DISEÑO E IMPLEMENTACIÓN DE FILTROS ADAPTATIVOS MEDIANTE UN SISTEMA DE DESARROLLO PSOC (PROGRAMMABLE SYSTEM ON CHIP)

JENNY ANDREA SUAREZ BARBOSA

NATHALY MURCIA SEPÚLVEDA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES (E3T)

BUCARAMANGA

2013

DISEÑO E IMPLEMENTACIÓN DE FILTROS ADAPTATIVOS MEDIANTE UN SISTEMA DE DESARROLLO PSOC (PROGRAMMABLE SYSTEM ON CHIP)

JENNY ANDREA SUAREZ BARBOSA

NATHALY MURCIA SEPÚLVEDA

TRABAJO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR AL TÍTULO DE:

INGENIERO ELECTRÓNICO

DIRECTOR

JAIME G. BARRERO PEREZ

MAGISTER EN INGENIERÍA ELÉCTRICA

CODIRECTOR

SALVADOR PACHECO

INGENIERO ELECTRÓNICO

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES (E3T)

BUCARAMANGA

2013

AGRADECIMIENTOS

Mis más sinceros agradecimientos:
A Dios que ha sido mi guía,
A mis padres por su esfuerzo y apoyo incondicional,
Al profesor Jaime Barrero y al Ingeniero Salvador Pacheco por su colaboración y asesorías,
Al grupo CEMOS y a la Universidad Industrial de Santander por brindarme las herramientas necesarias en la formación académica como Ingeniera Electrónica,
A mi amiga y compañera de estudio Nathaly Murcia, por su apoyo, su dedicación y por compartir su amistad en estos cinco años,
Y a todos mis compañeros que hicieron posible finalizar este proyecto.

Jenny Andrea Suárez

En primer lugar agradezco a Dios por darme la vida, la salud y las capacidades para superar con éxito los retos académicos y personales encontrados durante mi formación. Especialmente a mis padres, Irma E. Sepúlveda y Jaime Murcia por brindarme todo el amor y el apoyo en cada paso de mi vida. A mi compañera de tantas batallas y experiencias Jenny A. Suarez, por no desfallecer en los peores momentos y ser esa persona paciente, dedicada y excepcional. A la Universidad Industrial de Santander, el grupo de investigación CEMOS por brindarme los recursos. A todos los docentes que me formaron, a nuestro director, Msc. Jaime Barrero y codirector, Ing. Salvador Pacheco por su colaboración. A mis compañeros por su amistad brindada. Y a mi amigo y cómplice de estos últimos años, mi novio, por su compañía y por hacerme un poco más feliz cada día.

Nathaly Murcia Seúlveda

DEDICATORIA

A Dios que siempre ha guiado mis pasos, a mis padres quienes siempre me apoyado a lo largo de mi vida, además sin sus consejos, motivación y amor infinito no hubiera podido estar en esta etapa de mi vida, a mis hermanas y toda mi familia, quienes me ayudado a culminar con éxito este proyecto.

Jenny Andrea Suárez

A Dios y especialmente a mis padres, por hacerme la persona que soy al enseñarme valores como el amor, el respeto, la responsabilidad, el coraje y la perseverancia, por tantos sacrificios, por la paciencia y por siempre estar ahí para mí. A mi familia, mis hermanas, tíos, etc, por confiar en mis capacidades y por todos su buenos deseos.

Nathaly Murcia Seúlveda

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

RESUMEN

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE FILTROS ADAPTATIVOS MEDIANTE UN SISTEMA DE DESARROLLO PSOC (PROGRAMMABLE SYSTEM ON CHIP)[1]

AUTORES: Nathaly Murcia Sepúlveda, Jenny Andrea Suarez Barbosa [2].

PALABRAS CLAVE: Filtro adaptativo, LMS (Least Mean Square), PSoC (Programmable System on Chip), Wiener.

DESCRIPCIÓN: El presente trabajo plantea el diseño e implementación de dos filtros adaptativos (Filtro Wiener y Filtro de minímos cuadrados –LMS-) con el objetivo de reducir el efecto de señales no deseadas que podrían residir o afectar a una señal de referencia o deseada. Dichos filtros adaptativos fueron programados en el sistema de desarrollo PSoC 5 por medio de su propia herramienta de software (*PSoC Creator*).

Mediante la herramienta computacional LabVIEW™ se desarrolló un generador virtual, los datos generados a partir de éste se usaron como entradas al sistema de filtrado y constituyeron la base del proceso llevado a cabo. Además, tanto los datos de entrada como los de salida obtenidos desde el sistema de desarrollo PSoC, se visualizan en una interfaz de usuario amigable o si es necesario de guardan para su posterior análisis. El envío y recepción de datos se realizó por medio de un dispositivo UART (Receptor Transmisor Asíncrono Universal) conectado a través de un adaptador de puerto serial a USB a un computador.

En este orden, se describen las pruebas realizadas con cada tipo de filtro adaptativo ante la entrada de una señal contaminada con una interferencia tipo seno o con un ruido blanco uniforme para finalmente obtener los resultados y comparar el desempeño de los filtros adaptativos diseñados e implementados.

[1] Proyecto de grado desarrollado en la modalidad de investigación.
[2] Facultad de ingenierías Físico-mecánicas, Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Grupo de investigación en Control, Electrónica, Modelado y Simulación (CEMOS). Director: Msc. Jaime Guillermo Barrero. Co-Director: Salvador Pacheco.

ABSTRACT

TITLE: DESIGN AND IMPLEMENTATION OF ADAPTATIVE FILTERS TROUGH A DEVELOMENT SYSTEM PSOC (PROGRAMMABLE SYSTEM ON CHIP)[3]

AUTHORES: Nathaly Murcia Sepúlveda, Jenny Andrea Suarez Barbosa[4].

KEY WORDS: Adaptative Filter, LMS (Least Mean Square), PSoC (Programmable System on Chip), Wiener.

DESCRIPTION: The current project presents the design and implementation of two adaptive filters (Wiener Filter and Least Mean Square –LMS- Filter) in order to reduce the effect of unwanted signals that could reside or could affect a desired or reference signal. These adaptive filters were programmed into PSoC 5 development system through its own software tool (PSoC Creator).

By the computational tool LabVIEW ™ was developed a virtual generator, data generated from it were used as inputs to the filter system and formed the basis of the process undertaken. In addition, both the input data and the output above obtained from the PSoC development system are displayed in a user-friendly interface or if it's necessary, they are saved to make a subsequent analysis. The sending and receiving of data was performed using a UART (Universal Asynchronous Receiver Transmitter) device connected through a serial port adapter to USB to a computer.

In this order, tests performed are described. Those testing are made with each type of adaptive filter in the presence of an input signal contaminated with an interference of sine type or with uniform white noise to finally get the results and compare the performance of the design and implemented filters.

[3] Degree's project developed with research purposes.
[4] Facultad de ingenierías Físico-mecánicas, Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Grupo de investigación en Control, Electrónica, Modelado y Simulación (CEMOS). Advisor: Msc. Jaime Guillermo Barrero. Co-Advisor: Salvador Pacheco.

INTRODUCCIÓN

Todo sistema electrónico que vincule el procesamiento de señales o datos es vulnerable a distintos tipos de señales que las perturban y distorsionan las salidas de estos sistemas, dificultando un análisis adecuado. Este tipo de señales tanto en teoría como en práctica no se pueden eliminar, por lo cual se utilizan diferentes técnicas que permitan disminuir su efecto y extraer una señal deseada. Una de estas técnicas consiste en la implementación de filtros.

Cuando las señales a filtrar son variantes en el tiempo o se encuentran en ambientes muy contaminados los filtros de coeficientes constantes típicamente empleados no logran ser suficientes, pues trabajan con frecuencias fijas y como su nombre lo indica con coeficientes fijos, limitando su funcionamiento y evitando que se procese adecuadamente la señal.

Los filtros adaptativos permiten de una forma dinámica, confiable y segura la reducción de señales no deseadas. Estos filtros se configuran con el propósito de reducir a cero la señal de error, para ello se requiere que la salida de filtro sea igual a la señal deseada, actualizando de una muestra a la siguiente sus coeficientes. Dependiendo de cómo se minimiza el error se puede implementar el filtro adaptativo.

Este proyecto tiene como objetivo principal el diseño e implementación de dos tipos de filtros adaptativos como los son los filtros Wiener y LMS (*Least Mean Square* o Mínimo Cuadrado Medio). Para ello, se trabajó con una tecnología de microcontroladores conocida como PSoC[5]. Esta tecnología cuenta con un dispositivo físico y con unas herramientas de software que permiten al usuario personalizar el diseño con diferentes tipos de periféricos analógicos y digitales

[5] De sus siglas en ingles *Programable System on Chip*.

pre-caracterizados, además de permitir una comunicación *hardware/software* de esclavo y maestro I2C y SPI, *full-speed* USB 2.0, hasta 4 UART *full-dúplex*, entre otras características.

El presente documento recopila el proceso, y los resultados obtenidos, durante la investigación. Éste se organiza en siete capítulos, de la siguiente manera:

En el capítulo 1 se presenta el marco teórico donde se define brevemente el funcionamiento de los filtros digitales, los filtros adaptativos y se realiza un estudio detallado de los filtros Wiener y LMS.

En el capítulo 2 se presenta la implementación de los algoritmos de los filtros. Se explica paso a paso cómo se diseñaron y los procedimientos utilizados para su ejecución.

En el capítulo 3 se presenta el sistema de desarrollo PSoC. Se muestran varias de sus características principales, así como el respectivo entorno de trabajo al usar una de sus herramientas de software. Finalmente, se describe detalladamente la implementación de los filtros adaptativos.

En el capítulo 4 se presentan pruebas hechas mediante el software MATLAB® para determinar alguno parámetro necesarios en la implementación de los filtros.

En el capítulo 5 se presenta la interfaz gráfica implementada en LabVIEW™, que a su vez cumple con la función de generador de señales.

En el capítulo 6 se presenta las pruebas realizadas, el análisis y la comparación los resultados obtenidos.

Finalmente, en los capítulos 7 y 8 se presentan observaciones y conclusiones con base en los resultados obtenidos, además de recomendaciones para la optimización y trabajo a futuro del trabajo realizado.

DESCRIPCION DEL TRABAJO DE GRADO

El proyecto dejará como resultado un dispositivo que permitirá realizar el filtrado adaptativo de señales generadas con recepción y transmisión de datos desde y hacia una aplicación desarrollada en LabVIEW™ para la visualización y el análisis de los datos. El proyecto brindará a la Universidad Industrial de Santander un instrumento de adquisición y tratamiento de señales desarrollado en una tecnología correspondiente al sistema de desarrollo PSoC.

OBJETIVO GENERAL

Implementar filtros adaptativos usando tecnología PSoC (Programable System on Chip) orientados a señales conocidas de baja frecuencia para facilitar su análisis.

OBJETIVOS ESPECÍFICOS

- Evaluar el sistema de desarrollo PSoC 5 verificando sus ventajas y desventajas para la programación de filtros adaptativos (Ver capítulo 3).
- Programar filtros adaptativos (LMS y Wiener) con la ayuda de un sistema PSoC (Ver capítulo 2, 3 y 4).
- Valorar el desempeño de filtros adaptativos en la eliminación de ruido de señales conocidas de baja frecuencia generadas (Ver capítulo 6).
- Obtener las señales filtradas en una interfaz de usuario (Ver capítulo 5).

1 FUNDAMENTOS TEORICOS

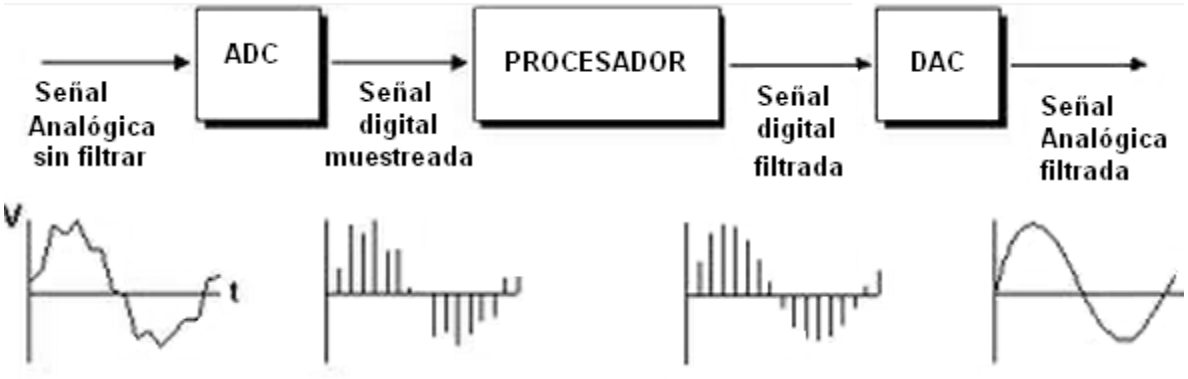
1.1 FILTROS

Un filtro es un sistema que, dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Hay dos tipos principales de filtros, filtros analógicos y filtros digitales, cada uno tiene diferente aspecto físico como funcionamiento.

Un filtro analógico emplea circuitos electrónicos con componentes discretos tales como resistencias, condensadores, amplificadores operacionales, entre otros, que sean requeridos para realizar el filtrado deseado. Algunas de sus aplicaciones son reducción de ruido, mejora de la calidad de vídeo, ecualizadores de gráficas.

Un filtro digital emplea un procesador digital que efectúa operaciones matemáticas en valores muestreados de la señal. El procesador puede ser de propósito general, tal como cualquier ordenador personal, un DSP (Procesador Digital de Señales) especializado. La señal de entrada analógica debe ser muestreada y digitalizada utilizando un ADC (Conversor Analógico Digital), el resultado corresponde a una serie de números binarios que representan los valores sucesivos muestreados. Estos son transferidos a un procesador encargado de realizarles operaciones matemáticas necesarias. Finalmente, si es necesario lo resultados son enviados por medio de un DAC (Conversor Digital Analógico) para devolver la señal a su forma analógica [8]. En la Figura 1, se observa el diagrama de bloques básico de un filtro digital [21].

Figura 1. Esquema básico filtro digital.

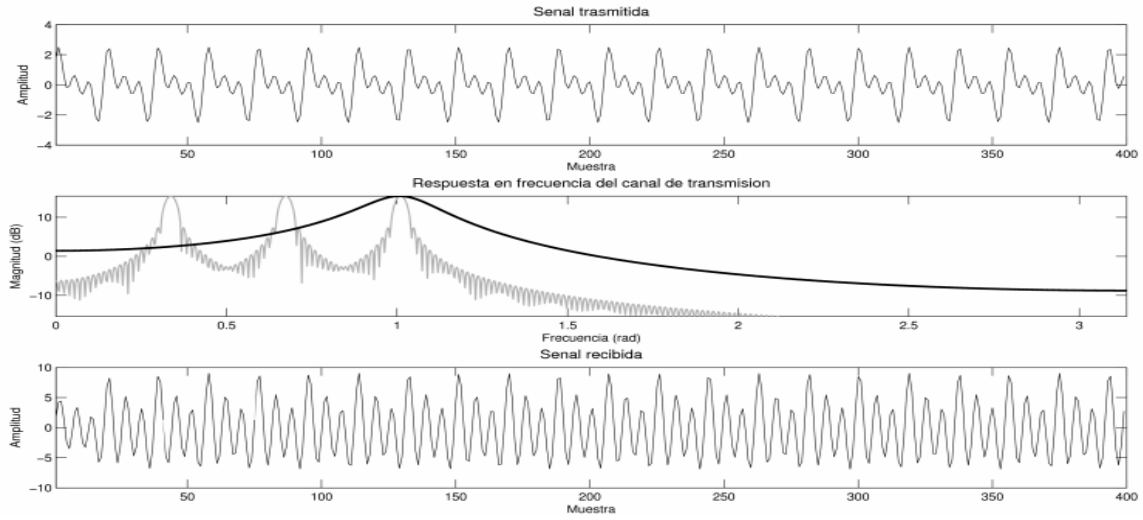


Fuente: Diseño de filtros digitales [21].

Algunas de las aplicaciones de un filtro digital:

- Separación de señales no deseadas (ruido, interferencias provenientes de otros sistemas) (Figura 2).
- Recuperación de señales distorsionadas.
- Síntesis de sonido: creación o modificación de señales para moldear espectros o formas de ondas y lograr el efecto auditivo deseado.
- Efectos de audio: coro, *flanger*, *phaser* o reverberación.

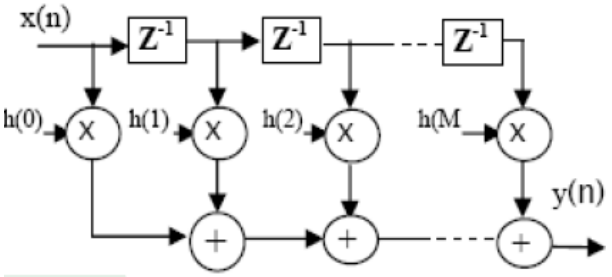
Figura 2. Ejemplo aplicación filtro digital.



Fuente: Introducción a la teoría de procesamiento digital de señales de audio [5].

Un filtro digital puede ser un sistema lineal e invariante en el tiempo (sistema LTI), al que se le aplica una señal con determinada conformación espectral y se obtiene a su salida una señal con una conformación espectral modificada de acuerdo a una función de transformación o función de conformación espectral [2]. Los filtros digitales LTI se pueden clasificar en dos grandes grupos, de respuesta infinita al impulso IIR y de respuesta finita al impulso FIR. Un esquema del filtro FIR se muestra Figura 3.

Figura 3. Filtro digital LTI del tipo FIR de orden M.



Fuente: Alba, et al [4].

La señal de salida se obtiene a través de su ecuación diferencial dada por:

$$y(n) = H^T x(n) \quad (1)$$

Donde

$$H^T = \{h(o)h(1)h(2) \dots h(M)\} \quad (2)$$

$$x(n) = \{x(n)x(n-1)x(n-2) \dots x(n-M+1)\}^T \quad (3)$$

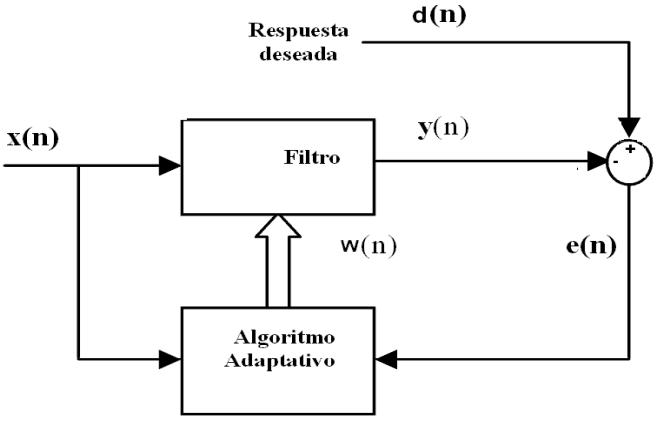
1.2 FILTROS ADAPTATIVOS

La mayoría de los filtros adaptativos de la actualidad se implementan por medio de sistemas digitales debido a que presentan algunas ventajas sobre filtros analógicos como alta confiabilidad, exactitud y menor sensibilidad a la temperatura

y el envejecimiento. Sin embargo, los filtros analógicos presentan una velocidad de convergencia mayor que la de los digitales [3], debido a que en estos últimos se requiere realizar un número elevado de operaciones de punto flotante [10].

La estructura de un filtro adaptativo es un sistema al que le llegan dos señales x(n) y e(n), ésta última se conoce como señal de error y es el resultado de la resta de la señal deseada, d(n), y la salida del filtro y(n). A los coeficientes o pesos (*weight*) del filtro se les llama w(n) y son los que multiplican a la entrada x(n) para obtener la salida como se muestra en la Figura 4. Dicho coeficientes se reprograman de una muestra a la siguiente a través de un algoritmo de adaptación y cambian su valor dinámicamente en el tiempo. Debido a esto, se hace necesario el uso de filtros estables por lo que se utilizan filtros FIR.

Figura 4. Estructura directa de un filtro adaptativo.



Fuente: Ruiz, J. [9].

$$y_n = w_n x_n \quad (4)$$

$$e_n = d_{n-} y_n \quad (5)$$

En principio el objetivo es hacer que la señal de error sea cero, para ello el sistema debe configurarse para que, a partir de la señal de entrada x(n) se genere una salida y(n) de forma que sea igual a la señal deseada d(n). Así como existen

diferentes formas de minimizar el error, existen diferentes tipos de filtros adaptativos [22]. Los algoritmos de adaptación más conocidos, son el algoritmo Wiener, el algoritmo RLS (*Recursive Least Square* – Mínimos Cuadrados Recursivos), y el algoritmo LMS (*Least Mean Square* – Mínimos Cuadrados Promediados).

