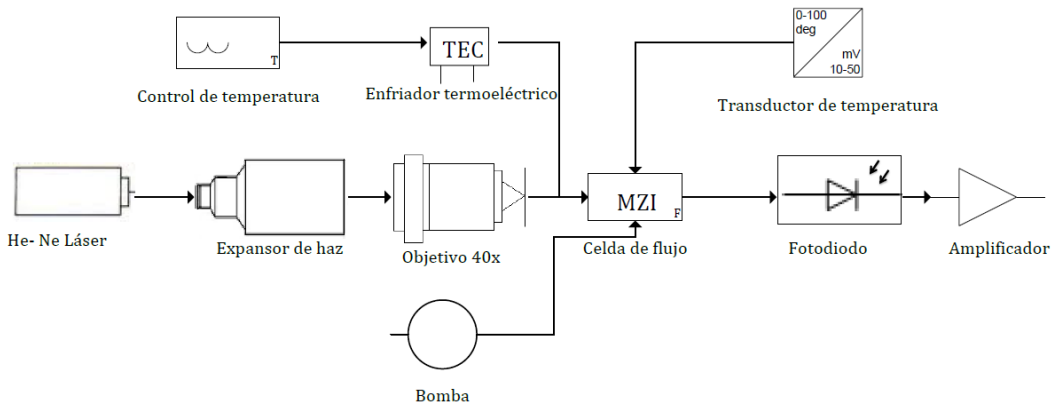
en el índice de refracción lo cual se evidencia en la parte de entrada de la señal. En la parte central esta fluyendo únicamente el analito, esta parte puede ser plana si no hay inmovilización biológica en la ventana sensora del dispositivo, pero tendrá una pendiente en caso contrario.

Finalmente vuelve a fluir el buffer, generando de nuevo variación en el índice de refracción como se observa en la parte de salida de la señal. Cuando por la ventana sensora fluya una única sustancia el índice de refracción se estabiliza y por tanto la señal se vuelve constante. La cantidad de material biológico inmoviliazado en la zona sensora corresponde a la diferencia de desfase entre la señal de entrada y la de salida.

1.6. MONTAJE EXPERIMENTAL DEL MZI

Como se evidencia en la figura 3, en el montaje experimental del MZI se emplea inicialmente un Láser de He-Ne de longitud de onda $\lambda = 632.8nm$, seguido por el expansor de haz y el objetivo 40x los cuales focalizan el haz para dirigirlo a la guía de onda.

Figura 3. Montaje experimental del MZI



Por su parte el sistema de microfluídica permite el flujo de las sustancias biológicas sobre la superficie del MZI. A la salida de la guía de onda se recoge la luz mediante fibra óptica u objetivo, llevándola al fotodetector. Seguidamente la señal es amplificada, transformada en voltaje y digitalizada empleando una tarjeta de adquisición.

El sistema además cuenta con sensado y control de temperatura, ya que la deriva térmica es una de las principales fuentes de error en la medida.

1.7. ADQUISICIÓN DE SEÑALES

En la naturaleza todas las magnitudes son analógicas, es decir que son continuas en el tiempo y en amplitud (pueden tomar cualquier valor).

Para el procesamiento de estas señales se pueden emplear sistemas analógicos, sin embargo realizar un procesamiento digital de una señal analógica tiene más ventajas que realizar el procesamiento en el dominio continuo. Una ventaja es que como un procesador de señales digitales puede ser un dispositivo programable, se tiene gran flexibilidad para realizar cambios de las operaciones ya que las modificaciones se realizan en software y no en hardware. Además el sistema digital proporciona mayor control de los requerimientos en cuanto a precisión se refiere.

Otra ventaja es la facilidad de almacenamiento de las señales digitales, lo cual brinda la posibilidad de realizar procesamiento fuera de línea, es decir el almacenamiento de las medidas para su posterior tratamiento. Adicionalmente como lo mencionan Proakis y Manolakis[24] los sistemas digitales facilitan la implementación de operaciones o algoritmos de cálculo complejos o sofisticados.

Lo anterior sumado a los avances en cuanto a intergración y desempeño de los procesadores, han propiciado que en la actualidad la mayoría de los sistemas de procesamiento empleados sean digitales.

1.8. TEORÍA DE ADQUISICIÓN DE SEÑALES.

El proceso de adquisicion de señales involucra basicamente 3 etapas, la captura de señales reales del mundo físico, la digitalización de estas muestras y el almacenamiento, análisis o visualización de los datos. Según Stewart y Hoffman[25] la digitalización de una señal analógical es el proceso por el cual se toman muestras de la misma por intervalos constantes de tiempo y se convierten en un conjunto de muestras digitales.

Este proceso de digitalización de una señal se compone esencialmente de tres partes, la primera es el muestreo y retención en el cual se capturan valores de la señal en instantes determinados separados por una cantidad constante de tiempo. Seguidamente se realiza la cuantización en la cual se aproximan los valores obtenidos a umbrales que corresponden a la división del valor máximo esperado de la señal de entrada entre la cantidad de bits del conversor analógico a digital y finalmente se realiza la codificación en la cual se convierte cada umbral en un valor binario que pueda ser tratado por un procesador.

Para elegir el sistema de adquisición de datos adecuado dependiendo de la aplicación específica

[24]PROAKIS, John y MANOLAKIS, Dimitris. Digital Signal Processing Principles, Algorithms and Applications. Tercera Edición. Prentice Hall. USA. 1996. 1033 p.

[25]STEWART, R.W. Y HOFFMAN, M.W. Digital Signal Processing An "A" to "Z". BlueBox Multimedia. 1998. 450 p.

es importante tener en cuenta los siguientes criterios:

- ⋆ Ancho de banda del bus: es la cantidad de datos que puede ser transmitido por el bus de datos en un periodo de tiempo. Se expresa en MegaBytes por segundo (MB/s). Es posible calcular el ancho de banda mínimo requerido tomando el número de bytes por muestra y multiplicándolo por la frecuencia de muestreo y el número de canales. De esta forma, comparando este dato con la velocidad del bus de transmisión se puede asegurar que este esté en capacidad de transmitir los datos adquiridos[26].
- ⋆ Portabilidad del sistema de adquisición: la portabilidad es un factor determinante en algunos escenarios y muchas veces es el responsable de la elección del hardware o sistema de adquisición y del bus de transmisión. Para ello, se puede optar por el uso de buses externos como USB o Ethernet e incluso la posibilidad de tener una estación remota y transmitir los datos por Wireless[27]. El bus de transmisión empleado en este proyecto es Universal serial Bus (USB) el cual proporciona una conexión económica y fácil de usar entre un sistema de adquisición y un PC. El USB 2.0 tiene un ancho de banda teórico de 60 MB/s, sin embargo este es compartido por los dispositivos conectados al mismo controlador USB. Otra de las ventajas que brinda emplear dispositivos conectado por USB es la posibilidad de usarlos luego de conectarlos sin necesidad de reiniciar el PC ni configurarlos manualmente, ya que cuenta con detección automática de dispositivos.
- ⋆ Frecuencia de muestreo(Fs): se refiere a la velocidad a la cual los canales del sistema de adquisición toman muestras de la señal medida en Hertz (Hz). Éste parámetro está determinado por la frecuencia máxima de la señal que se desea muestrear ya que según el teorema de Nyquist la frecuencia de muestreo debe ser por lo menos 2 veces la frecuencia máxima de la señal a adquirir. Sin embargo en la práctica se obtienen mayor precisión cuando la relación es por lo menos de 10 veces.
- ⋆ Resolución: es la menor variación de la señal que puede ser detectada pero el sistema de adquisición y está asociada con la cantidad de bits del conversor analógico a digital lo que influye en la cantidad de niveles de cuantificación del mismo. Al dividir el rango de la medición (por ejemplo +/- 10V) entre la cantidad de niveles se obtiene el valor mínimo de cambio que puede ser detectado por el sistema.

1.9. HARDWARE DE ADQUISICIÓN DE DATOS DE NATIONAL INSTRUMENTS

En 2005 National Instruments adquirió la compañía Measurement Computing, un proveedor de sistemas de adquisición de datos de bajo costo, con lo cual se fortaleció en el mercado como proveedor de software, hardware y servicios.

Los productos de hardware fabricados por National Instruments para la adquisición de datos están comprendido en dos grandes grupos, los de solo un dispositivo y los de un sistema completo.

Los que son un solo dispositivo a su vez se dividen en DAQ portátil y DAQ de escritorio, mientras que los sistemas se dividen en NI CompactDAQ y Paltaforma PXI.

Como se observa en la tabla 1 las tarjetas de adquisición de un dispositivo portátil emplean distintos buses de comunicación como USB, Wi-fi y Ethernet, tienen una buena tasa de muestreo y con un bajo precio son dispositivos con muy buena relación costo- beneficio.

---

[26]NATIONAL INSTRUMENTS, [Citado en 15 de noviembre de 2011] Disponible en internet: http://www.ni.com/LabVIEW/whatis/esa/

[27]Ibíd

Tabla 1. Características de los sistemas de adquisición de National Instruments

Características	DAQ Portátil	DAQ de Escritorio	NI CompactDAQ	Plataforma PXI
Bus	USB, Wi-Fi, Ethernet	PCI, PCI Express	USB, Wi-Fi, Ethernet	PXI, PXI Express
Portabilidad	Excelente	Bueno	Mejor	Bueno
Número de Canales de E/S	1 a 100	1 a 100	1 a 250	1 a 1000+
Configuración de E/S	Fijo	Fijo	Modular	Modular
Razón de Muestreo Máx.	2 MS/s	10 MS/s	1 MS/s	10 MS/s
Acondicionamiento de Señales Integrado	Disponible	No	Sí	Disponible
Sincronización /Disparo	Bueno	Mejor	Mejor	Excelente
Sistemas Operativos	Windows, Linux, Mac OS X	Windows, Linux, Mac OS X, Real-Time	Windows	Windows, Linux, Real-Time

Figura 4. Tarjetas de adquisición de datos del grupo DAQ USB

Las tarjetas de adquisición de datos de la familia DAQ USB cuentan con entradas y salidas analógicas y digitales las cuales se conectan por medio de terminales de tornillo, BNC o terminación masiva. Estos sistemas de adquisición se caracterizan por ser altamente portables, sencillos de usar y de alto rendimiento. Algunas referencias de estos dispositivos se muestra en la figura 4.

Para este tipo de tarjetas National Instruments brinda un software llamado $NI - DAQmx$ o $NI - DAQmx\ Base$ el cual además de funcionar como controlador proporciona una interfaz intuitiva para programar el hardware, herramientas de configuración del hardware, asistentes para la selección de entradas o salidas y brinda soporte para controlar el hardware con software de programación diferente a LabVIEW.

2. DISEÑO METODOLÓGICO

Existen numerosas herramientas, lenguajes y metodologías de programación cada una con sus ventajas y facilidades, pero también con sus requerimientos y dificultades al momento de emplearlos. A continuación se describirán las estrategias y metodologías que se emplearon en el diseño y desarrollo del software que es el objeto principal de este proyecto.

2.1. PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos es una metodología de diseño de software y un estilo o paradigma de programación que permite el desarrollo de grandes programas mediante la unión de elementos más simples que se diseñan de forma independiente y que se asemejan a los objetos que existen físicamente [28]. Esto permite programar de forma modular facilitando la ampliación y mantenimiento del software, así como reutilizar estos módulos en distintos programas.

Este paradigma de programación es soportada por numerosos lenguajes incluyendo C++ y Java, y ha sido adaptada a entornos de programación gráfica como LabVIEW.

En esta metodología una clase es una plantilla de objetos, es decir la definición de propiedades y métodos que no pueden tomar ningún valor ni ser modificados de ninguna forma ya que son solo una abstracción[29]. Por otra parte un objeto es una instancia de una clase, es decir la creación de una entidad con la estructura de una clase definida. Es la forma de "materializar" las propiedades y métodos descritos en una clase.

Cada clase está compuesta por propiedades y métodos. Las propiedades son datos internos del objeto que definen sus características y su estado mientras que los métodos son las funciones o acciones que puede realizar el objeto las cuales definen su comportamiento.

Las propiedades y los métodos pueden tener diversos niveles de visibilidad: privados, protegidos o públicos. Estos niveles expresan desde donde se pueden modificar o acceder, o qué método o parte del programa los puede emplear.

Además en la programación orientada a objetos se han definido tres propiedades principales de los objetos[30]:

- ⋆ El encapsulamiento es la propiedad de ver al objeto como un "todo", lo que implica que los datos o métodos pueden estar "ocultos" es decir que no son accesibles sino por medio de un método destinado para tal fin.
- ⋆ La herencia es la propiedad de una clase para especializar otra, es decir una clase puede estar basada en otra más general, dando lugar a relaciones de tipo jerárquico. La clase hija guarda la relación "es un" con la clase padre.
- ⋆ El polimorfismo es la habilidad de asociar el mismo nombre a funciones diferentes, pero dependiendo de los parámetros o datos con que sea invocado, se decide cuál es la función que se requiere.

---

[28] GARCÍA, Javier; RODRIGUEZ, José; SARIEGUI, José y BRAZÁLEZ, Alfonso. Aprenda C++ como si estuviera en primero. Universidad de Navarra. San Sebastian. Abril, 1998. 83 p.

[29] BITTER, Rick; MOHIUDDIN, Taqi y NAWROCKI, Matt. *Object-Oriented Programming in LabVIEW*, En: LabVIEW Advanced Programming Techinques.: CRC Press LLC, 2001.

[30] LAJARA, José y PELEGRÍ, José. LABVIEW. Entorno gráfico de programación. Marcombo S.A., Barcelona 2007. 374 p.

Por otra parte se han dado también unas pautas para el desarrollo de software, clases, propiedades y métodos de forma efectiva con POO. Una de las pautas que se ha considerado en el desarrollo de este proyecto son los principios SOLID.

Los principios S.O.L.I.D. fueron reunidos por Robert Martin en 1995 y buscan guiar al desarrollador en la gestión de dependencias al desarrollar programación orientada a objetos, con el fin de lograr un sistema que sea más fácil de mantener y ampliar en el tiempo, y se enuncian continuación[31]:

- ⋆ Principio de Única Responsabilidad (Single responsibility principle): una clase así como todos sus métodos deben tener una única responsabilidad.
- ⋆ Principio Abierto/Cerrado: las entidades (clases o métodos) deben estar abiertas a la extensión pero cerradas a la modificación.
- ⋆ Principio de sustitución de Liskov: Los objetos de un programa deben poder ser reemplazados por instancias de sus subtipos sin alterar el correcto funcionamiento del programa, es decir que un método debe funcionar de forma adecuada con un objeto de su misma clase (T) y con un objeto de una clase hija que herede de T.
- ⋆ Principio de Segregación de la Interface (Interface segregation principle): Este principio propone que que no se obligue al usuario a depender de interfaces o clases que no sean indispensables.
- ⋆ Principio de Inversión de Dependencia (Dependency inversion principle): Enuncia que una clase no debe depender directamente de otra sino de su abstracción.

2.2. REPRESENTACIÓN DE SOFTWARE EMPLEANDO UML

El Lenguaje de Modelado Unificado (UML) es una notación utilizada para modelar y diseñar sistemas de software antes de su construcción, ya que permite visualizar, construir, detallar y plasmar una idea para facilitar el desarrollo de software como lo presentan Fowler y Scott[32].

Empleando esta notación se pueden desarrollar diferentes tipos de diagramas. Inicialmente es conveniente realizar diagramas de casos de uso como el de la figura 5 para describir actividades, objetivos y requierimientos del software. Estos diagramas también pretenden describir el comportamiento del sistema frente a las acciones de los actores externos y las funciones internas que debe desempeñar. Este diagrama es muy general por cuanto presenta qué hace el sistema pero no especifica cómo lo hace.

Estos diagramas muestran las interacciones entre el usuario y el sistema. En este sentido ilustra por una parte los actores externos que influyen en el funcionamiento del software, y por otra parte las acciones que realiza cada actor o recaen sobre éste, y que corresponden a las tareas o actividades que el sistema debe desempeñar. El esquema general del un diagrama de casos de uso se muestra en la figura 5.

Otro tipo de diagrama empleado para la planificación del software es el de estados, en el cual se describe el sistema como un proceso, que como tal tiene un estado de inicio, otro de fin, y numerosos estados intermedios conectados por transiciones los cuales siguen un orden o una secuencia dependiendo de las opciones de decisión que tiene el usuario para elegir.

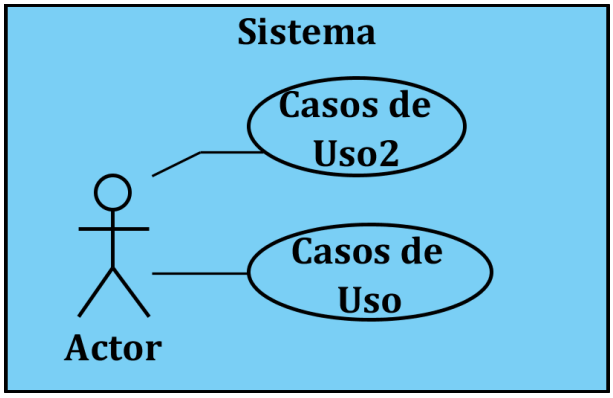
Sin embargo este diagrama no sólo se aplica para el sistema global, sino que puede ser usado para describir el comportamiento de un objeto particular.

---

[31] BLÉ, Carlos, y colaboradores. *Diseño ágil con TDD*. SafeCreative. 2010. 308 p.

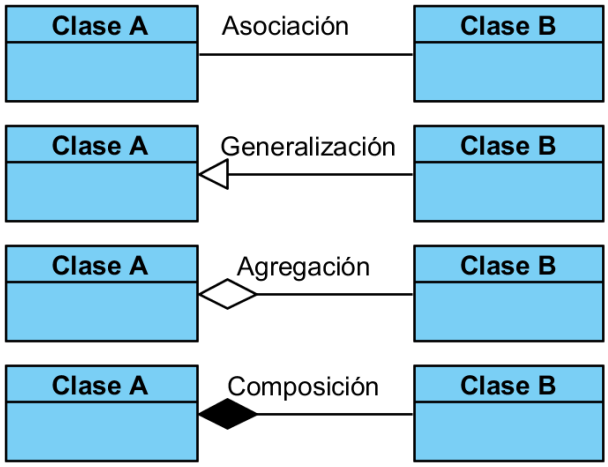
[32] FOWLER, Martin y SCOTT, Kendall. UML gota a gota. Traducido por Jaime González. Editorial Pearson. México. 1999. 224 p.

Figura 5. Esquema de diagrama de casos de uso



UML también tiene notación para el desarrollo de diagramas mucho más detallados como el diagrama de estructura estática de clases, en el cual se definen las clases que se implementarán en el software, sus atributos y acciones y las distintas relaciones que tendrán entre sí.

Figura 6. Tipos de relaciones entre clases



Este diagrama es quizás el que esta mas asociado con la programación orientada a objetos y tiene tres términos principales que establecen los tipos de relaciones entre clases[33] y su representación se muestra en la figura 6:

- Asociación: Son relaciones entre instancias de clases.
- Generalización: Hace referencia a la herencia es decir la relación entre una clases principal y sus subtipos
- Agregación y composición: Es la relación de componente es decir de un todo con sus partes.

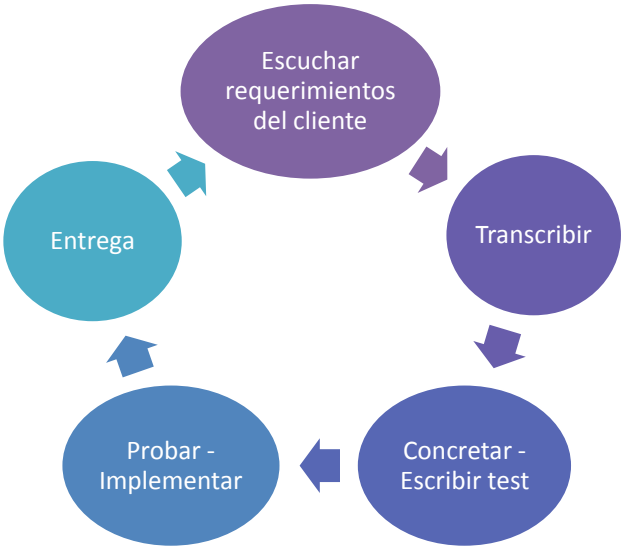
No obstante hoy en día no se suele diseñar software por medio de UML. En lugar de esto se emplea como una herramienta para representar el funcionamiento del software o su arquitectura. Ésto se debe a que el diseño con UML es costoso de mantener ya que con cada cambio de requerimientos o especificaciones se deben modificar los diagramas y con ellos el código.

[33]FOWLER, Martin y SCOTT, Kendall. UML gota a gota. Traducido por Jaime González. Editorial Pearson. México. 1999. 224 p.

2.3. DESARROLLO GUIADO POR PRUEBAS (TDD)

El proceso más empleado incluso en la actualidad para el desarrollo de software recibe el nombre de cascada y consiste en recibir los requerimientos del cliente, abstraerlos, modelarlos o realizar la planificación, programar o implementar el código, realizar algunas pruebas y entregar. A pesar de su uso masivo se ha encontrado que no es muy efectivo ni funcional ya que es muy común que el usuario del software detecte fallas, encuentre que el sistema no hace lo que él pidió o peor aún, que encuentre que nadie más le puede hacer mantenimiento ni modificaciones.

Figura 7. Proceso del desarrollo guiado por pruebas



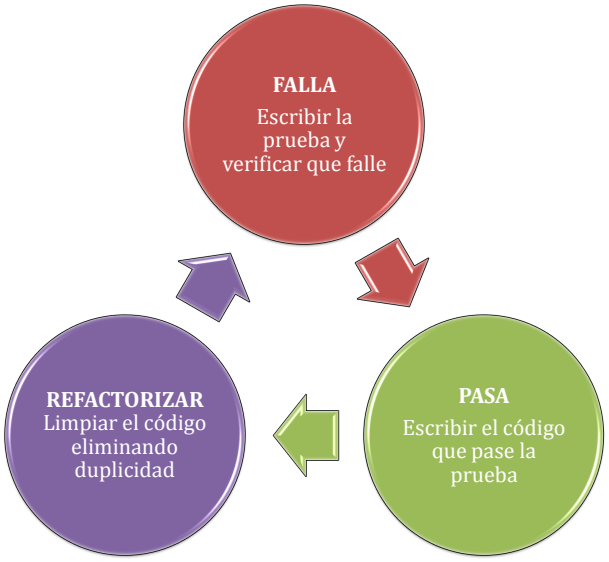
En la última década se han plantado diferentes metodologías de desarrollo de software basadas del manifiesto ágil publicado tras una reunión de especialistas en el tema que se llevo a cabo en el 2001[34]. Éste determina una nueva forma de entender el desarrollar software con otras prioridades y otras estrategias al tiempo que plantea un conjunto de mejores prácticas para combatir los problemas mencionados. La secuencia que propone para el desarrollo de software ágil se muestra en la figura 7.

El agilismo en las etapas de concretar y probar e implementar software hace uso de Desarrollo Orientado por Pruebas (TDD) y Desarrollo Dirigido por Pruebas de Aceptación (ATDD) como técnicas de diseño e implementación de software. Mediante estas técnicas cada requerimiento del cliente se convierte en un conjunto de pruebas y más que eso en un grupo de ejemplos de cómo se espera que funcione el software en diferentes situaciones y de los posibles casos que deben generar fallos.

El ciclo de desarrollo con TDD que se ilustra en la figura 8 comienza con la implementación de una prueba que verifica un requisito del software, seguidamente se comprueba que la prueba falle (Rojo), ya que no hay código que la satisfaga y luego se programa la cantidad mínima de código que satisfaga el test (Verde), y finalmente se limpia el código eliminando duplicidad y haciendo mejoras oportunas si es necesario (Refactorizar). Así continúa el proceso hasta que se implementen todas las especificaciones requeridas, lo que implica que el sistema hace exclusivamente lo que solicitó el cliente, y que funcione de forma adecuada. A medida que avanza el desarrollo se obtiene un grupo de pruebas que permiten después de cada modificación comprobar que no se vea afectada ninguna funcionalidad.

[34]BLÉ, Carlos, y colaboradores. *Diseño ágil con TDD*. SafeCreative. 2010. 308 p.

Figura 8. Ciclo del desarrollo guiado por pruebas



Según el lenguaje sobre el cual se esté programando es posible que estas pruebas se realicen y verifiquen de forma automática con algún software adicional, para el caso del LabVIEWse puede emplear el complemento VITester.

Es así como ATDD permite concretar lo que quiere el cliente y TDD como lo hace posible el desarrollador, y combinadas según Blé[35] brindan gran cantidad de ventajas como:

- ⋆ Se implementa exclusivamente lo necesario.
- ⋆ La calidad del software se incrementa y el código es más reutilizable.
- ⋆ Reducción de fallas.
- ⋆ Tener un programa completo en continua producción.
- ⋆ Resulta más sencillo mantener el código y/o realizarle cambios.
- ⋆ Se tienen revisiones periódicas por parte del cliente lo que permite cumplir los requisitos que este plantea.

La figura 9 muestra varias clases de pruebas que se pueden implementar TDD en el desarrollo de software. Las pruebas de aceptación comprueban que se cumplen los requisitos de negocio. Éstas se dividen en funcionales o no funcionales, las primeros prueban que se cumplan los requisitos del software mientras que las no funcionales pueden evaluar aspectos de rendimiento que van mas alla del funcionamiento adecuado.

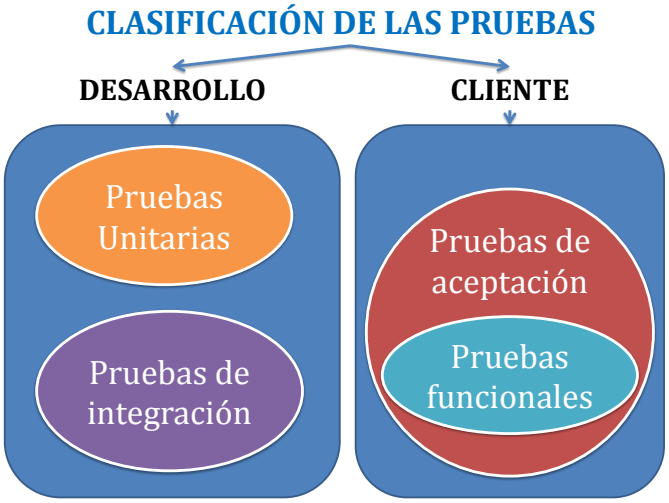
Las pruebas unitarias son las unidades básicas ya que prueban la mínima funcionalidad posible. Éstas se caracterizan por ser atómicas, inocuas o repetibles, rápidas e independientes. Por su parte las pruebas de integración rompen con varios principios de las unitarias principalmente porque prueban funcionalidades más amplias.

2.4. LABVIEW

LabVIEW (por sus siglas Laboratory Virtual Instrument Engineering Workbench) es un software corporativo desarrollado por National Instruments, la empresa que en 1986 lanzó la versión

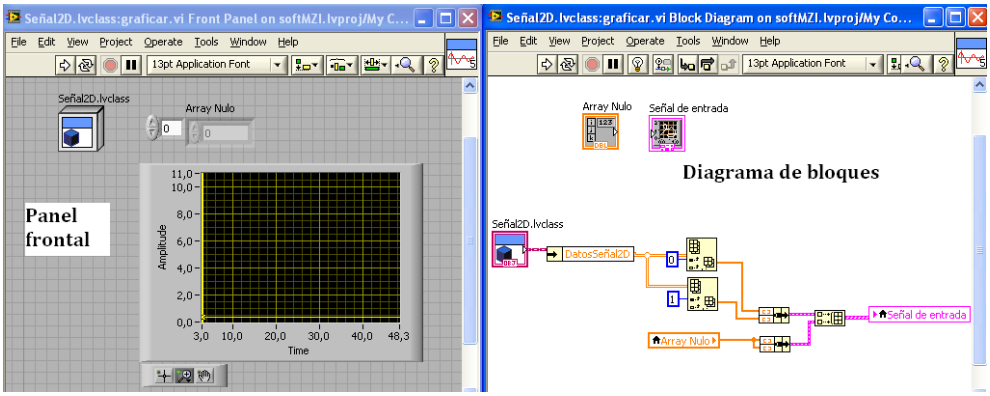
[35]BLÉ, Carlos, y colaboradores. *Diseño ágil con TDD*. SafeCreative. 2010. 308 p.