Las características más importantes de un algoritmo adaptativo son:

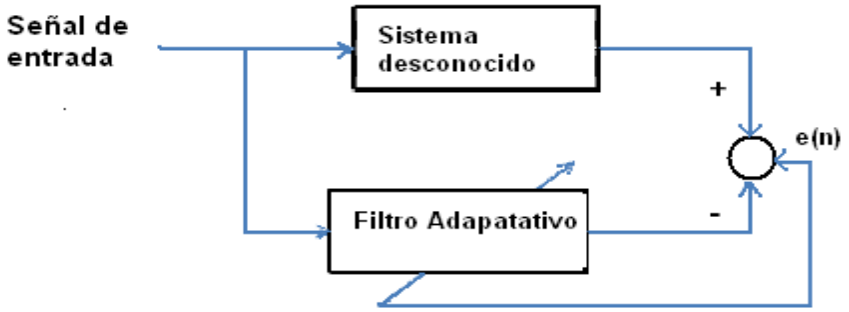
1. Complejidad computacional alta.
2. Mayor estabilidad.
3. Menor velocidad inicial de convergencia.
4. Consistencia de la velocidad de convergencia con variaciones en el condicionamiento de la señal.
5. Habilidad para seguir características variables en el tiempo.
6. Robustez (inmunidad) al ruido.

Un filtro adaptativo puede operar satisfactoriamente en un ambiente desconocido y seguir las variaciones en el tiempo de señales de entrada, lo cual lo hace un dispositivo sumamente poderoso para el procesamiento de señales y aplicaciones de control. Los filtros adaptativos pueden ser empleados en diversos campos como las comunicaciones, sismología, ingeniería biomédica, acústica, etc. Aunque estas aplicaciones pueden ser muy diferentes entre sí, tiene una característica en común: una señal de referencia y una señal deseada son utilizadas para estimar el error, el cual es usado para controlar los valores de los coeficientes del filtro. La principal diferencia entre las diversas aplicaciones del filtrado adaptativo consiste en la manera en que se extrae la señal deseada. En este contexto se puede distinguir cuatro clases de aplicaciones del filtrado adaptativo [3]:

1. Identificación. En esta clase de aplicaciones, el filtro adaptativo es usado para proveer un modelo lineal que representa un sistema desconocido. El sistema y el filtrado adaptativo tiene la misma señal de entrada. La salida

del sistema proporciona una señal deseada para el filtro adaptativo. En la Figura 5 se observa este tipo de aplicación.

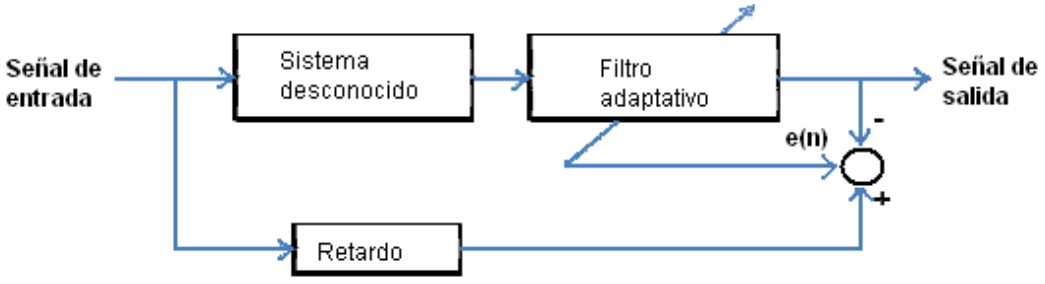
Figura 5. Clase de Filtro Adaptativo: Identificación.



Fuente: Duran, C [3].

2. Modelado inverso. La función del filtro adaptativo en este tipo de aplicaciones tiene la función de proveer un modelo que representa un sistema desconocido con ruido. En el caso de un sistema lineal ideal, el modelo tiene una función de transferencia igual el reciproco (inverso) de la función de transferencia del sistema desconocido. La Figura 6 muestra el diagrama de bloques de este tipo de aplicaciones.

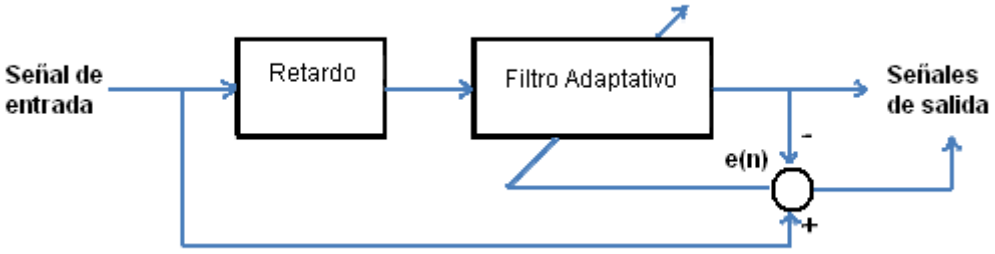
Figura 6. Clase de Filtro Adaptativo: Modelado Inverso.



Fuente: Duran, C [3].

3. Predicción. La función del filtro adaptativo es proveer la mejor predicción del valor presente de una señal aleatoria. El valor presente de la señal es la señal deseada para el filtro. Los valores pasados de la señal son la señal de entrada del filtro adaptativo. En la Figura 7 se puede observar cómo está configurada este tipo de aplicación.

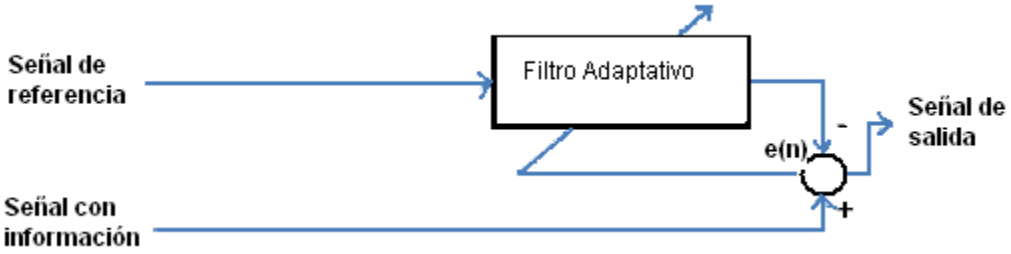
Figura 7. Clase de Filtro Adaptativo: Predicción.



Fuente: Duran, C [3].

4. Cancelación de Interferencia. El filtro adaptativo es usado para cancelar una interferencia desconocida contenida en una señal con información útil. La señal con información útil tiene la función de ser la señal deseada. Se emplea una señal de referencia como entrada del filtro adaptativo. Esta señal de referencia es obtenida de sensores instalados de tal manera que la señal con información útil sea débil o indetectable. La Figura 8 muestra el diagrama de esta serie de aplicaciones.

Figura 8. Clase de Filtro Adaptativo: Cancelación de Interferencia.



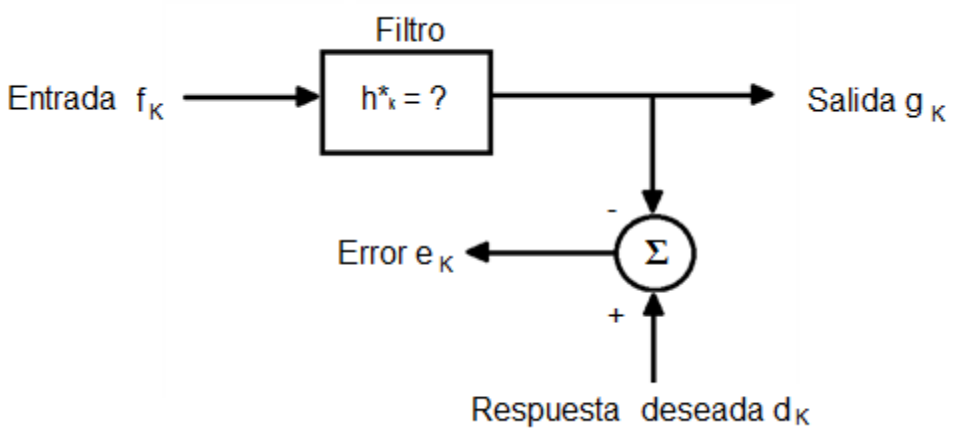
Fuente: Duran, C [3].

En general los filtros adaptativos tienen dos procesos básicos: un proceso de filtrado donde se obtiene un dato de salida en respuesta a un dato de entrada y un proceso adaptativo donde se ajustan los coeficientes del filtro de acuerdo a un algoritmo. Estos dos procesos trabajan de manera interactiva.

1.3 FILTRO WIENER

Los filtros Wiener son filtros lineales de mínimos cuadrados que pueden ser usados para predicción, estimación, interpolación, filtrado de señal y ruido, etc. Para diseñarlos se necesita tener un conocimiento previo de las propiedades estadísticas de la señal de entrada de tal modo que la señal de salida del filtro se aproxime lo más posible (en el sentido cuadrático medio) a una señal deseada. El problema reside en que el conocimiento de estas propiedades estadísticas generalmente no se puede obtener. En su lugar se usan filtros adaptativos, que hacen uso de los datos de entrada para aprender los datos estadísticos requeridos.

Figura 9. Filtro Wiener digital.



Fuente: Caizer, et al [11].

El filtro de Wiener es uno de los filtros lineales óptimos más importantes. En su forma más general, consiste en una señal de entrada, *x(n)*, una respuesta deseada, *d(n)*, y un filtro lineal de respuesta al impulso *h(n)*. Este filtro es alimentado por *x(n)* y produce a su salida *y(n)*. La diferencia entre la señal de salida del filtro *y(n*) y la señal deseada *d(n)* es el error de la estimación *e(n)* como se puede observar en la Figura 9 [17].

El objetivo del filtrado de Wiener es determinar la respuesta impulsional *h(n)* de forma que el error *e(n)* sea, en un sentido estadístico, "lo más pequeño posible". El criterio elegido es la minimización del valor cuadrático medio del error

$$x = E\{|e(n)|^2\} \quad (6)$$

Esta función de coste posee una dependencia de segundo orden con los coeficientes del filtro, y tiene un mínimo que determina los coeficientes óptimos, es decir, el mejor diseño del filtro.

Si se considera un filtro causal de longitud *p*, orden *p*-1, de respuesta impulsional *w(n)*, y función de red [16]:

$$\mathrm{W}(z) = \sum_{n=0}^{p-1} w(n) z^{-n} \quad (7)$$

Si *x(n)* es la entrada al filtro, la salida es la estimación de *d(n)*, obtenida como la convolución de *w(n)* con *x(n)*

$$\mathrm{W}(z) = \sum_{l=0}^{p-1} w(l) x(n-l) \quad (8)$$

Se debe encontrar los coeficientes que minimizan la función de coste

$$\xi = \mathrm{E}\{|e(n)|^2\} = \mathrm{E}\left\{\left|d(n) - \hat{d}(n)\right|^2\right\} \quad (9)$$

Por tanto, derivando x respecto a *w*(k)* e igualamos las derivadas a cero para los valores posibles de *k* (*k*= 0, 1, ..., *p*-1).

$$\frac{\partial \xi}{\partial \mathrm{w}^*(k)} = \frac{\partial}{\partial \mathrm{w}^*(k)} E\{e(n)e^*(n)\} = E\left\{e(n)\frac{\partial \mathrm{e}^*(n)}{\partial \mathrm{w}^*(k)}\right\} = 0 \quad (10)$$

Con

$$e(n) = d(n) - \sum_{l=0}^{p-1} w(l) x(n-l) \quad (11)$$

Entonces, la derivada de *e*(n)* es:

$$\frac{\partial \mathrm{e}^*(n)}{\partial \mathrm{w}^*(k)} = -x^*(n-k) \quad (12)$$

Y sustituyendo se obtiene el principio de ortogonalidad o teorema de proyección:

$$\mathrm{E}\{e(n)x^*(n-k)\} = 0 \quad ; \quad k = 0,1,\ldots,p-1 \tag{13}$$

Este teorema indica que x es mínimo y los coeficientes del filtro asumen sus valores óptimos cuando *e(n)* es ortogonal a cada muestra de entrada *x(n)* que es utilizada para el cálculo de la estimación. Como consecuencia, el error también es ortogonal a la salida del filtro. Este principio establece una condición suficiente y necesaria para la optimización.

Aplicando la expresión del error *e(n)* se tiene

$$E\{d(n)x^*(n-k)\} - \sum_{l=0}^{p-1} w(l)E\{x(n-l)x^*(n-k)\} = 0 \tag{14}$$

Con la suposición de que *x(n)* y *d(n)* son procesos estacionarios en sentido amplio (WSS), $E\{x(n-l)\,x^*(n-k)\} = r_x(k-l)$ y $E\{d(n)\,x^*(n-k)\} = r_{dx}(k)$, por tanto,

$$\sum_{l=0}^{p-1} w(l)r_x(k-l) = r_{dx}(k) \quad ; \quad k = 0,1,\ldots,p-1 \tag{15}$$

Se tiene un conjunto de *p* ecuaciones lineales con *p* incógnitas *w(k)*, para *k* = 0, 1, ..., *p*-1. En forma matricial, utilizando el hecho de que la secuencia de autocorrelación posee simetría conjugada, $r_x(k) = r_x^*(-k)$, se puede expresar las ecuaciones de *Wiener-Hopf* como

$$\begin{bmatrix} r_x(0) & r_x^*(1) & \ldots & r_x^*(p-1) \\ r_x(1) & r_x(0) & \ldots & r_x^*(p-2) \\ r_x(2) & r_x(1) & \ldots & r_x^*(p-3) \\ \ldots & \ldots & \ldots & \ldots \\ r_x(p-1) & r_x(p-2) & \ldots & r_x(0) \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \ldots \\ w(p-1) \end{bmatrix} = \begin{bmatrix} r_{dx}(0) \\ r_{dx}(1) \\ r_{dx}(2) \\ \ldots \\ r_{dx}(p-1) \end{bmatrix} \tag{16}$$

y en notación vectorial

$$R_x w = r_{dx} \quad (17)$$

donde R_x es la matriz de autocorrelación toeplitz hermítica p x p, w es el vector de coeficientes del filtro, y r_{dx} es el vector de correlación cruzada entre la señal deseada $d(n)$ y la señal recibida $x(n)$.

El error cuadrático medio puede obtenerse de la siguiente manera:

$$\xi_{min} = E\{|e(n)|^2\} = E\left\{e(n)\left[d(n) - \sum_{l=0}^{p-1} w(l)x(n-l)\right]^*\right\} = E\{e(n)d^*(n)\} - \sum_{l=0}^{p-1} w^*(l)E\{e(n)x^*(n-l)\} \quad (18)$$

Si los coeficientes w son la solución de las ecuaciones de *Wiener-Hopf*, $E\{e(n)\, x^*(n-k)\} = 0$ y se tiene el error cuadrático medio (e.c.m.) mínimo.

$$\xi_{min} = E\{e(n)d^*(n)\} = E\left\{\left[d(n) - \sum_{l=0}^{p-1} w(l)x(n-l)\right]d^*(n)\right\} \quad (19)$$

Tomando el valor esperado,

$$\xi_{min} = r_d(0) - \sum_{l=0}^{p-1} w(l)r_{dx}^*(l) \quad (20)$$

Y en notación vectorial

$$\xi_{min} = r_d(0) - r_{dx}^H w \quad (21)$$

Como $w = R_x^{-1} r_{dx}$, el e.c.m. mínimo puede expresarse en términos de la matriz de autocorrelación R_x y el vector de correlación cruzada r_{dx}:

$$\xi_{min} = r_d(0) - r_{dx}^H R_x^{-1} r_{dx} \quad (22)$$

De forma más general, este estudio puede realizarse considerando que los coeficientes del filtro pueden ser complejos, y así se puede expresar:

$$Rw_{opt} = p \quad (23)$$

$$z_{min} = r_d(0) - p^H w_{opt} = r_d(0) - {w_{opt}}^H R w_{opt} \quad (24)$$

Donde

$$x = [x(n)\ x(n-1)\ \dots\ x(n-p+1)]^T \quad (25)$$

$$R = E[xx^H] = \begin{bmatrix} r(0) & r(1) & \dots & r(p-1) \\ r^*(1) & r(0) & \dots & r(p-2) \\ \dots & \dots & \dots & \dots \\ r^*(p-1) & r^*(p-2) & \dots & r(0) \end{bmatrix} = \begin{bmatrix} r(0) & r(1) & \dots & r(p-1) \\ r(-1) & r(0) & \dots & r(p-2) \\ \dots & \dots & \dots & \dots \\ r^*(-p+1) & r^*(-p+2) & \dots & r(0) \end{bmatrix} \quad (26)$$

$$p = E[xd^*] = [r_{xd}(0)\ r_{xd}(-1) \dots r_{xd}(-p+1)]^T \quad (27)$$

1.3.1 FUNCIÓN CORRELACIÓN

En procesamiento de señales, la correlación cruzada (o a veces denominada "covarianza cruzada") es una medida de la similitud entre dos señales, frecuentemente usada para encontrar características relevantes en una señal desconocida por medio de la comparación con otra que sí se conoce.

Dadas dos secuencias de señal reales *x(n)* y *y(n)* cada una con energía finita. La correlación cruzada de *x(n)* y *y(n)* es una secuencia $r_{xy}(l)$ definida como:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l) \qquad l = 0, \pm 1, \pm 2, \ldots \tag{28}$$

O, equivalente a

$$r_{yx}(l) = \sum_{n=-\infty}^{\infty} x(n+l)y(n) \qquad l = 0, \pm 1, \pm 2, \ldots \tag{29}$$

El índice *l* es el parámetro de desplazamiento y los subíndices *xy* en la secuencia de correlación cruzada $r_{yx}(l)$ indican las secuencias que están correlacionadas. El orden de los subíndices indica la dirección en que una secuencia es desplazada respecto a la otra [6].

1.3.2 MATRIZ TOEPLITZ

En el álgebra lineal, una matriz de Toeplitz es una matriz cuadrada con todas sus diagonales de izquierda a derecha paralelas numéricamente. Una matriz de Toeplitz presenta la siguiente estructura [19]:

$$T = \begin{pmatrix} a & b & c & d & k \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ j & h & g & f & a \end{pmatrix} \tag{30}$$

En términos matemáticos:

$$\forall\, a_{ij} \in T \rightarrow a_{i,j} = a_{i+1,j+1} \tag{31}$$

1.3.3 DESCOMPOSICIÓN CHOLESKY

Una matriz *A* simétrica y positiva definida puede ser factorizada de manera eficiente por medio de una matriz triangular inferior y una matriz triangular superior. Para una matriz no singular la descomposición *LU* lleva a considerar una descomposición de tal tipo *A = LU;* dadas las condiciones de *A*, simétrica y definida positiva, no es necesario hacer pivoteo, por lo que ésta factorización se hace eficientemente y en un número de operaciones la mitad de *LU* tomando la forma $A=LL^T$, donde *L* (la cual se puede considerar como la raíz cuadrada de *A*) es una matriz triangular inferior donde los elementos de la diagonal son positivos [21].

Para resolver un sistema lineal *Ax = b* con *A* simétrica definida positiva y dada su factorización de Cholesky $A=LL^T$, primero se debe resolver $Ly = b$ y entonces resolver $L^T x=y$ para lograr *x*.

Para encontrar la factorización $A=LL^T$, bastaría ver la forma de *L* y observar las ecuaciones que el producto derecho conduce al igualar elementos:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ l_{31} & l_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ 0 & 0 & \cdots & l_{n3} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix} \quad (32)$$

Así se obtendría que:

$$a_{11} = {l_{11}}^2 \qquad \rightarrow \; l_{11} = \sqrt{a_{11}}$$

$$a_{21} = l_{21} l_{11} \qquad \rightarrow l_{21} = \frac{a_{21}}{l_{11}}, \ldots \; l_{n1} = \frac{a_{n1}}{l_{11}}$$

$$a_{22} = {l_{21}}^2 + {l_{22}}^2 \qquad \rightarrow l_{22} = \sqrt{\left(a_{22} - {l_{21}}^2\right)}$$

$$a_{32} = l_{31} l_{21} + l_{32} l_{22} \rightarrow l_{32} = \frac{(a_{32} - l_{31} l_{21})}{l_{22}}, etc$$

Y de manera general, para *i=1,…,n* y *j=i+1,…,n*:

$$l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)} \quad (33)$$

$$l_{ji} = \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}\right) \Big/ l_{ii} \quad (34)$$

Entonces, ya que *A* es simétrica y definida positiva, se puede asegurar que los elementos sobre la diagonal de *L* son positivos y de esta manera los restantes elementos son reales.

1.3.4 FACTORIZACIÓN LU E INVERSIÓN DE MATRICES

La descomposición LU proporciona un método eficiente para calcular la matriz inversa, la cual tiene un número importante de aplicaciones en la práctica de la ingeniería [20]. La eliminación de Gauss está diseñada para resolver sistemas de ecuaciones algebraicas lineales,

$$[A]\{X\} = \{B\} \quad (35)$$

La ecuación (35) puede redondearse para dar,

$$[A]\{X\} - \{B\} = 0 \quad (36)$$

Suponga que la ecuación (36) se puede expresar como un sistema triangular superior:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} \quad (37)$$

Se observa que se usa la eliminación para reducir el sistema a una forma triangular superior. La ecuación (37) puede también ser expresada en notación matricial y ordenada para dar

$$[U]\{X\} - \{D\} = 0 \quad (38)$$

Ahora, suponga que existe una matriz diagonal inferior con números 1 sobre la diagonal,

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \quad (39)$$

Que tiene la propiedad de que cuando se multiplica por ecuación (38), el resultado es la ecuación (36). Esto es,

$$[L]\big\{[U]\{X\} - \{D\}\big\} = [A]\{X\} - \{B\} \quad (40)$$

Si esta ecuación se cumple, de las reglas de multiplicación de matrices se obtendría

$$[L][U] = [A] \quad (41)$$

y

$$[L][D] = [B] \quad (42)$$

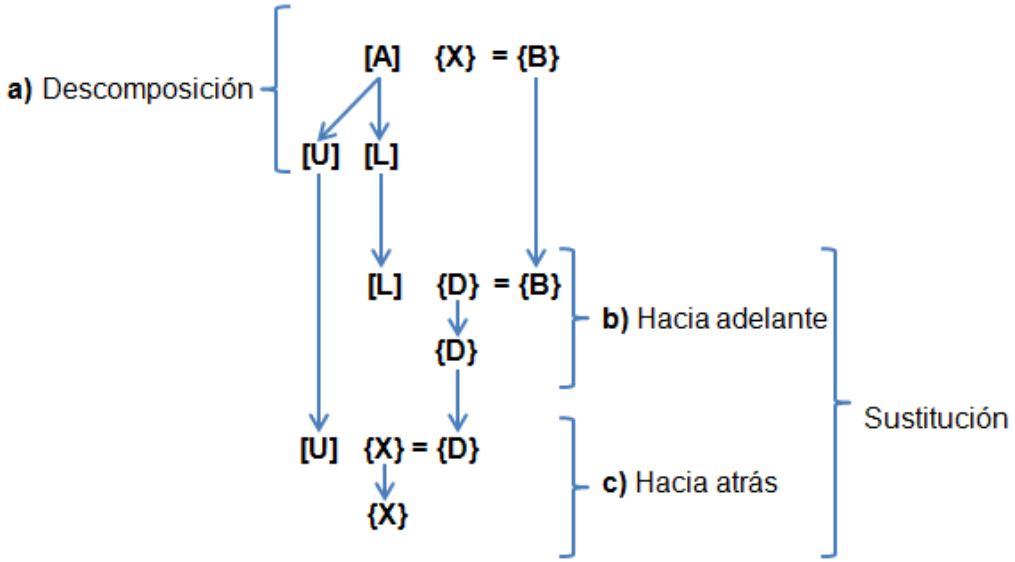
Una estrategia de dos pasos (véase

Figura **10**) para obtener soluciones tiene su base en las ecuaciones (38), (41) y (42):

1. *Paso de descomposición LU.* [*A*] se factoriza o “descompone” en matrices triangulares inferior [*L*] y superior [*U*].

2. *Paso de sustitución.* [*L*] y [*U*] se usan para determinar una solución {X} para un lado derecho {B}. este paso, a su vez, se divide en dos. Primero, la ecuación (42) se usa para generar un vector intermedio {*D*} por sustitución hacia adelante. Luego, el resultado es sustituido en la ecuación (38), la que se resuelve por sustitución hacia atrás para {*X*}.

Figura 10. Procedimientos en la descomposición LU.



Fuente: Chapra, et al [20].

La inversa se puede calcular en forma de columna-por-columna para generar soluciones con vectores unitarios que están en el lado derecho de la matriz como constantes. Por ejemplo, si las constantes del lado derecho de la ecuación tienen un número “1” en la primera posición y ceros en las otras,

$$\{b\} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \quad (43)$$

La solución resultante será la primera columna de la matriz inversa. En forma similar, si se usa un vector unitario que tiene 1 en el segundo renglón, el resultado será la segunda columna de la matriz inversa.

De esta manera, cuando se tiene la descomposición LU, es decir, expresada una matriz A en las matrices triangulares inferior *L* y superior *U*, se procede a determinar la primera columna de la matriz inversa efectuando el procedimiento de solución. Primero, usando la sustitución hacia adelante con un vector unitario (con el número 1 en el primer renglón) como el vector de términos independientes. Así, de acuerdo con la ecuación (42), el sistema con diagonal inferior se puede fijar como

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \quad (44)$$

Y resolviendo con la sustitución hacia adelante para $\{D\}^T = [d_1\ d_2\ d_3]$. Este vector se puede usar en el lado derecho con la forma de la ecuación (37),

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} \quad (45)$$

la cual puede resolverse por sustitución hacia atrás para $\{X\}^T = [x_1\ x_2\ x_3]$, lo cual es la primera columna de la matriz,

$$[A]^{-1} = \begin{bmatrix} x_1 & 0 & 0 \\ x_2 & 0 & 0 \\ x_3 & 0 & 0 \end{bmatrix} \quad (46)$$

Para determinar la segunda columna se debe realizar el mismo procedimiento expuesto pero esta vez la ecuación (42) se formula como

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \quad (47)$$

En esta medida, cuando se varía la posición del "número 1" en la ecuación (42) y se realiza el procedimiento indicado, se encuentran las respectivas columnas de la matriz inversa.

1.3.5 TEOREMA DE CONVOLUCIÓN

Cuando se trata de hacer un procesamiento digital de señal no tiene sentido hablar de convoluciones aplicando estrictamente la definición ya que solo disponemos de valores en instantes discretos de tiempo. Es necesario, pues, una aproximación numérica. Para realizar la convolución entre dos señales, se evaluará el área de la función $x(\tau) * h(t-\tau)$. Para ello, se debe disponer de muestreos de ambas señales en los instantes de tiempo nt , que se llamará $x[k]$ y $h[n-k]$ (donde n y k son enteros). El área es, por tanto [1],

$$y[n] = \sum_{k=-\infty}^{\infty} t.x[k].h[n-k] = t.\left[\sum_{k=-\infty}^{\infty} x[k].h[n-k]\right] \quad (48)$$

La convolución discreta se determina por un intervalo de muestreo $t = 1$:

$$[22] \quad y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k] \qquad (49)$$

1.4 FILTRO LMS (LEAST MEAN SQUARE)

El filtro LMS consiste en la obtención de coeficientes que permiten adquirir el valor esperado mínimo del cuadrado de la señal de error, definida como la diferencia entre la señal deseada y la señal producida a la salida del filtro. Una característica importante del LMS es su simplicidad pues no requiere el uso de funciones complejas [15].

El algoritmo LMS es un algoritmo de filtrado lineal adaptativo que, en general, consiste de dos procesos básicos:

- Un *proceso de filtrado*, que involucra, el cómputo de la salida de un filtro lineal en respuesta a una señal de entrada, y la generación de una estimación del error mediante la comparación de esta salida con la señal deseada.
- Un *proceso adaptativo*, que involucra, el ajuste automático de los parámetros del filtro de acuerdo al error estimado.

Si fuera posible obtener medidas exactas del vector gradiente $\nabla\zeta(n)$ en cada iteración *n*, y se dispusiera del parámetro μ adecuadamente elegido, el vector de pesos del filtro convergería a la solución óptima de Wiener. Pero en la realidad no dispone de estas medidas exactas del vector gradiente, ya que no se conoce la matriz de autocorrelación de la señal de entrada al filtro ni el vector de correlación

cruzada entre esta señal de entrada al filtro y la respuesta deseada. Por tanto, el vector gradiente ha de ser estimado a partir de los datos.

La manera más sencilla de estimar el vector gradiente consiste en sustituir en la ecuación

$$\nabla\zeta(n) = -2p + 2Rw(n) \quad (50)$$

R y *p* por estimaciones instantáneas constituidas por los valores de señal de entrada al filtro y por la respuesta deseada,

$$\hat{R}(n) = x(n)x^H(n) \quad (51)$$

$$\hat{p}(n) = x(n)d(n) \quad (52)$$

Y se obtiene así una estimación instantánea del vector gradiente

$$\hat{\nabla}\xi(n) = -2x(n)d^*(n) + 2x(n)x^H(n)w(n) \quad (53)$$

Esta estimación puede verse como el resultado de aplicar el operador gradiente ∇ al error instantáneo $|e(n)|^2$. Si sustituimos la estimación del vector gradiente en la ecuación de actualización de los pesos utilizada en el algoritmo *Steepest-Descent.*

$$w(n+1) = w(n) - (1/2)\,\mu(-\nabla\zeta(n)) \quad (54)$$

Se obtiene la relación recursiva siguiente:

$$\mathrm{w}(n+1) = w(n) + \mu x(n)[d^*(n) - x^H(n)w(n)] \quad (55)$$

Se puede expresar esta solución con las relaciones siguientes [22]:

Salida del filtro $$y(n) = w^H(n)x(n) \quad (56)$$

Estimación del error $$e(n) = d(n) - y(n) \quad (57)$$

Adaptación de los pesos del filtro $$w(n+1) = w(n) + \mu x(n)e^*(n) \quad (58)$$

En cada iteración, la actualización de un peso requiere dos multiplicaciones complejas y una suma. Por tanto, para un filtro de *p* coeficientes, se realizan 2*p*+1 multiplicaciones complejas y 2*p* sumas complejas por iteración. Es decir, la carga computacional es del orden de *p* (O(*p*)).

Se puede observar que la estimación del error, definida por las ecuaciones (56) y (57), depende del vector de pesos actual *w(n)*, y el término μ *x(n) e*(n)* representa la corrección aplicada al vector de pesos *w(n)*. El algoritmo iterativo ha de comenzar con un valor inicial *w(*0*)*. También se puede apreciar que en cada iteración se debe conocer el valor de *x(n)*, *d(n)* y *w(n)*.

El algoritmo LMS, para un filtro de orden *M*, puede resumirse de la siguiente manera:

- Parámetros, M, orden del filtro. μ, tamaño del paso.
- Inicialización, si se dispone de información acerca del vector de coeficientes del filtro, usarla para elegir un valor apropiado para $\hat{\mathbf{w}}(0)$. En caso contrario, usar $\hat{\mathbf{w}}(0) = 0$.
- Datos, dados la señal de entrada en el instante n

$$u(n) = [u(n), u(n-1), \dots, u(n-M+1)]^T \quad (59)$$

$d(n)$: Señal deseada a la salida del filtro a calcular

$$\hat{\mathrm{w}}(\mathrm{n}+1) = [\hat{\mathrm{w}}_0(\mathrm{n}+1), \hat{\mathrm{w}}_1(\mathrm{n}-1), \ldots, \hat{\mathrm{w}}_{M-1}(\mathrm{n}+1)]^T \quad \textbf{(60)}$$

Estimación del vector de coeficientes del filtro en el instante *n* + 1.

Para $n = 0,1,2,\ldots$, calcular:

Señal error $e(n) = d(n) - \hat{\mathrm{w}}^{\boldsymbol{H}}(\mathrm{n}).\mathbf{u}(\mathrm{n})$ **(61)**

Adaptación de los coeficientes del filtro $\hat{\mathrm{w}}(\mathrm{n}+1) = \hat{\mathrm{w}}(\mathrm{n}) + \mu\, e^*(n)\boldsymbol{u}(n)$ **(62)**

El superíndice *T* denota trasposición, el superíndice *H* denota traspuesta conjugada, el asterisco denota conjugación y $\hat{\mathrm{w}}^{\boldsymbol{H}}.\boldsymbol{u}(n)$ es la salida del filtro, que se calcula como el producto interno entre el vector de coeficientes del filtro $\hat{\mathrm{w}}(\mathrm{n})$, cuyos componentes suelen llamarse *pesos* o *weights*, y el vector de datos de entrada al filtro $\mathrm{u}(\mathrm{n})$.

En este filtro los pesos se ajustan hasta que el error cuadrático medio sea minimizado. El factor de convergencia μ determina el error mínimo local, así como la velocidad de convergencia, siendo directamente proporcional a la velocidad de convergencia e inversamente proporcional al error cuadrático medio mínimo; es decir, cuando μ es grande la velocidad de convergencia es alta pero el error cuadrático medio es más alto, y cuando μ es pequeño, aunque la velocidad de convergencia se reduce, el error cuadrático medio es menor [13].

Para mantener la estabilidad, el factor de convergencia debe cumplir lo siguiente:

$$|1 - 2\mu\lambda_i| < 1 \;\; ; \;\; i = 1,2,3,\ldots N-1 \quad \textbf{(63)}$$

De otra forma se puede escribir como:

$$0 < \mu < \frac{1}{\lambda_{max}} \quad (64)$$

Donde λ corresponde a los valores propios de la matriz de autocorrelación de la entrada.

2 IMPLEMENTACIÓN DE ALGORITMOS DE FILTROS ADAPTATIVOS

Los filtros adaptativos usados para la cancelación de ruido debieron implementarse mediante la elaboración algoritmos y códigos de programación. Inicialmente dichos filtros se elaboraron en el software MATLAB®. Por medio de este software se facilitó la programación de las diferentes operaciones empleadas en el diseño de los filtros, especialmente en el caso del filtro Wiener, pues el software MATLAB® provee funciones prediseñadas tales como correlación cruzada, autocorrelación, creación de la matriz toeplitz, división de matrices y filtrado digital (ver anexo A).

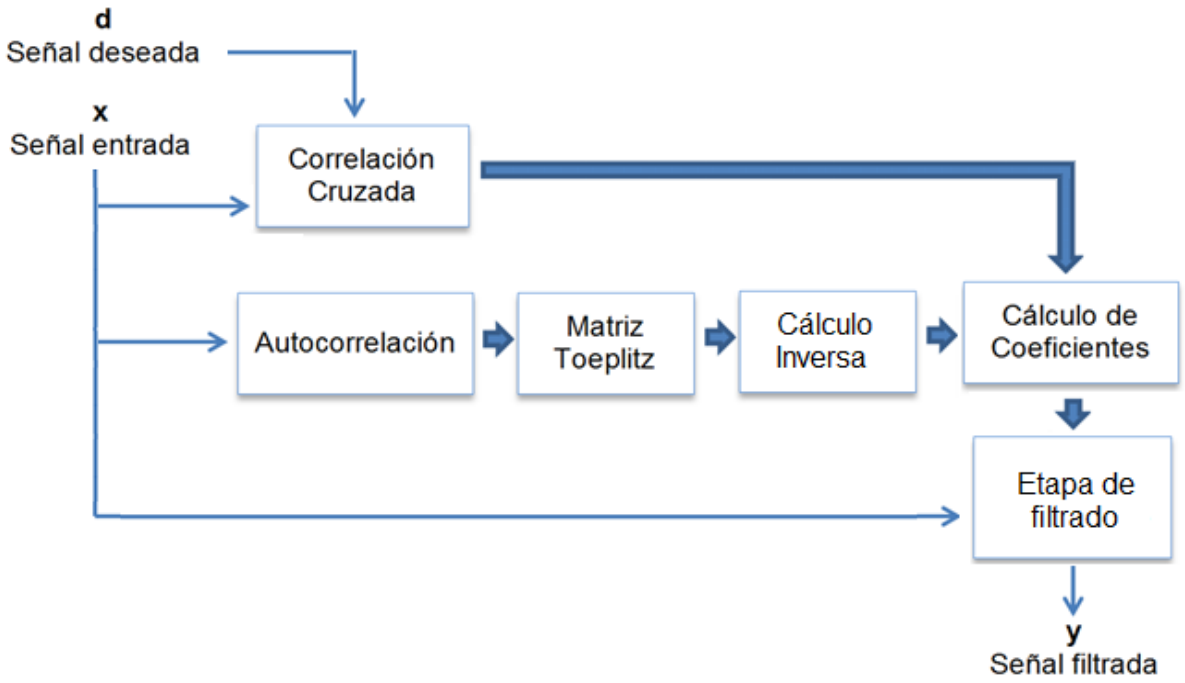
Posteriormente, se hizo uso de un compilador c (en este caso DEV-C++) para convertir las funciones usadas en estructuras sencillas basadas en el lenguaje c, pues dicho lenguaje de programación es el utilizado en el sistema de desarrollo PSoC específicamente PSoC *designer*.

2.1 FILTRO WIENER

Como se puede observar en la ecuación (17) para encontrar los coeficientes óptimos del filtrado Wiener se debe primero encontrar la matriz de autocorrelación toeplitz hermítica, y el vector de correlación cruzada entre la señal deseada *d(n)* y la señal recibida *x(n)* para posteriormente realizar el filtrado entre dichos coeficientes y x(n). Para llevar a cabo este proceso se debe conocer además el número de muestras “n” y seleccionar el grado del filtro “p” que en este caso correspondió a 250 y 40 respectivamente (ver capítulo 4).

En la Figura 11 se presenta de forma general las etapas necesarias para la implementación del filtro Wiener.

Figura 11. Diagrama de bloques filtro Wiener

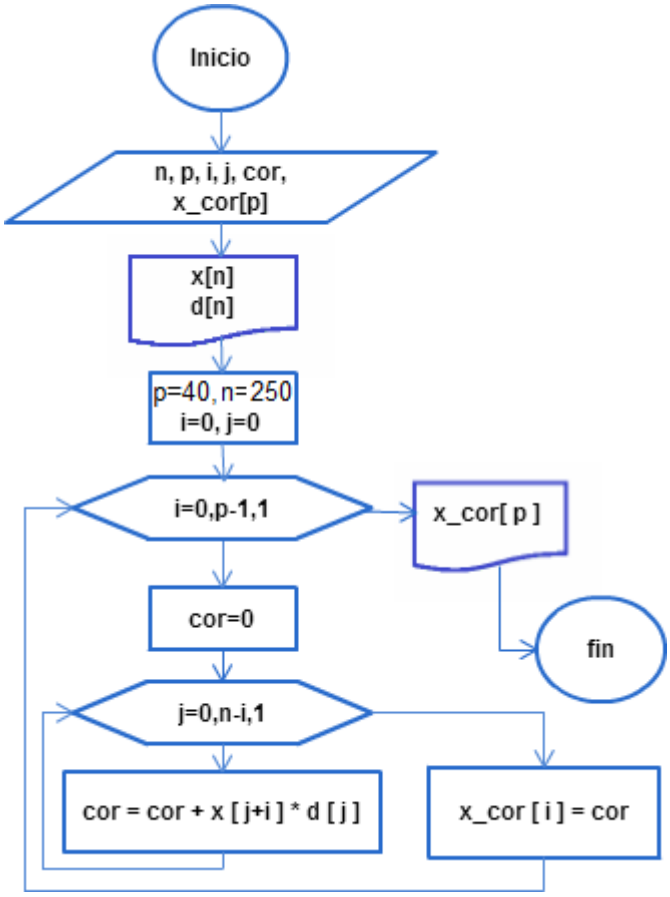


Fuente: Autores.

2.1.1 CORRELACIÓN CRUZADA

Para realizar la correlación cruzada entre la señal de deseada y la señal de entrada se elaboró un algoritmo basado en la ecuación (29), el diagrama de flujo de dicho algoritmo se ilustra en la Figura 12.

Figura 12. Diagrama de Flujo algoritmo Correlación Cruzada.