Figura 9. Clasificación de las pruebas



1.0 de este nuevo lenguaje y entorno de programación gráfica, en principio para ordenadores Macintosh, y en el año de 1992 para sistema operativo Windows[36]. Una aplicación desarrollada en LabVIEW se conoce como Instrumento Virtual (VI), el cual consta de una ventana con un panel frontal, que es semejante al panel de un instrumento de medida físico ya que esta conformado por controles e indicadores, y otra ventana que contiene el diagrama en bloques del código fuente[37].

Figura 10. Entorno de desarrollo de Labview



Por ser gráfico LabVIEW ofrece una gran facilidad y practicidad en la programación, así como gran potencia en la visualización de señales, LabVIEW se destaca por permitir simular un instrumento físico por cuanto genera y adquiere señales del exterior, pero además cuenta con librerías de funciones y herramientas para hacer procesamiento, almacenamiento y análisis de la información. También cuenta con librerías para realizar interfaces de comunicación con los puertos de salida del computador y/o con tarjetas de adquisición de datos propias de la misma compañía o de otros fabricantes.

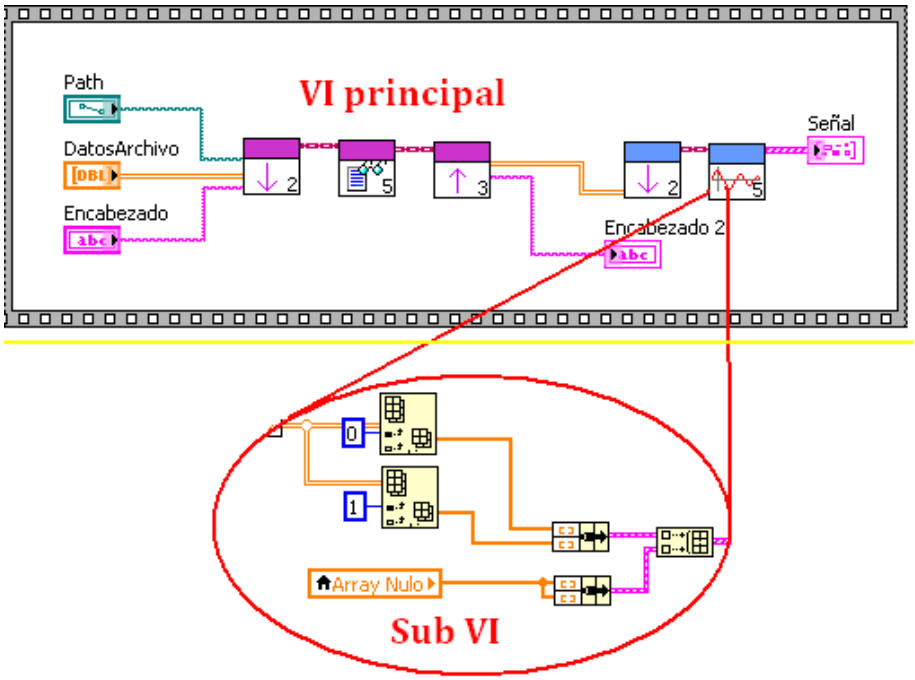
El entorno de desarrollo de LabVIEW se compone de dos ventanas principales que se evidencian

[36]NATIONAL INSTRUMENTS, [Citado en 15 de noviembre de 2011] Disponible en internet: http://www.ni.com/LabVIEW/whatis/esa/

[37]LÁZARO, Antoni; DEL RÍO FRENÁNDEZ, Joaquín. LabVIEW: Programación gráfica para el control de instrumentación. Editorial Paraninfo, Madrid España- 1997, ISBN: 84-283-2339-9.

en el figura 10: el *Diagrama de bloques*, el cual contiene el código donde las funciones son bloques con entradas y salidas y las variables son controles (entradas) indicadores (salidas) que tienen su elemento asociado en el *Panel frontal*, es cual corresponde a la interfaz de usuario.

Figura 11. Ejemplo de SubVI



La programación en LabVIEW se caracteriza por ser modular, lo cual permite que al interior de un VI se pueda llamar y emplear otro VI, que para el caso, recibe el nombre de subVI como se observa en el ejemplo de la figura 11. Cada VI es independiente y al emplearlo al interior del programa principal o de otro subprograma se presenta como un bloque, al cual se le puede diseñar un icono y configurar entradas y salidas[38].

LabVIEW cuenta además con una gran cantidad de funciones entre las cuales se pueden destacar las funciones para tratamiento de arreglos de datos (crear, concatenar, indexar, seleccionar, determinar máximos y mínimos, etc.) la generación y procesamiento de señales (interpolación, normalización, detección de cruces por cero), y las opciones de manejo de archivos (crear, abrir, cerrar, borrar, etc).

Un proyecto está compuesto por VIs. A partir de un proyecto es posible crear el ejecutable y el instalador del aplicativo desarrollado. El ejecutable sólo podrá funcionar en un computador que tenga instalado el *Run Time Engine* de LabVIEW, mientras que el instalador incluye algunas herramientas livianas (entre ellas el *Run Time Engine*) que permiten instalar y ejecutar el aplicativo aún en un computador que no tenga instalado el software LabVIEW.

2.4.1. Programación orientada a objetos en LabVIEW. LabVIEW no es propiamente un lenguaje orientado a objetos, sin embargo cuenta con una representación de clases y objetos que puede ser empleada en el desarrollo de aplicaciones en LabVIEW y que permiten implementar algunos de los beneficios de la orientacion a objetos.

[38]KEHTARNAVAZ, Nasser y KIM, Namjin. Digital signal processing system-level design: using LabVIEW. Newness. 2005. 305 p.

Como se mencionó anteriormente la clase es la plantilla y los objetos son la instancia de la clase.

En LabVIEW las propiedades de la clase se definen en un cuadro parecido a un cluster. En ese sentido la clase se define como una estructura que contiene variables de diferentes tipos similar a un estructura en lenguaje C/C++. Sin embargo para cumplir con la propiedad de encapsulamiento que consiste en ocultar los datos previniendo la manipulación de estos por agentes externos, en LabVIEW los datos de una clase estan declarados por defecto como privados y no se puede modificar el parámetro de visibilidad, es decir que solo pueden ser leídos o modificados por los métodos de la propia clase.

Por el contrario los métodos si pueden ser declarados como privados, públicos o protegidos con lo cual se gestionan los permisos de uso de los mismos.

El tipo de programación Lvoop (Labiew object Oriented Programming) brinda numerosos beneficios dentro de los que se puede resaltar:

- Escalabilidad: Al tener código modular y con funciones bien definidas es más sencillo gestionar el crecimiento del software.
- Mantenimiento: como los objetos se crean a partir de una clase, al realizar alguna modificación en esta, la nueva estructura se actualiza automáticamente en todos los objetos de ese tipo.
- Reutilización de código: Al definir una clase se pueden crear todos los objetos de ese tipo que se requieran. Así mismo poder crear clases basadas en otras (herencia) es otra forma de reutilizar el código. Finalmente si las clases también pueden reutilizarse en diferentes aplicativos.
- Desarrollo colaborativo: Por ser cada clase independiente el código es modular y facilita la intervención de varios programadores en el desarrollo del aplicativo.

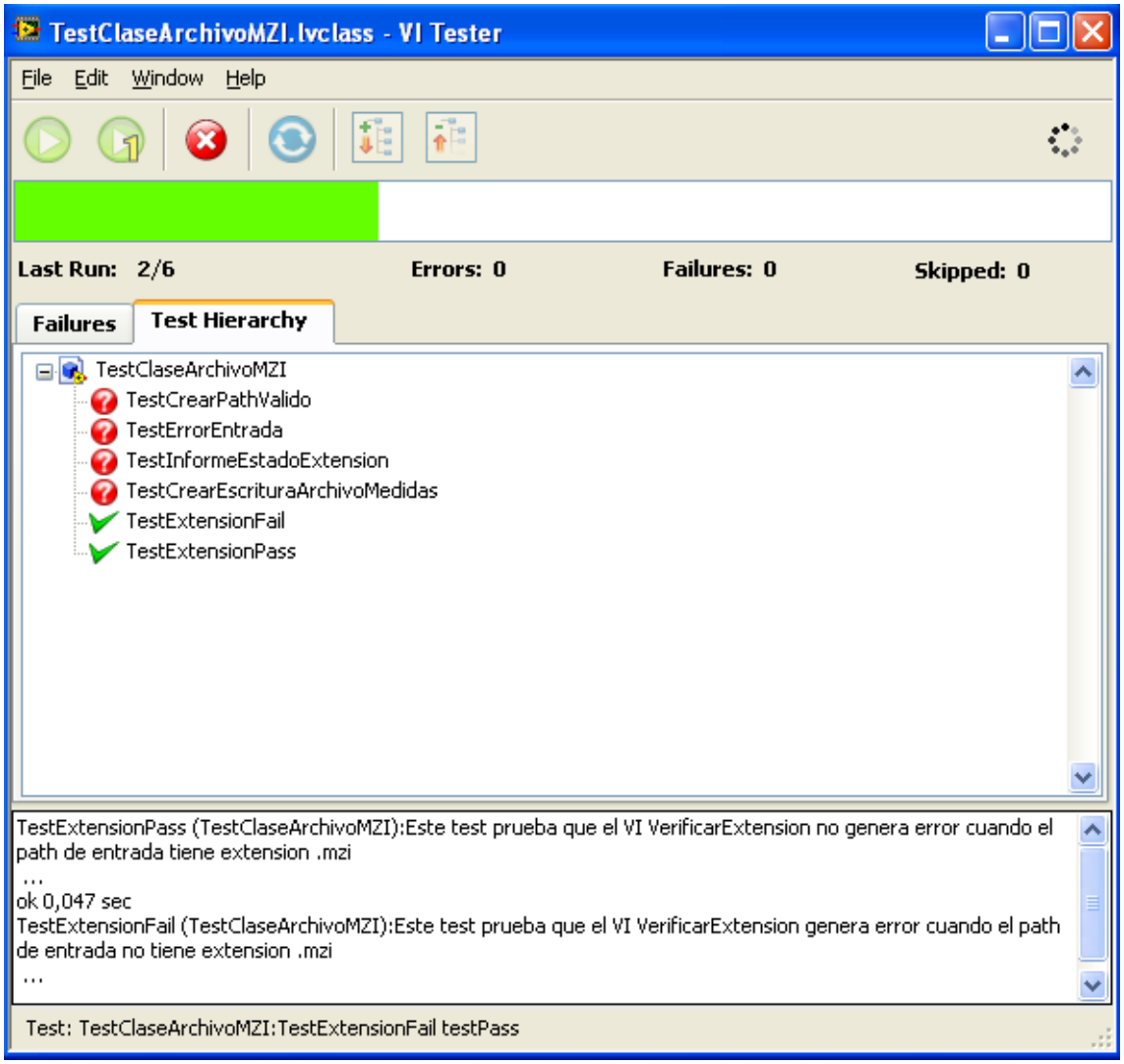
A pesar de los beneficios mencionados implementar Lvoop no siempre es la mejor opción para cualquier tipo de aplicativo. Estas ventajas se aprovechan cuando el software a desarrollar es de gran tamaño, implica un tiempo considerable de programación y debe ser mantenido por mucho tiempo y en los proyectos de trabajo colaborativo que implique un grupo de programadores desarrollando el código.

Tabla 2. Diferencias entre OOP el C++ y LVOOP

C++	**LabVIEW**
Lenguaje basado en línea de comandos (texto)	Lenguaje gráfico basado en el flujo de datos
Tiene constructores	No necesita constructores
Tiene destructores	No necesita destructores
Permite el paso de objetos por valor o por referencia	Solo permite el paso de objetos por valor
Tiene plantillas	No tiene plantillas
Permite herencia múltiple	No permite herencia múltiple

2.4.2. TDD en LabVIEW. LabVIEW cuenta con la Red de Herraientas de LabVIEW que corresponde a un conjunto de herramientas y complementos desarrollados por terceros pero con compatibilidad con LabVIEW avalada por National Instruments. Una de estas herramientas es el VI Package Manager desarrollado por la empresa JKI, brinda una lista de los complementos disponibles según la versión de LabVIEW disponible y facilita su descarga e instalación.

Figura 12. Interfaz de usuario del VI Tester



Esta empresa en su línea *JKILabs* distribuye una estructura de pruebas llamada *VITester* para implementar TDD en la programación en LabVIEW. Esta herramienta facilita la programación de pruebas en LabVIEW y la ejecución automática de las mismas logrando probar el aplicativo en cuestión de segundos.

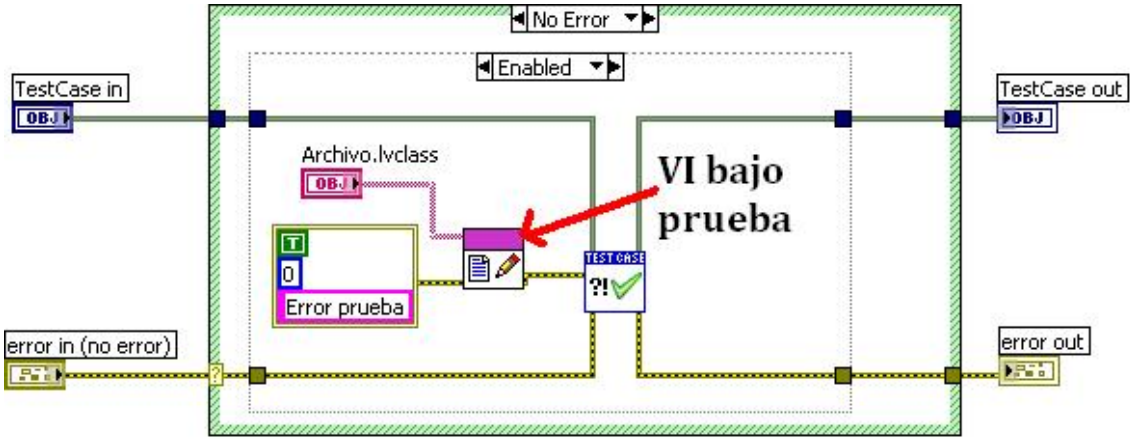
Este software fue desarrollado con la arquitectura de test xUnit basado en el original SUnit, y fue seleccionado para este proyecto ya que su distribución es libre. Esta estructura permite la creación de *TestCase* que son conjuntos de test agrupados en una clase.

Este framework esta conformado por 3 partes:

1. Test: es un VI que se escribe para probar uno o varios VIs.
2. Interfaz de Usuario para ejecutar los test que se muestra en la figura 13.
3. Framework de test basado en programación orientada a objetos.

Empleando este complemento de LabVIEW resulta rápido y sencillo probar el funcionamiento de un aplicativo desarrollado en labview, incluso es posible evidenciar su comportamiento con

Figura 13. Ejemplo de prueba



condiciones de error o falla.

2.5. SISTEMA DE CONTROL DE VERSIONES

Un sistema de control de versiones (SCV) permite almacenar archivos así como registrar y gestionar los cambios realizados por medio de un registro histórico de los mismos. Esto hace posible regresar a versiones anteriores de los archivos y guardar copias de seguridad de los datos.

Esta herramienta también facilita realizar trabajo colaborativo ya que el repositorio global se guarda en un servidor principal, del cual los usuarios pueden en diferentes computadores conectados a este servidor o incluso en el mismo generar copias o clones del repositorio. De esta forma es posible realizar modificaciones en el repositorio local sin afectar directamente el global y en cierta medida aislar los cambios realizados por cada usuario pero luego sincronizar para unir todas las modificaciones.

Los sistemas de control de versiones se clasifican según la forma como almacenan los datos en centralizados y distribuidos, las diferencias entre las dos clases se ilustran en el cuadro comparativo de la figura 14.

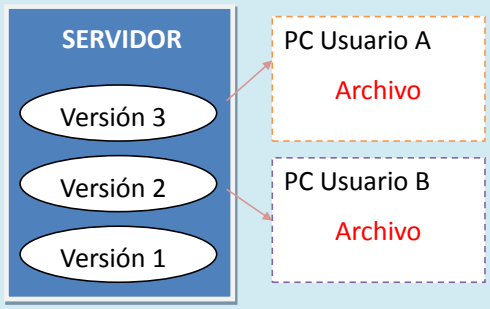
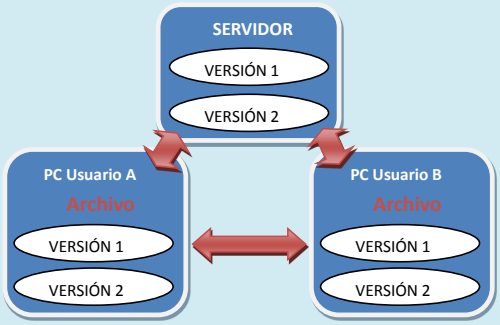
2.5.1. Características del Sistema de control de versiones GIT. El sistema de control de versiones GIT fue desarrollado en el 2005 por Linus Torvalds para la gestión del kernel de Linux luego de que tuviera inconvenientes con la compañía desarrolladora de BitKeeper, el SCV empleado por él hasta ese momento.

El SCV Git se caracteriza por ser rápido, eficiente con proyectos grandes y contar con un sistema de ramificaciones apropiado para el desarrollo no lineal como lo manifiesta Chacón[39]. GIT a diferencia de otros sistemas de control de versiones trata los archivos del repositorio como un mini sistema de archivos así que al guardar una versión nueva almacena copias en ese instante de todos los archivos en lugar de guardar solo las modificaciones realizadas (modo de funcionamiento de sistemas como Subversion).

Otra de las ventajas de usar GIT es que localmente existe una copia de todo el repositorio en una base de datos lo que hace que el acceso sea casi inmediato y brinda la posibilidad de hacer

[39]CHACON, Scott Pro Git, 2009, USA. 275 p.

Figura 14. Diferencias entre el control de versiones centralizado y el distribuido

Sistema de control de versiones centralizado	Sistema de control de versiones distribuido
Los cambios se realizan directamente en el servidor.	Las modificaciones se realizan en el computador local y luego se sincronizan con el servidor
Todo el repositorio se encuentra guardado en el servidor, así que si éste falla no es posible realizar modificaciones y si se daña, es muy probable que se dañe o pierda toda la información.	Al abrir los archivos en un PC local se replica todo el repositorio así que se puede trabajar localmente aun si el servidor no está disponible, y si se dañan los datos en él existe una copia local para recuperarlos.
Son ejemplos de este sistema de control: CVS, Subversion, y Perforce	Son ejemplos de este sistema de control: Git, Mercurial, Bazaar o Darcs.
	

modificaciones asi el servidor principal este fuera de línea.

Por otra parte, la mayoría de las operaciones con el GIT agregan información esto con el fin de no emplear procedimientos de borrado que puedan dañar el repositorio. Adicionalmente en la transmisión de datos realiza suma de verificación (checksum) para evitar errores.

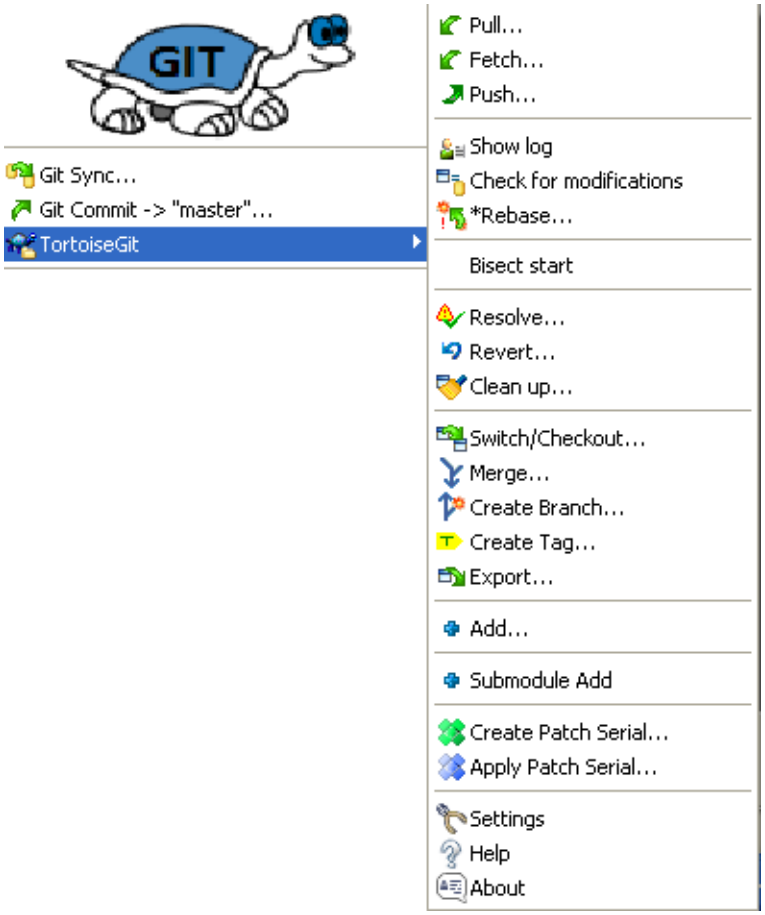
Instalando los archivos fuentes del GIT es posible usarlo por medio de comandos en modo consola o con una interfaz gráfica, sin embargo en este caso se instaló el cliente llamado TortoiseGit el cual integra las opciones del GIT en el explorador de Windows facilitando así su uso al seleccionar la opción requerida directamente sobre el directorio o archivo deseado como lo ilustra la figura 15. Este software es de libre distribución, tiene licencia GNU y pede descargarse de [40]

2.5.2. Gestión de un repositorio empleando GIT. Es posible crear un repositorio a partir de una carpeta que contenga los archivos que van a componer el repositorio. Seguidamente agregar los archivos y realizar los cambios. Sin embargo en el caso de necesitar un repositorio compartido que facilite el trabajo colaborativo es mejor crear un repositior principal llamado *Bare*, el cual contiene la información del sistema de control de veriosnes pero no los archivos de trabajo.

Para crear un nuevo repositorio *Bare* debe seleccionarse un directorio vacío y emplear la opción *Git create repository here*, y seleccionar la opción *Make it Bare*. Allí se almacenarán los archivos

[40] http://code.google.com/p/tortoisegit/downloads/list.

Figura 15. Menú del TortoiseGIT



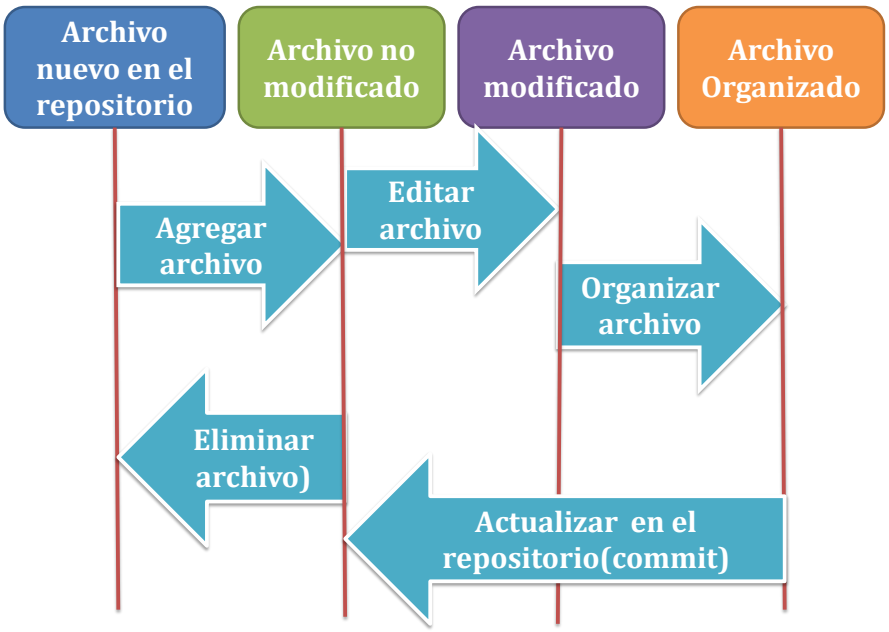
internos propios del Git, sin embargo para agregar o tener acceso a los archivos se debe crear un clon de este repositorio.

Para añadir archivos al repositorio deben copiarse en la carpeta del clon, y por primera vez ser agregados mediante el comando *add*, luego se pueden realizar y guardar modificaciones, sin embargo estas sólo se actualizan en el repositorio principal cuando se realiza una confirmación de cambios mediante el comando *commit* seguido del envío al repositorio principal con el comando *push*. Al realizar una confirmación de cambios es necesario agregar un comentario alusivo o descriptivo de las modificaciones realizadas, lo cual es una buena herramienta para documentar el proceso de desarrollo y para saber a qué versión regresar al momento de necesitar un punto de restauración. En caso de tener múltiples clones y realizar trabajo colaborativo es importante antes de modificar emplear el comando *pull* para actualizar el repositorio local con el global.

El ciclo de vida de un archivo en un repositorio gestionado por GIT se muestra en la figura 16.

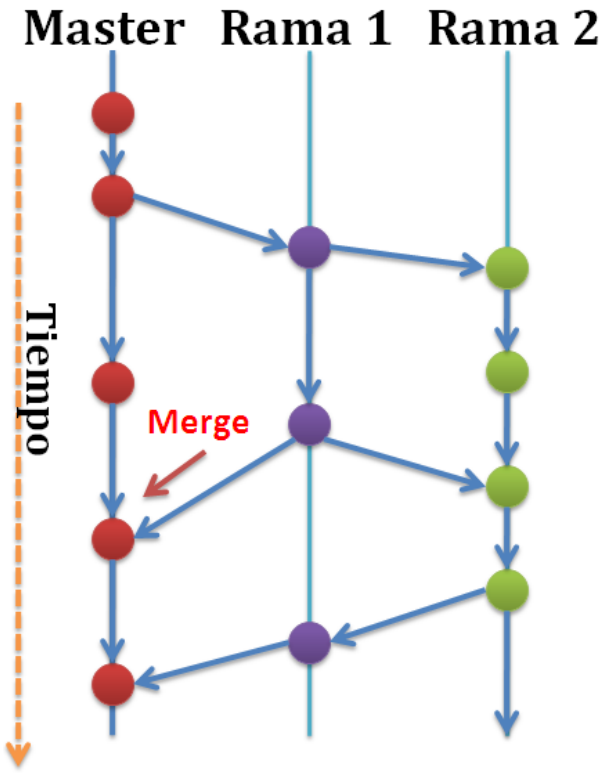
2.5.3. Ramificaciones en GIT. El SCV GIT tiene una línea de trabajo principal llamada *master* que guarda las modificaciones de forma consecutiva, sin embargo brinda la posibilidad de crear ramas para hacer desarrollo no lineal. Una rama es una derivación de la línea principal de trabajo con un apuntador móvil a la última confirmación de cambios de la rama de la cual fue creada. Esto es equivalente a crear una copia de una carpeta con todo su contenido de forma que se puede modificar sin alterar la carpeta inicial pero con la diferencia de que por la forma en que GIT opera una rama es realmente un archivo que contiene 40 datos de una suma

Figura 16. Ciclo de vida de un archivo en el GIT



de control SHA-1 lo que hace que su creación sea un proceso instantáneo[41].