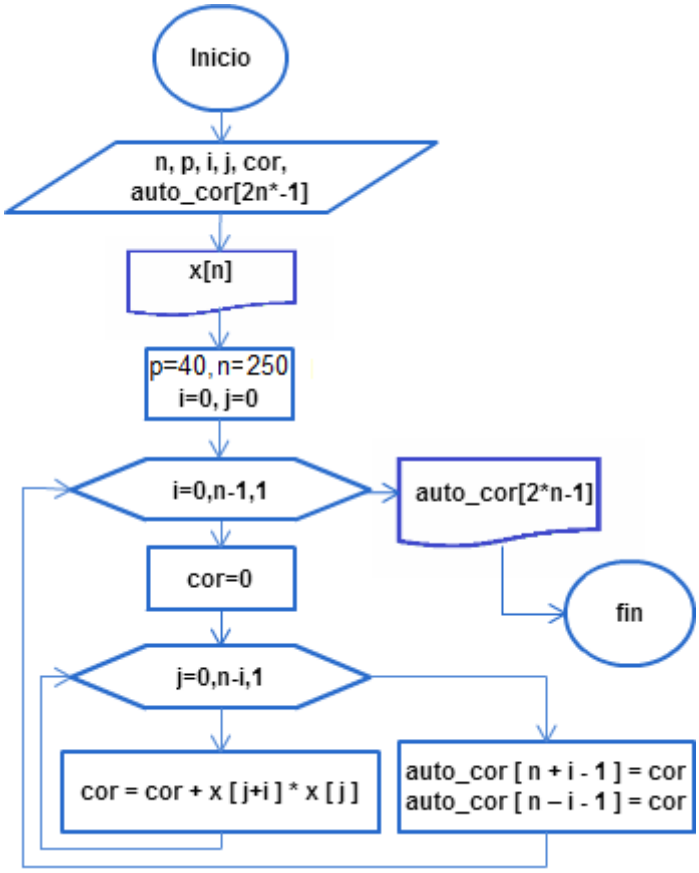


Fuente: Autores.

2.1.2 AUTOCORRELACIÓN

También es necesario utilizar la función correlación para determinar la matriz de autocorrelación toeplitz. En este caso, ya que la autocorrelación se define como la correlación cruzada de la señal consigo misma, se utiliza un algoritmo similar al de la función anterior pero esta vez la señal de deseada es reemplazada por la señal de entrada. En la Figura 13 se ilustra el diagrama de flujo de la función autocorrelación.

Figura 13. Diagrama de flujo algoritmo autocorrelación

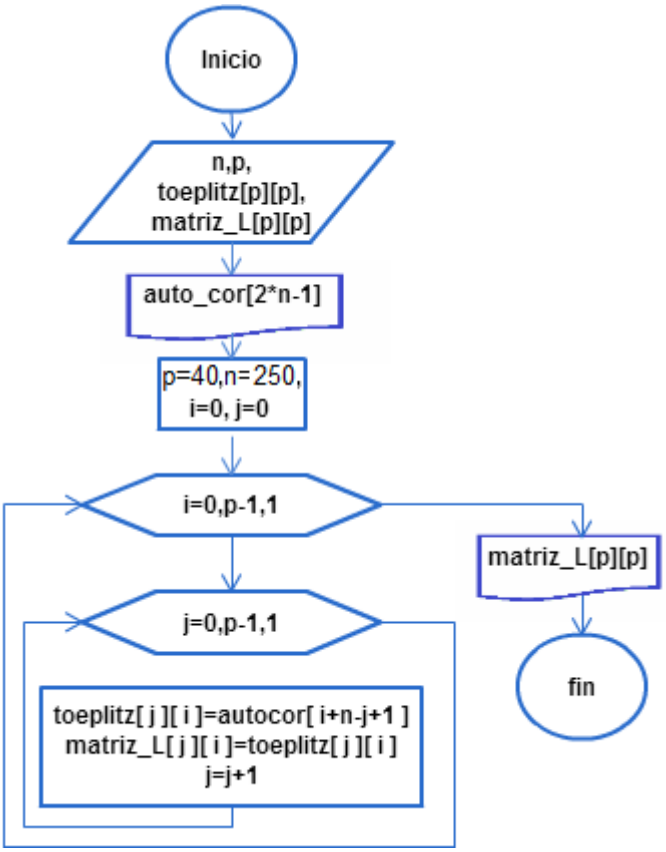


Fuente: Autores.

2.1.3 MATRIZ TOEPLITZ

En esta etapa se emplea la salida de la función autocorrelación, pero solo son seleccionados ciertos elementos del vector *auto_cor,* dependiendo del grado del filtro como puede observarse en la ecuación (16). Para ilustrar esta etapa se elaboró un diagrama de flujo basado en la ecuación (31) que puede ser observado en la Figura 14.

Figura 14. Diagrama de flujo algoritmo Matriz Toeplitz



Fuente: Autores.

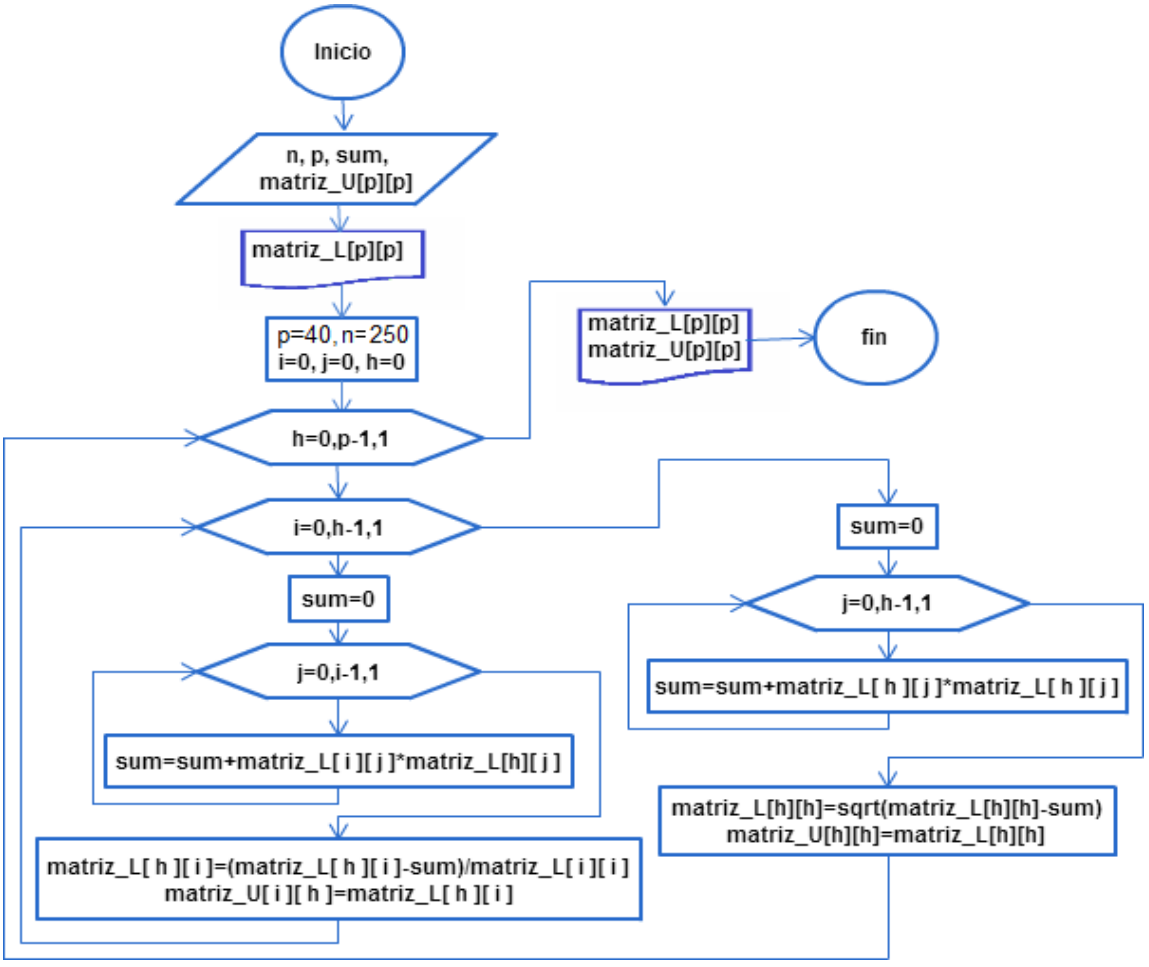
2.1.4 CÁLCULO INVERSA

Para calcular los coeficientes del filtro además de calcular la matriz de autocorrelación toeplitz se hace necesario encontrar su matriz inversa, ya que al despejar la ecuación (17) se tiene que $w = {R_x}^{-1} r_{dx}$. Entonces, para llevar a cabo dicha operación se empleó en una primera instancia la factorización Cholesky y en una segunda, una serie de sustituciones para el cálculo de la matriz inversa requerida.

- **FACTORIZACIÓN CHOLESKY**

La factorización Cholesky se deriva de la factorización LU y se pretende encontrar una matriz triangular inferior matriz_L y una matriz triangular superior matriz_U matrices, que serán usadas en la siguiente sub-etapa para el cálculo de la matriz inversa. Las ecuaciones (33) y (34) son las usadas para dicha descomposición y el conjunto de operaciones empleadas pueden ser observadas en el diagrama de flujo de la Figura 15.

Figura 15. Diagrama de flujo algoritmo Factorización Cholesky



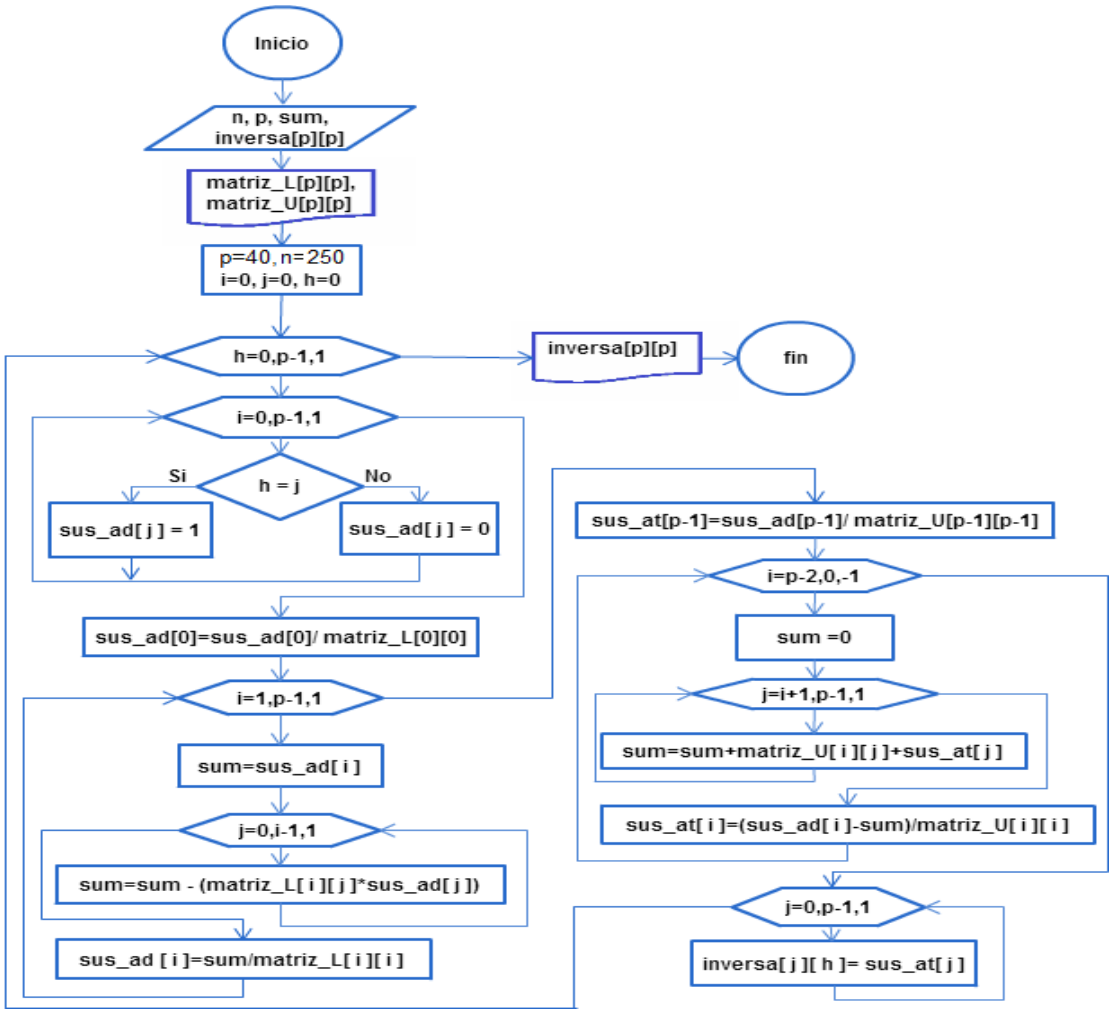
Fuente: Autores.

- **INVERSA**

En esta sub-etapa se realiza una serie de sustituciones llamadas hacia adelante y hacia atrás que pueden verse mejor ilustradas en el diagrama de la

Figura **10** se expone un método para la inversión de matrices a través de una previa descomposición LU. El algoritmo empleado en esta etapa corresponde al diagrama de flujo de la Figura 16.

Figura 16. Diagrama de flujo algoritmo Matriz Inversa

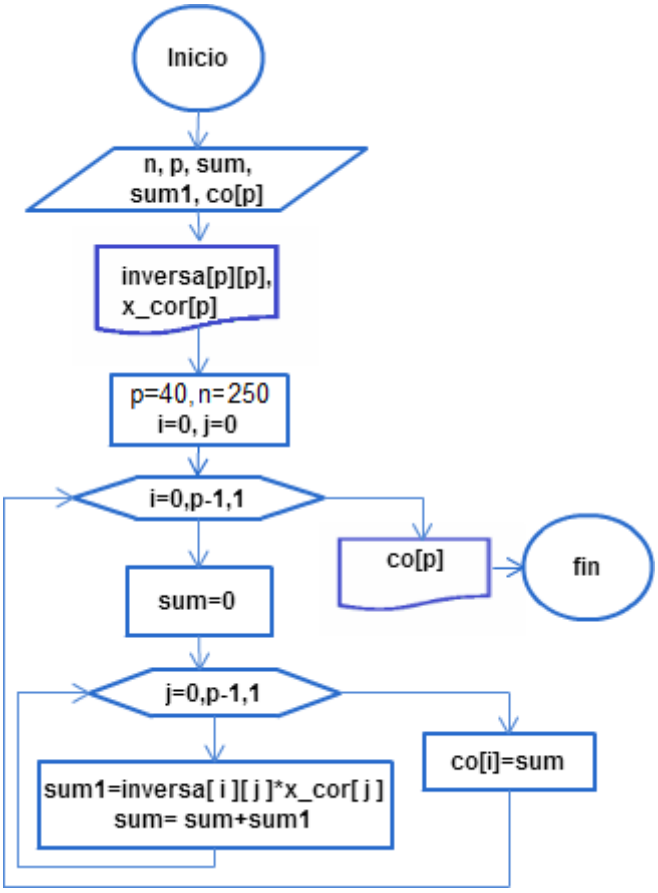


Fuente: Autores.

2.1.5 CÁLCULO DE COEFICIENTES

En esta etapa simplemente se realiza una multiplicación de matrices o más exactamente entre un vector y una matriz. Para esto se debió elaborar un algoritmo sencillo ilustrado en la el diagrama de flujo de la Figura 17.

Figura 17. Diagrama de flujo algoritmo para el cálculo de coeficientes

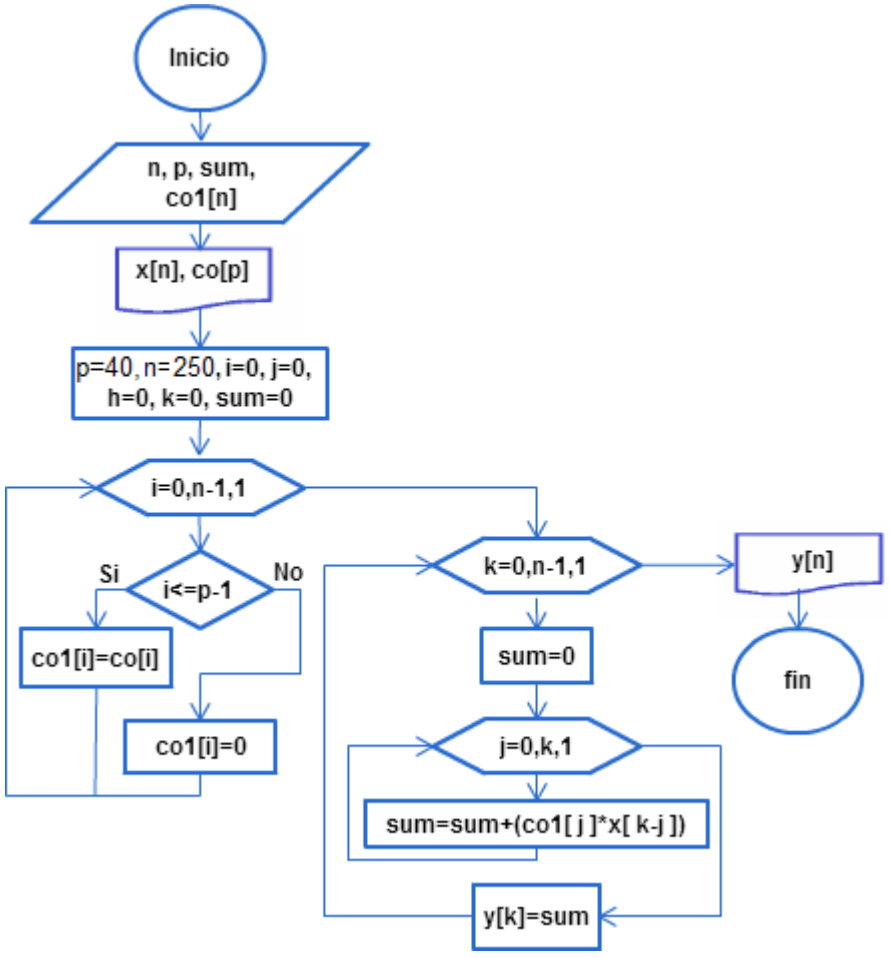


Fuente: Autores.

2.1.6 ETAPA DE FILTRADO

Luego de obtener los coeficientes del filtro, solo queda aplicar los mismos a la entrada x(n) para obtener la salida del filtro y(n). Dicho procedimiento se puede observar en la Figura 18 y se basa en la ecuación (49).