Figura 17. Esquema de ramificación en GIT



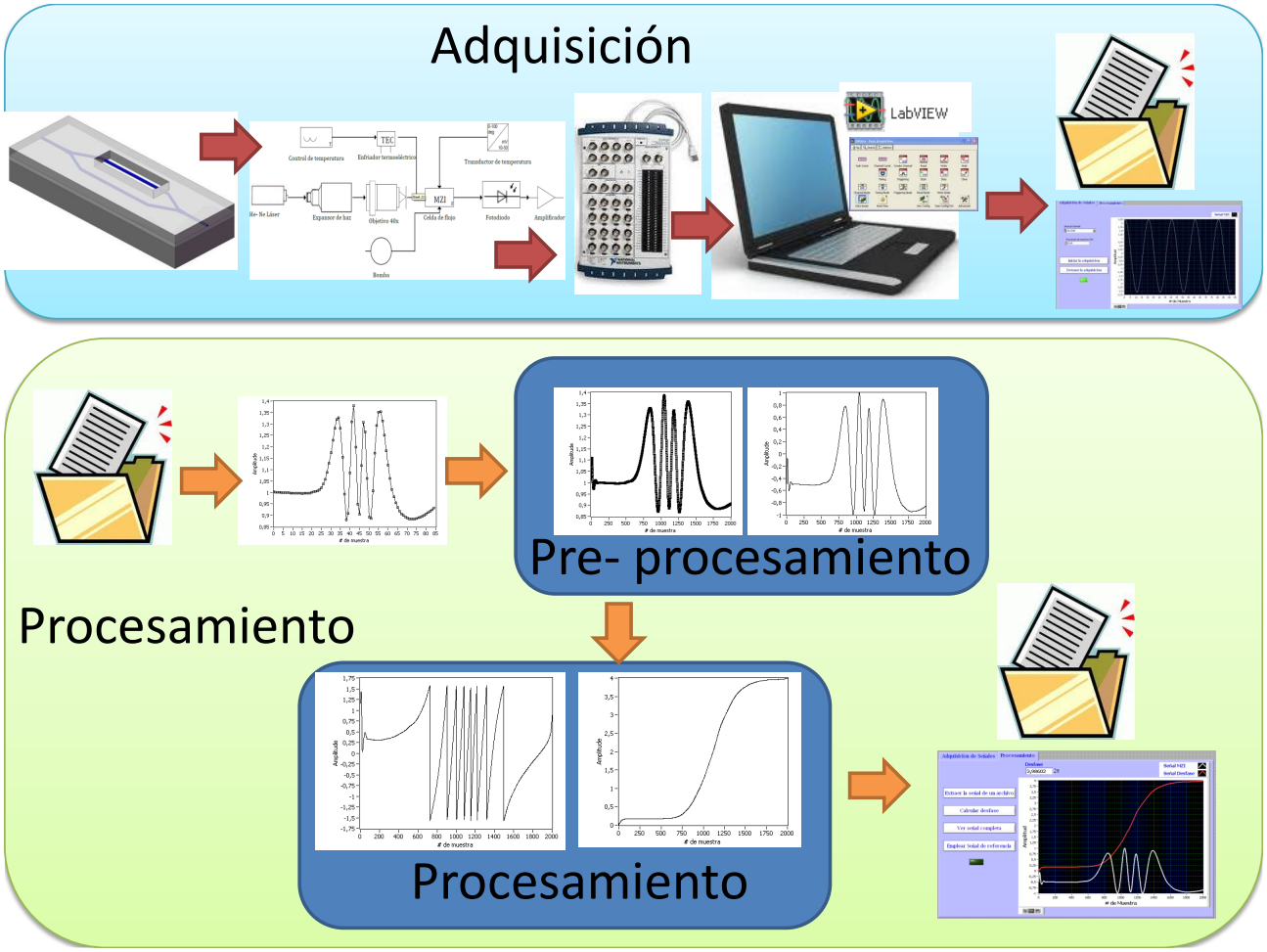
[41] CHACON, Scott Pro Git, 2009, USA. 275 p.

En GIT es muy fácil y rápido crear ramas, pasar de una rama a otra o unirlas; además se crea un registro de todas las modificaciones hechas en cada rama con lo cual gráficamente se puede evidenciar el cambio que ha tenido el repositorio. Un ejemplo de ramificación se muestra en la figura 17.

3. EJECUCIÓN DEL PROYECTO

Teniendo bases claras con respecto al funcionamiento del biosensor MZI descrito en el Marco teórico y las metodologías mencionadas en el capítulo destinado al Diseño metodológico se describirá ahora como se integró todo esto para cumplir con los objetivos del proyecto y satisfacer los requerimientos dados por el dueño del software. La figura 18 resume las etapas del sistema de adquisición y procesamiento.

Figura 18. Esquema general del software

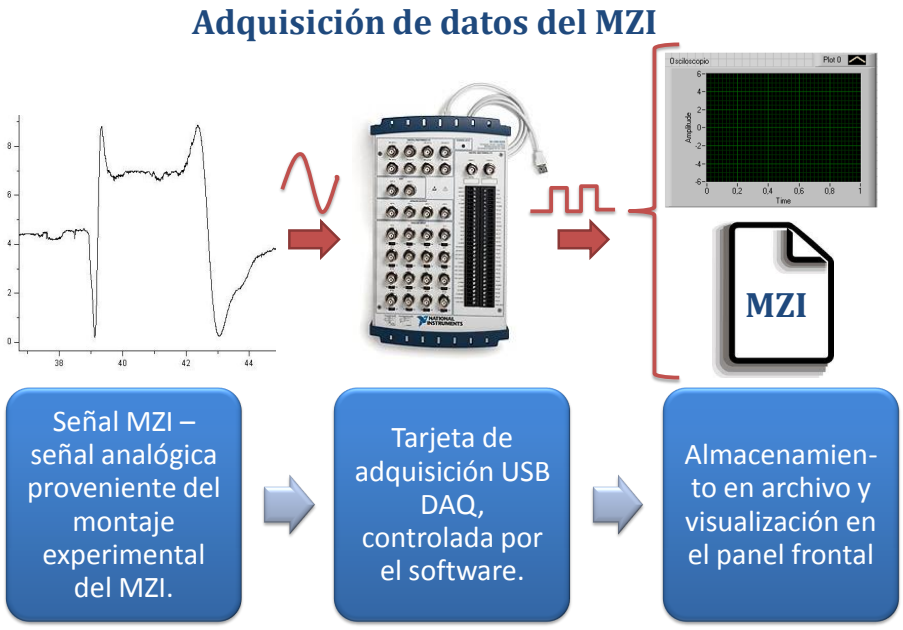


3.1. ADQUISICIÓN DE LAS SEÑALES DEL MZI

En esta etapa del software se realiza la captura de las señales a la salida del montaje experimental del MZI por medio de una tarjeta USB DAQ la cual entrega una señal digital para su visualización y almacenamiento como lo ilustra la figura 19. Los requerimientos del dueño del software en este sentido establecen que empleará un solo canal analógico de entrada con un frecuencia de adquisición a lo sumo de 50Hz.

La tarjeta de adquisición cuenta con un bloque de memoria en el cual se almacena la muestra digitalizada. Dicha memoria es leída por el PC y descargada en un buffer o sección de memoria

Figura 19. Adquisición de señales del MZI



destinada para ello a través del USB .

Desde el PC empleando software corporativo de National Instruments es posible configurar la tarjeta de adquisición. En este sentido la interfaz gráfica cuenta con controles para seleccionar el canal físico y la frecuencia de muestreo y 2 botones para iniciar y detener la medida respectivamente. Al iniciar la adquisición se sugiere un nombre para el archivo (basado en la fecha y hora del sistema) y el directorio en el cual se almacenarán las medidas, sin embargo el usuario los puede modificar. Adicionalmente en ese momento un indicador tipo led se enciende indicando que se están capturando datos. Cada dato que se captura se muestra en un cuadro de gráfico en la interfaz de usuario y se almacena directamente en el archivo en formato ASCII previniendo que ante una falla (por ejemplo un corte del suministro eléctrico o una repentina desconexión del hardware) se pierda toda la información.

Para la realización de pruebas se emplearon 2 tarjetas de adquisición particularmente, aunque el sistema es compatible en general para las tarjetas de la familia DAQ USB. Las características del hardware se describen en la tabla 3.

Tabla 3. Características de las tarjetas de adquisición empleadas

Característica	**NI USB 6008**	**NI USB 6251**
Conector	Tornillo	BNC y Tornillo
Entradas analógicas	4 diferenciales	8 diferenciales
Resolución	12 bits	16 bits
Máxima frecuencia de muestreo	10KS/s	1.25 MS/s
Salidas analógicas	2	2
Entradas y salidas digitales	12	24

El código programado en LabVIEW para la adquisición de señales se observa en las figuras 67,

68 y 69 del anexo E.

Por otra parte, aprovechando el empleo de múltiples hilos o procesos por parte de LabVIEW, de forma que ejecuta tareas de forma paralela, se estructuró el software para que la adquisición se realice en un bucle independiente al procesamiento y de esa forma se puedan realizar las dos tareas paralelamente y se incremente el rendimiento del software.

3.2. TÉCNICAS DE PRE-PROCESAMIENTO DEL LAS SEÑALES DEL MZI

El pre-procesamiento de la señal del MZI corresponde a su adecuación para su visualización, almacenamiento y posterior procesamiento. El objetivo de implementar esta etapa es enfrentar problemas tales como el ruido, la falta de datos, o la presencia de datos no coherentes con el conjunto, características que son fuentes de error.

La primera operación del preprocesamiento es normalizar lo que corresponde a escalar los valores a un rango de valores más adecuado. Para esto existen varias estrategias: empleando el mínimo y el máximo de la señal para transformar los demás valores de acuerdo a esos límites, conservando la relación entre ellos; extraer la media y la desviación estándar de conjunto de datos y emplear estas dos medidas para escalar todos los valores; o mover los puntos decimales de los valores de conjunto escalándolos a otro orden de magnitud como lo mencionan Hernandez y Rodriguez[42]. La estrategia empleada en este consiste en hallar el mínimo y máximo de la señal y con ellos el valor medio, para centrar la señal en cero y luego dividirla entre el valor máximo. De esta forma se consigue una señal entre -1 y 1. Se eligieron estos límites por la similitud de la señal con una seno pura y con el fin de poder emplear funciones trigonométricas.

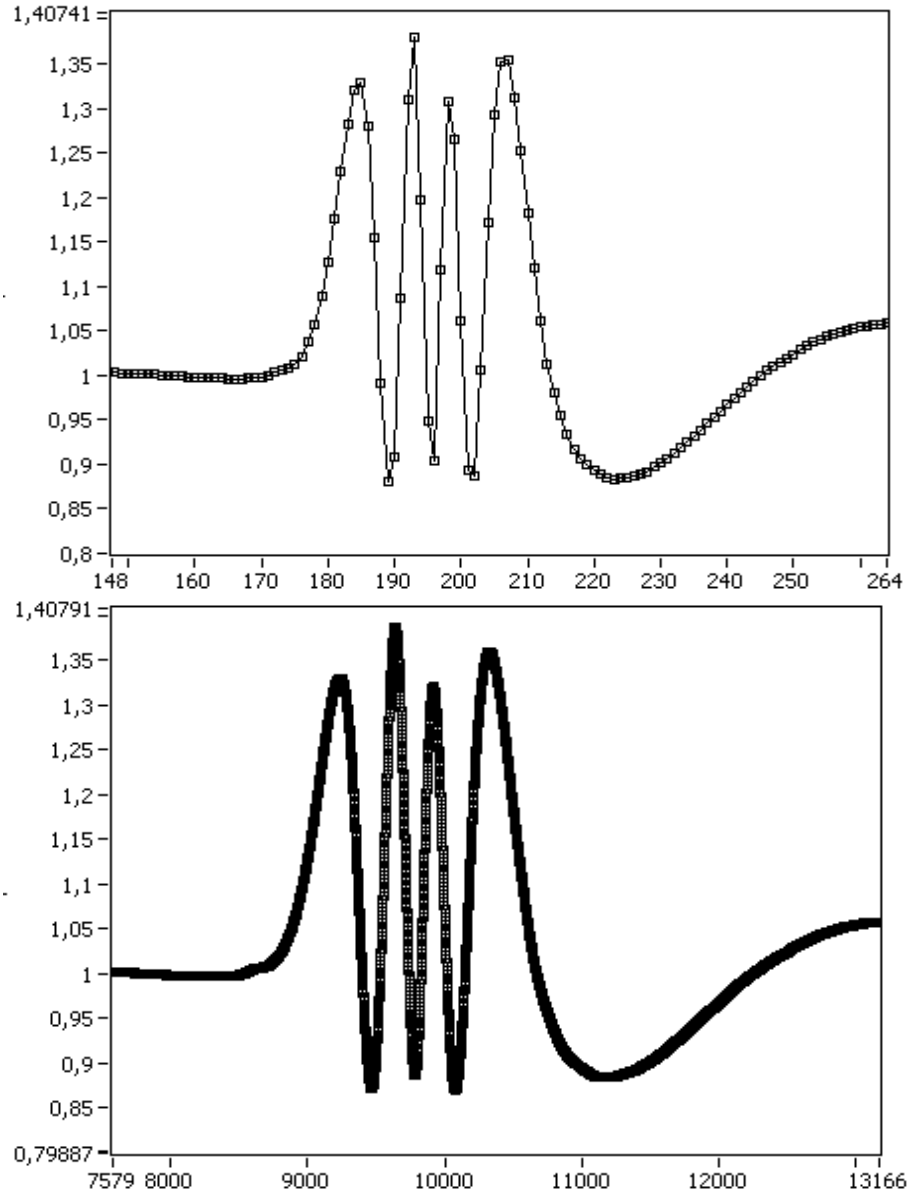
Sin embargo el software brinda la posibilidad de seleccionar una parte de la señal lo que permite que existan señales menores a un ciclo, las cuales al normalizarlas pueden generar inconsistencias por no estar el máximo y mínimo de la señal contenidos allí. Para esto se permite seleccionar una señal de referencia contra la cual normalizar, es decir que de la referencia se extraen los límites (conocidos como el factor de visibilidad de la señal MZI) y con ellos se normaliza el trozo de señal menor a un ciclo.

Por otra parte, para el tratamiento de una señal del MZI es necesario que haya sido adquirida con la resolución y la frecuencia suficientes de forma que tengan una cantidad óptima de puntos y favorezcan el desarrollo de un buen análisis. Sin embargo independiente del sistema de adquisición se implementó una etapa que determina si la señal tiene la cantidad adecuada de puntos o si es necesario realizar una interpolación o remuestreo y de ser así en que factor se debe realizar. Para esto y teniendo en cuenta que las señales no son periódicas, se normaliza y luego se halla la cantidad mínima de puntos que hay entre cruces por cero consecutivos, con este valor y la constante de 50 (cantidad de puntos considerada óptima para favorecer el procesamiento sin aumentar drásticamente la carga computacional) se halla la razón de remuestreo. A continuación se empleó una función de LabVIEW para realizar interpolación la cual cuenta con cuatro métodos:

- Coaccionar: Inserta los nuevos puntos del mismo valor que la muestra mas cercana, teniendo como resultado una señal a pasos.
- Lineal: Ubica los puntos intermedios formando una línea recta.
- Spline: Ubica los puntos formando una curva empleando para ello polinomios de bajo grado.
- Filtro FIR: Emplea una respuesta a impulso finita para la interpolación.

[42]HERNANDEZ, Claudia y RODRIGUEZ, Jorge *Preprocesamiento de datos estructurados.* En: Vinculos, Junio, 2008. vol. 4, no. 2, pp. 27-48.

Figura 20. Ejemplo de una parte de una señal MZI antes y después de la interpolación



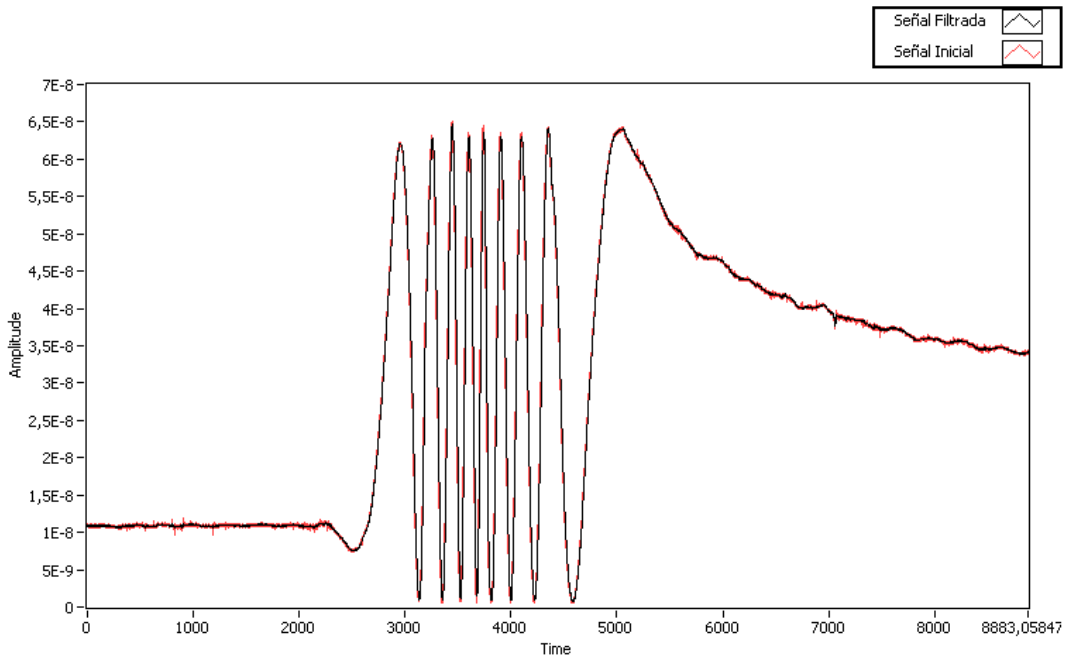
Teniendo en cuenta que las señales de este proyecto son de tipo sinusoidal se eligió la interpolación mediante el método Spline ya que conserva la tendencia de la curva y suaviza la señal. Los resultados de esta técnica se evidencian en el ejemplo de la figura 20.

Finalmente se encontró que la señal tiene ruido en las partes "planas" o no sinusoidales lo que es perjudicial para el procesamiento posterior, así que se empleó una técnica para suavizar la señal. El módulo *Advanced Signal Processing* de LabVIEW tiene una función llamada *TSA Exponential Average* la cual realiza una media exponencial de una serie de tiempo, de forma que suaviza la señal. Esta función calcula los valores promedio de la asignacion de pesos exponencialmente decrecientes de acuerdo con la ecuación:

$$X_a(i) = \alpha * X_t(i) + (1 - \alpha)X_a(i - 1) \tag{3}$$

Un ejemplo de los resultados obtenidos con esta funcion se muestran en la figura 21.

Figura 21. Reduccion del ruido suavizando la señal



3.3. DESCRIPCIÓN DEL ALGORITMO PARA EL CÁLCULO DEL DESFASE

En esta etapa se debe hallar la fase de la señal, para ello se exploraron diferentes estrategias para extraer un valor correspondiente al ángulo de cada punto.

Se han seleccionado básicamente dos estrategias para el cálculo de la fase. El primer método está basado en la *Transformada de Hilbert* y el segundo en la acumulación sucesiva de desfases calculados entre puntos consecutivos.

En físico inglés Dennis Gabor propuso un método para obtener una señal compleja única a partir de una señal real, la cual es conocida como señal analítica $z(t)$. La transformada gabor puede entenderse como un tratamiento localizado de la señal mediante filtros pasabanda deslizantes de ancho de banda constante. Por otra parte la señal $sin(wt)$ se formuló como la transformada Hilbert de $cos(wt)$, con lo que se conoce a la transformada Hilbert como un operador de desplazamiento de $+/-\pi/2$. Esta transformada según Díaz y Esteller en [43] está definida como:

$$z(t) = s(t) + i\hat{s}(t) = Ae^{i\phi(t)} \tag{4}$$

donde $\hat{s}(t)$ es la transformada Hilbert de $s(t)$, esta transformada se puede hallar de la siguiente forma:

$$\hat{s}(t) = \frac{1}{\pi} p.v. \int_{-\infty}^{+\infty} \frac{s(\tau)}{t-\tau} d\tau \tag{5}$$

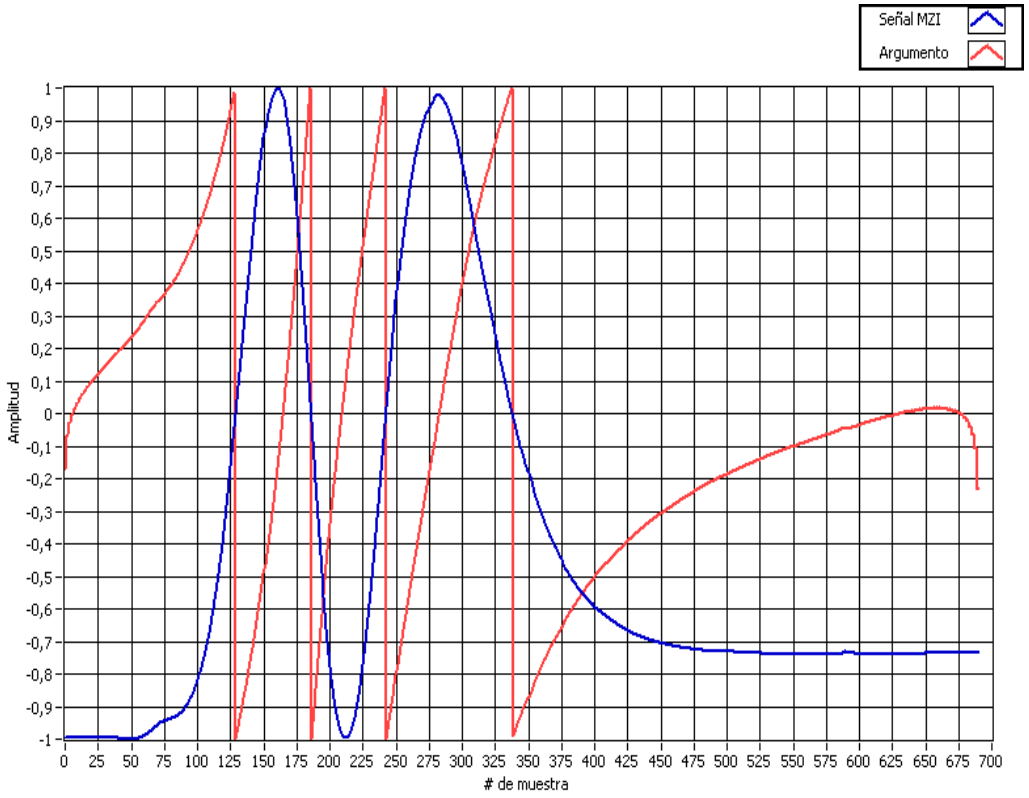
siendo $p.v.$ el valor principal de Cauchy de la integral. Con esto, la fase de la señal $\phi(t)$ se

[43] DIAZ, M y ESTELLER, R. Comparación del operador de energía no lineal y la transformada de Hilbert en la estimación de la amplitud y frecuencia instantáneas. En: IEEE Latin America Transactions, vol 5. no 1. 2007

encuentra calculando el argumento de la señal analítica de la siguiente forma [44]:

$$\phi(t) = \arctan\left(\frac{\hat{s}(t)}{s(t)}\right) \tag{6}$$

Figura 22. Fase calculada mediante método basado en la Transformada Hilbert



Al implementar este método se encontró que es muy efectivo para medir la variación de fase de ciclos completos, es decir que detecta con precisión el cambio de π entregando una señal diente de sierra como se observa en la figura 22. La discontinuidad en la señal se debe a la naturaleza de la función arcotangente la cual se indetermina en $\pi/2$ y $-\pi/2$ pero en ese rango esta definida.

Este método tambien presenta resultados erróneos cuando hay presente ruido alrededor de 0 ya que los detecta como cruces por cero. Por esto se hizo una modificación en la cual el algoritmo verifica que el trozo de señal comprendido entre los cruces por cero detectados supere unos límites y no corresponda a ruido.

Por otra parte este mecanismo no funciona bien para medir el cambio de fase de la parte inicial (antes del primer cruce por cero) y final (después del último cruce por cero) ya que como se observa da valores erróneos. Debido a esto se adoptó otro método de cálculo de desfase exclusivamente para esas partes descritas.

En este segundo método de cálculo se halla el ángulo en cada punto empleando la relación trigonométrica inversa *arcoseno* y se halla la diferencia de fase entre puntos consecutivos. Esta

[44]KRASKOV, Alexander; KREUZ, Thomas; ANDREZEJAK, Ralph; STÖGBAUER, Harald; NADLER, Walter y GRASSBERGER, Peter. Extracting Phases from Aperiodic Signals. Febrero 2008. Alemania. 3 p.

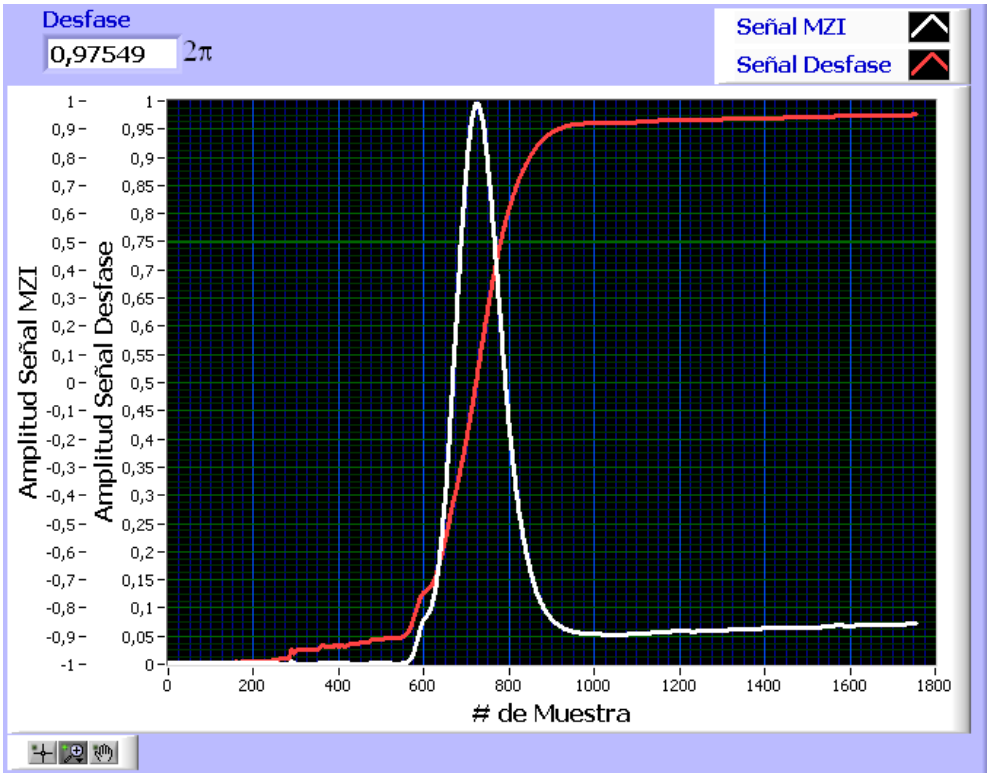
diferencia se acumula (suma o resta dependiendo de un criterio basado en la pendiente) hasta finalizar la señal.

Teniendo en cuenta las discontinuidades producto del empleo de la *Transformada de Hilbert* y que los resultados deben ser entregados gráficamente es muy importante que sea fácil visualizar el incremento continuo de la fase por esto se ajustan los resultados de los dos métodos empleados para dar lugar a la gráfica creciente.

3.4. POST-PROCESAMIENTO, VISUALIZACIÓN Y ALMACENAMIENTO DE RESULTADOS

En esta parte del proceso se ordena la presentación de los resultados y el almacenamiento de los mismos.

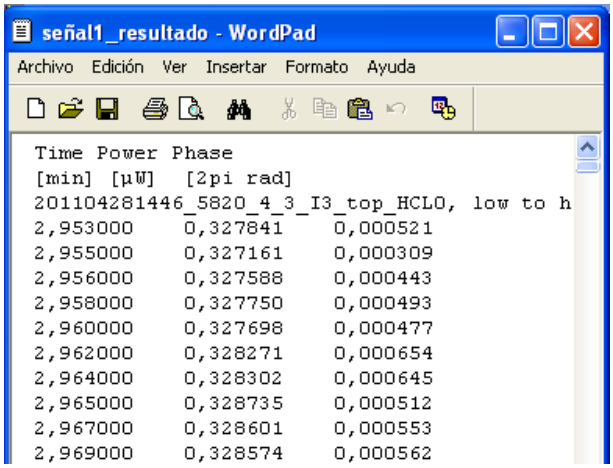
Figura 23. Visualizacion de resultados en la Interfaz de usuario



En cumplimiento de los requisitos de este proyecto, los valores del desfase punto a punto se muestran gráficamente en la interfaz de usuario del aplicativo que muestra la señal MZI y su desfase en la misma gráfia per con escalas independientes. Por otra parte el acumulado o total se muestra en un indicador numérico como se observa en la figura 23. Adicionalmente se almacenan los datos finales en un archivo cuyo nombre y directorio es sugerido por el aplicativo tomando el mismo nombre del archivo leído, agregándole la palabra *resultado* y conservando la extensión *.mzi*; sin embargo el usuario puede modificar el nombre y debe decidir si quiere o no almacenar los datos.

Como se observa en la figura 24 la información almacenada consta de dos partes el encabezado y los datos. Para el encabezado se conserva el que tenía el archivo leído pero se le agrega la tercera columna con el nombre *Phase* y la unidad de medida 2π *rad*. Los datos consisten en tres columnas con los valores de tiempo, potencia y desfase separados por un *Tab* (Código 09

Figura 24. Archivo de registro de resultados



ASCII).

Es importante que los datos del archivo puedan ser exportados a un software de análisis como Origin o Matlab por lo que se almacenan en un archivo de texto.

3.5. METODOLOGÍAS INTEGRADAS EN EL DESARROLLO DEL SOFTWARE

Para iniciar el desarrollo del software empleando TDD y las demás metodologías mencionadas se especificó un listado de requerimientos del software el cual se empleó para la planificación y construcción de las pruebas o ejemplos. Este listado de requerimientos se muestra en el Anexo A.