Figura 18. Diagrama de flujo algoritmo de filtrado

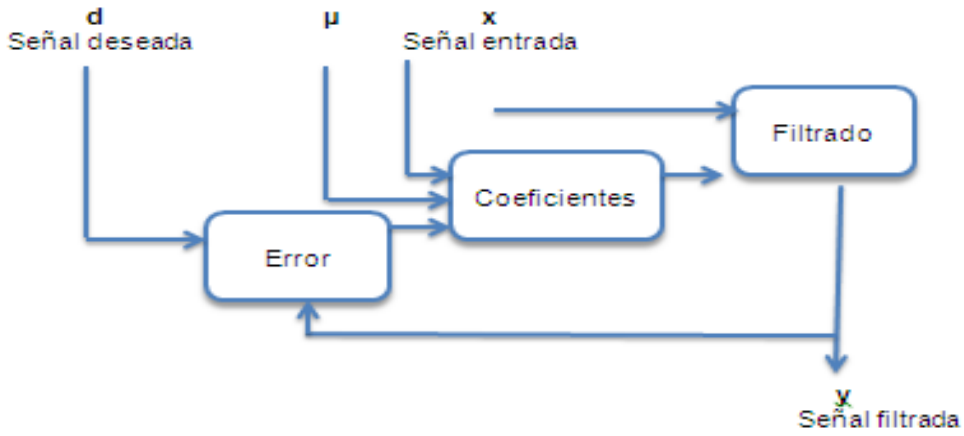


Fuente: Autores

2.2 FILTRO LMS (LEAST MEAN SQUARE)

En la Figura 19 se observa un diagrama de bloques del algoritmo del filtro LMS

Figura 19. Diagrama de Bloques filtro LMS



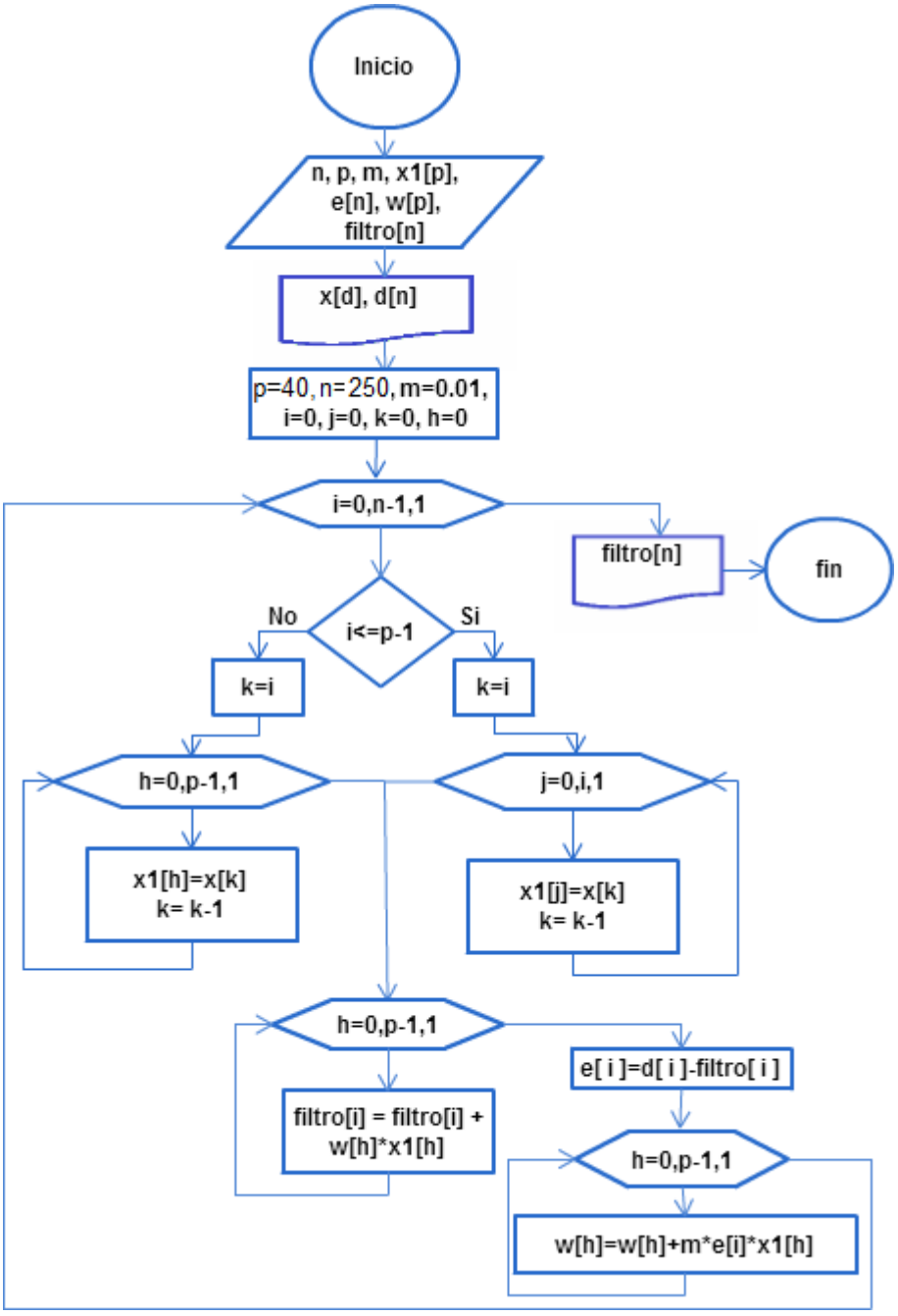
Fuente: Autores.

El diseño de este algoritmo se basó en las ecuaciones (56), (57) y (58), las cuales son necesarias para conocer la salida del filtro, estimar el error y calcular los coeficientes respectivamente. Una mejor forma de ilustrar este algoritmo es por medio de un diagrama de flujo como el que se muestra en la Figura 20.

A diferencia del algoritmo del filtro Wiener, este tipo de filtrado no requiere varias funciones para poder realizar el filtrado. El filtrado LMS solo depende de la velocidad de convergencia (μ), de las señales de entrada x(n) y deseada d(n), además el número de muestras “n” y el grado del filtro “p” que en este caso correspondió a 250 y 40 respectivamente (ver capítulo 4)

.

Figura 20. Diagrama de flujo filtro LMS



Fuente: Autores.

Para poder hallar la salida del filtro LMS, se debe partir del vector de pesos w(n). En esta medida, se hace necesario conocer su valor inicial w(0)

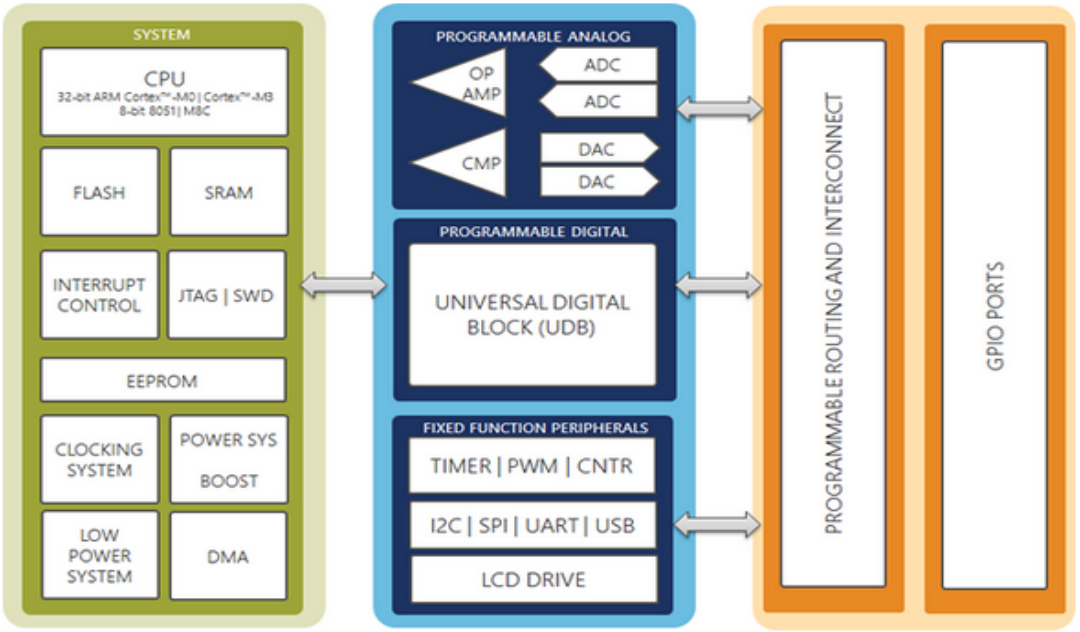
que en el caso del diseño propuesto corresponde a un valor igual a cero. Ahora, para calcular los valores actuales de los pesos es necesario estimar el error *e(n)* (comparando la salida del filtro *y(n)* con la señal deseada *d(n)*) y finalmente ajustar el valor de los pesos del filtro, todo esto asignado a μ un valor determinado.

3 SISTEMA DE DESARROLLO PSoC

CARACTERÍSTICAS GENERALES

Es una tecnología de microcontroladores conocida por su abreviatura PSoC desarrollada por la compañía Cypress Semiconductor en el año 2002. La cual incorpora un núcleo, 12 bloques analógicos y 16 bloques digitales de diferentes dispositivos electrónicos para luego ser programados mediante lenguaje C o Asembler. En la Figura 21 se puede ver la estructura genérica de un PSoC.

Figura 21. Estructura genérica de un PSoC.

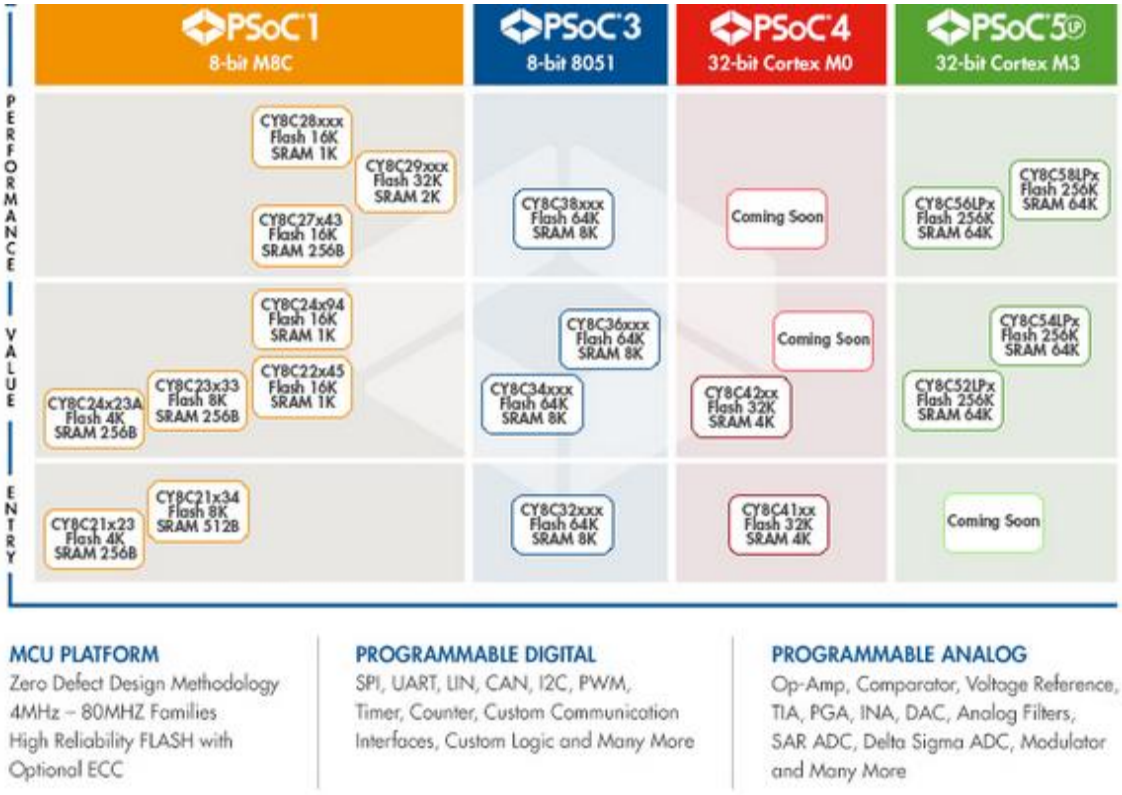


Fuente: PSoC [18].

Actualmente, se han desarrollado cuatro tecnologías delimitadas por sus arquitecturas y el tipo de procesador que tiene incorporado. PSoC1 le corresponde la serie de CY8C2XXXX con CPU MC8, PSoC3 con serie CY8C2XXXX con CPU 8051(Conocida también como Intel MSC-51), PSoC4 y PSoC5 con serie

CY8C5XXXX con CPU ARM Cortex M3. En la Figura 22 se puede observar una comparación de estas 4 tecnologías.

Figura 22. Características Dispositivos PSoC.



Fuente: PSoC [18].

Cypress proporciona el software que ayudan a la programación de estos dispositivos. El primer IDE (*Integrated device Electronics*) fue *PSoC Designer* utilizado para diseñar y depurar los dispositivos PSoC1, *PSoC Creator* fue el segundo IDE utilizado en dispositivos PSoC3, PSoC4 y PSoC5 el cual combina una interfaz gráfica con un entorno de diseño hardware/software . Este programa permite al usuario configurar y conectar circuitos existentes en el chip y los componentes equivalente a periféricos MCU, además PSoC *Creator* también permite a los usuarios conectar cualquier periférico a cualquier pin y por último *PSoC Programmer* (PSoC1, PSoC3, PSoC4 y PSoC5) [18].

Cypress Semiconductor provee junto con el dispositivo físico una serie de herramientas de software (PSoC *Designer*, PSoC *Programmer* y PSoC *Creator*) que se pueden utilizar para personalizar PSoC y satisfacer sus necesidades específicas de aplicación. Sus aplicaciones se desarrollan utilizando un catálogo de periféricos analógicos y digitales pre-caracterizados en un entorno de diseño de arrastrar y soltar. De hecho, es posible personalizar el diseño aprovechado una Interfaz de programación de aplicaciones de código generadas dinámicamente, depurar y probar los diseños con el entorno de depuración integrado que incluye la emulación de circuitos y características estándar de depuración de un software.

PSoC Creator es un entorno de diseño integrado (IDE) fácil de usar que permite el diseño de *hardware/firmware* (bloque de instrucciones de máquina para propósitos específicos, “software que controla hardware”) concurrente de sistemas PSoC basado en la introducción del esquema clásico.

Con *PSoC Creator* es posible:

- Crear y compartir periféricos personalizados definidos por el usuario mediante el diseño esquemático jerárquico y la entrada Verilog.
- Colocar y enrrutar automáticamente componentes seleccionados e integrar lógica sencilla que reside habitualmente en multiplexores discretos.
- Intercambio entre *hardware* y *software* de las consideraciones de diseño que le permite centrarse en lo que importa.

PSoC Creator también permite aprovechar todo un sistema de herramientas dentro de un compilador integrado y mejores programadores de producción para apoyar PSoC 3, 4 y PSoC 5.

Algunas características de dichos software son las siguientes:

- Aplicación GUI Editor de configuración del módulo del dispositivo y de usuario y la reconfiguración dinámica.
- Amplio catálogo de módulos de usuario.
- Editor integrado de código fuente (C y montaje).
- Compilador C libre sin restricciones de tamaño o límites de tiempo.
- Depurador integrado.
- Emulación de Circuito Integrado (ICE).

Otra de las características corresponde al soporte integrado para los interfaces de comunicación:

- Hardware y software de esclavos I2C y maestros.
- Full-Speed USB 2.0.
- Hasta 4 UART full-dúplex, maestro y esclavo SPI, y Wireless.

PSoC ofrece una amplia gama de bloques analógicos y digitales pre-caracterizados y configurables para la aplicación de una serie de funciones y periféricos que suelen ser piezas separadas que no se pueden configurar (o reconfigurar) sobre la marcha. La capacidad de configuración de los módulos de usuario tiene capacidad para adaptación específica de la aplicación y los cambios de última hora. Ejemplos de módulos de usuario provistos por *PSoC* incluyen:

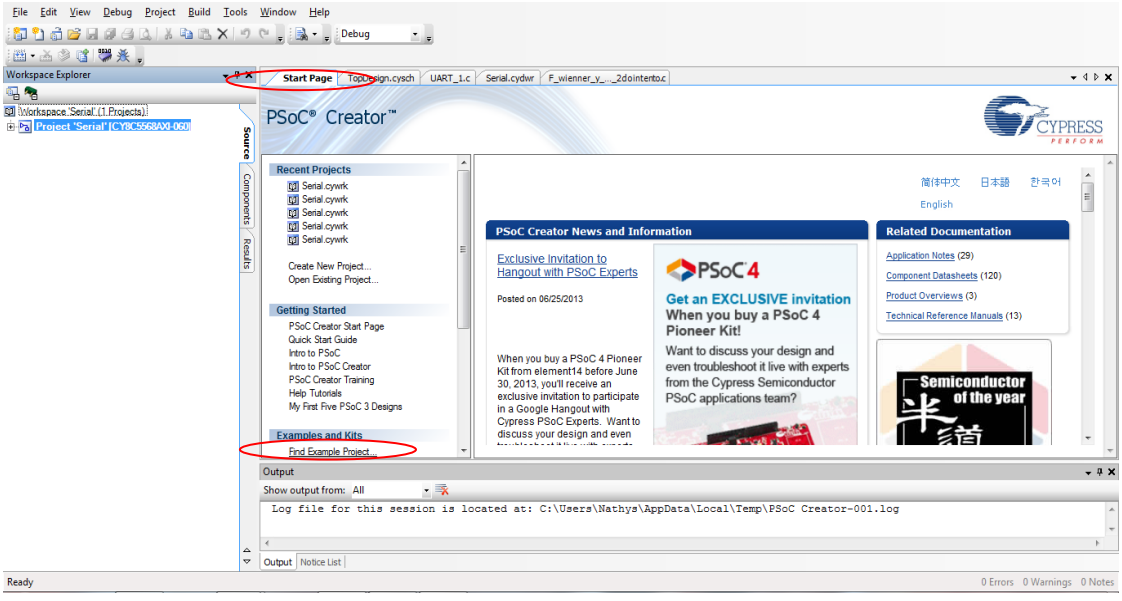
- ADCs y DACs.
- Amplificadores, comparadores y multiplexores analógicos.
- Filtros.
- Contadores y temporizadores.
- PWMs.

- *Star Network Protocol* (SNP).
- Medición de la temperatura.
- Bloques de comunicación digital como I2C, UART, SPI.
- Bloques de funciones especiales como LED / LCD Drivers.
- Soluciones táctil capacitivas (pantallas táctiles, botones, barras de desplazamiento, sensores de proximidad, etc.).

ENTORNO DE TRABAJO

Al iniciar el software *PSoC Creator* el usuario es remitido a una página de inicio como la de la Figura 23. En este entorno es posible crear un nuevo proyecto o abrir tanto proyectos elaborados como ejemplos que provee el software eligiendo la opción "*Open Example Project"* en donde es posible seleccionar una gran cantidad de aplicaciones para cada módulo o bloque interno.

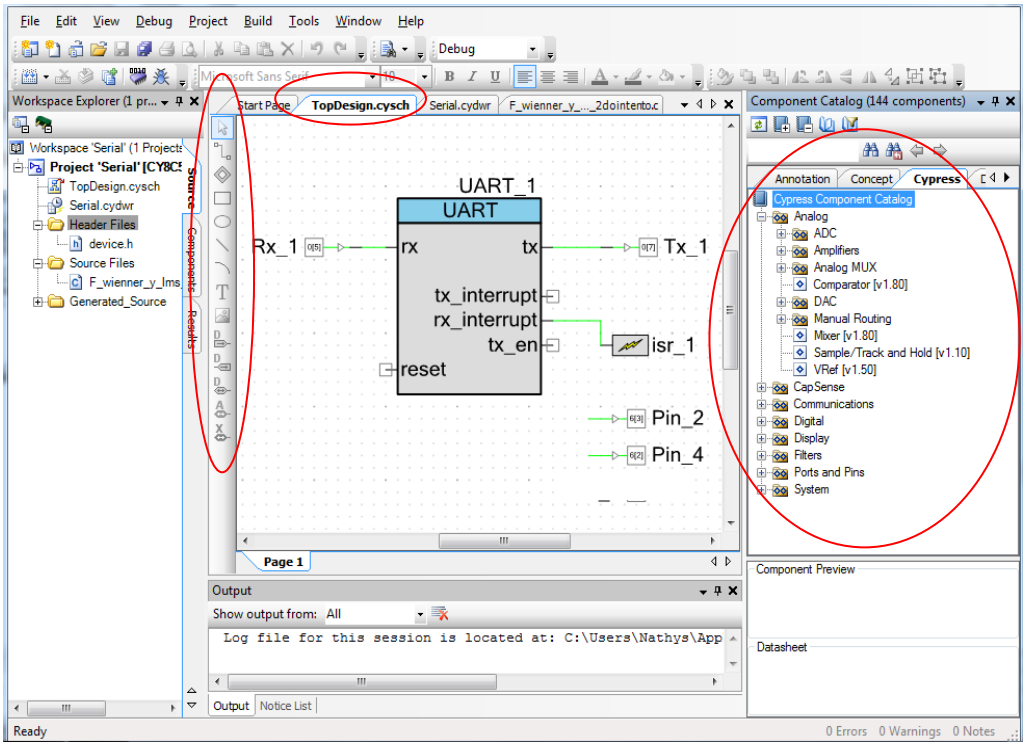
Figura 23. Página de inicio *PSoC Creator*



Fuente: PSoC Creator.

Al agregar un proyecto al espacio de trabajo se deben configurar diferentes fuentes generadas, una de ellas es la correspondiente al diseño gráfico en donde se hace uso de bloques prediseñados. En este caso la fuente es llamada *TopDesign* y su extensión es "*.cysch*". Dicho diseño se puede observar en la Figura 24.

Figura 24. Entorno gráfico de *PSoC Creator*



Fuente: PSoC Creator.