3.5.1. Diseño de software. En lo que respecta al diseño del software se crearon diagramas de caso de uso, de estados y de clases los cuales permitieron plasmar los requerimientos iniciales del software. Puede existir alguna contradicción al intentar emplear UML para diseñar el software y TDD para programar con el único fin de satisfacer las pruebas planteadas (lo que supone no tener un modelo a priori del software). No obstante UML se empleó con el fin de documentar el funcionamiento más que para hacer un diseño riguroso y específico antes de la programación. En este sentido los modelos se fueron desarrollando y modificando por iteraciones conforme avanzó la programación del aplicativo.

Las figuras 25 y 26 muestran los diagramas de caso de uso del software desarrollado. En éstos uno de los actores externos es el *Usuario* visto desde dos perspectivas, inicialmente como el que activa, configura, pone en funcionamiento y detiene el software y por otro lado el *Usuario* como el actor que recibe los resultados del procesamiento y los visualiza. Sin embargo se puede notar que existen acciones intermedias asociadas o incluidas por otras que no se relacionan directamente con los actores, pero que reflejan requisitos del sistema y por tanto es importante plasmarlos en el diagrama.

Por otra parte la figura 27 muestra los diagramas de estado que describen el funcionamiento del software. Además del diagrama general se desarrollaron diagramas de los estados por lo que pasa un objeto de la clase señal, y otro de la clase archivo.

El diagrama de la izquierda muestra el funcionamiento general del software. Inicialmente el usuario puede seleccionar si realiza adquisición de datos o procesamiento de señales. En caso de elegir la primera opción, se procede a hacer la configuración de la tarjeta de adquisición, se

Figura 25. Diagrama de caso de uso

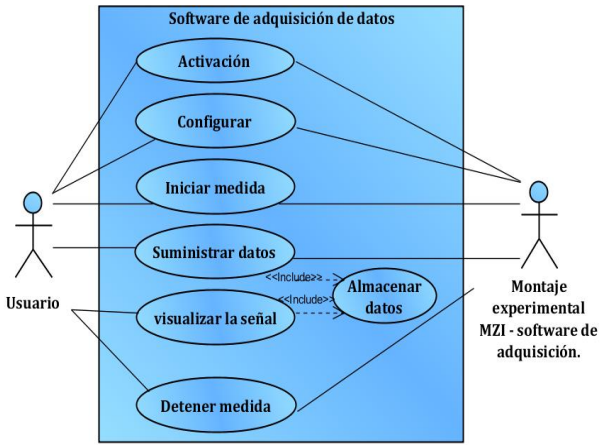
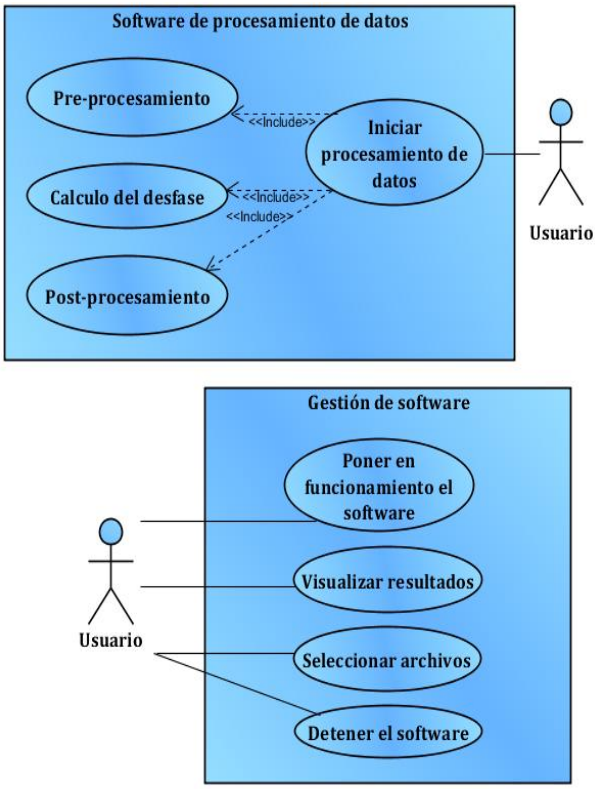


Figura 26. Diagrama de caso de uso

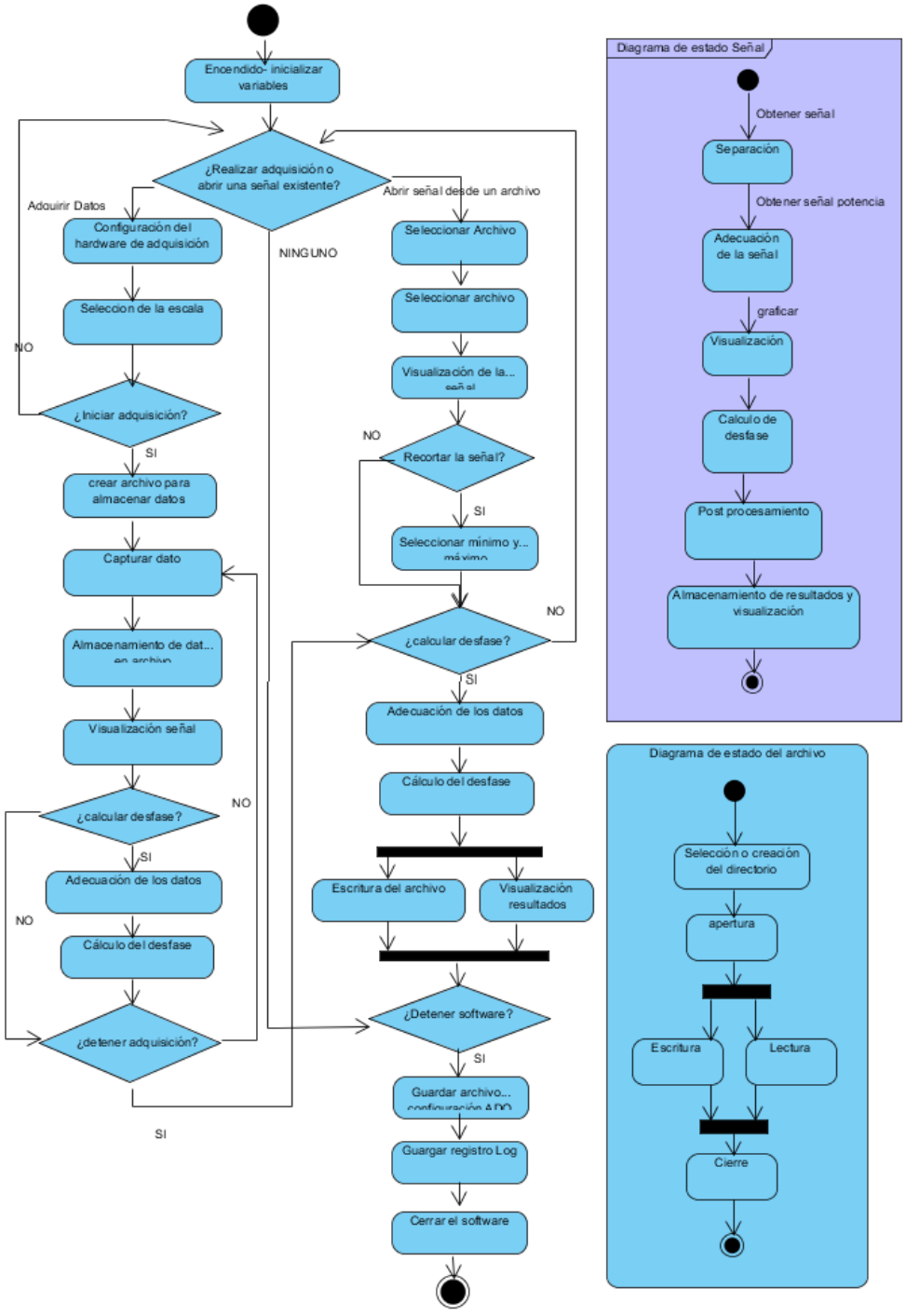


selecciona la escala con el mismo valor del amplificador y el sistema queda disponible a la espera de que el usuario presione el botón *Iniciar Adquisición*; cuando el usuario lo selecciona se crea el archivo para almacenar los datos y se adquiere una dato a la vez. Cada dato se visualiza en la gráfica, y se almacena en el archivo, así sigue consecutivamente hasta que el usuario decida detener la adquisición.

Por otra parte si el usuario decide procesar señales, el software dispone una ventana para seleccionar el archivo, se verifica que tenga la extensión MZI y en caso afirmativo se abre graficando la señal en el cuadro de gráfico. Una vez graficada, el usuario decide si quiere seleccionar una parte de la señal o si desea emplear una señal de referencia. En el último caso

debe seleccionar el botón *Emplear referencia* y seleccionar el archivo que contiene la señal. Para realizar el procesamiento el sistema espera a que el usuario seleccione el botón *Calcular desfase*, al seleccionarlo el sistema realiza el pre-procesameinto, luego el cálculo de desfase y finalmente almacena los resultados en un archivo seleccionado por el usuario y los grafica.

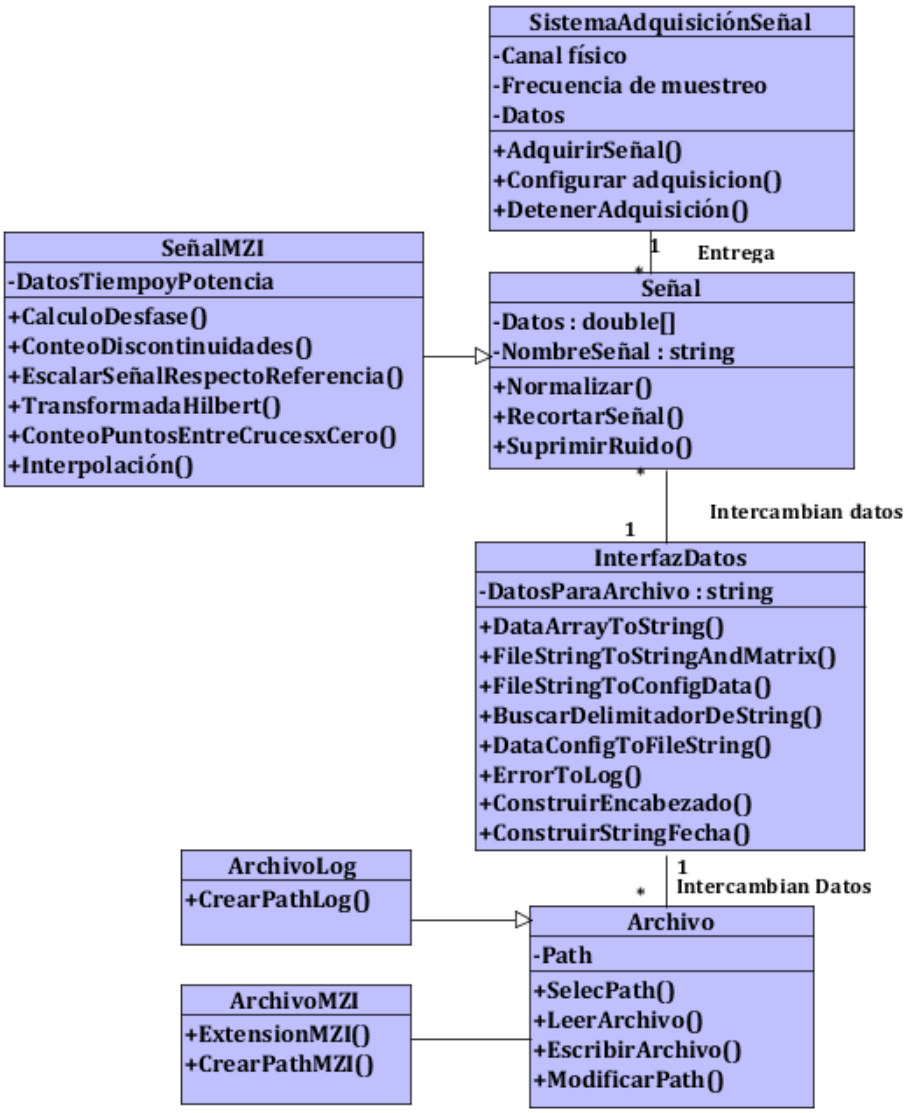
Figura 27. Diagrama de estados



El usuario puede realizar procesamientos consecutivos y repetidas adquisiciones de señales. Sin embargo también puede hacer las dos operaciones simultáneamente.

Finalmente el usuario puede detener el software en cualquier momento.

Figura 28. Diagrama de clases



Por otra parte se modeló el diagrama de clases respectivo que se muestra en la figura 28, en el cual se encuentran las 7 clases que se implementaron en el software con sus propiedades y métodos. También se puede evidenciar la visibilidad de estos, siendo (-) el signo que representa privado y (+) el que denota público. En este sentido todas las variables o propiedades son privadas y todos los métodos son públicos.

En lo que respecta a las relaciones entre las clases se evidencia que existe generalización entre la clase Señal (clase padre) y la clase SeñalMZI (clase hija), la misma relación se conserva entre la clase Archivo y sus subclases ArchivoMZI y ArchivoLog.

Por otra parte la clase SistemaAdquisiciónSeñal está asociada con la clase Señal con una multiplicidad (1...*) es decir que un solo sistema de adquisición puede entregar múltilpes señales. La clase Señal a su vez está asociada con la clase InterfazDatos ya que intercambian datos con una multiplicidad (*..1) es decir que múltiples señales pueden pasar por la interfaz de datos. Finalmente la clase Archivo y la clase InterfazDatos están relacionados por cuando existe también un intercambio de datos.

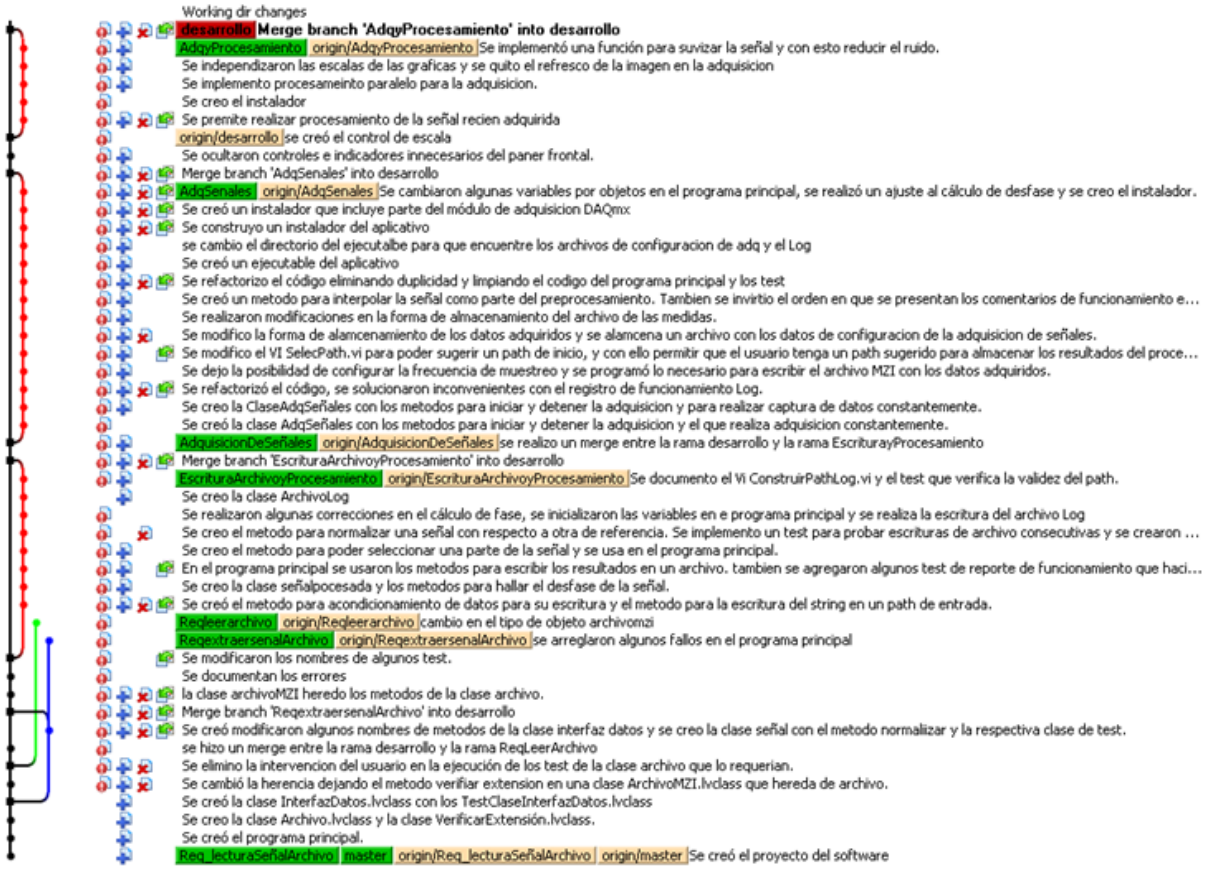
3.5.2. Implementación del código. Una vez se tuvo una descripción concreta de los requisitos se procedió a la programación del software, para lo cual se decidió emplear el lenguaje y entorno de programación gráfico LabVIEW por su facilidad de programación, por soportar POO y TDD, por ser compatible con el hardware de adquisición de datos y principamente para facilitar su mantenimiento ya que es una herramienta usada con mucha frecuencia en el centro de investigación para el cual se desarrolló este proyecto. Los componentes principales del software desarrollado en LabVIEW se presentan en el Anexo E.

Como se mencionó anteriormente el montaje experimental del MZI y los usuarios finales pertenecen al grupo de investigación *Nano Biosensors and Bioanalytical Applications Group (nanoB2A)* con sede en la ciudad de Barcelona por lo que se hizo necesario buscar una plataforma que permitiera compartir los archivos del software y además tener registro de las modificaciones y versiones que se hicieran de los mismos.

Para esto se utilizó un sistema de control de versiones con el fin de almacenar, registrar y gestionar los cambios realizados en los archivos fuentes del aplicativo. Como se mencionó en el marco teórico este sistema permite regresar a versiones anteriores de los archivos y guardar copias de seguridad de los datos.

En la selección de la herramienta adecuada con la cual implementar una estructura colaborativa que permitiera la realización de las pruebas del software con los sistemas biosensores, se pusieron en consideración un control de versiones de cada tipo. Fue así como se estudiaron las ventajas y desventajas de Subversion que es de tipo centralizado y Git que es de tipo distribuido, lo que concluyó en la decisión de usar GIT por las ventajas que se mencionaron con anterioridad.

Figura 29. Ramificaciones del proyecto en GIT



También se adoptó el desarrollo por sprints para una mejor gestión de la implementación del software. Un sprint es un ciclo de producción dentro de un proceso de desarrollo de software iterativo con una duración promedio de una semana en el cual se ha implementado el código necesario para el cumplimiento de determinados requisitos. Los sprints se presentan en el Anexo B y cada Sprint dió lugar a una rama en el empleo del SCV GIT.

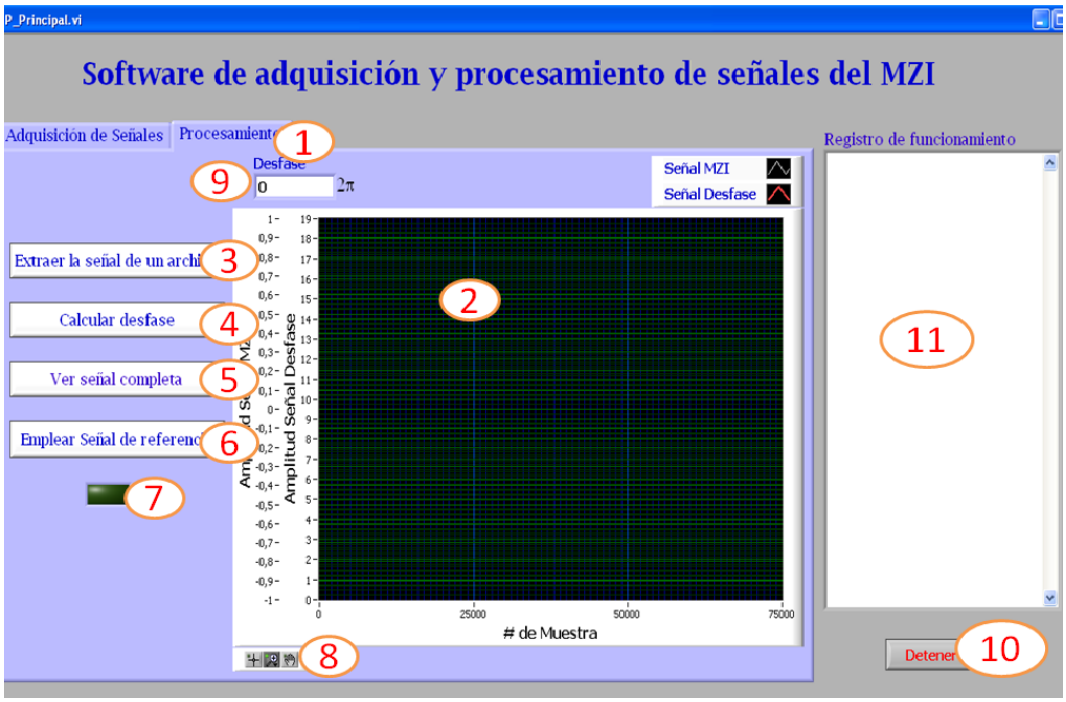
Fue así como se aprovecharon las ventajas del desarrollo por ramificaciones. De esta forma se creó una rama llamada *desarrollo* la cual contiene las versiones estables del aplicativo y a partir de esta se crearon varias ramas en cada una de las cuales se implementó el código para cumplir los requerimientos de cada sprint. Una vez los requerimientos de la nueva rama se cumplieron se unió con la rama principal actualizando así la versión estable del software.

La ramificación que se logró en este proyecto se muestra en la figura 29 e ilustra todo el desarrollo del software empleando TDD. Con esta estrategia primero se programaron los test de prueba y luego las clases, atributos y métodos para cumplir con ellos. Los test que se implementaron por clase se pueden evidenciar en la tabla del Anexo C.

La interfaz de usuario del software desarrollado se ilustra en las figuras 30 y 31 las cuales muestran las pestañas correspondientes al procesamiento y a la adquisición de señales respectivamente.

Las partes que componen la interfaz de usuario se describen a continuación:

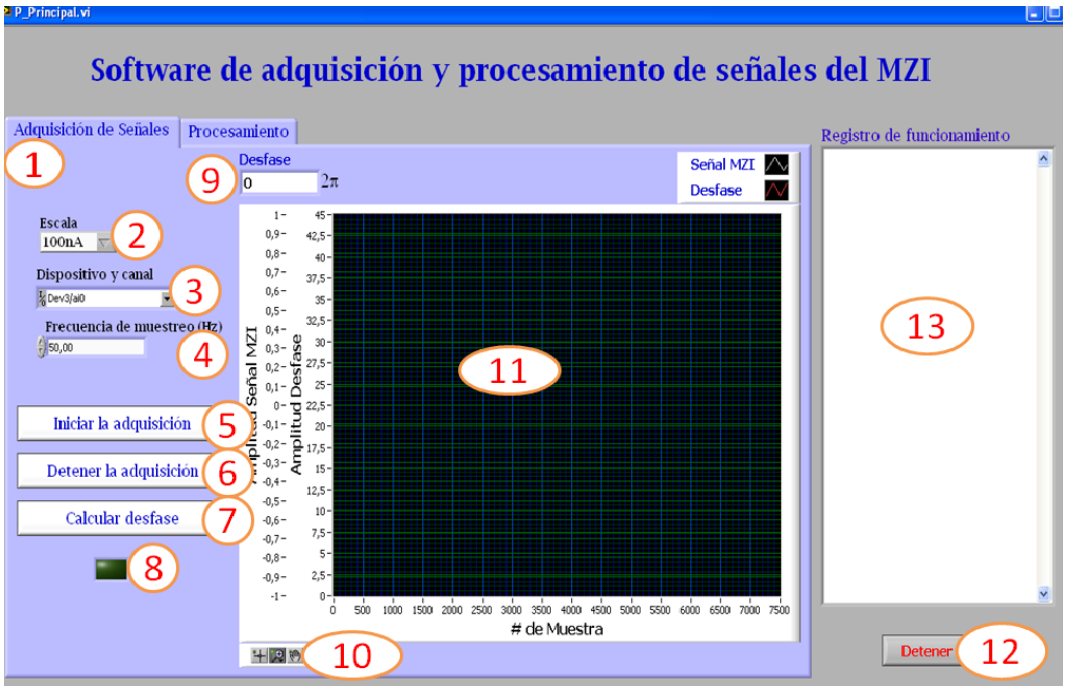
Figura 30. Interfaz de Usuario del software (Pestaña de Procesamiento)



1. Pestaña de Procesamiento
2. Cuadro de gráficas donde se muestra la señal cuando se carga desde un archivo y al seleccionar calcular desfase se muestra la señal MZI y el Desfase.
3. Botón que despliega una ventana para seleccionar el archivo y cargar la señal del MZI.
4. Botón para realizar el cálculo de desfase, al terminar el procesamiento despliega una ventana para seleccionar el archivo en el cual se almacenarán los resultados.
5. Botón que muestra la señal cargada desde archivo completa.

6. Botón que despliega una ventana para seleccionar el archivo que contiene la señal que se desea usar como referencia. Esta señal se debe cargar después de cargar la señal MZI a procesar.
7. Indicador que se enciende al emplear una señal de referencia.
8. Herramientas del gráfico que permiten seleccionar una parte de la señal.
9. Indicador numérico que muestra el valor final del desfase en múltiplos de 2π al realizar el procesamiento.
10. Botón que detiene el software y cierra la ventana.
11. Cuadro de texto donde se muestran los mensajes de funcionamiento del software.

Figura 31. Interfaz de Usuario del software (Pestaña de adquisición)



1. Pestaña para adquisición de datos.
2. Selector de escala.
3. Control para seleccionar el dispositivo y canal de la tarjeta de adquisición.
4. Control para fijar la frecuencia de muestreo.
5. Botón para iniciar la adquisición, despliega una ventana para seleccionar el nombre del archivo en el cual se almacenarán los datos.
6. Botón para detener la adquisición.
7. Botón para calcular el desfase en cualquier momento durante la adquisición.
8. Indicador luminoso que se enciende cuando se estan capturando datos.
9. Indicador numérico que muestra el valor final del desfase en múltiplos de 2π al realizar el procesamiento.
10. Herramientas del gráfico que permiten seleccionar una parte de la señal.
11. Cuadro de gráfico donde se muestran los datos adquiridos.
12. Botón que detiene el software y cierra la ventana.
13. Cuadro de texto donde se muestran los mensajes de funcionamiento del software.

En el desarrollo se tuvo en cuenta que la operación del software debe ser intuitiva y para facilitar al usuario realizar un seguimiento de las operaciones del mismo se implementó un sistema que informa sobre el funcionamiento presentando reportes de estado y de errores en la interfaz. Para ello se empleó el cluster de error que maneja LabVIEW por defecto pero insertando en él mensajes de funcionamiento, de las operaciones realizadas y de las fallas cuando se presentan.

Otro aspecto a destacar fue que en LabVIEW se emplearon eventos para hacer más eficiente el software por cuanto se reduce la carga computacional y no se sobrecarga el procesador ni el sistema operativo. Sin embargo puede resultar desventajoso para realizar adquisición de señales a una frecuencia mayor a 1KHz pero como las señales del MZI tienen un cambio lento son adquiridas a una frecuencia mucho menor a ésta por lo que no representa ninguna dificultad.

Para usar el software primero se instala el software con el procedimiento que se describe en el Anexo D.

Cuando se inicia el software este verifica que en el directorio donde se instaló exista una carpeta llamada *data* y dentro de ella dos carpetas: $MediasMZI$ y *ArchivoLog* en estas dos carpetas se almacenan por defecto los archivos .mzi de datos adquiridos y los archivos de registro de funcionamiento del software respectivamente.

Al iniciar también se inicializan todas las variables y objetos. En este proceso se realiza la lectura del archivo de configuración de la adquisición contenido también en la carpeta data en el cual se encuentran los últimos valores de frecuencia y canal físico empleados por el usuario.

Para procesar una señal primero se selecciona la pestaña *Procesamiento* y se selecciona la señal con el botón *Extraer la señal de un archivo*. Luego se debe seleccionar el botón *Calcular desfase* y el archivo donde se guardarán los resutados. Seguidamente se mostrarán en la interfaz la gráfica del desfase, el valor total y el registro de todas las operaciones realizadas por el software.

Para adquirir una señal se selecciona la pestaña *Adquisición de señales*, se selecciona el canal físico de la tarjeta de adquisición, la frecuencia de muestreo, la escala que corresponde a la misma del amplificador físico y se presiona el botón *Iniciar Adquisición*. Finalmente se selecciona el archivo para almacenar los datos. En ese momento se enciende el indicador de adquisición y los datos aqduiridos se muestran en el cuadro de gráfico y se almacenan en el archivo. Los mensajes de funcionamiento del software se muestran en el cuadro de registro.

Para detener el software se selecciona el botón *Detener*, en ese momento se almacenan los últimos datos de frecuencia y canal en el archivo de configuración, se almacenan los mensajes de registro en el archivo Log (con un nombre creado basado en la fecha y hora) y se cierra el aplicativo.

Por otra parte, por la forma en que está estructurado el software es posible realizar procesamiento y adquisición de datos simultáneamente ya que por emplear procesos independientes se paraleliza el funcionamiento en el procesador. Adicionalmente se emplean eventos, de forma que el sistema esta monitoreando la activación de estos cada determinado tiempo y el resto del tiempo se encarga del evento por defecto que es el *Timeout*.

4. PRUEBAS EXPERIMENTALES Y RESULTADOS OBTENIDOS

4.1. PRUEBAS DE ADQUISICIÓN

Se realizaron pruebas con la tarjeta de adquisición DAQ USB 6808 y un generador de onda sinusoidal. En la figura 32 se muestra la interfaz de usuario en funcionamiento mientras adquiere datos.

Figura 32. Ejemplo de adquisición de una señal MZI

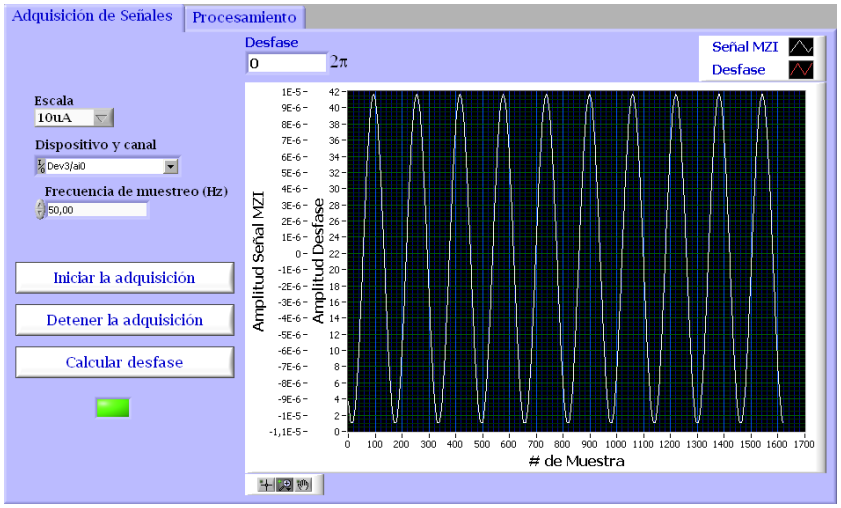
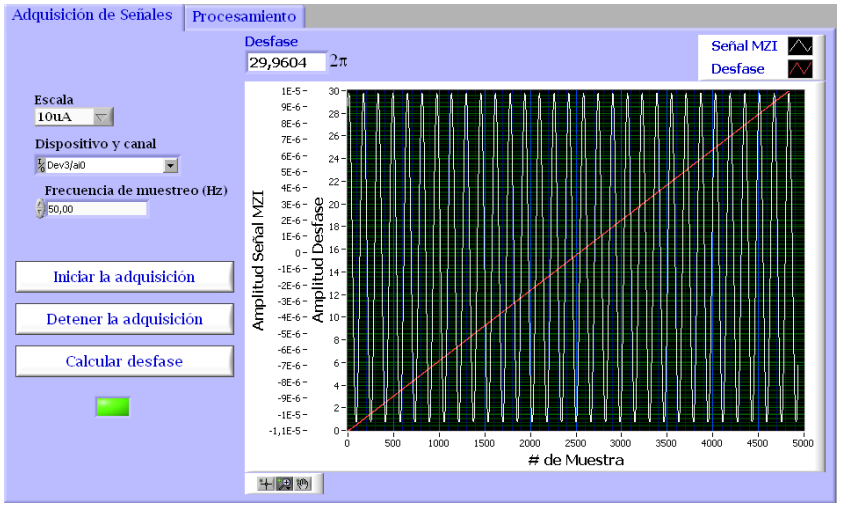


Figura 33. Ejemplo de adquisición de una señal MZI



Por su parte la figura 33 muestra el cálculo de fase mientras se captura la señal. Como se observa la gráfica de desfase se muestra en el mismo gráfico de la señal MZI, pero continúa la adquisición. Finalmente la figura 34 muestra el cálculo de fase al terminar la adquisición.

Se realizaron pruebas de adquisición de señales con el montaje experimental del MZI con frecuencias entre 10 y 15 Hz. La figura 35 muestra un ejemplo de una señal adquirida con la tarjeta USB 6251 y el software desarrollado. La figura 36 muestra como quedan almacenados los datos en el archivo.

Figura 34. Ejemplo de adquisición de una señal MZI

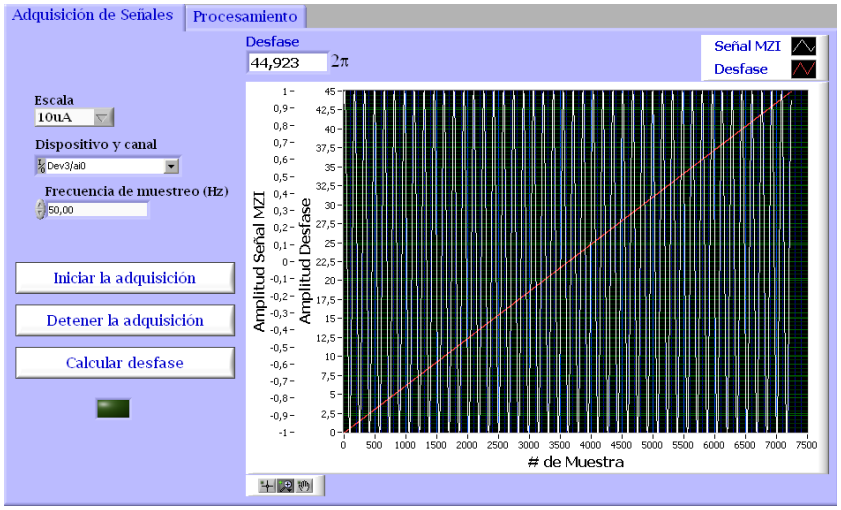


Figura 35. Ejemplo de adquisición de una señal MZI

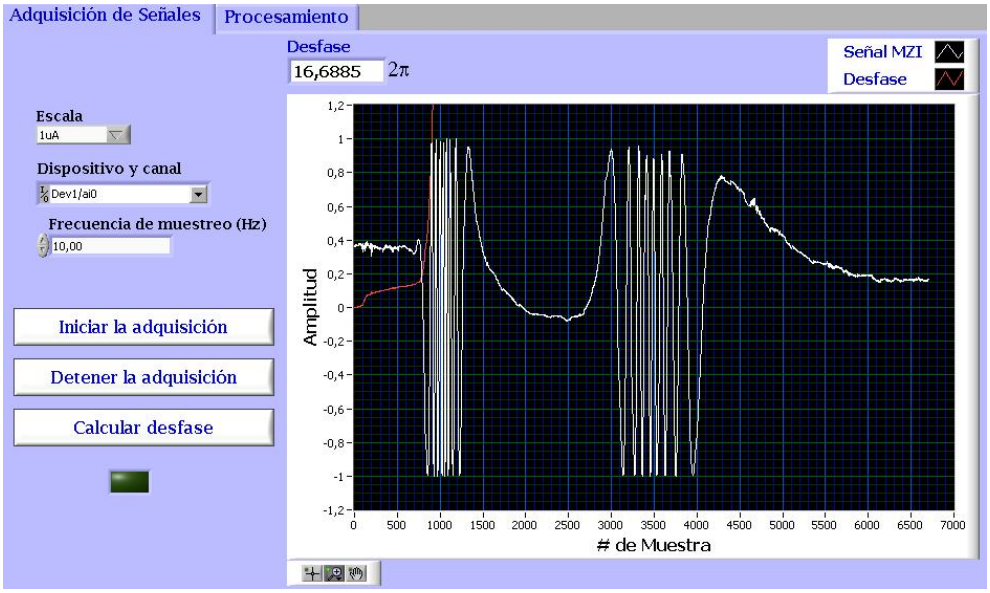
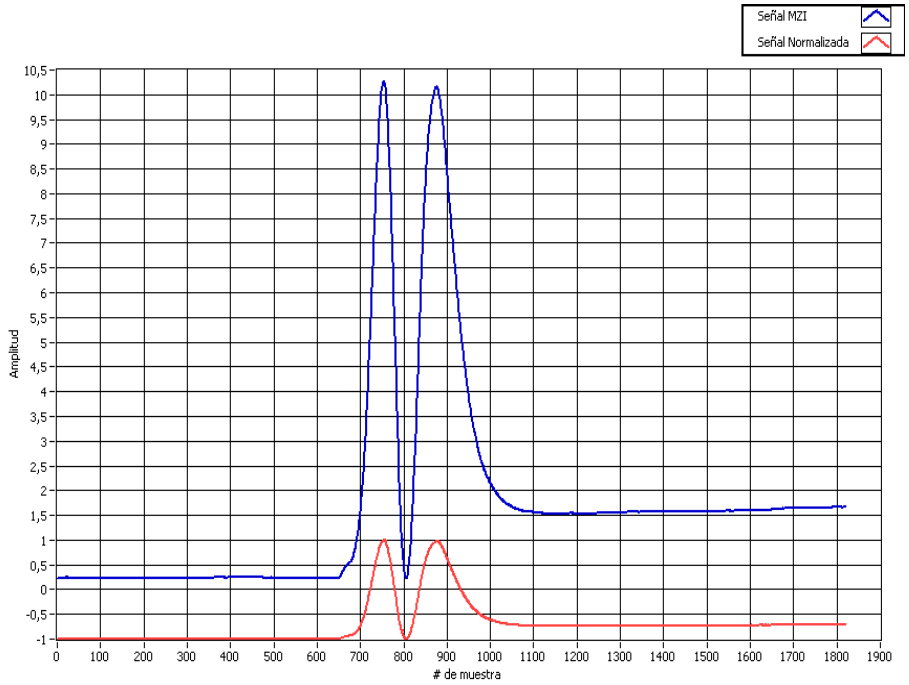


Figura 36. Ejemplo de archivo de medida

4.2. PRUEBAS DEL PREPROCESAMIENTO

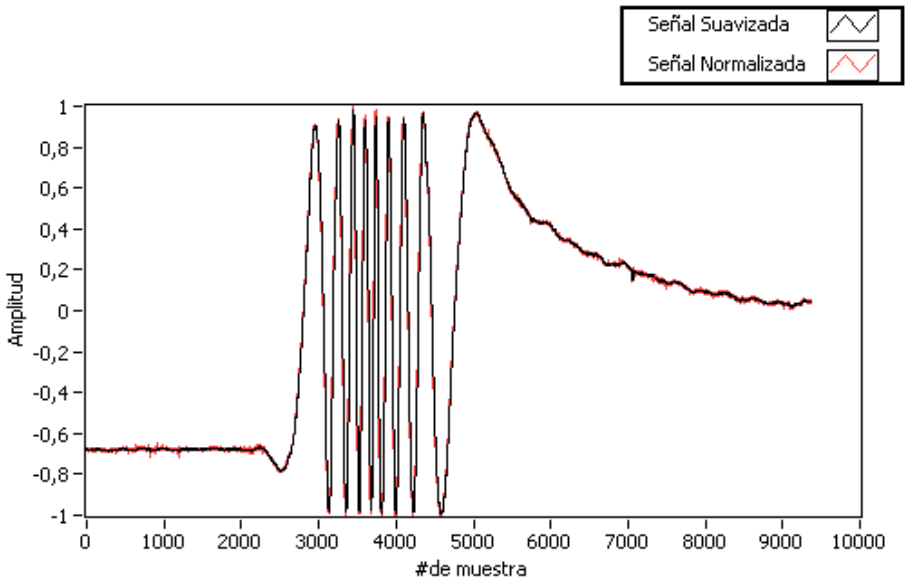
La figura 37 muestra un ejemplo de la normalización que realiza la etapa de pre-procesamiento de las señales. Mediante este proceso se escala la señal de el tamaño que tenga a valores entre -1 y 1 empleando el máximo, el mínimo y el valor medio como se describió con anterioridad.

Figura 37. Ejemplo de normalización de una señal MZI



La figura 38 muestra un ejemplo de una señal MZI tras ser normalizada y suavizada con la función anteriormente descrita. En esta señal se puede evidenciar que se reduce notablemente el ruido.

Figura 38. Ejemplo de una señal MZI normalizada y suavizada



4.3. PRUEBAS DEL PROCESAMIENTO

Para probar el desempeño de la etapa de procesamiento el Grupo de investigación en calidad de dueño del proyecto proporcionó un banco de señales con el valor del desfase esperado. Estas señales se probaron en el software y los resultados obtenidos en comparación se observan en la tabla 6. Adicionalmente de la figura 39 a la figura 45 se muestran las gráficas del desfase calculado con cada señal de prueba.

Tabla 4. Comparación ente los valores de referencia y los calculados

Valor Referencia	**Valor Calculado**	**Error(%)**
0,97610	0,97549	0,0625
1,93300	1,91743	0,8054
3,26170	3,30347	1,2806
7,1157	7,19534	1,1192
7,0686	7,21136	2,0513
3,2317	3,30718	2,3356
1,8465	1,81701	1,5970

En las pruebas realizadas se encontró un margen de error inferior al 2,5% lo que es aceptable en este procesamiento lo que no implica que el cálculo sea erróneo sino que la repetitibilidad de la medida posee bajo error y el sistema es confiable.

Figura 39. Ejemplo de cálculo de desfase de una parte de una señal MZI (menos de 1π)

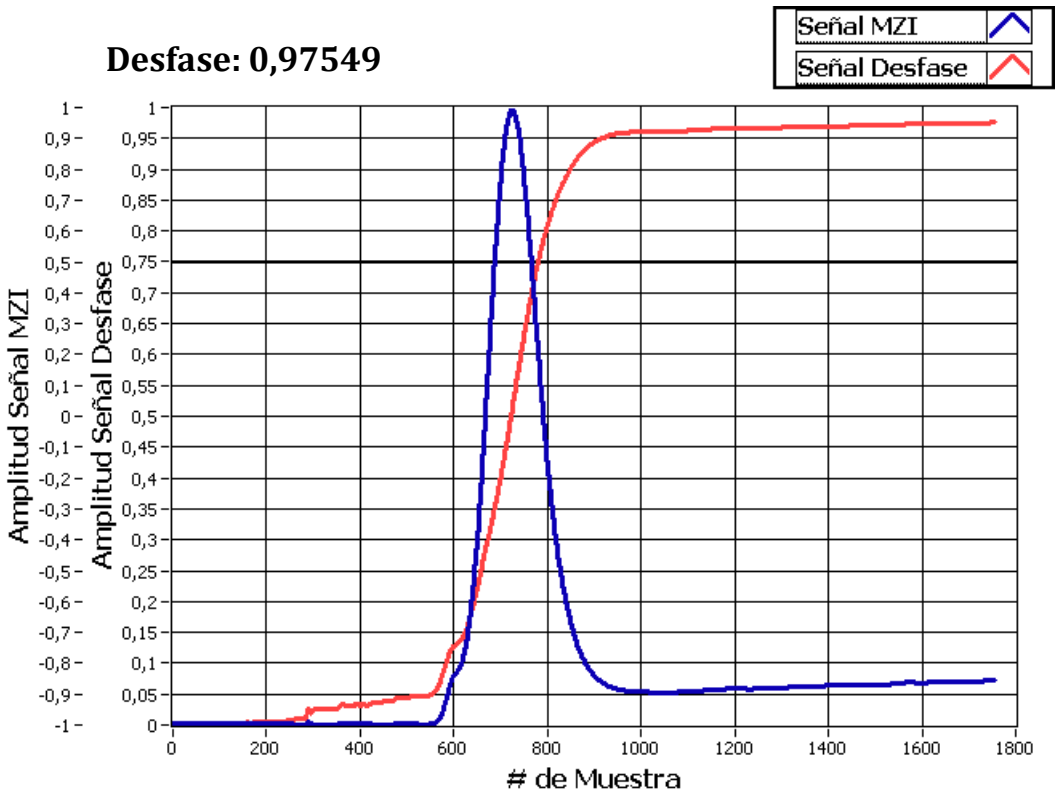


Figura 40. Ejemplo de cálculo de desfase de una parte de una señal MZI (menos de 2π)

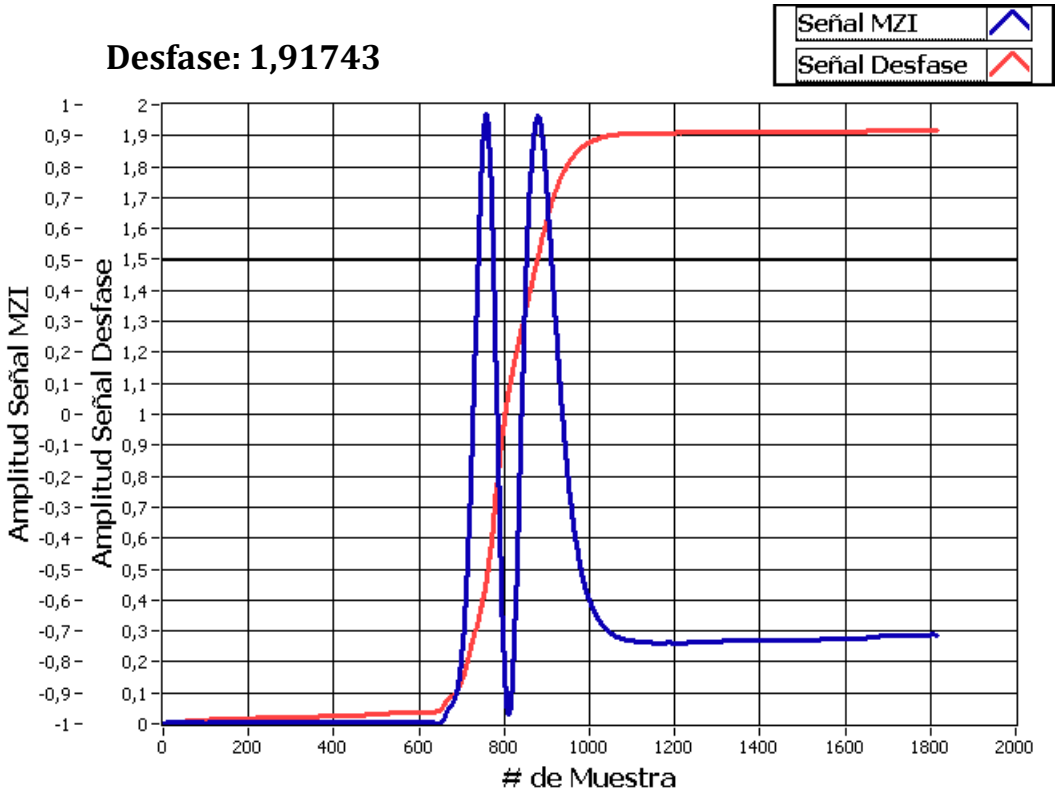


Figura 41. Ejemplo de cálculo de desfase de una parte de una señal MZI (menos de 4π)

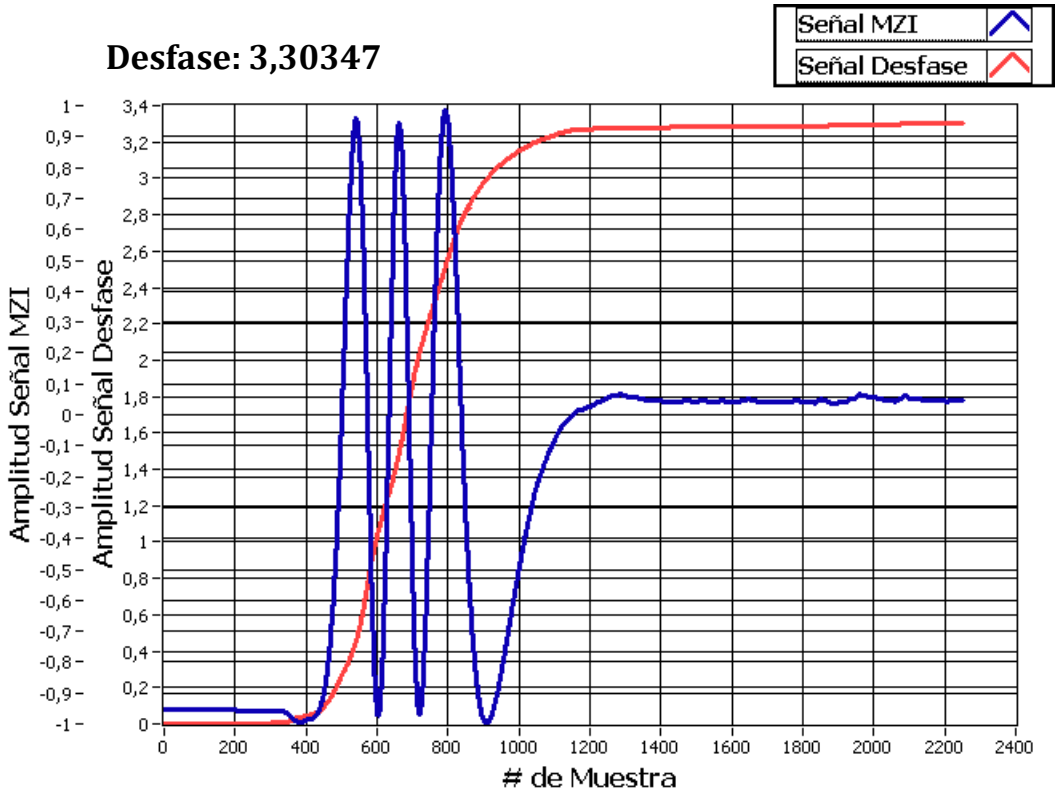


Figura 42. Ejemplo de cálculo de desfase de la parte de la entrada de una señal MZI

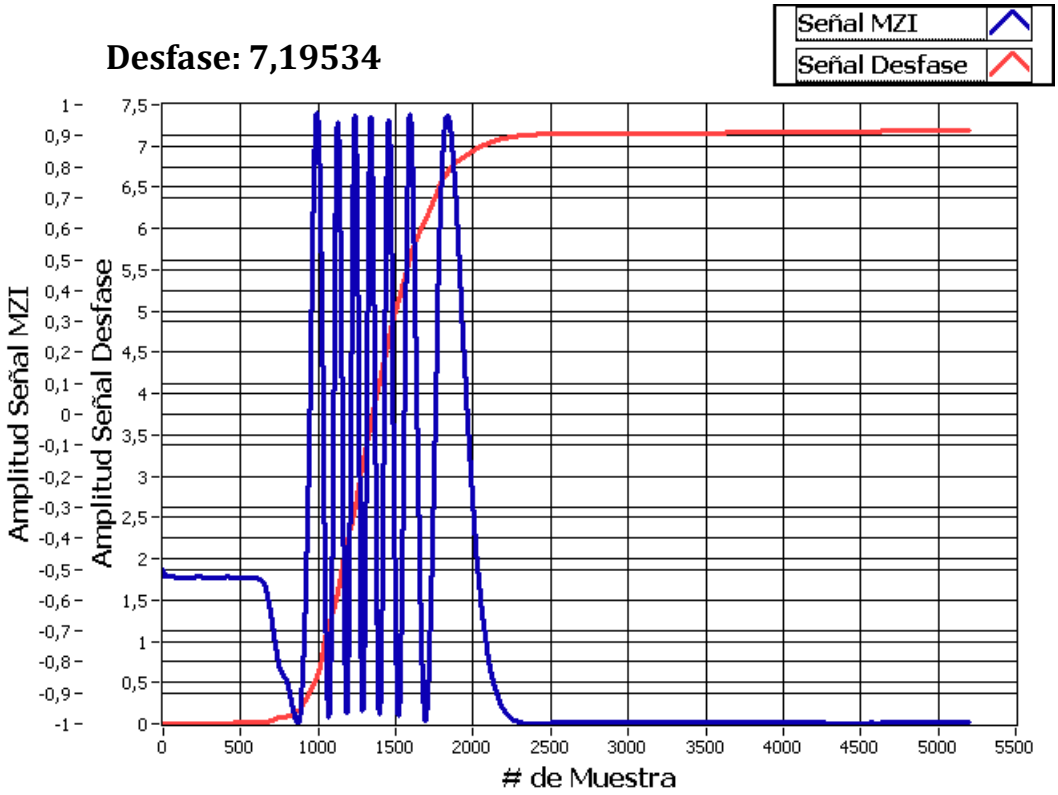


Figura 43. Ejemplo de cálculo de desfase de la parte de la salida de una señal MZI

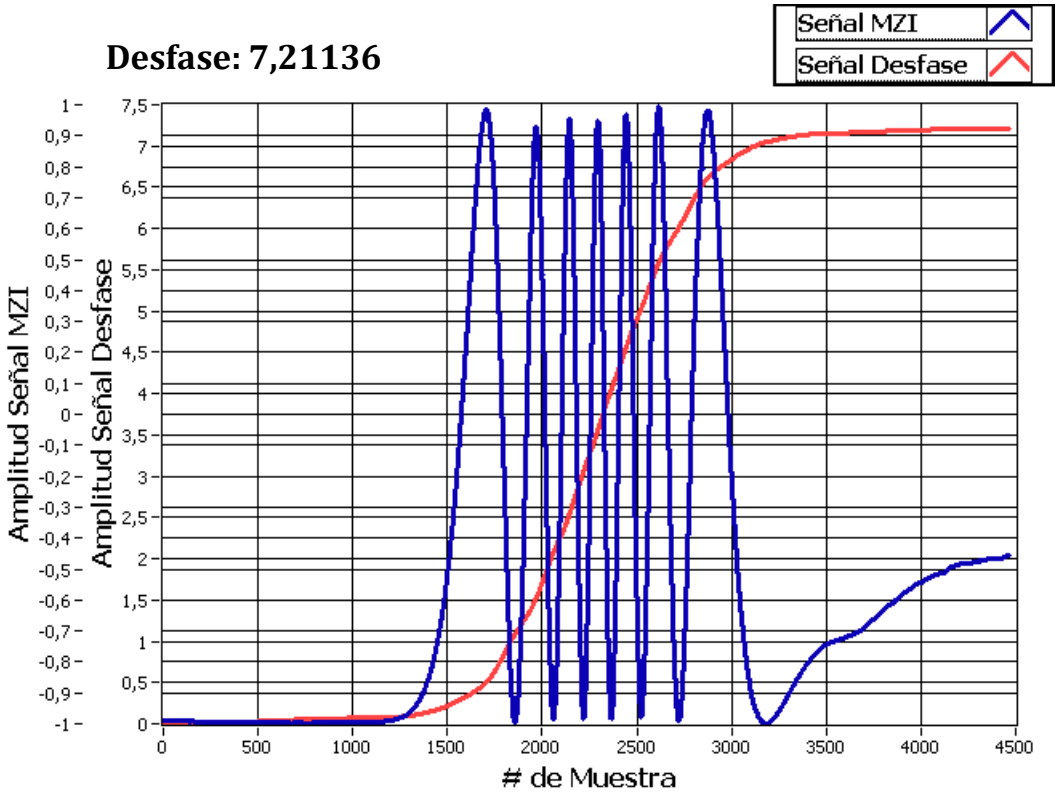


Figura 44. Ejemplo de cálculo de desfase de una señal MZI (menos de 4π)