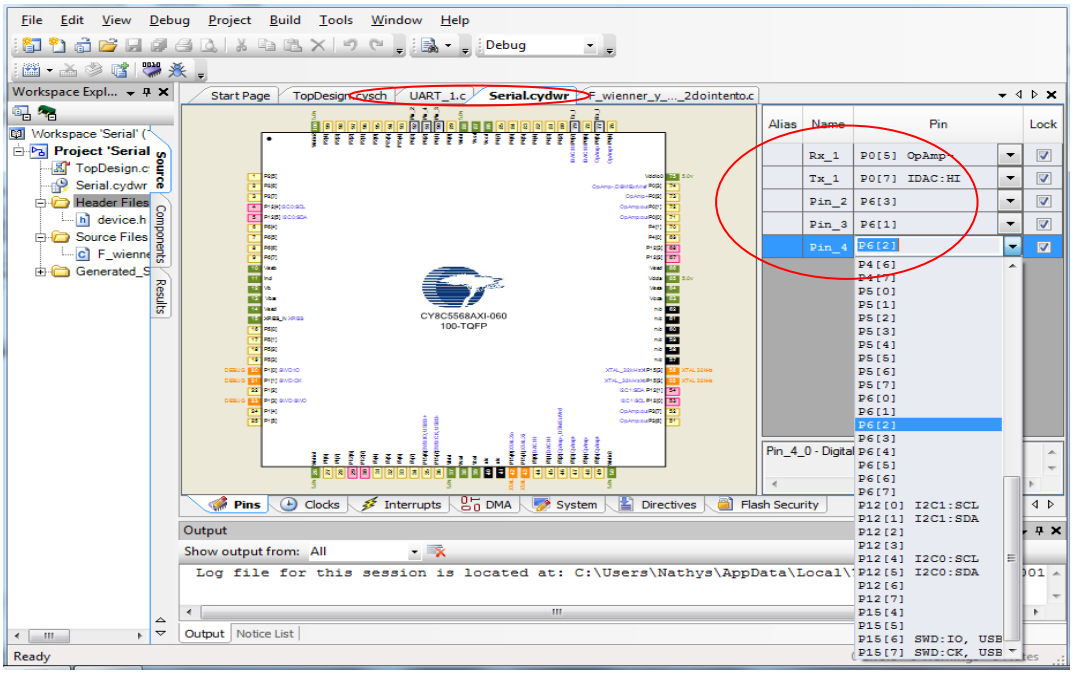
En este diseño es posible agregar cualquiera de los bloques disponibles desde la ventana derecha de la Figura 24. Dichos bloques se encuentran clasificados en varios grupos. El primero de ellos *"Annotation"* donde es posible realizar anotaciones de diferentes elementos de circuitos pasivos o activos, fuentes sensores y demás, pero solo funcionan como indicaciones no afectan la programación o funciones del diseño. Otro de los grupos corresponde a *"Concept"* donde encontramos algunos prototipos de comunicación o funciones digitales. Y

finalmente, el grupo de bloques “Cypress” de donde se extraen los bloques funcionales y configurables para el diseño que se quiere construir (mencionados anteriormente).

También, es posible realizar conexiones, anotaciones o diferentes figuras usando la tabla de herramientas ubicada en la parte izquierda del diagrama.

Otra de las fuentes configuradas es aquella cuya extensión corresponde a “.cydwr” en este caso llamada *Serial* en la que es posible asignar pines analógicos a los elementos agregados en la fuente anterior. Dichas asignaciones pueden ser visualizadas en la Figura 25.

Figura 25. Asignación de pines en *PSoC Creator.*

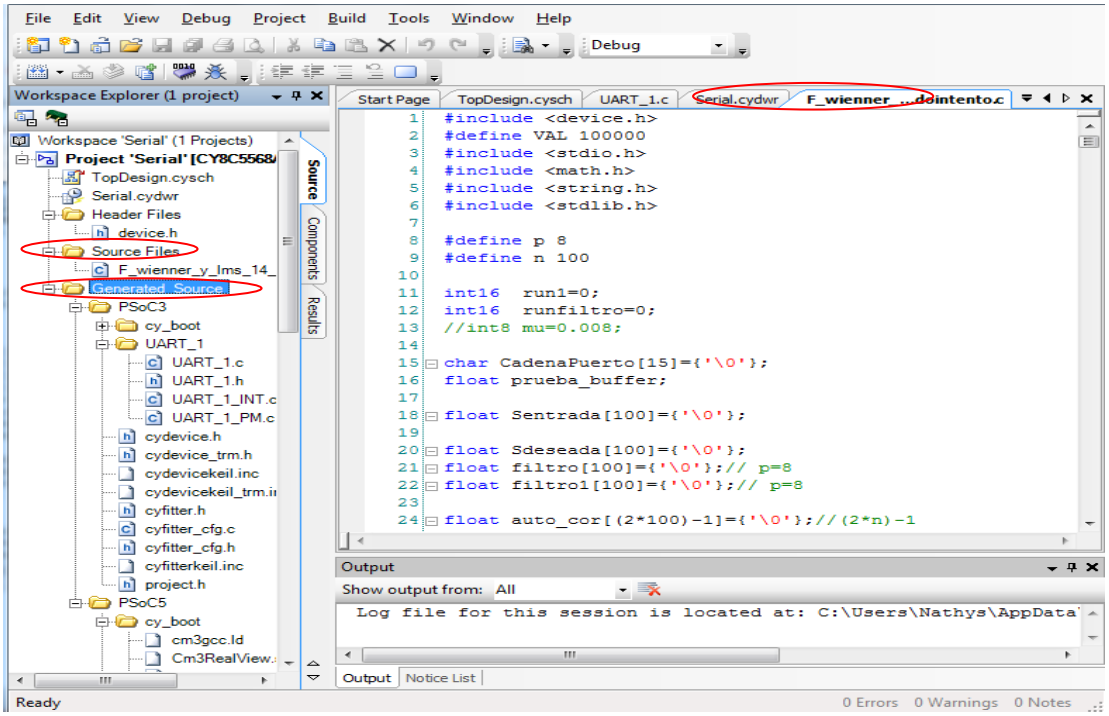


Fuente: PSoC Creator.

También existen una serie de fuentes generadas automáticamente al agregar elementos o seleccionar características internas de los dispositivos y se

encuentran en la carpeta *"Generated Sources"* la cual puede ser observada en la Figura 24.

Figura 26. Fuente de código c de *PSoC Creator.*



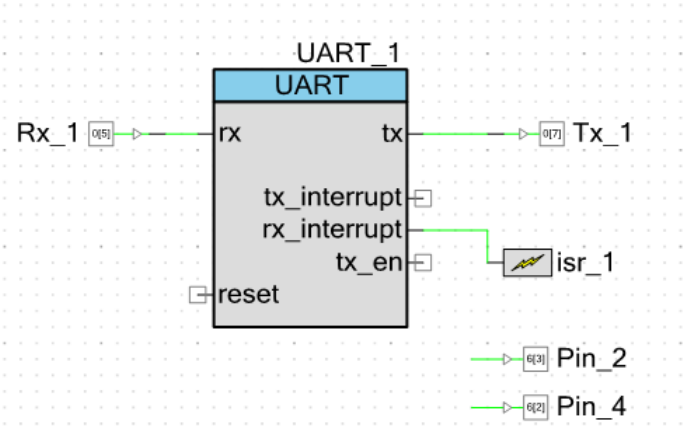
Fuente: PSoC Creator.

La última fuente trabajada corresponde a la fuente de extensión ".c" dentro de la carpeta "Source Files" Figura 26. Dicha fuente es una de las más importantes en el presente proyecto, de hecho en la misma se describe una serie de procedimientos empleados para el filtrado de una señal. En dicha fuente es posible por medio del lenguaje de programación C y de algunos comandos propios de cada bloque funcional realizar una completa descripción del diseño usando líneas de código.

DESCRIPCIÓN DEL DISEÑO

Para realizar el filtrado adaptativo de una señal es necesario además de adquirir la señal de entrada, adquirir una señal deseada. En esta medida la tarjeta PSoC recibe desde un generador externo (ver capítulo 5) estas señales. Para la adquisición de las mismas se debió usar un bloque funcional llamado “UART_1” por medio del cual se realiza la transmisión y recepción de datos entre PSoC y el software LabVIEW™, este bloque involucra la utilización de 2 pines “Rx_1” y “Tx_1” conectados al receptor y transmisor respectivamente, así como la conexión a otro bloque llamado “isr_1” por medio del cual es posible generar una interrupción en este caso en la recepción de datos pues se encuentra conectado a la salida “rx_interrupt” del UART. Por otra parte, fue de gran utilidad el uso de algunos pines conectados a LEDs exteriores para indicar o no la ejecución de ciertas órdenes (ver Figura 27).

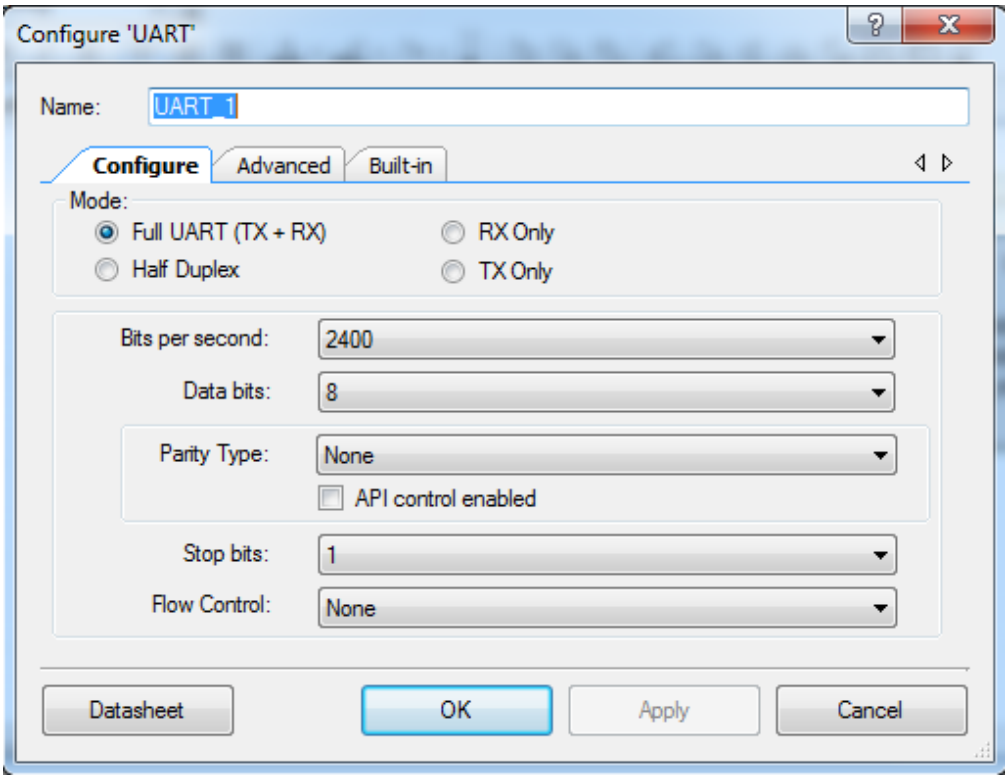
Figura 27. Bloques y conexiones del diseño gráfico.



Fuente: PSoC Creator.

Dentro de esta misma fuente es posible modificar algunas características del bloque UART tales como velocidad y modo de trabajo. Esta configuración puede ser vista en la Figura 28.

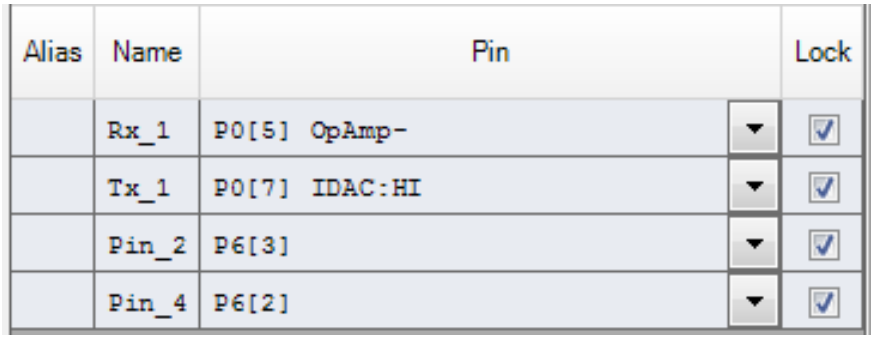
Figura 28. Opciones de configuración UART.



Fuente: PSoC Creator.

Otra de las fuentes modificadas corresponde a aquella en la que se asignan los pines digitales en el diagrama gráfico a los pines físicos de la tarjeta (ver Figura 29).

Figura 29. Asignación de pines en la tarjeta.



Fuente: PSoC Creator.

En este caso se eligieron los pines P0[5] y P0[7] para las conexiones del receptor y transmisión respectivamente, dichos pines pueden ser utilizados debido a que no se están ocupando los pines ni del OpAmp (Amplificador operacional) ni del DAC (Conversor análogo digital) asignados por defecto. También fueron asignados los pines P6[3] y P6[2] que corresponden a LEDs de la tarjeta física.

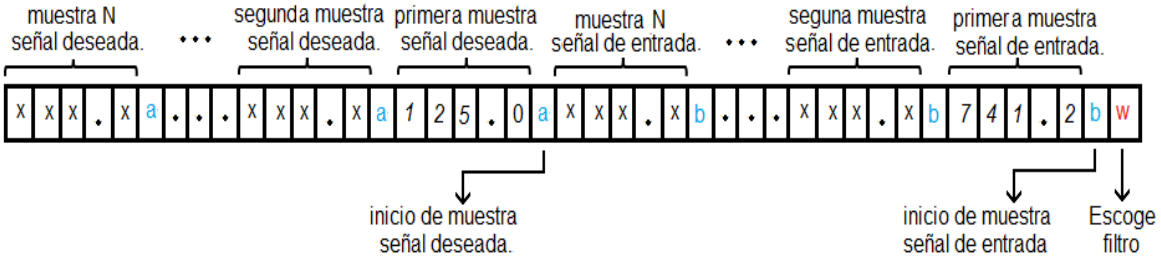
DESCRIPCIÓN DE CÓDIGO C

Como ya fue mencionado es necesario modificar la fuente ".c". En dicha fuente se realiza una serie de procedimientos para el adecuado filtrado de una señal de entrada respecto a una señal deseada. En el diagrama de la Figura 31 se enuncia la estructura general del proceso de filtrado.

Inicialmente se realiza la declaración de librerías, variables y funciones a usar. Después de esta etapa y dentro de la interrupción enunciada en el numeral anterior se empieza a realizar el proceso de adquisición de datos a través del receptor del UART.

En la etapa de adquisición de datos (ver Figura 30) inicialmente se recibe una letra "l" o "w" para la selección del tipo de filtro, luego se empieza a recibir muestra por muestra los valores de la señal de entrada, cada uno de estos valores poseen un carácter de inicio correspondiente a la letra "b" y además 3 dígitos decimales, dichos caracteres deben ser inicialmente almacenados en un *buffer* y ser convertidos en un número por medio de la función *attof* para finalmente ser almacenados en las posiciones el vector de la señal de entrada. En seguida, se empieza a recibir la señal deseada, señal a la cual se le debe realizar el mismo procedimiento de lectura y grabado pero esta vez teniendo en cuenta que el carácter de inicio de cada dato corresponde a la letra "a".

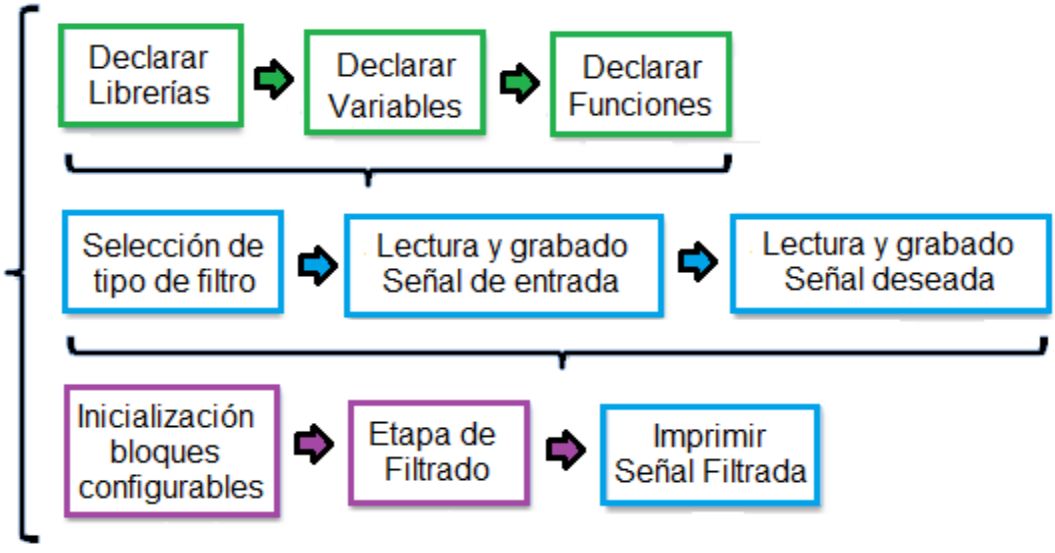
Figura 30. Diagrama de Recepción de datos



Fuente: Autores.

Ahora, una vez se haya escogido el filtro y se hayan almacenado las señales de entrada y deseada es posible realizar el filtrado adaptativo. Antes de realizar el filtrado en el *main* del código fuente (ver anexo B) se inicializan los bloques de PSoC. Y para realizar el filtrado se hace necesario el uso de las funciones creadas en la parte final del código (ver capítulo 2). Finalmente, se procede a realizar el envío de la salida del filtro, es decir la señal filtrada, ésta señal de salida se imprime en la interrupción y se realiza una sola vez después de haber realizado todos los procedimientos mencionados.

Figura 31. Estructura general del código c.

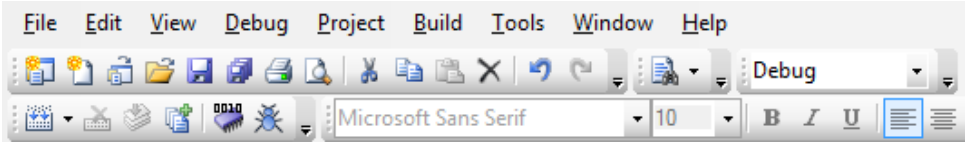


Fuente: Autores.

EJECUCIÓN DEL SOFTWARE

PSoC Creator posee una serie de herramientas en la parte superior de la interfaz para el procesamiento, verificación e implementación del diseño creado (ver Figura 32).

Figura 32. Barra de herramientas de *PSoC Creator*



Fuente: Autores.

Los iconos de la parte superior sirven para abrir proyectos o fuentes, guardar cambios, volver a la acción anterior o posterior, entre otras. Pero las funciones que realmente se quieren indicar son en orden de derecha a izquierda: construir serial, cancelar construcción, compilar, generar aplicación, programar y depurar.

De esta manera, después de finalizar el diseño es posible *compilar* el proyecto para buscar errores o advertencias, las cuales aparecen en la parte inferior de la Figura 26 eligiendo la pestaña “Notice list” con sus respectivas descripciones.

Ahora, para ejecutar la función *programar* es necesario conectar la tarjeta PSoC vía USB y de este modo ejecutar el diseño creado. Pero además, si se hace necesario el estudio detallado del funcionamiento del software es posible recurrir a la *depuración* donde es útil el uso de *“breakpoints”* para correr el programa hasta determinado punto, así como el uso de ventanas en las que se vigilan las variables de interés. Todo esto se puede elegir en el menú superior de la Figura 32 accediendo a la opción “*Debug>>insert breakpoint” o* “*Debug>>windows>>Add watch”*.

4 DETERMINACIÓN DE PARÁMETROS EN MATLAB®

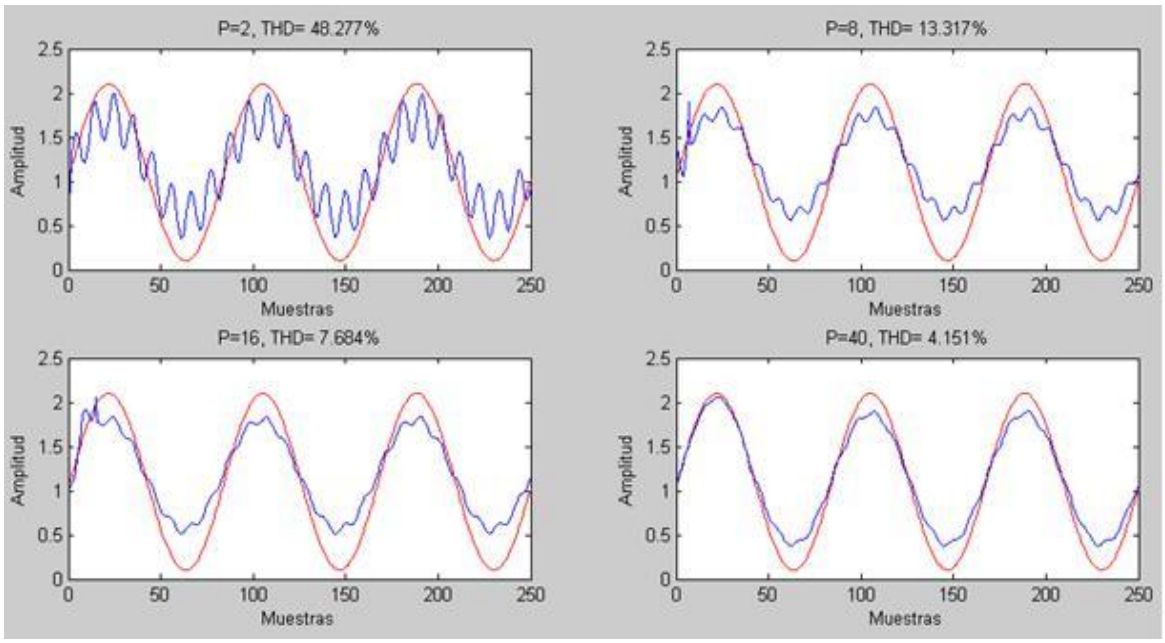
4.1 ORDEN DEL FILTRO

Inicialmente se realizaron pruebas en MATLAB® para conocer un valor adecuado del orden del filtro “p”, con un rango de valores $1 < p \leq N$, donde N es el número de muestras. Se asignó a N un valor de 250. Estas pruebas se realizaron con el código del filtro Wiener (Ver anexo A) donde la entrada corresponde a una señal seno de baja frecuencia (3 Hz) mezclada con una señal de interferencia, en este caso una señal seno de mayor frecuencia (24 Hz).