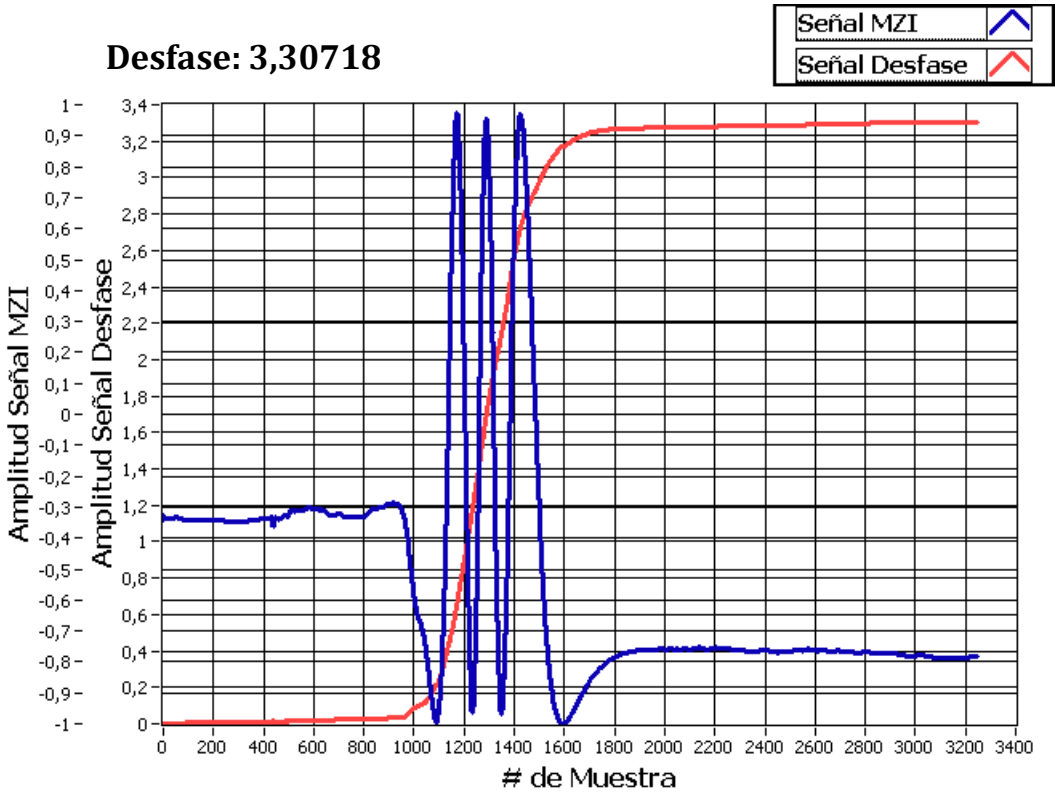


Figura 45. Ejemplo de cálculo de desfase de una señal MZI (menos de 2π)

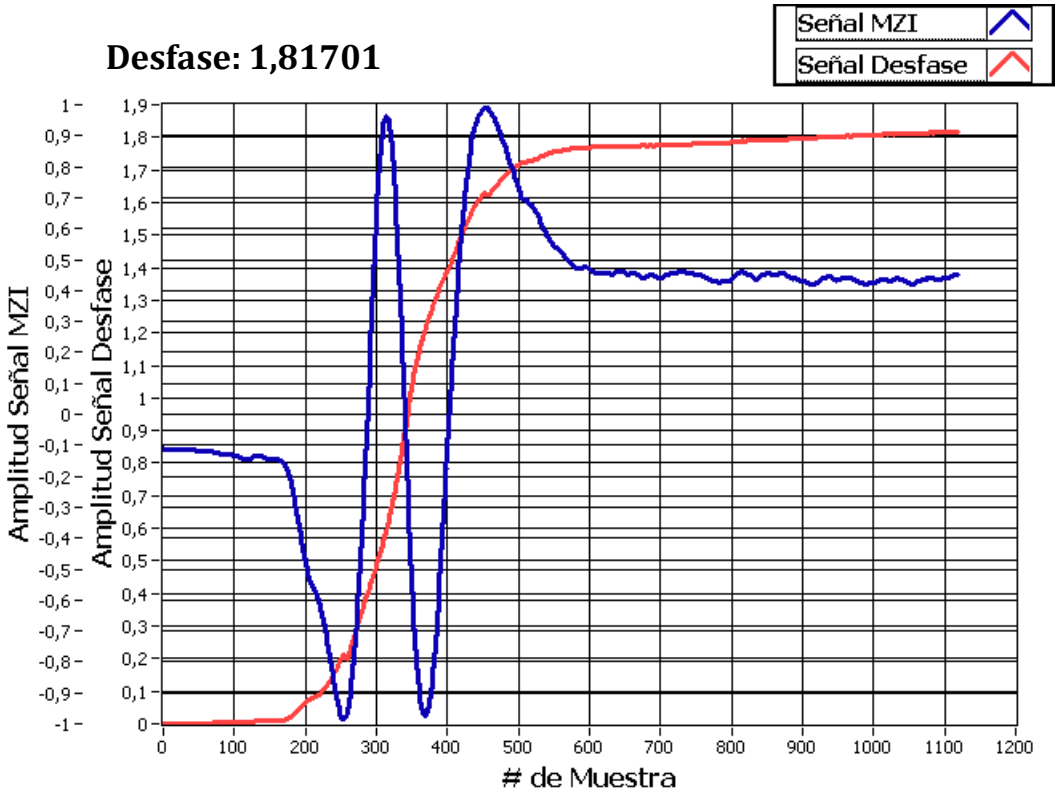
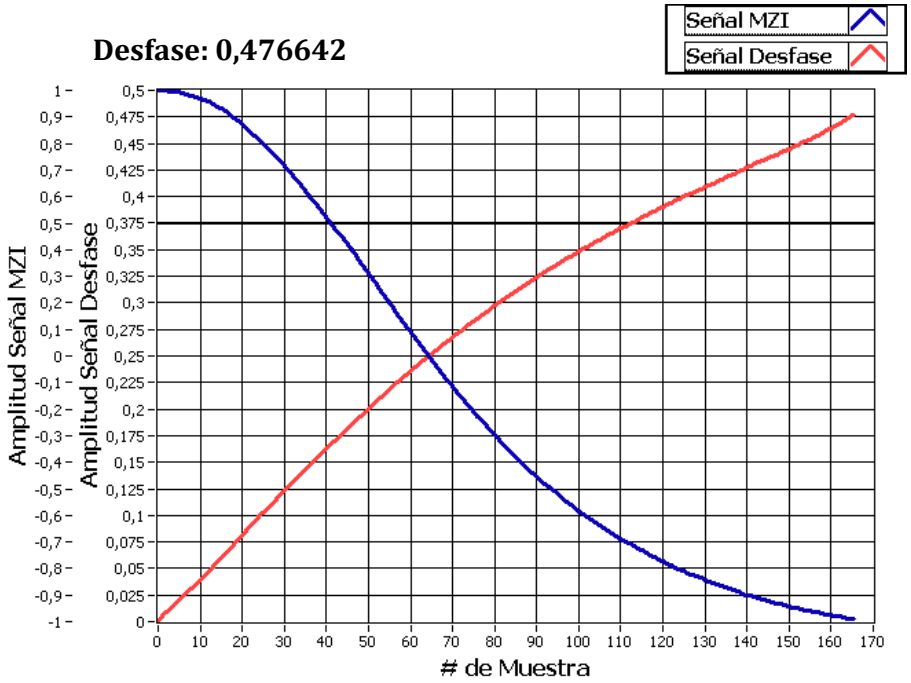
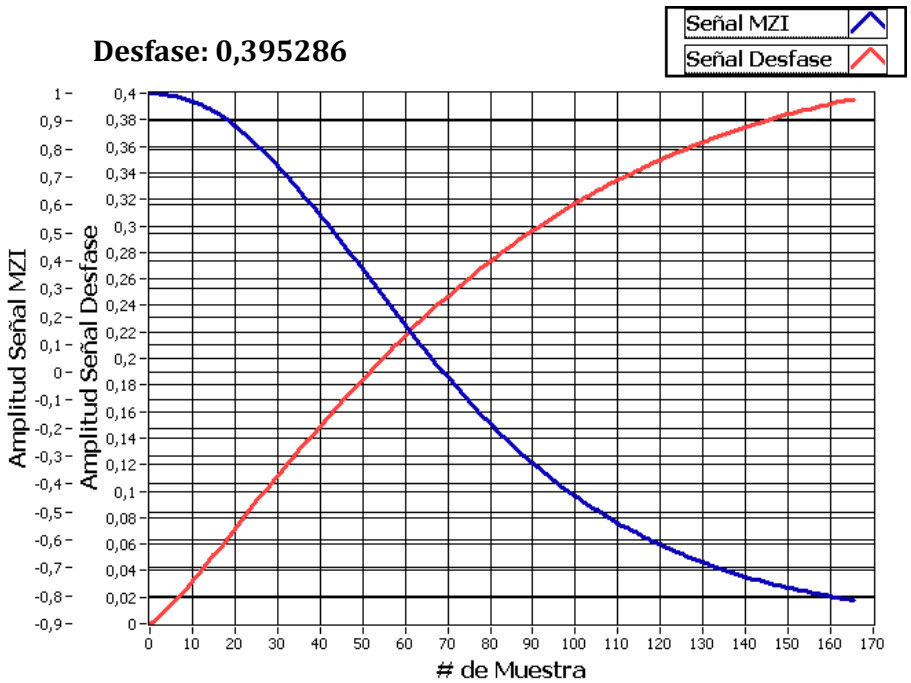


Figura 46. Ejemplo de cálculo de desfase de una señal MZI menor de *pi* sin emplear referencia



Las gráficas 46 y 47 ilustran el funcionamiento del procesamiento empleando la señal de referencia. Para este caso se tomó una parte de una señal MZI que tiene longitud menor a π, como se observa si a esta señal se le calcula el desfase, el proceso normaliza con los límites locales lo cual da un valor erroreo de fase. En cambio si se selecciona la señal, luego se selecciona la referencia y se calcula el desfase, la señal MZI se normaliza con respecto a la señal de referencia. Con estos datos el procesamiento si corresponde con el valor adecuado.

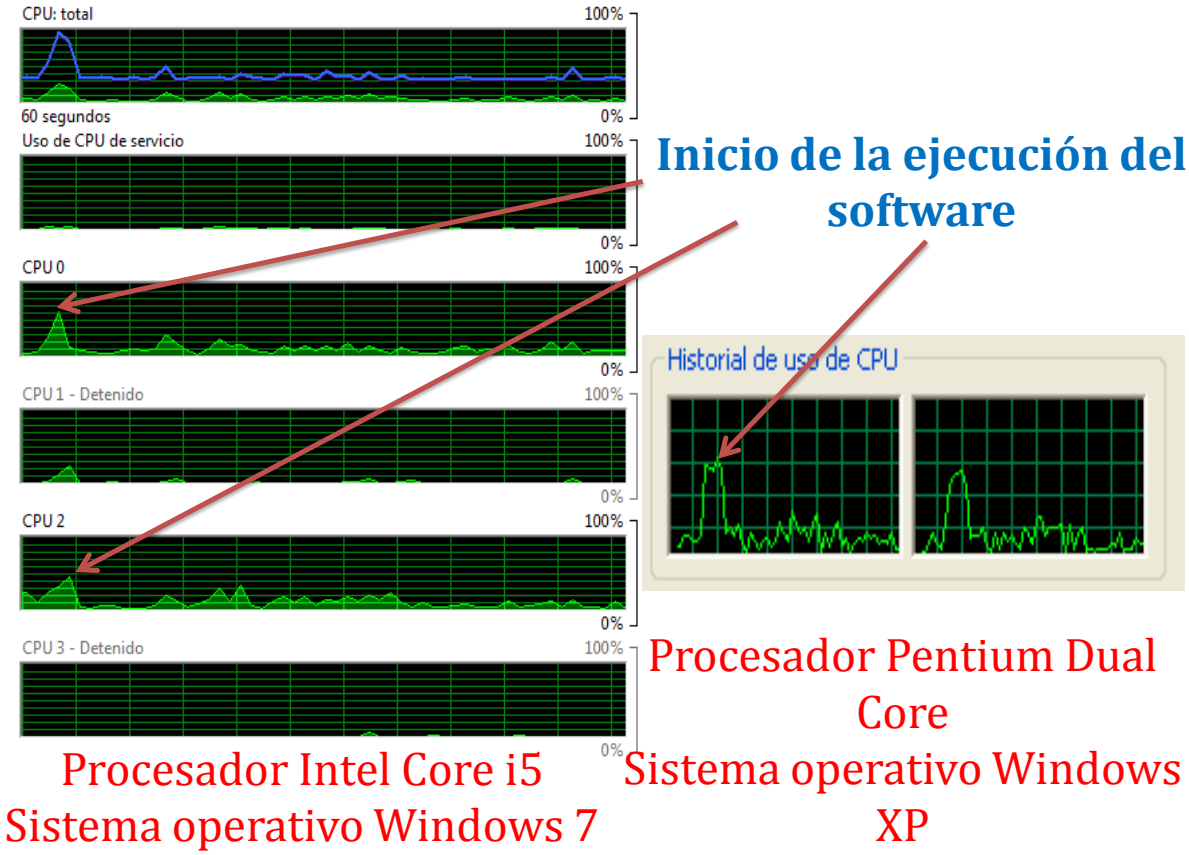
Figura 47. Ejemplo de cálculo de desfase de una señal menor de *pi* MZI empleando referencia



4.4. PRUEBAS DE RENDIMIENTO

Para probar el rendimiento del software se revisó la carga computacional la cual se observa en las gráficas de consumo de CPU de la figura 48. Como se evidencia, sólo se tiene una carga significativa al iniciar la ejecución del software donde alcanza un 60% del consumo de la CPU pero en el resto del tiempo adquiriendo datos o realizando procesamiento esta al rededor de el 20% del consumo. Esto aplica casi independiente del procesador en el que se instale el software.

Figura 48. Carga computacional del software



CONCLUSIONES

Se realizó una buena documentación de las características de los biosensores y de la señal que entrega para formular estrategias para su procesamiento.

Se implementó un sistema para adquirir señales compatible con el hardware de National Instruments seleccionado, que cumple con las especificaciones del dueño del software

Se encontró que la transformada Hilbert puede ser empleada para el cálculo de la fase de una señal con gran precisión aun cuando esta no sea periódica. Sin embargo también se hallaron algunos casos en los que da valores erróneos por lo que se debe complementar con otro método.

Se diseñó y probó un algoritmo para el cálculo del desfase de las señales provenientes del biosensor MZI el cual mostró un porcentaje de error inferior al 2,5

Inicialmente se planteó la inquietud de realizar procesamiento en línea o no. En este sentido evaluando las etapas del procesamiento se optó por no realizar procesamiento en tiempo real debido a que son necesarios los límites de la señal completa para las operaciones como normalización e interpolado y para el cálculo de la fase. Sin embargo el procesamiento es muy rápido y se puede realizar en cualquier momento durante la adquisición sin detenerla así que no representa mayor retardo para conocer el desfase de alguna señal.

Se estudiaron, documentaron y emplearon diferentes metodologías para el desarrollo del software con el fin de hacerlo además de funcional, fácil de mantener, estable y con buenas características de rendimiento. Entre estas metodologías se implementaron Programación orientada a objetos y Desarrollo guiado por pruebas. Así mismo se desarrollaron modelos empleando UML.

Todo lo anterior se reúne en la entrega de un software de adquisición y procesamiento de señales que cumple con las especificaciones de funcionamiento.

Se comprobaron las ventajas de emplear un sistema de control de versiones en el desarrollo de un proyecto de software.

BIBLIOGRAFÍA

BIOSENS, [citado en 10 de octubre de 2011], Disponible en: http://www.imm.cnm.csic.es /RedBiosensores/tecnologia-de-biosensores.html.

BITTER, Rick; MOHIUDDIN, Taqi y NAWROCKI, Matt. *Object-Oriented Programming in LabVIEW*, En: LabVIEW Advanced Programming Techinques.: CRC Press LLC, 2001.

BLANCO, Francisco; AGIRREGABIRIA, M.; BERGANZO, Javier; MAYORGA. K.; ELIZALDE, J.; CALLE, Ana;DOMINGUEZ, Carlos y LECHUGA, Laura *Microfluidic-optical integrated CMOS compatible devices for label-free biochemical sensing* En: Journal of Micromechanics and Microengineering, vol. 16, no. 5, Mayo 2006, pp. 1006-1016.

BLÉ, Carlos, y colaboradores. *Diseño ágil con TDD*. SafeCreative. 2010. 308 p.

CHACON, Scott *Pro Git*, 2009, USA. 275 p.

DIAZ, M y ESTELLER, R. Comparación del operador de energía no lineal y la transformada de Hilbert en la estimación de la amplitud y frecuencia instantáneas. En: IEEE Latin America Transactions, vol 5. no 1. 2007

FOWLER, Martin y SCOTT, Kendall. UML gota a gota. Traducido por Jaime González. Editorial Pearson. México. 1999. 224 p.

GARCÍA,Javier; RODRIGUEZ, José; SARIEGUI, José y BRAZÁLEZ, Alfonso. Aprenda C++ como si estuviera en primero. Universidad de Navarra. San Sebastian. Abril, 1998. 83 p.

GIRALDO, Jairo; GONZÁLEZ, Edgar y GÓMEZ Fernando. *Nanotecnociencia: Nociones preliminares sobre el universo nanoscópico*, Bogotá D.C.: Ediciones Buinaima, 2007. 254 p.

HERNANDEZ, Claudia y RODRIGUEZ, Jorge *Preprocesamiento de datos estructurados*. En: Vinculos, Junio, 2008. vol. 4, no. 2, pp. 27-48.

KEHTARNAVAZ, Nasser y KIM, Namjin. Digital signal processing system-level design: using LabVIEW. Newness. 2005. 305 p.

KITSARA, María; MISIAKOS, Konstatinos; RAPTIS, Ioannis y MAKARONA, Eleni. *Integrated optical frequency-resolved Mach-Zehnder interferometers for label-free affinity sensing.* En: Optics express, 2010, vol. 18 no.8, p. 8193-8206. Disponible en Internet: http://www.ncbi.nlm.ni

KRASKOV, Alexander; KREUZ, Thomas; ANDREZEJAK, Ralph; STÖGBAUER, Harald; NADLER, Walter y GRASSBERGER, Peter. Extracting Phases from Aperiodic Signals. Febrero 2008. Alemania. 3 p.

LAJARA, José y PELEGRÍ, José. LABVIEW. Entorno gráfico de programación. Marcombo S.A., Barcelona 2007. 374 p.

LÁZARO, Antoni; DEL RÍO FRENÁNDEZ, Joaquín. LabVIEW: Programación gráfica para el control de instrumentación. Editorial Paraninfo, Madrid España- 1997, ISBN: 84-283-2339-9.

LECHUGA, Laura; ZINOVIEV, Kirill; CARRASCOSA, Laura y MORENO, M. *Nanodevices for Biosensing: Design, Fabrication and Applications.* En: Nanotechnologies for the Life Sciences, Septiembre, 2007, vol. 10, p. 317-344.

LECHUGA, Laura; ZINOVIEV, Kirill; HIDALGO, Orlando; DOMINGUEZ, Carlos y ELIZALDE J. *Biosensing microsystems: fast, label-free, real-time clinical testing.* Diciembre 2008, SPIE Newsroom, 2p.

NANO BIOSENSORS AND BIOANALYTICAL APPLICATIONS, [citado en 17 de septiembre de 2011], Disponible en Internet: http://www.cin2.es/ biosensores/

NATIONAL INSTRUMENTS, [Citado en 15 de noviembre de 2011] Disponible en internet: http://www.ni.com/LabVIEW/whatis/esa/

PALLÁS, Ramón. *Sensores y acondicionadores de señal*, 4 ed., Barcelona: Ed. Marcombo. 2008. 217 p.

PÉREZ, Concepción *Sensores ópticos*, Universidad de Valencia, 1996.

PRIETO Francisco; SEPÚLVEDA, Borja; CALLE, Ana; LLOBERA, Andreu; DOMINGUEZ, Carlos; ABAD, Antonio; MOTOYA, Ángel y LECHUGA, Laura. *An integrated optical interferometric nanodevice based on silicon technology for biosensor applications* Nanotechnology, vol. 14, Aug. 2003, pp. 907-912.

PROAKIS, John y MANOLAKIS, Dimitris. Digital Signal Processing Principles, Algorithms and Applications. Tercera Edición. Prentice Hall. USA. 1996. 1033 p.

SANCHEZ, José y LECHUGA, Laura. Desarrollo de un biosensor fotónico de alta sensibilidad basado en interferómetros Mach-Zehnder integrados en tecnología de silicio. Universidad Autónoma de Madrid. Facultad de Ciencias. Dpto. de Física de Materiales, 2007.

STEWART, R.W. Y HOFFMAN, M.W. Digital Signal Processing An "A" to "Z". BlueBox Multimedia. 1998. 450 p.

WILLIAMS, Linda y WADE, Adam *Nanotechnology Demystified*, Ed. McGraw-Hill, 2007.

ZHANG, Yan. *Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor Miniature fiber-optic multicavity Fabry-Perot interferometric biosensor*,Trabajo de grado Doctor of Philosophy in Electrica Engineering, Virginia Polytechnic Institute and State University, Diciembre 2005.

ZINOVIEV, Kirill;CARRASCOSA, Laura; SANCHEZ, José; SEPÚLVEDA, Borja; DOMINGUEZ, Carlos, y LECHUGA, Laura. Silicon Photonic Biosensors for Lab-on-a-Chip Applications, En: Advances in Optical Technologies, Abril, 2008, pp. 1-7.

ANEXOS

A. REQUERIMIENTOS DEL SOFTWARE

1. Se debe poder adquirir una señal.
2. Se debe poder cargar la señal de un archivo existente.
3. El sistema debe brindar información de errores o eventos al usuario en términos comprensibles para él.
4. El sistema debe permitir realizar varias mediciones o procesos de medida y/o procesamiento consecutivo sin reiniciarse.
5. El sistema permitirá hacer procesos de adquisición y procesamiento simultáneamente.
6. Se debe poder iniciar la adquisición cuando lo desee el usuario, no automáticamente.
7. Cuando se quiera detener el software mientras se encuentra realizando adquisición o procesamiento debe esperar a que se detenga la captura y/o finalice el procesamiento.
8. Al iniciar el software se deben leer de un archivo de configuración los valores por defecto para los parámetros de adquisición.
9. En el entorno gráfico del software deben ser configurables los parámetros de la adquisición de datos (como la frecuencia de muestreo, el canal de adquisición, entre otros).
10. Al detener el software se deben almacenar en un archivo los valores actuales de los parámetros de la adquisición.
11. Mientras se estén adquiriendo datos no se deben poder modificar los parámetros de configuración.
12. Si los parámetros de configuración de la adquisición no son válidos (por ejemplo una frecuencia 0 o negativa), el sistema debe avisar a usuario y no permitir que se inicie la adquisición en esas condiciones.
13. Cuando se están adquiriendo datos se debe poder detener la adquisición en cualquier momento.
14. Cuando no se están adquiriendo datos y se selecciona detener la medida no debe suceder nada.
15. El sistema debe dar un encabezado por defecto para ser almacenado con la señal adquirida, sin embargo el usuario debe poder agregar un nombre a la señal desde el entorno del aplicativo.
16. Si se selecciona adquirir datos pero el sistema encuentra que el hardware no está disponible o no se encuentra bien conectado debe avisa al usuario .
17. Al iniciar el software los cuadros de gráficos deben estar limpios o vacios.
18. Cuando se está midiendo se debe observar en tiempo real una gráfica de los datos capturados.
19. Mientras el sistema este adquiriendo datos debe almacenarlos.
20. Para almacenar los datos el usuario podrá seleccionar el directorio y el nombre del archivo, sin embargo el sistema dará uno por defecto.
21. Una vez han finalizado la adquisición de datos el sistema quedará disponible para decidir de nuevo si se quiere adquirir o abrir un archivo.
22. Si se selecciona cargar los datos de un archivo se debe poder seleccionar el archivo verificando que tenga la extensión adecuada.
23. Al abrir el archivo se debe poder visualizar la señal en una gráfica.
24. Mediante un botón se debe poder iniciar el procesamiento, el cual debe arrojar una grafica del desfase y un valor total de desfase.
25. El procesamiento debe entregar un porcentaje de error o un margen de tolerancia.
26. Al terminar el procesamiento debe ser posible almacenar los datos del desfase con un path que se dará por defecto con la extensión .mzi aunque el usuario lo podrá modificar.
27. El archivo que se cree con los resultados del procesamiento deberá contener 3 columnas: tiempo, potencia y desfase.
28. Los datos deben ser exportados en un formato legible por otro software de análisis de

datos como Origin, donde sea fácil además de ver los datos saber a qué corresponde cada columna y la unidad de medida en la que esta expresado.

29. Se debe poder seleccionar de la gráfica de la señal solo una parte de la misma.
30. Una vez seleccionada una porción de la señal debe ser posible volver a visualizar la señal completa.
31. La ejecución del software debe consumir los recursos mínimos, y tener la menor carga computacional posible.
32. El procesamiento debe tomar el mínimo tiempo posible.
33. La ventana debe tener los botones típicos de ventana de Windows para minimizar, maximizar y cerrar, y contará con un botón diferente para detener el software.

B. DESARROLLO POR SPRINTS

Sprint	Requisitos implementados
1	• Lectura de archivos de texto y extracción de los datos de la señal MZI contenidos en este. • Creación de un sistema de Log que informe al usuario • Mostrar en la interfaz de usuario La señal leída y el reporte de funcionamiento
2	• Creación de un sistema de registro que informe al usuario el funcionamiento del software empleando el cluster de error • Creación de la clase señal para completar el requisito de extraer las señales desde un archivo.
3	• Procesamiento de las señales para hallar el desfase • Escritura de datos en un archivo de texto
4	• Permitir que el usuario seleccione solo una parte de la señal para ser procesada. • Realizar mejoras al procesamiento de la señal para hallar el desfase
5	• Adquisición de datos (configuración del hardware, conversión de los datos a la escala adecuada, creación y escritura de los archivos de medidas). • Almacenamiento de datos capturados
6	• Realizar procesamiento de datos adquiridos sin interrumpir la adquisición y posibilidad de procesamiento de datos al finalizar a captura. • Creación del ejecutable e instalador del software.

C. TEST IMPLEMENTADOS

Clase	**Test Implementados**	**Tiempo (s)**
Interfaz Datos	TestErrorWithEmptyString (TestClaseInterfazDatos):Este test verifica que el VI ArchivoToSignal genere error si el String esta vacio.	0,047
	TestOutWithEmptyString (TestClaseInterfazDatos):Este test prueba que si el string esta vacio el método ArchivoToSignal dara a la salida un arreglo vacio tambien.	0,078
	TestExtraccionDatosPotencia (TestClaseInterfazDatos):Este test prueba que el método ArchivoToSignal extraiga de forma adecuada el arrego de los valores de potencia.	0,062
	TestExtraccionDatosPotencia2 (TestClaseInterfazDatos):Este test prueba que el método ArchivoToSignal extraiga de forma adecuada el arrego de los valores de potencia.	0,062
	TestExtraccionDatosPotencia3 (TestClaseInterfazDatos):Este test prueba que el método ArchivoToSignal extraiga de forma adecuada el arrego de los valores de potencia.	0,062
	TestExtraccionDatosPotencia4 (TestClaseInterfazDatos):Este test prueba que el método ArchivoToSignal extraiga de forma adecuada el arrego de los valores de potencia.	0,047
	TestEscrituraString (TestClaseInterfazDatos):Este test prueba que el método DataArrayToString escriba los datos del string y de arreglo de entrada en el string a la salida.	0,062
	TestInformeestadoDatArrayToString (TestClaseInterfazDatos):Este VI verifica que el VI dataArrayToString cuando lea adecuadamente el archivo almacene un string para documentacion.	0,062
	TestExtraccionEncabezado (TestClaseInterfazDatos):Este test prueba que el método StringToStringAndMatrix escriba el texto del encabezado en el string de salida.	0,062
	TestBuscarDelimitador1Espacio (TestClaseInterfazDatos):Este Test verifica que el matodo BuscarDelimitador funcione adecuadamente para encontrar un espacio	0,062

Continúa en la siguiente página

Clase	Test Implementados	Tiempo (s)
	TestBuscarDelimitador2Espacios (TestClaseInterfazDatos):Este Test verifica que el matodo BuscarDelimitador funcione adecuadamente para encontrar un delimitador de 2 espacios	0,062
	TestBuscarDelimitador1Tab (TestClaseInterfazDatos):Este Test verifica que el matodo BuscarDelimitador funcione adecuadamente para encontrar un Tab	0,062
	TestExtraccionMatrizDatos (TestClaseInterfazDatos):Este test prueba que el método StringToStringAndMatrix escriba los datos de tiempo y potencia del string en la matriz de datos a la salida.	0,062
	TestEscrituraStringTiempoyPotencia (TestClaseInterfazDatos):Este test prueba que el método DataArrayToString escriba los dato del string y de arreglo de entrada en el string a la salida.	0,078
	TestEscrituraStringRecortada (TestClaseInterfazDatos):Este test prueba que el método DataArrayToString escriba los dato del string y de arreglo de entrada en el string a la salida.	0,062
	TestEscrituraStringSinMinYMax (TestClaseInterfazDatos):Este test prueba que el método DataArrayToString escriba los datos del string y de la matriz de entrada en el string a la salida verificando que si no se colocan valores minimo y maximo no se realice recorte de la matriz	0,062
	TestEscrituraStringSinDesfase (TestClaseInterfazDatos):Este test prueba que el método DataArrayToString escriba los datos del encabezado y el arreglo de tiempo y potencia aun cuando no tenga datos de desfase.	0,062
	TestConstruirEncabezado (TestClaseInterfazDatos):Este test prueba que el método ConstruirMatrizMedidayEncabezado contruyael encabezado	0,047
	TestReporteLogConstruirMatrizMediday Encabezado (TestClaseInterfazDatos):Este VI verifica que el VI ConstruirMatrizMediday Encabezado documente su funcionamiento por medio del cluster de error.	0,062
	TestFileStringToDataConfig (TestClaseInterfazDatos):Este test prueba que el VI FileStringToConfigData extraiga adecuadamente el nombre del canal fisico	0,062
	TestFileStringToDataConfig2 (TestClaseInterfazDatos):Este test prueba que el VI FileStringToConfigData extraiga adecuadamente la frecuencia	0,062

Continúa en la siguiente página

Clase	Test Implementados	Tiempo (s)
	TestDataConfigToStringFile (TestClaseInterfazDatos):Este test prueba que el VI DataConfigToFileString funcione adecuandamente creando un string con los dtos del canal fisico y frecuencia	0,125
ArchivoLog	TestPathValido (TestClaseArchivoLog):Este test prueba que el VI ConstruirPathLog entregue un path valido.	0,078
	TestInformeEstadoPathLog (TestClaseArchivoLog):Este test prueba que el VI ConstruirPathLog documente su estado pro medio del cluster de error	0,078
	TestErrorEntrada (TestClaseArchivoLog):Este test prueba que el VI ConstruirPathLog entregue un path valido.	0,078
	TestSalidaConEntradaError (TestClaseArchivoLog):Este test prueba que el VI ConstruirPathLog entregue un path nulo cuando recibe como entrada un error	0,062
ArchivoMZI	TestExtensionPass (TestClaseArchivoMZI):Este test prueba que el VI VerificarExtension no genera error cuando el path de entrada tiene extension .mzi	0,062
	TestExtensionFail (TestClaseArchivoMZI):Este test prueba que el VI VerificarExtension genera error cuando el path de entrada no tiene extension .mzi	0,062
	TestCrearEscrituraArchivoMedidas (TestClaseArchivoMZI):Este test prueba que se pueda realizar una escritura de un archivo MZI de forma correcta.	1,437
	TestInformeEstadoExtension (TestClaseArchivoMZI):Este test prueba que el VI VerificarExtension no genera error cuando el path de entrada tiene extension .mzi	0,078
	TestErrorEntrada (TestClaseArchivoMZI):Este test prueba que el VI VerificarExtension tenga como salida error cuando le ingresa un error.	0,109
	TestCrearPathValido (TestClaseArchivoMZI):Este test prueba que el VI CrearPathMZI entregue un path válido	0,125
Archivo	TestObjeto (TestClaseArchivo):Este Test verifica que la el objeto de los metodos de la clase Archivo tienen a la salida este un objeto de este tipo.	0,062
	TestErrorEntradaSelectPath (TestClaseArchivo):Este test comprueba que al colocar un error a la entrada del VI SelectPath, este no hace nada diferente a reflejar el error a la salida.	0,062
	TestErrorEntradaLeerArchivo (TestClaseArchivo):Este test comprueba que al colocar un error a la entrada del VI SelectPath, este no hace nada diferente a reflejar el error a la salida.	0,125

Continúa en la siguiente página

Clase	Test Implementados	Tiempo (s)
	TestErrorEntradaEscribirArchivo (TestClaseArchivo):Este test comprueba que al colocar un error a la entrada del VI SelectPath, este no hace nada diferente a reflejar el error a la salida.	0,062
	TestErrorEntradaModificarPath (TestClaseArchivo):Este test comprueba que al colocar un error a la entrada del VI SelectPath, este no hace nada diferente a reflejar el error a la salida.	0,125
	TestPropiedadTipoPath (TestClaseArchivo):Este test verifica que la propiedad del objeto archivo sea tipo path.	0,062
	TestLectura (TestClaseArchivo):Este test verifica con un ejemplo específico que la lectua de datos de un archivo con un path dado se realice adecuadamente y que los datos se almacenen debidamente en un String conectado a la salida.	0,062
	TestInformeEstadoLeerArch (TestClaseArchivo):Este test verifica que el registro del funcionamiento del metodo LeerArchivo se escriba en el cluster de error.	0,078
	TestErrorConPathVacio (TestClaseArchivo):Este test verifica que si el objeto que le ingresa al método leerArchivo no tiene un path almacenado, genera error a la salida.	0,062
	TestLecturaConsecutiva (TestClaseArchivo):Este test verifica con un ejemplo específico que la lectua de datos de un archivo con un path dado se realice repetidas veces de forma adecuada y que los datos se almacenen debidamente en un String conectado a la salida.	0,062
	TestNoPathEscritura (TestClaseArchivo):Este test verifica que si e objeto que le ingresa al método leerArchivo no tiene un path almacenado, genera error a la salida.	0,062
	TestEscrituraStringVacio (TestClaseArchivo):Este test verifica quese puedan realizar escrituras consecutivas sin marcar error.	0,094
	TestModificarPath (TestClaseArchivo):Este test prueba que el método ModificarPath funcione adecuadamente.	0,109
	TestInformeEstadoEscriArch (TestClaseArchivo):Este test verifica que el registro del funcionamiento del metodo Escribir archivo se escriba en el cluster de error.	0,219
	TestInformeEstadoModifPath (TestClaseArchivo):Este test verifica que el registro del funcionamiento del metodo ModificarPath se escriba en el cluster de error.	0,078
	TestEscrituraConsecutiva (TestClaseArchivo):Este test verifica que se puedan realizar escrituras consecutivas de archivos adecuadamente	0,156
	TestLecturaArchivoVacio (TestClaseArchivo):Este test verifica que el VI LeerArchivo notifique al usuario si el archivo con el path almacenado esta vacio.	0,109

Continúa en la siguiente página

Clase	Test Implementados	Tiempo (s)
	TestInformeEstadoModificarPath (TestClaseArchivo):Este test prueba que el método ModificarPath funcione adecuadamente.	0,125
	TestEscrituraConsecutivaDelMismoArchivo (TestClaseArchivo):Este test verifica que se escriba adecuadamente el mismo archivo consecutivamente.	0,297
	TestEscrituraArchivo (TestClaseArchivo):Este test verifica que se escriba un archivo adecuadamente	0,156
Adquisición Señales	TestErrorChannelEmpty (TestClaseAdqSeñales):Este test prueba que el VI AdquirirSeñal marque error si no tiene una canal fisico válido almacenado en el objeto de entrada.	1,062
	TestCanalNoValidoConfigAdq (TestClaseAdqSeñales): Este tst prueba que el VI ConfigAdq marque error si el canal que tene almacenado no es válido	0,203
	TestCanalNoValidoDetenerAdquisicion (TestClaseAdqSeñales):Este test prueba que el VI DetenerAdquisicion marque error si en el objeto de entrada no hay almacenado un canal fisico valido	0,297
	TestStringSalidaNoVacio (TestClaseAdqSeñales):Este Test prueba que el VI AdquirirSeñal construya un string válido	0,109
	TestObjetoSistAdq (TestClaseAdqSeñales):Este VI prueba que el objeto existe.	0,094
	TestCanalVacioConfigAdq (TestClaseAdqSeñales)	0,078
	TestFrecuenciaNoValidoConfigAdq (TestClaseAdqSeñales):Este test prueba que el VI ConfigAdq marque error cuando tiene un valor negativo o nulo en la propiedad frecuencia	0,125
Señal	TestNormalizacion (TestClaseSeñal):Este test prueba que la normalizacion se realice correctamente para un arreglo de datos de entrada dado	0,109
	TestRecortarSenal (TestClaseSeñal):Este test prueba que la normalizacion se realice correctamente para un arreglo de datos de entrada dado	0,094
	TestNormalizacionArregloVacio (TestClaseSeñal):Este test verifica que el Vi normalizar marque error si no tiene una señal de entrada en el objeto Señal	0,125
	TestInformeEstadoRecorteSenal (TestClaseSeñal):Este test verifica que el Vi RecortarSeñal informe al usuario de su funcionamiento a traves del cluster de error.	0,047
	TestRecortarArregloVacio (TestClaseSeñal):Este test verifica que el Vi RecortarSeñal informe al usuario cuando el arreglo del objeto señal se encuentre vacio	0,094

Continúa en la siguiente página

Clase	Test Implementados	Tiempo (s)
	TestInformeEstadoNormalizacion (TestClaseSeñal):Este test verifica que el Vi normalizar informe al usuario de su funcionamiento a traves del cluster de error.	0,047
	TestNormalizacionErrorEntrada (TestClaseSeñal):Este test prueba que la normalizacion se realice correctamente para un arreglo de datos de entrada dado	0,109
	TestRecortarErrorEntrada (TestClaseSeñal):Este test prueba que la normalizacion se realice correctamente para un arreglo de datos de entrada dado	0,094
SeñalMZI	TestDocumentaTransforHilbert (TestClaseSeñalMZI):Este Test prueba que el VI transformada Hilbert informe al usuario sobre su funcionamiento a traves del cluster de error.	0,156
	TestDocumentaConteoDiscontinuidades (TestClaseSeñalMZI):Este Test prueba que el VI ConteoDiscontinuidades informe al usuario sobre su funcionamiento a traves del cluster de error.	0,062
	TestDocumentaCalculoDesfase (TestClaseSeñalMZI):Este Test prueba que el VI CalculoDesfase informe al usuario sobre su funcionamiento a traves del cluster de error.	0,094
	TestCalculoDesfase (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final	0,359
	TestCalculoDesfase2 (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final	0,094
	TestCalculoDesfase3 (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final	0,094
	TestCalculoDesfase4 (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final	0,109
	TestCalculoDesfase5 (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final	0,125
	TestEscalarSenalConReferencia (TestClaseSeñalMZI):Este test verifica que el VI EscalarSeñalRespectoReferencia fuencione de forma adecuada	0,109

Continúa en la siguiente página

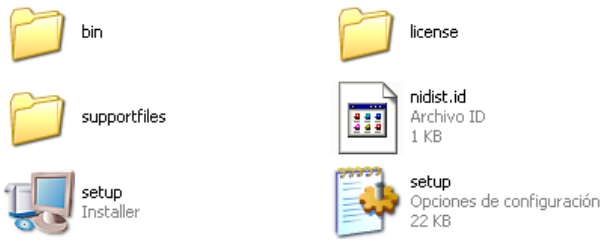
Clase	Test Implementados	Tiempo (s)
	TestErrorConObjetoSinSenal (TestClaseSeñalMZI):Este Test prueba que el VI Normalizar marque error si se conecta un objeto tipo señalMZI a la entrada el cual no tiene datos de señal.	0,109
	TestInterpolacion (TestClaseSeñalMZI):Este test prueba que el vi interpolacion funcione adecuadamente	0,062
	TestDocumentaInterpolacion (TestClaseSeñalMZI):Este Test prueba que el VIInterpolacion informe al usuario sobre su funcionamiento a traves del cluster de error.	0,078
	TestDocumentaConteoPuntosCrucex0 (TestClaseSeñalMZI):Este Test prueba que el VI ConteoPuntosEntreCucesPorCero informe al usuario sobre su funcionamiento a traves del cluster de error.	0,125
	TestCalculoDesfase6 (TestClaseSeñalMZI):Este test verifica que el sistema pueda calcular el desfase de señales dadas comparando con valores dados por el usuario final, Verificando que realice correcciones a la transformada Hilbet cuanod la señal tiene ruido al rededor de 0.	0,266
	TestInterpolacionSinCrucesPorCero (TestClaseSeñalMZI):Este test prueba que el VI interpolación No interpole si la señal tiene longitud menor a pi/2 (no tiene al menos 2 cruces por cero)	0,172

D. MANUAL DE USUARIO

D.1. INSTALACIÓN DEL SOFTWARE

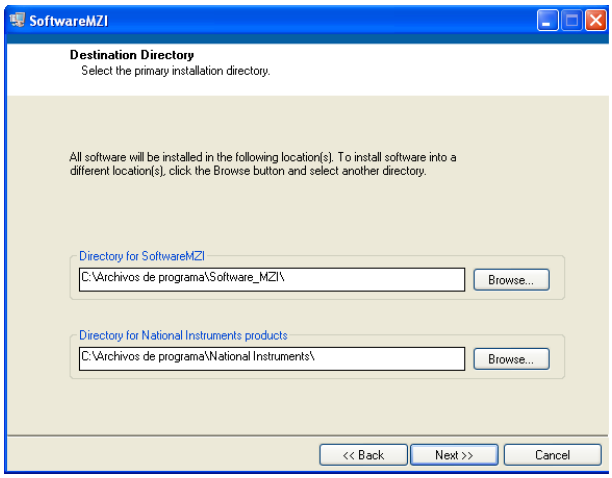
En la carpeta del instalador de la figura 49 se selecciona el Setup.

Figura 49. Carpeta del instalador del software



En la siguiente ventana (figura 50) se selecciona el directorio donde se instalará el software y se selecciona *Next*.

Figura 50. Proceso de instalación: Seleccionar directorio



Luego se acepta la licencia y se continua con el botón *Next*.

La siguiente ventana (Figrura 52) confirma el software que será instalado.

Con el último *Next* se inicia el proceso de instalación hasta que salga el anuncio de la figura 54. En esta ventana se selecciona *Finish* y con esto ha concluido el proceso de instalación.

En el directorio seleccionado para la instalación se crea una carpeta con los archivos del la figura 55. Para ejecutar el programa se puede seleccionar el ejecutable *SoftwareMZI* de esta carpeta o seleccionar el programa con la ruta Inicio → Programas → SoftwareMZI → SoftwareMZI.

Después de ejecutar el programa por primera vez, al interior de la carpeta *data* se crean las carpetas *ArchivoLog* y *MedidasMZI* como se muestra en la figura 56. Adicionamente se encuentra el archivo de configuracion de la adquisición con los datos de canal y frecuencia como lo ilustra la figura 57.

Figura 51. Proceso de instalación: Aceptar la licencia

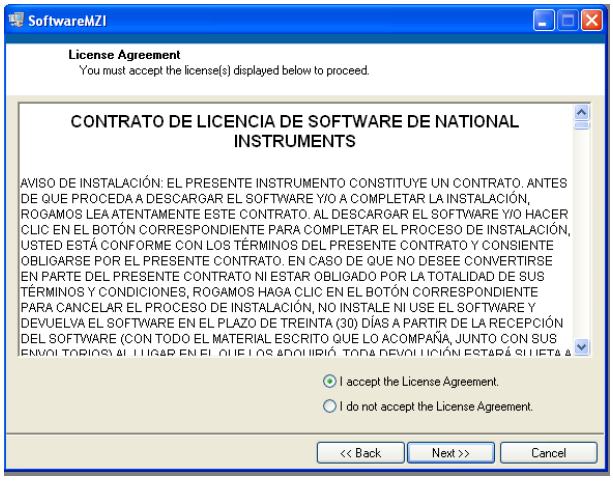


Figura 52. Proceso de instalación: confirmación del software a instalar

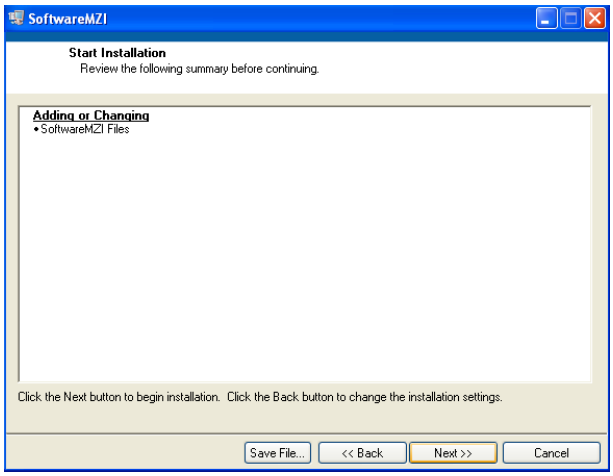
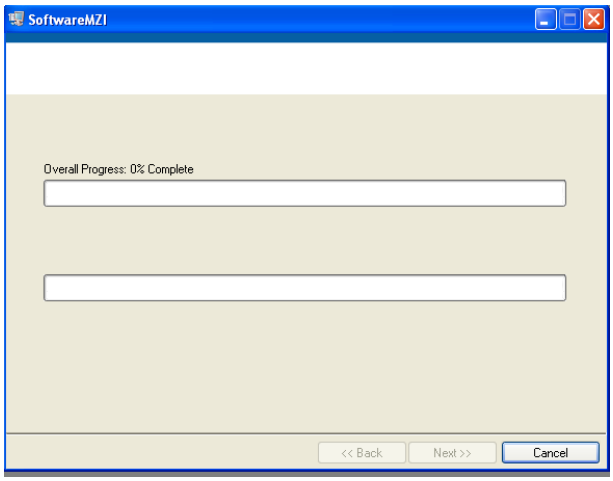


Figura 53. Proceso de instalación



Por otra parte en la carpeta *ArchivoLog* se almacenan los archivos de registro de funcionamiento los cuales tienen la estructura de la figura 58.

En la carpeta *MedidasMZI* se almacenan los archivos de los datos adquiridos, los cuales tienen

Figura 54. Final del proceso de instalación

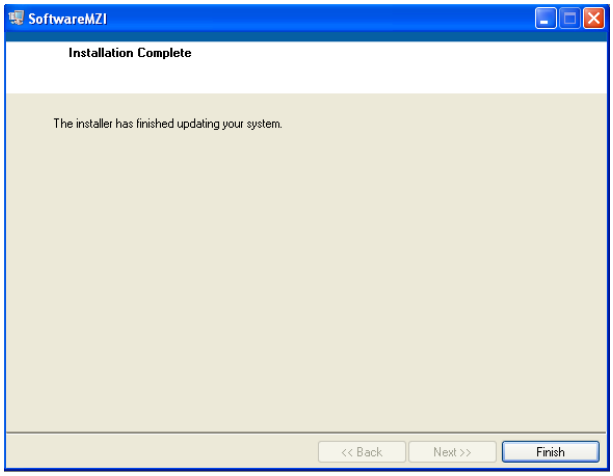
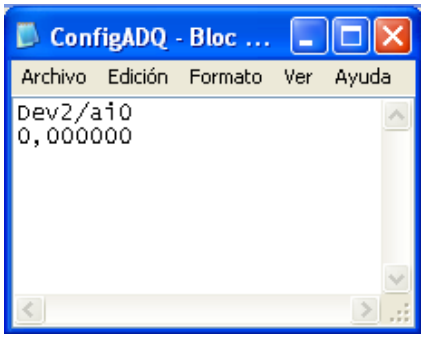


Figura 55. Contenido de la carpeta donde se instaló el software

Figura 56. Contenido de la carpeta *data*

Figura 57. Archivo de configuración



la estructura de la figura 59, Es decir que tiene un encabezado compuesto por las tres primeras líneas, seguido por las dos columnas de datos: Tiempo y Amplitud.

Figura 58. Archivo de registro de funcionamiento

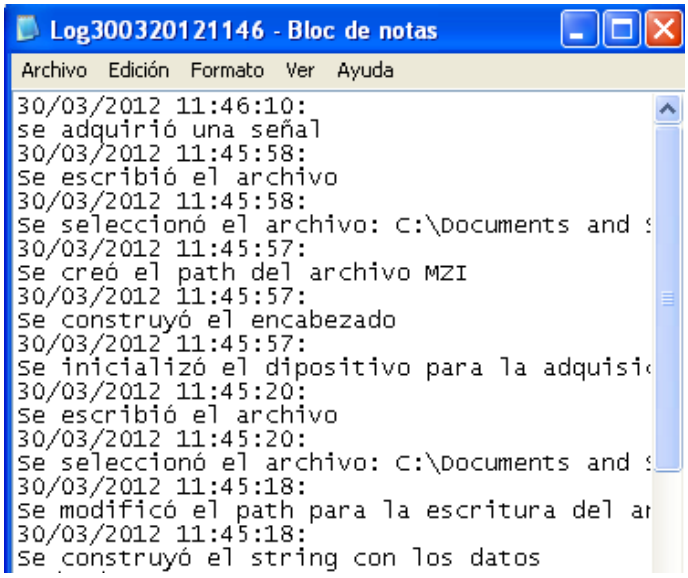
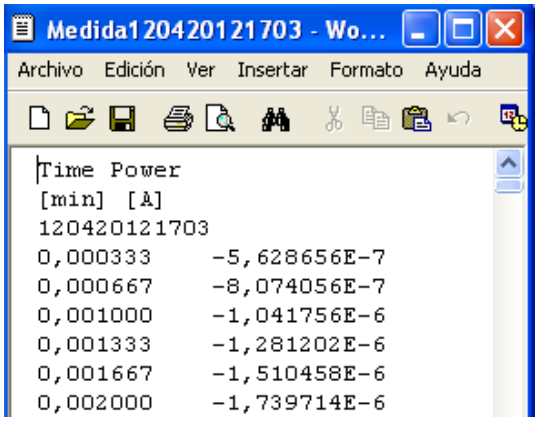


Figura 59. Archivo de medida



D.2. INTERFAZ DE USUARIO

La interfaz de usuario del programa se ilustra en las figuras 60 y 61 las cuales muestran las pestañas correspondientes al procesamiento y a la adquisición de señales respectivamente. Las partes que componen la interfaz de usuario se describen a continuación:

1. Pestaña de Procesamiento
2. Cuadro de gráficas donde se muestra la señal cuando se carga desde un archivo y al seleccionar calcular desfase se muestra la señal MZI y el Desfase.
3. Botón que despliega una ventana para seleccionar el archivo y cargar la señal del MZI.
4. Botón para realizar el cálculo de desfase, al terminar el procesamiento despliega una ventana para seleccionar el archivo en el cual se almacenarán los resultados.
5. Botón que muestra la señal cargada desde archivo completa.
6. Botón que despliega una ventana para seleccionar el archivo que contiene la señal que se desea usar como referencia. Esta señal se debe cargar después de cargar la señal MZI a procesar.
7. Indicador que se enciende al emplear una señal de referencia.
8. Herramientas del gráfico que permiten seleccionar una parte de la señal.
9. Indicador numérico que muestra el valor final del desfase en múltiplos de 2π al realizar el

procesamiento.

10. Botón que detiene el software y cierra la ventana.
11. Cuadro de texto donde se muestran los mensajes de funcionamiento del software.

Figura 60. Interfaz de Usuario del software (Pestaña de Procesamiento)

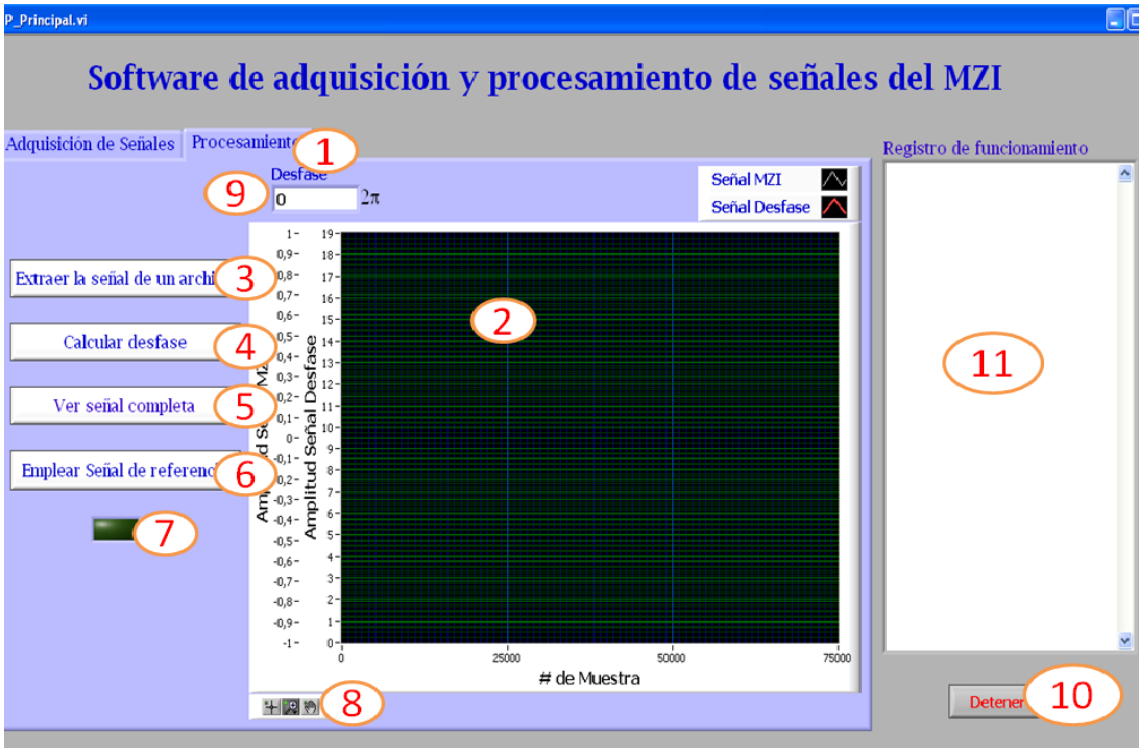
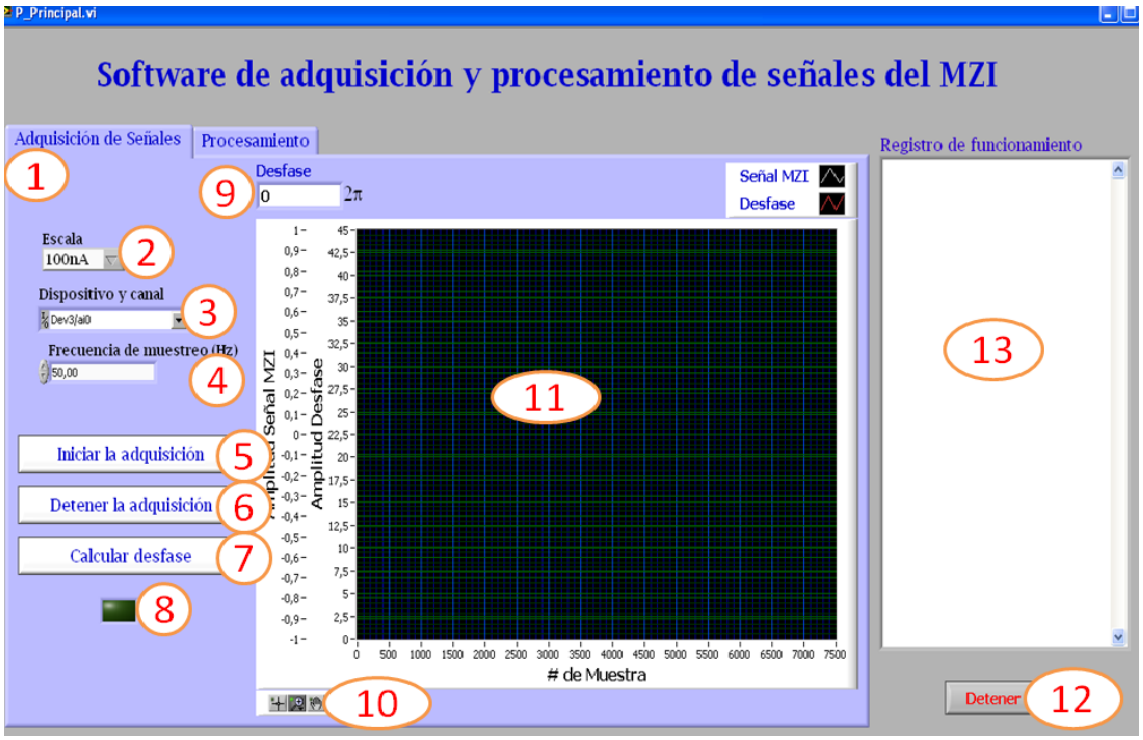


Figura 61. Interfaz de Usuario del software (Pestaña de adquisición)



1. Pestaña para adquisición de datos.
2. Selector de escala.
3. Control para seleccionar el dispositivo y canal de la tarjeta de adquisición.
4. Control para fijar la frecuencia de muestreo.
5. Botón para iniciar la adquisición, despliega una ventana para seleccionar el nombre del archivo en el cual se almacenarán los datos.
6. Botón para detener la adquisición.
7. Botón para calcular el desfase en cualquier momento durante la adquisición.
8. Indicador luminoso que se enciende cuando se estan capturando datos.
9. Indicador numérico que muestra el valor final del desfase en múltiplos de 2π al realizar el procesamiento.
10. Herramientas del gráfico que permiten seleccionar una parte de la señal.
11. Cuadro de gráfico donde se muestran los datos adquiridos.
12. Botón que detiene el software y cierra la ventana.
13. Cuadro de texto donde se muestran los mensajes de funcionamiento del software.