Inicialmente se realizaron una serie de pruebas con un valor de N=100 (Ver anexo C), pero finalmente se decidió trabajar con N=250 debido a que se deseaba tomar mayor número de muestras de la señal de interferencia, además se ampliaba la posibilidad de asignar a esta señal una frecuencia de mayor (f<125 Hz). También se realizaron pruebas hasta con 800 muestras, las cuales pueden ser consultadas en el anexo C.

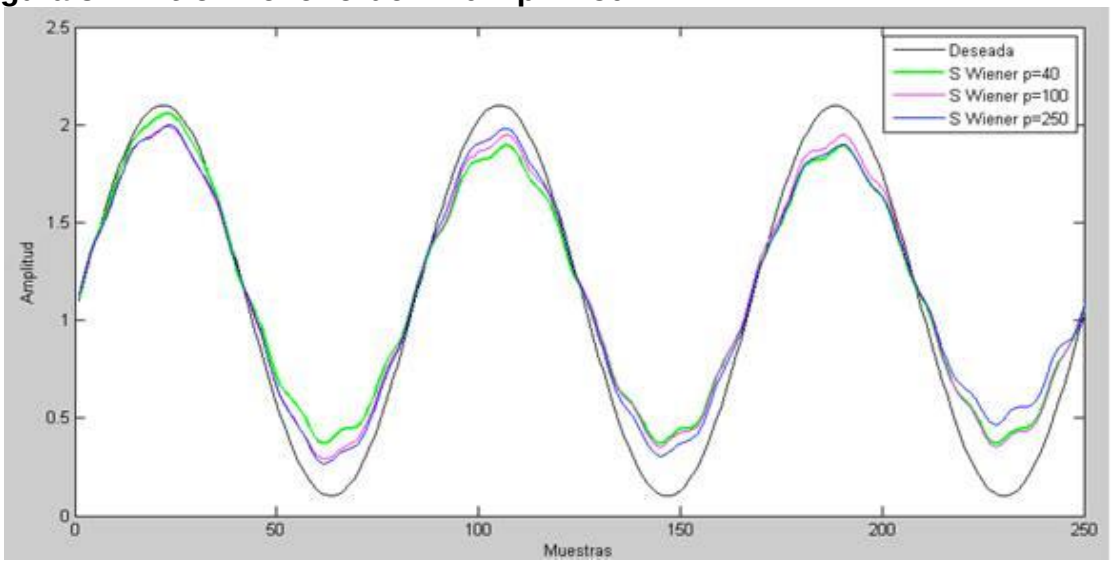
En la Figura 33 y la Figura 34 es posible notar que a medida que aumenta el valor del orden del filtro, mejora la calidad de la señal filtrada. En esta medida, se decide tomar un orden de filtro 40 ya que realiza un filtrado razonable. El criterio tomado para esta selección correspondió a una distorsión armónica total (THD) menor al 5%. En la Tabla 1 es posible observar los THD para diferentes valores de p.

Figura 33. Filtro Wiener orden: 2 < p < 40



Fuente: Matlab®.

Figura 34. Filtro Wiener orden: 40 < p < 250



Fuente: Matlab®.

Tabla 1. THD de salidas Wiener y LMS variando p.

N	% THD Wiener	% THD LMS	%THD promedio
8	13.32	15.74	14.53
16	7.69	9.13	8.41
32	3.80	6.74	5.27
40	4.15	5.58	4.87

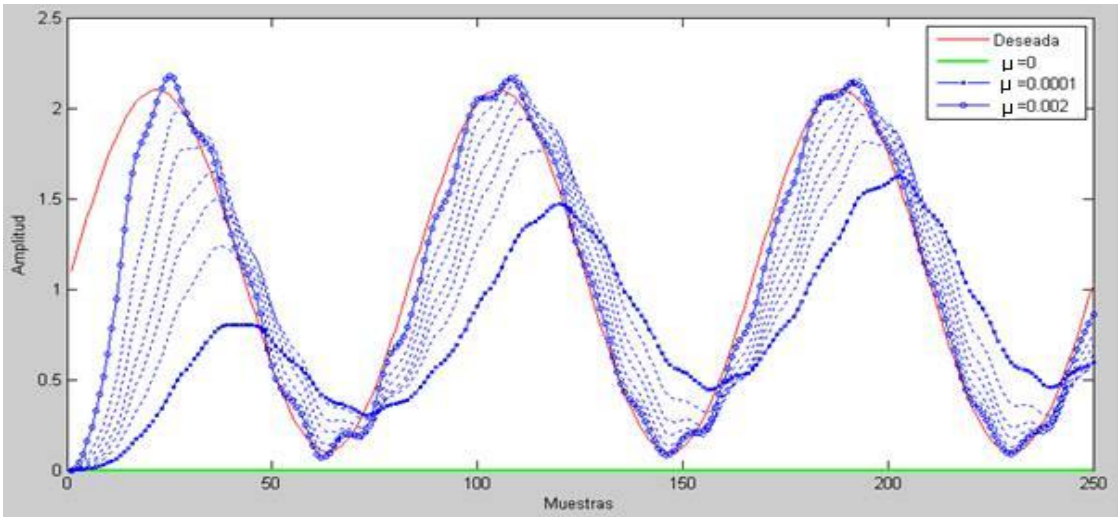
Fuente: Autores.

4.2 VELOCIDAD DE CONVERGENCIA

Además del orden del filtro, definido en la sección anterior, para realizar el filtrado LMS es necesario conocer el valor de la velocidad de convergencia (μ) óptimo, para ello se realiza un análisis gráfico en Matlab® con los que se obtiene que:

1. Para una rango de $0<\mu<0.002$ Figura 35, se obtiene una señal de salida nula cuando $\mu=0$. Pero a medida que μ aumenta, el valor de la señal de salida comienza a crecer pareciéndose cada vez más a la señal deseada.

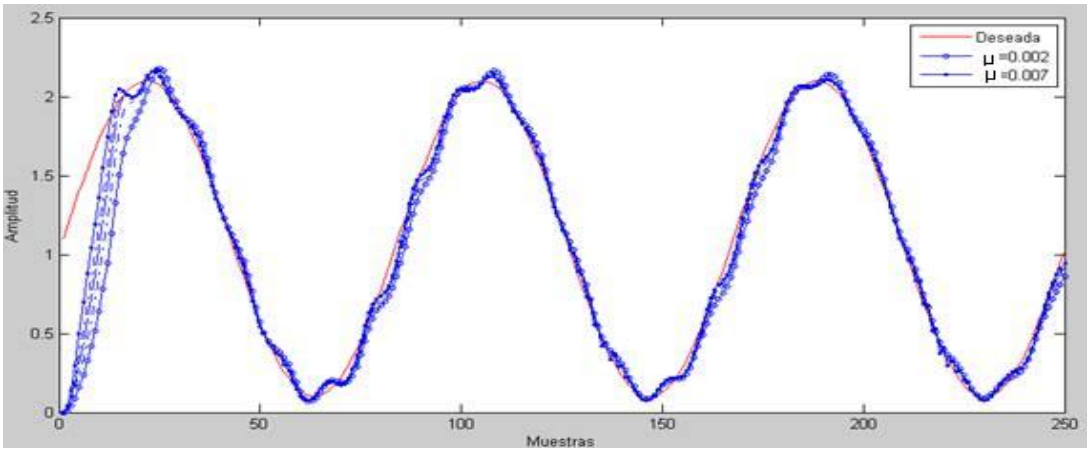
Figura 35. Rango: 0<mu<0.002



Fuente: Matlab®.

2. Para el rango entre 0.002<μ<0.007 Figura 36, cuando μ aumenta la señal de salida se asemeja un cada vez más a la señal deseada.

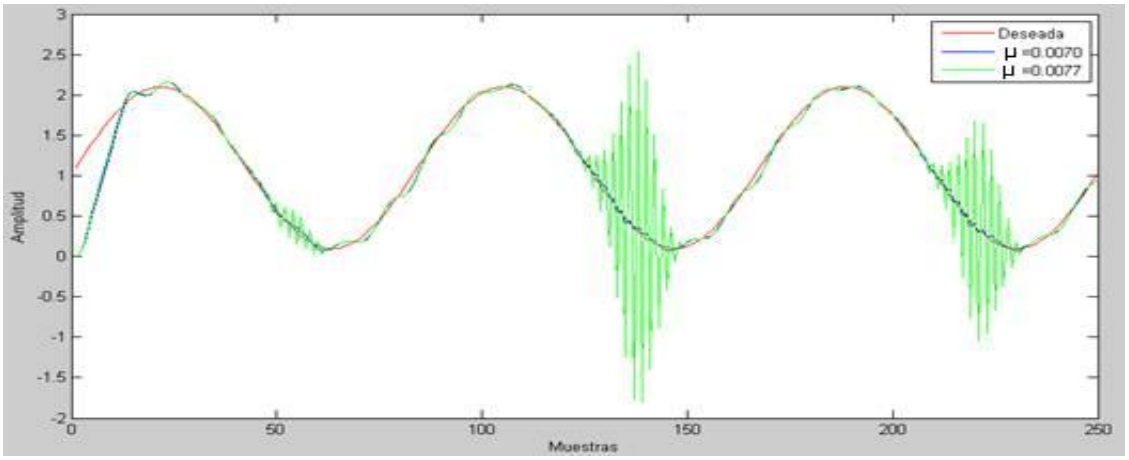
Figura 36. Rango: 0.002<mu<0.007



Fuente: Matlab®.

3. Por último, en el rango de 0.007<μ<0.0077 Figura 37, la señal de salida comienza a distorsionarse cada vez más, hasta perder las características de la señal deseada y del filtro.

Figura 37. Rango: 0.007<mu<0.0077



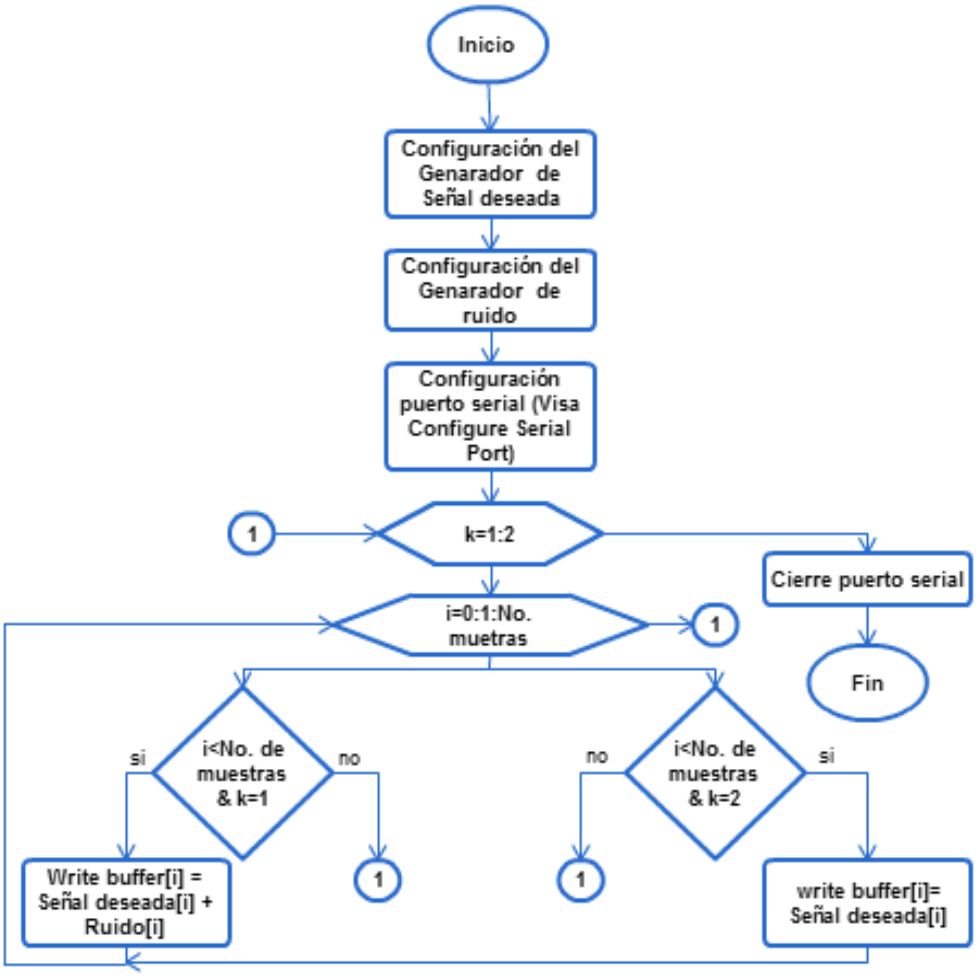
Fuente: Maltab®

A partir de lo anterior se le asigna a la velocidad de convergencia μ un valor de 0.006.}

5 INTERFAZ DE USUARIO

El hardware de adquisición de datos y el software LabVIEW™ 10.0.1 son herramientas que permiten la manipulación de los filtros adaptativos programados en la tarjeta descrita en el capítulo anterior. Este software es una plataforma con un lenguaje de programación gráfico que permite diseñar un conjunto de señales para evaluar la eficiencia de los filtros implementados.

Figura 38. Diagrama de Flujo Primera Etapa (Envío de Datos)

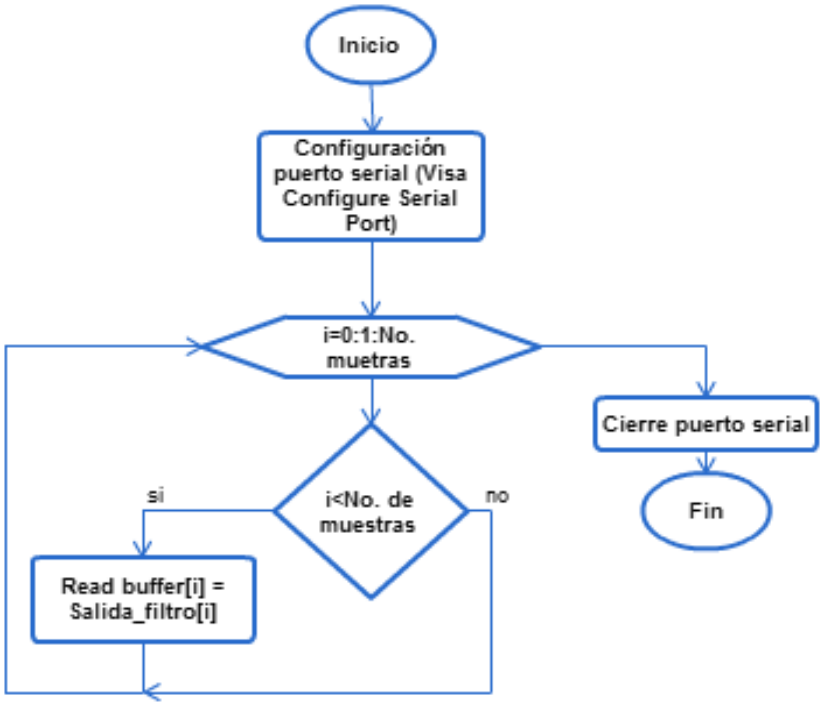


Fuente: Autores.

El objetivo es crear una interfaz gráfica que permita al usuario generar una señal deseada y seleccionar diferentes tipos de ruidos. El panel frontal permite la configuración y la visualización de estas señales así como observar la señal obtenida en la salida de cada filtro. Para ello fueron necesarias dos etapas, la primera etapa de *envió de datos* y la segunda de *recepción de datos*, como se muestran en la Figura 38 y Figura 39 respectivamente. Ambas etapas están controladas por medio de un pulsador nombrado *"Start"* que permite el funcionamiento de ambas etapas simultáneamente.

En las siguientes secciones se puede observar tanto la visualización como la programación de todos los módulos requeridos para su funcionamiento.

Figura 39. Diagrama de Flujo Segunda Etapa (Recepción de Datos)



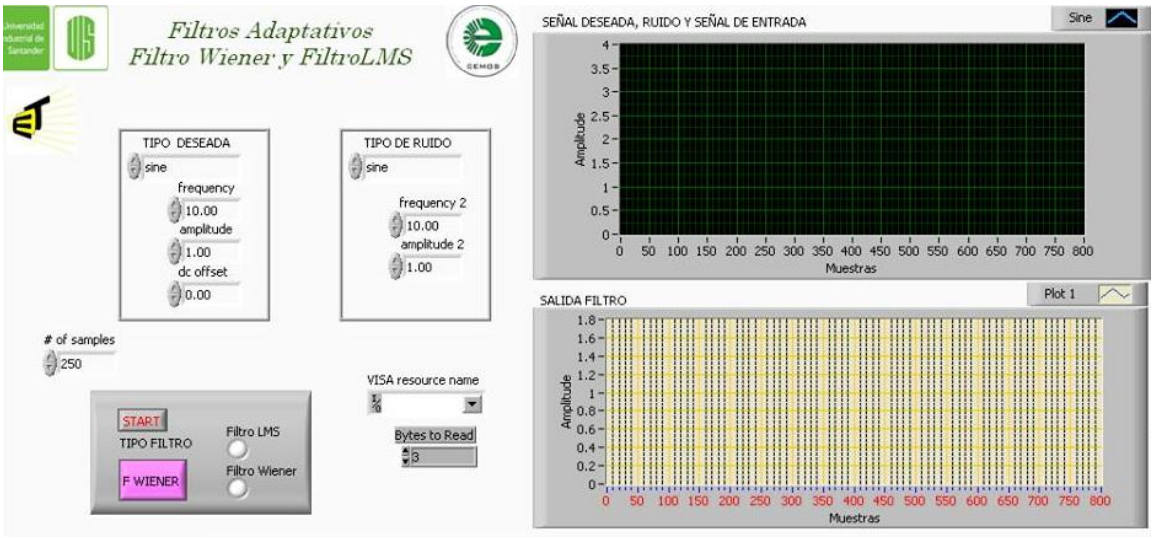
Fuente: Autores.

5.1 VISUALIZACIÓN

En la Figura 40 se observa el panel frontal de LabVIEW™, en éste se puede observar dos secciones, en la parte izquierda la configuración de las siguientes características:

- Señal deseada
- Tipo de ruido
- Tipo de filtro
- Puerto serial

Figura 40. Panel Frontal LabVIEW™



Fuente: LabView™.

Las características de la señal deseada y ruido, como tipo de señal (seno, coseno, triangular o cuadrada), amplitud, frecuencia y componente de continua son independientes entre sí. Es necesario que ambas señales tengan el mismo número de muestras, dicho valor determinará el límite de frecuencia de las

señales generadas. De esta manera, si se tienen 250 muestras, el valor de dicha frecuencia será inferior a 125Hz, cumpliendo así el teorema de Nyquist.
En la sección de tipo filtro, se tiene dos opciones, Filtro Wiener y Filtro LMS cada uno de ellos cuenta con un LED indicador. Y por último, en la configuración del puerto serial, el número de bytes tanto de lectura como de escritura configurados en la tarjeta PSoC que corresponde a 5 y el nombre de puerto serial que se está utilizando.