D.3. ADQUISICIÓN DE UNA SEÑAL

Para realizar la adquisición de una señal siga los pasos descritos a continuación:

1. Seleccionar la pestaña *Adquisición de señales*
2. Seleccionar el canal físico de la tarjeta de adquisición, la frecuencia de muestreo. Existen unos valores por defecto pero el usuario los puede modificar.
3. Seleccionar la escala para almacenar los datos dependiendo de la escala del amplificador físico del montaje experimental
4. Presionar el botón *Iniciar Adquisición*
5. Seleccionar el nombre del archivoy el directorio para almacenar los datos. Por defecto el directorio es \data\MedidasMZI y el programa sugiere un nombre de archivo, sin embargo el usuario puede modificarlos.
6. En ese momento se enciende el indicador de adquisición y los datos aqduiridos se muestran en el cuadro de gráfico y se almacenan en el archivo.
7. Para detener la adquisición se preciona el botón *Detener la adquisición*
8. Los mensajes de funcionamiento del software se muestran en el cuadro de registro en la interfaz gráfica y tambien se alamacenana en el ArchivoLog al detener el software.

D.4. PROCESAMIENTO DE UNA SEÑAL

Para realizar el procesamiento de una señal siga los pasos descritos a continuación:

1. Seleccionar la pestaña *Procesamiento*
2. Presionar el botón *Extraer la señal de un archivo*
3. Seleccionar el archivo con la señal MZI (con extensioón .mzi)
4. Si se quiere emplear señal de referencia se debe presionar el botón *Emplear señal de referencia* y seleccionar el archivo correspondiente.
5. Seleccionar el botón *Calcular desfase*
6. Seleccionar el archivo donde se guardarán los resutados.
7. En ese momento se mostrarán en la interfaz la gráfica del desfase, el valor total y el registro de todas las operaciones realizadas por el software.

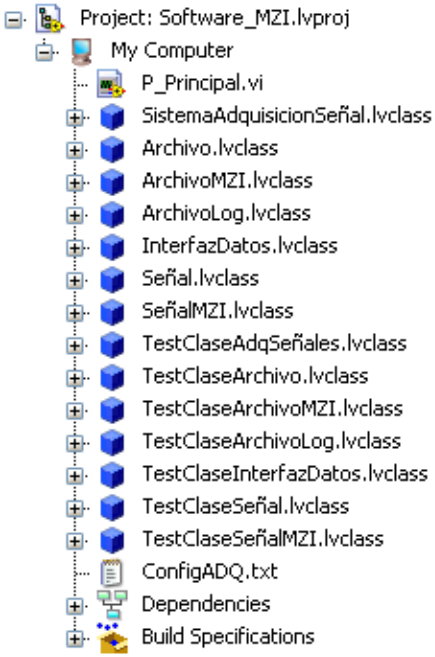
D.5. DETENER LA EJECUCIÓN DEL PROGRAMA

Para detener el software se selecciona el botón *Detener*, en ese momento se almacenan los últimos datos de frecuencia y canal en el archivo de configuración, se almacenan los mensajes de registro en el archivo Log (con un nombre creado basado en la fecha y hora) y se cierra el aplicativo.

E. CÓDIGO LABVIEW

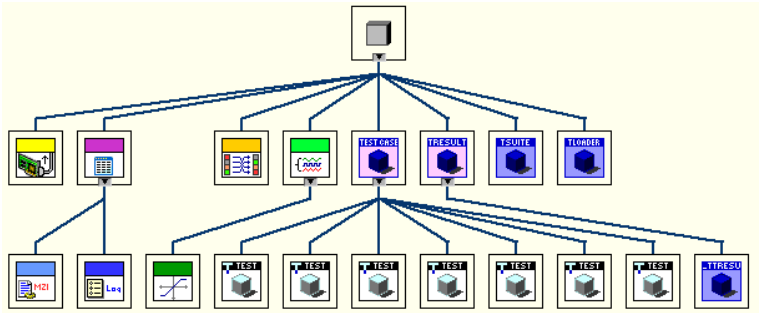
La figura 62 muestra el proyecto en LabVIEW que contiene todas las clases, incluso las que contienen las pruebas que se ejecutan mediante el VITester. Además se observa el programa principal que no pertenece a ninguna clase particular sino que es un VI del proyecto en general.

Figura 62. Proyecto en LabVIEW



Por otra parte la figura 63 muestra la jerarquía de las clases implementadas en LabVIEW la cual conserva la generalización plasmada en el diagrama de clases presentado anteriormente. En este sentido se observan las clases *ArchivoMZI* y *ArchivoLog* como subclases de la Clase Archivo, y la clase *SeñalMZI* que hereda de la clase *Señal*. Por su parte las clases *SistemaAdquisicionSeñal* y *InterfazDatos* no tienen relaciones de generalización.

Figura 63. Diagrama de clases



Las figuras 64, 65 y 66 muestran el código del programa principal con la orientación a eventos.

Figura 64. VI Programa principal

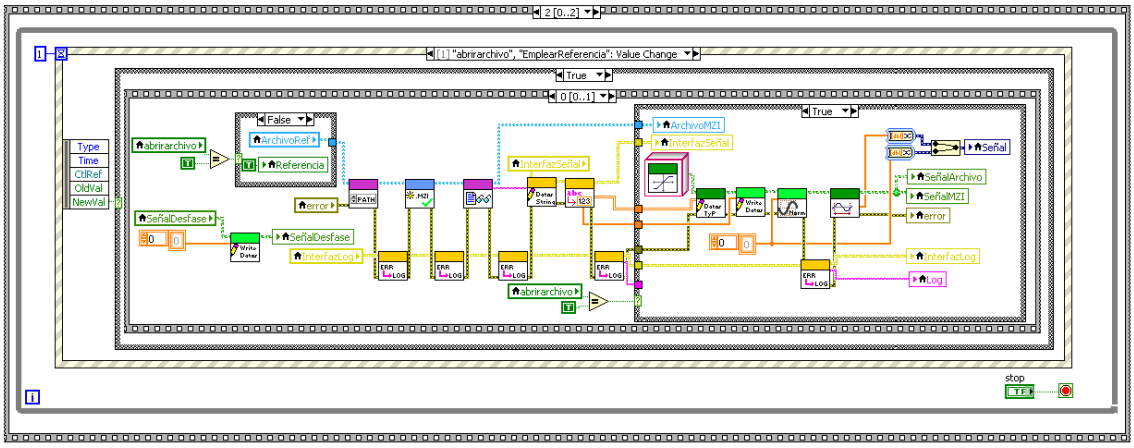
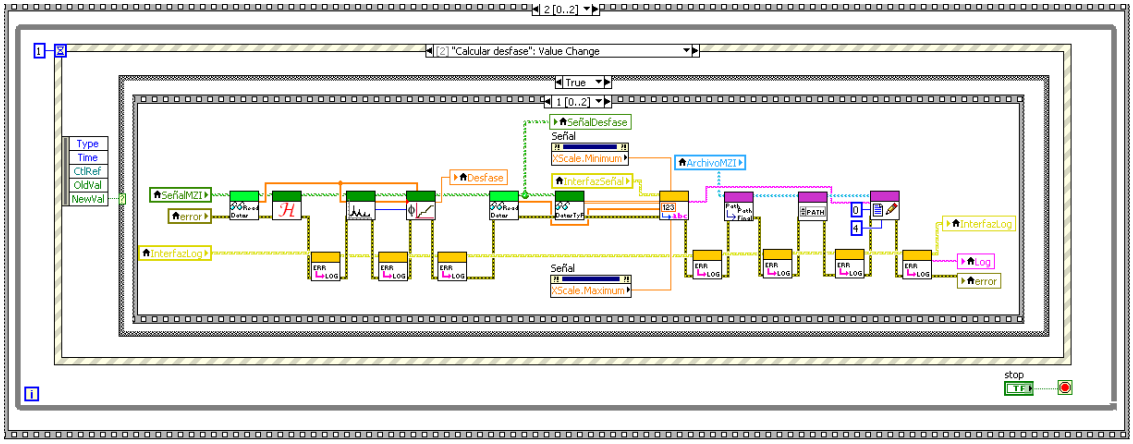


Figura 65. VI Programa principal



Las figuras 69, 67 y 68 muestran el código que se implementó para la adquisición de datos. El VI *ConfigurarAdquisición* selecciona una entrada de Voltaje de la tarjeta de adquisición, fija los límites de la señal de entrada en +/-10V, se configura el número de muestras por canal, la frecuencia de muestreo y el modo de adquisición como muestras contínuas.

El VI *DetenerAdquisicion* detiene el muestreo de datos y limpia el canal para que quede disponible para una nueva adquisición. Por otra parte el VI Adquirir Señal esta configurado para adquirir una muestra en formato *Double*.

El VI *EscribirArchivo* dependiendo de la entrada *operation* abre o crea el archivo del path almacenado en la propiedad del objeto de entrada y dependiendo de la entrada *from* puede escribir el archivo desde el principio o a partir del último dato.

El VI *DataArrayToString* toma el string que contiene el texto del encabezado, la matriz de tiempo y potencia y el arreglo de desfases para construir un solo string con esos datos, el cual se pueda escribir en un archivo.

El VI *FileStringToStringAndMatrix* extrae del String de entrada el texto del encabezado y una

Figura 66. VI Programa principal

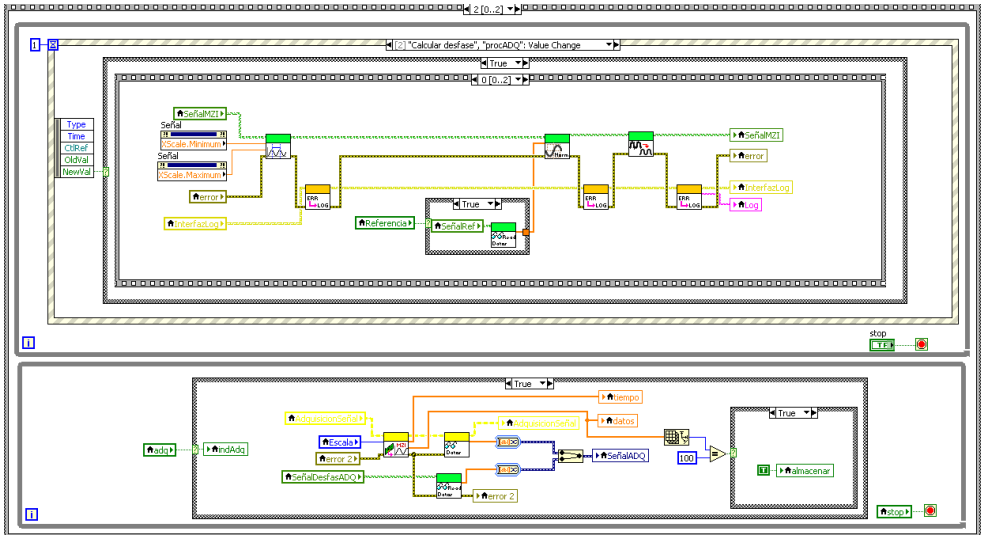


Figura 67. VI ConfigurarAdquisición

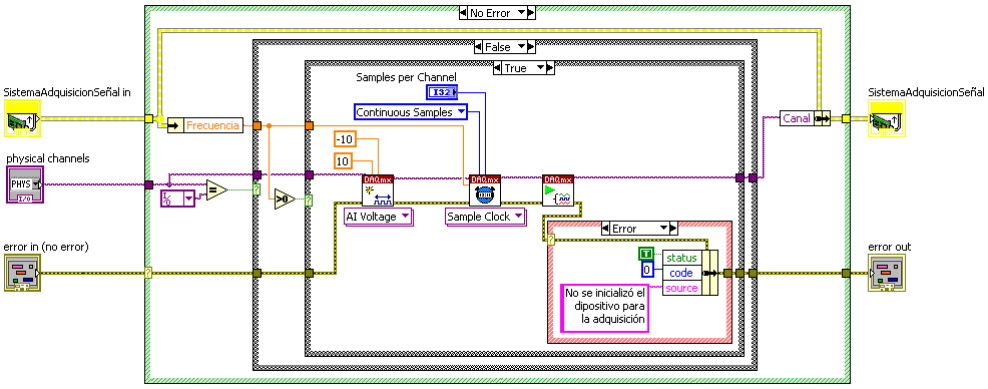
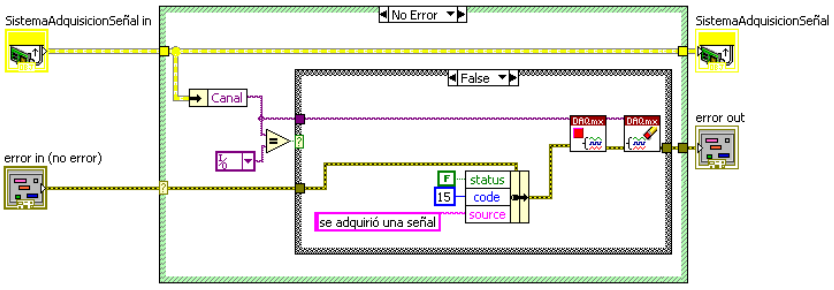


Figura 68. VI DetenerAdquisicion



matriz con los datos de tiempo y potencia.

La figura 73 muestra la sección de código encargada de la interpolación de la señal en la etapa de pre-procesamiento.

La figura 79 muestra la sección de código que suaviza la señal en la etapa de pre-procesamiento.

Figura 69. VI AdquirirSeñal

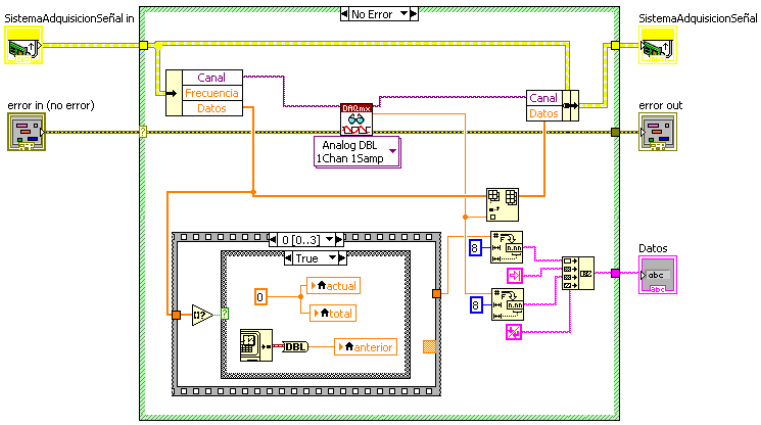


Figura 70. VI EscribirArchivo

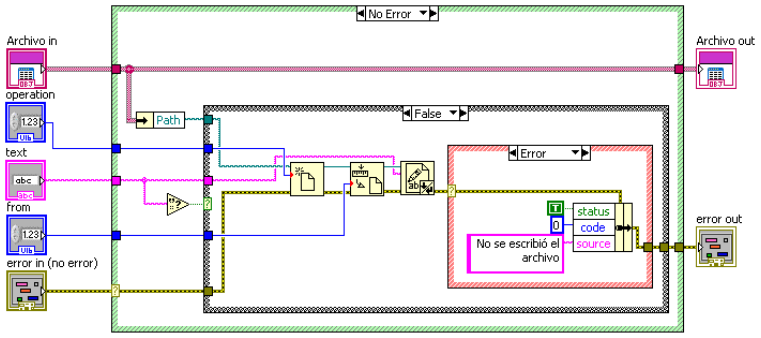
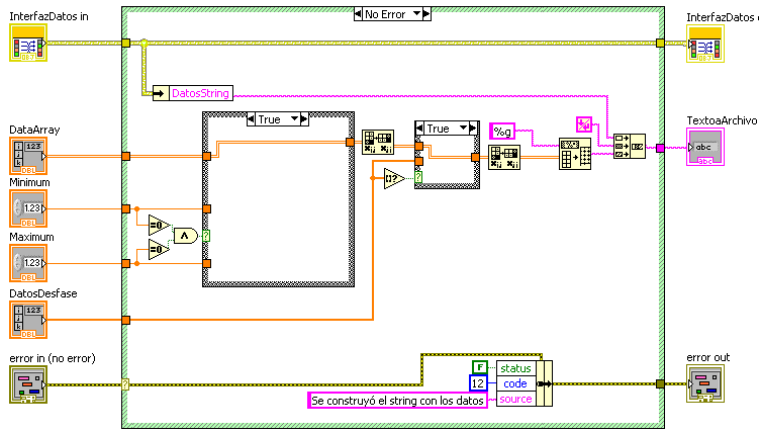


Figura 71. VI DataArrayToString



Los VIs *ConteoDiscontinuidades*, *TransformadaHilbert* y *CalculoDesfase* son los reponsables del cálculo de desfase. El primer VI se halla la derivada de la señal de entrada para detectar las discontinuidades de la señal y cuenta los picos.

El VI *TransformadaHilbert* calcula el argumento de la función analítica donde la parte real es

Figura 72. VI FileStringToStringAndMatrix

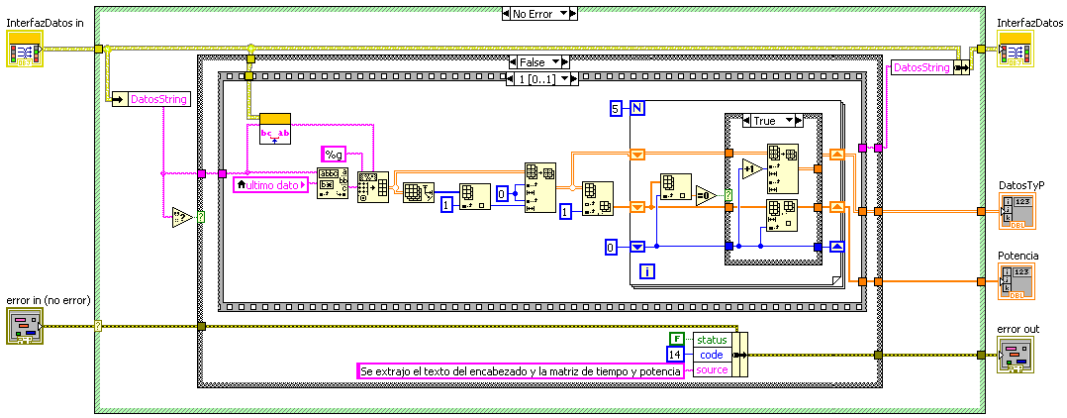


Figura 73. VI Interpolación

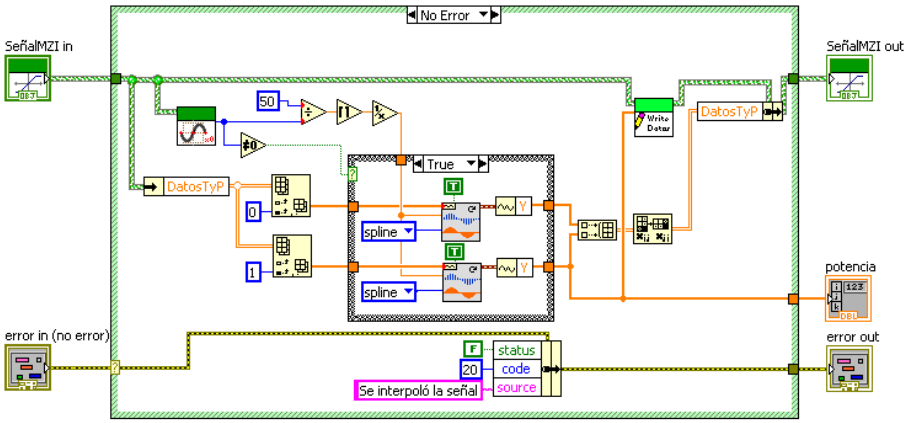
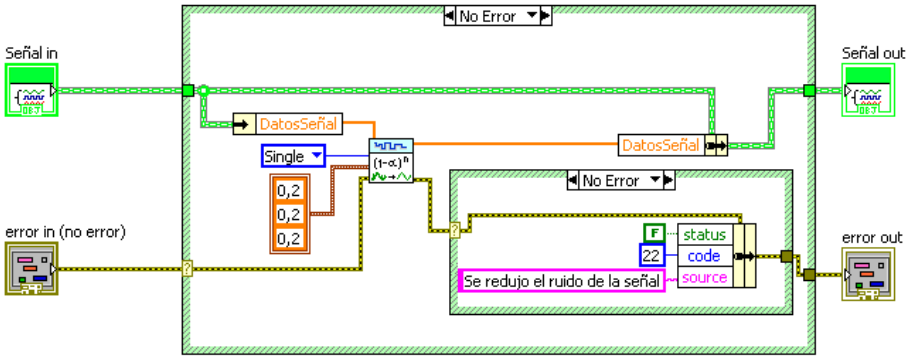


Figura 74. VI SuprimirRuido



la señal, y la parte compleja es su transformada Hilbert.

La sección de código de las figuras 77, 78 y 79 ilustra parte del procesamiento empleado para el cálcuo del desfase por el segundo método.

Figura 75. VI ConteoDiscontinuidades

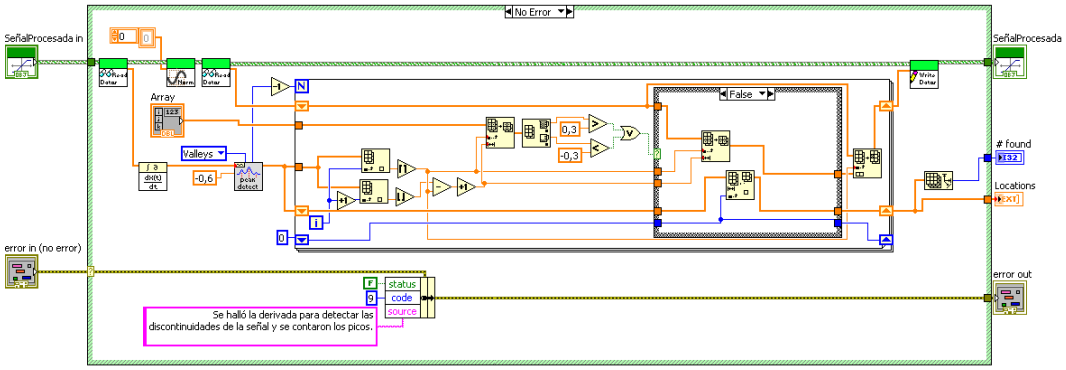


Figura 76. VI TransformadaHilbert

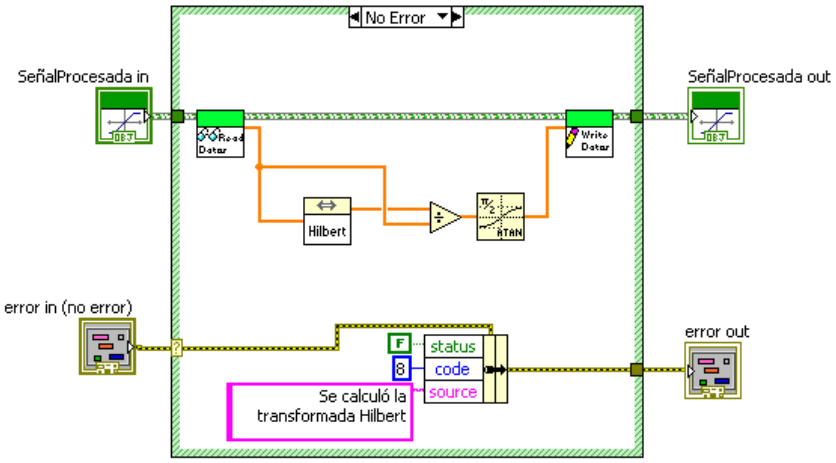


Figura 77. VI CalculoDesfase

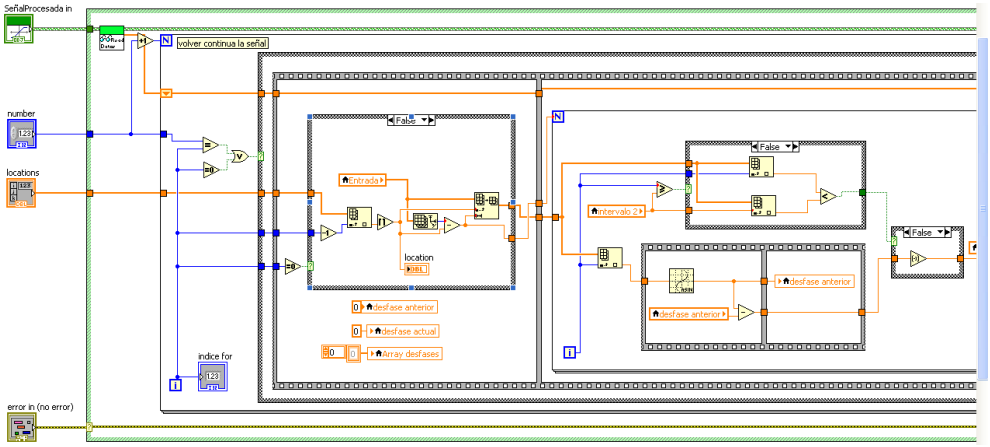


Figura 78. VI CalculoDesfase

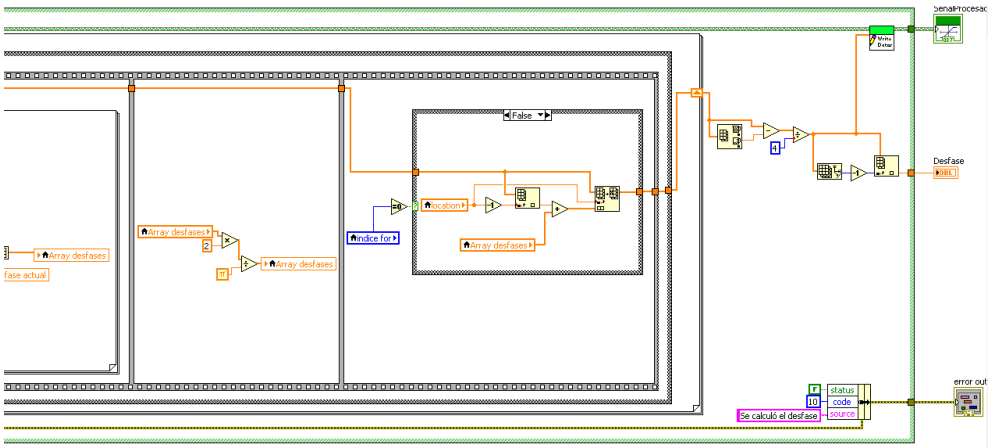
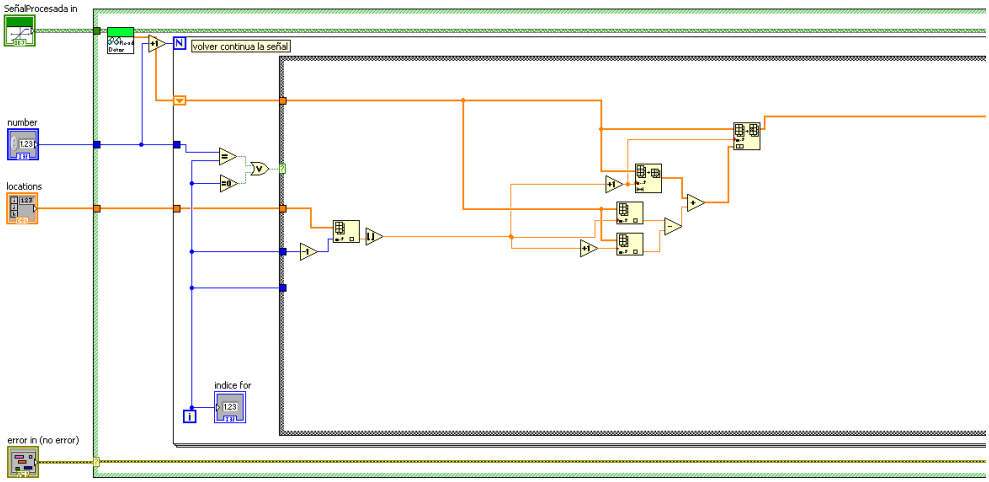


Figura 79. VI CalculoDesfase



F. HOJAS DE ESPECIFICACIONES DEL HARDWARE EMPLEADO