La sección derecha corresponde a la visualización de las señales, en la primera gráfica se muestran las señales de entrada al filtro, como los son la señal deseada y la de entrada (señal deseada + ruido) además de la señal de ruido. En la segunda gráfica se muestra la salida del filtro, es decir la señal filtrada dependiendo del tipo de filtro seleccionado en la sección anterior.

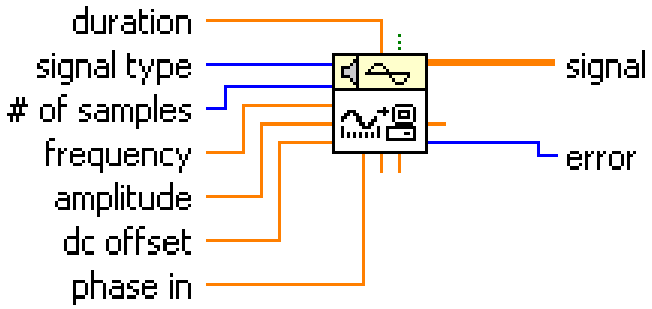
5.2 PROGRAMACIÓN

Como se muestra en la Figura 38, para poder realizar esta etapa se requieren tres pasos indispensables:

1. Configuración de las señales y selección de tipo de filtro.
2. Configuración del Puerto serial.
3. Envió de datos.

Para poder generar las señales seno se utiliza un subVi del banco datos de LabVIEW™ que se conoce como "*Signal Generator by Duration*" (Figura 41). En él se configuran las características de la señal como su amplitud, frecuencia, componente de continua, fase, tipo de señal, entre otras.

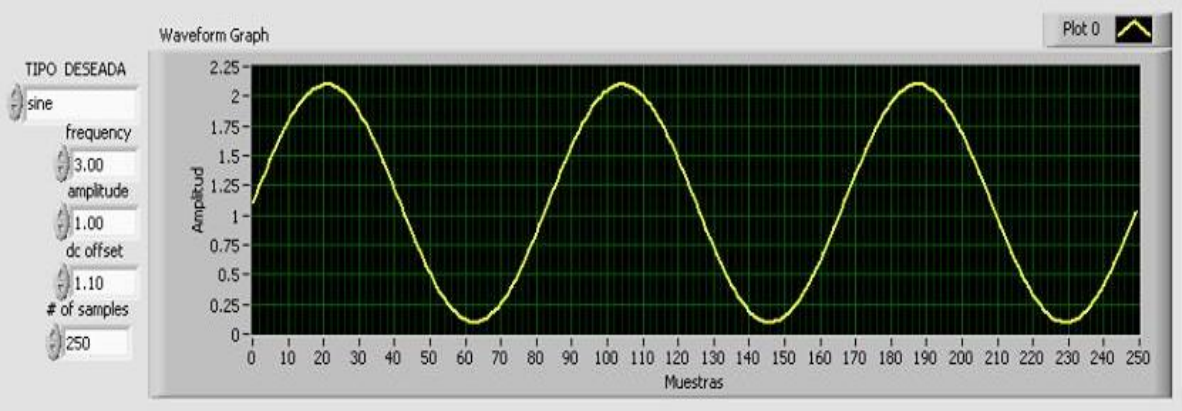
Figura 41. Ícono Signal Generator by Duration VI.



Fuente: LabView™.

La señal deseada fue una de las señales seno necesarias para el filtrado adaptativo, para este caso se seleccionó una frecuencia baja como se puede observar en la Figura 42.

Figura 42. Señal seno de baja frecuencia (3 Hz).

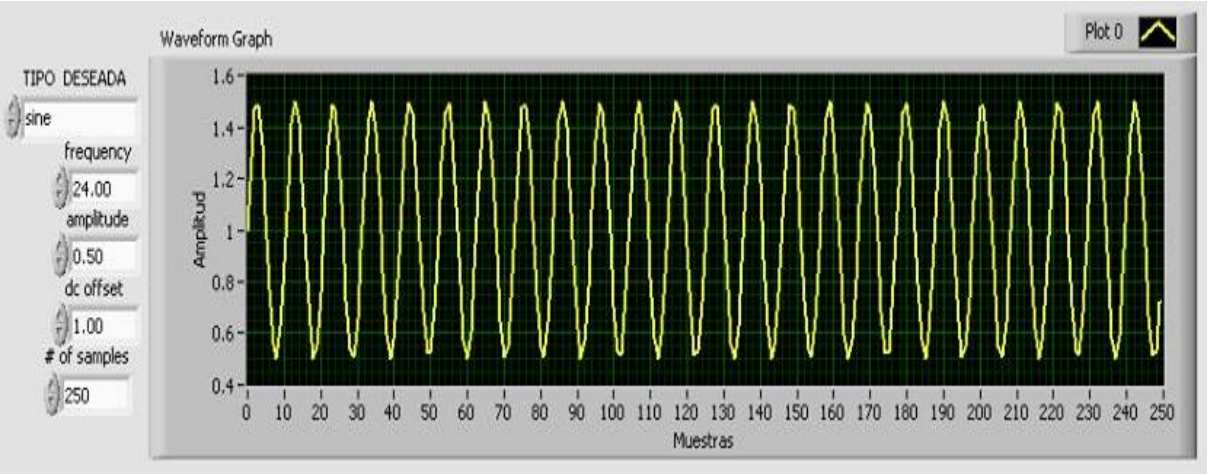


Fuente: LabView™.

Otra de las señales involucradas en el filtrado adaptativo fue la señal de ruido que para una de las pruebas correspondió también a una señal seno pero en este caso con un valor de frecuencia mayor (

Figura **43**).

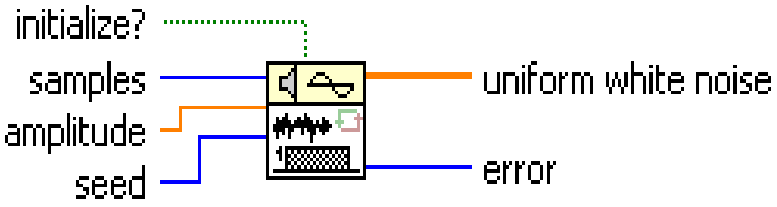
Figura 43. Señal seno de mayor frecuencia (24 HHz).



Fuente: LabView™.

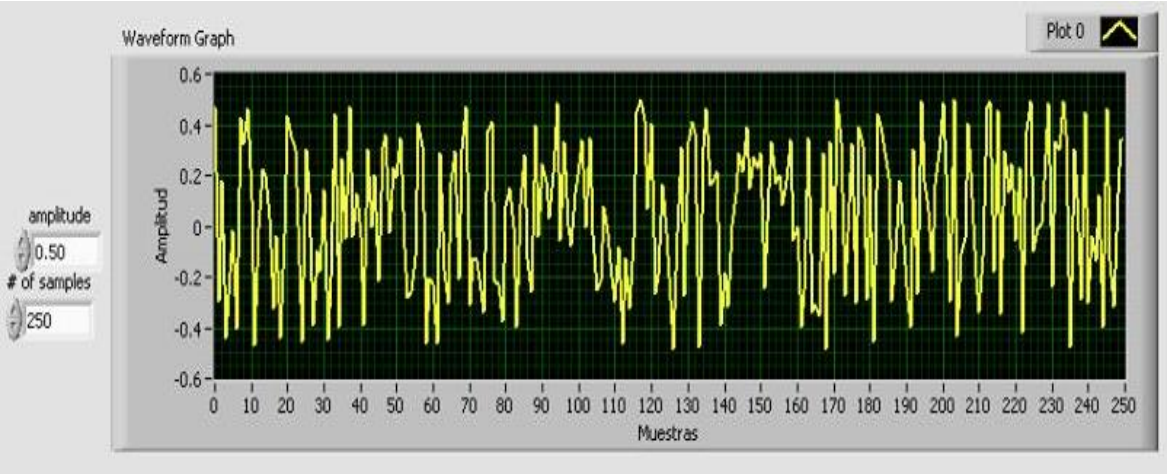
Otra de las pruebas realizadas consistió en cambiar el tipo de ruido, por lo que se recurrió a otro subVi del banco datos de LabVIEW™ que se conoce como *"Uniform White Noise"* (Figura 44). En éste es posible configurar algunas características de la señal como su amplitud y número de muestras. En la Figura 45 es posible apreciar gráficamente la señal ruido generada.

Figura 44. Ícono *Uniform White Noise.*



Fuente: LabView™.

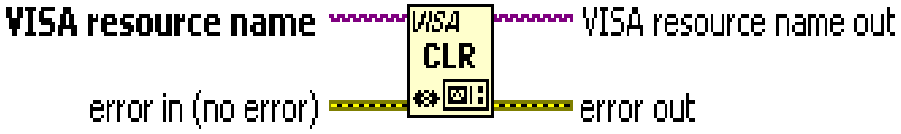
Figura 45. Señal de ruido blanco uniforme.



Fuente: LabView™.

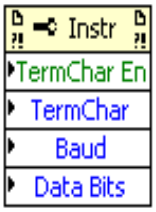
Por otra parte, se utiliza la función "*Visa Clear"* Figura 46 para la limpieza de los *buffers* de entrada y salida, evitando así datos no deseados o posibles errores. En seguida, se realiza la configuración del puerto serial con la función "*Property Node"* Figura 47*.* El número de *bytes* y el nombre del puerto se modifican en el Panel Frontal. La velocidad en baudios del puerto es 2400 bps, la misma velocidad especificada en el software *PSoC designer*. Todos los módulos de VISA se conectan de manera serial, evitando realizar la configuración de cada uno por separado.

Figura 46. Ícono *Visa Clear.*



Fuente: LabView™.

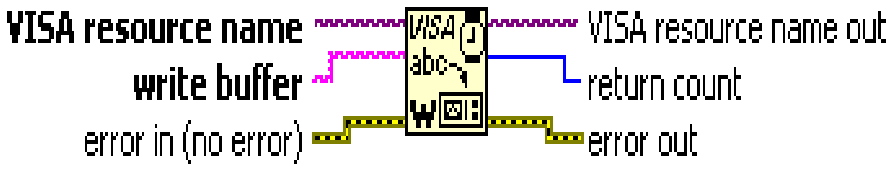
Figura 47. Ícono *Property Node.*



Fuente: LabView™.

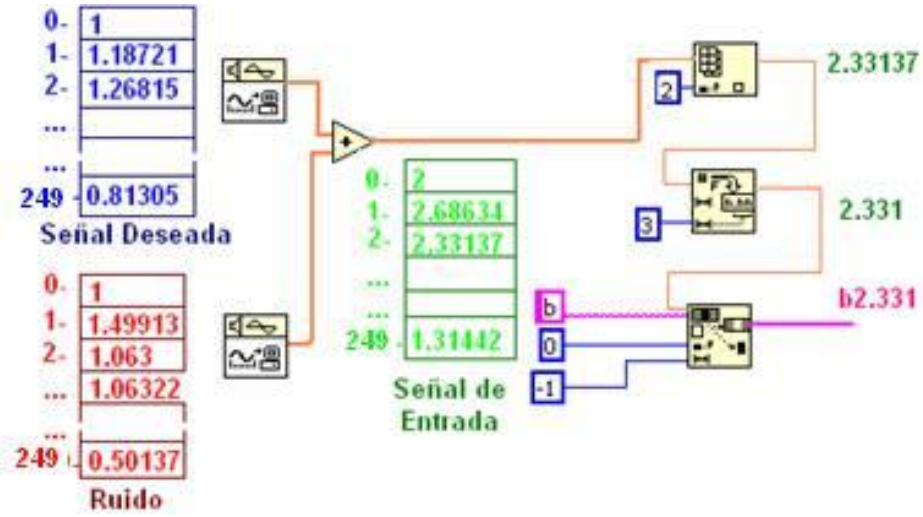
Antes de realizar el envío de datos de las señales de entrada al filtro, el usuario tiene la opción de seleccionar el tipo de filtro que se desea, ya sea Filtro Wiener o LMS. Específicamente, se envía la letra "*w*" ó "*l*" según el filtro seleccionado. En el envío de datos hacia la tarjeta se realiza la lectura dato por dato del vector de salida, se redondea a 3 dígitos decimales y en seguida se adicionan un bit de inicio, para este caso es la letra "*b*" ó *"a"* dependiendo de la señal enviada (entrada o deseada respectivamente). Por último los datos son enviados, utilizando la función "*Visa Write"* Figura 48. Este proceso puede ser observado en la Figura 49.

Figura 48. Ícono *Visa Write.*



Fuente: LabView™.

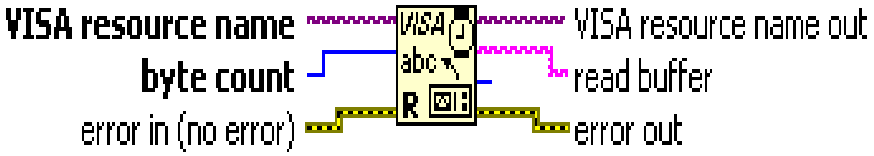
Figura 49. Proceso de envío de datos de la señal de entrada.



Fuente: LabView™.

La tarjeta PSoC realiza el filtrado de la señal, como se especificó en los capítulos anteriores y se espera la respuesta correspondiente. Con la ayuda de adaptador de puerto serial a USB conectado a un computador y con la función *Visa Read* Figura 50, se puede realizar esta lectura. Dicha lectura correspondiente a la salida del filtro es graficada e incluso almacenada para su posterior estudio y análisis.

Figura 50. Ícono Visa Read.



Fuente: LabView™.

6 PRUEBA Y RESULTADOS

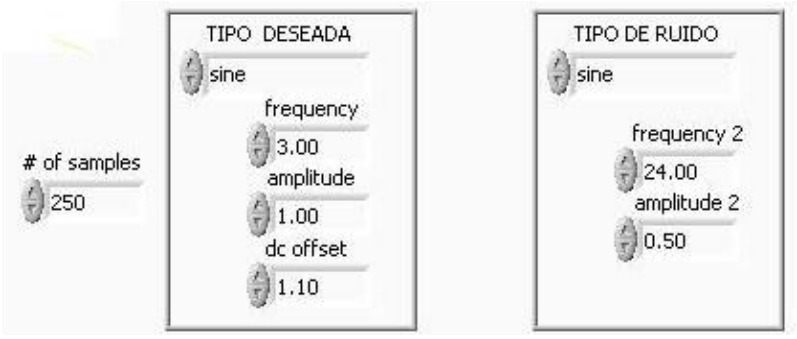
Después de la implementación del diseño en PSoC y en LabVIEW™ como se describió en el capítulo 3 y capítulo 5 respectivamente, y ya con los parámetros N y p definidos (ver capítulo 4) se realizaron una serie de pruebas en dos escenarios, el primero de ellos tomando como señal de entrada la suma de la señal deseada con una señal seno conocida y el segundo con un ruido blanco uniforme. Posteriormente y mediante simulaciones en el software MATLAB® se realizaron una serie de comparaciones y análisis para la evaluar del desempeño de los filtros.

6.1 FILTRO WIENER

6.1.1 RUIDO SENO

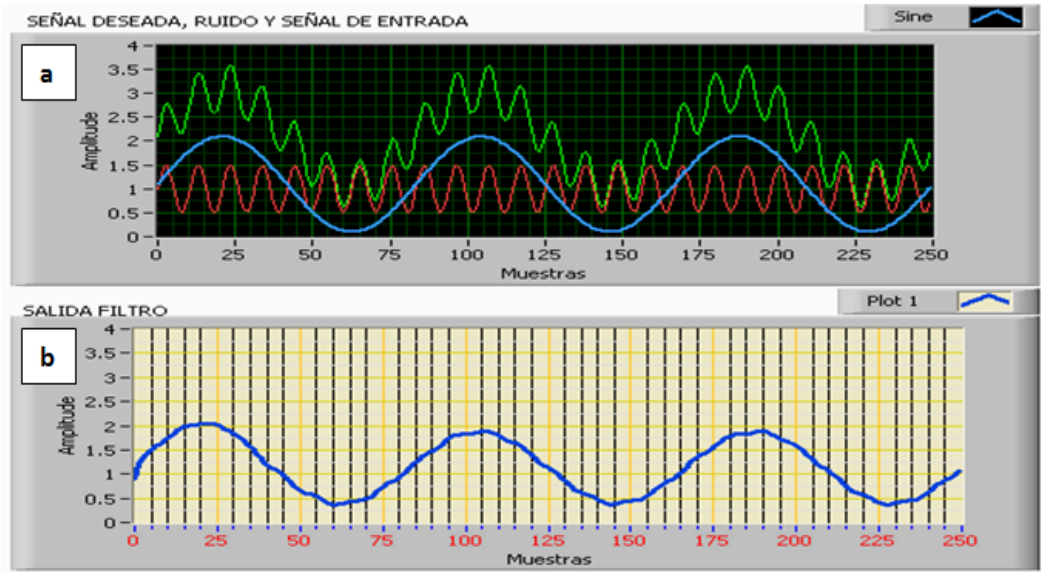
En un primer escenario, se selecciona como ruido una señal seno de mayor frecuencia y menor amplitud a la señal deseada a partir de los controles en LabVIEW™ (ver Figura 51). En la Figura 52 se observa la señal de salida del filtro Wiener, donde por inspección se observa la eliminación de la componente de continua y la atenuación del ruido.

Figura 51. Valores asignados a señal deseada y ruido.



Fuente: LabView™.

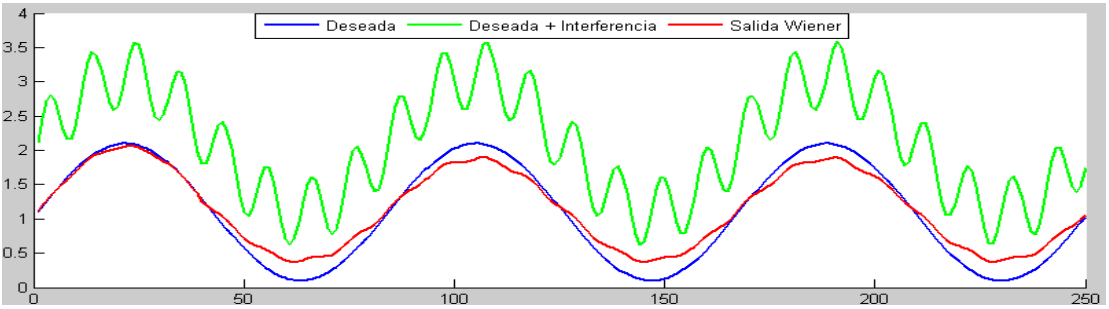
Figura 52. (a) Azul, señal deseada. Rojo, ruido. Verde, señal de entrada (Señal deseada + ruido). (b) Salida Filtro Wiener.



Fuente: LabView™.

Para realizar un estudio detallado de los resultados se diseñó un codigo en MATLAB® que permitiera leer los datos almacenados de las señales generadas en LabVIEW™ y realizar el filtro correpondiente. En la Figura 53, se puede observar la señal deseada, la señal de entrada y la salida del filtro simuladas.

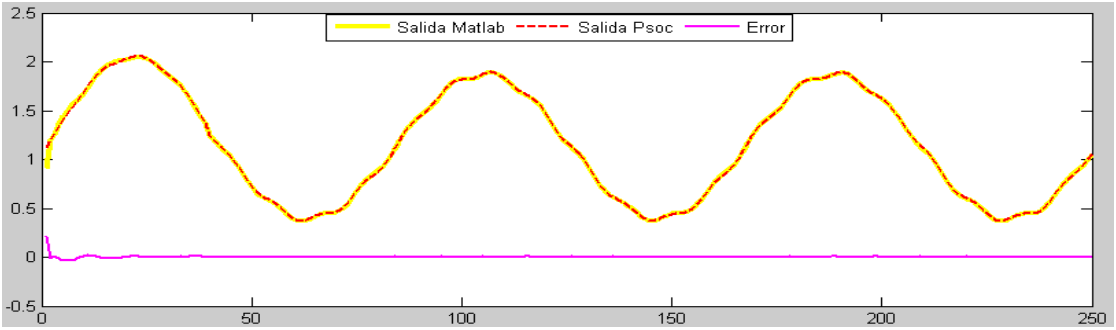
Figura 53. Simulación Filtro Wiener MATLAB®.



Fuente: Matlab®.

En la Figura 54 se realiza una comparación entre los datos de la señal filtrada obtenida con la tarjeta PSoC y la obtenida mediante simulación MATLAB® para evaluar que tan similares son los resultados de las mismas. De esta manera, es posible notar que las dos respuestas son similares y por ello se produce un error promedio de 0.0049. Esta pequeña diferencia puede haber sido ocasionada por el uso de las funciones prediseñadas de MATLAB®.

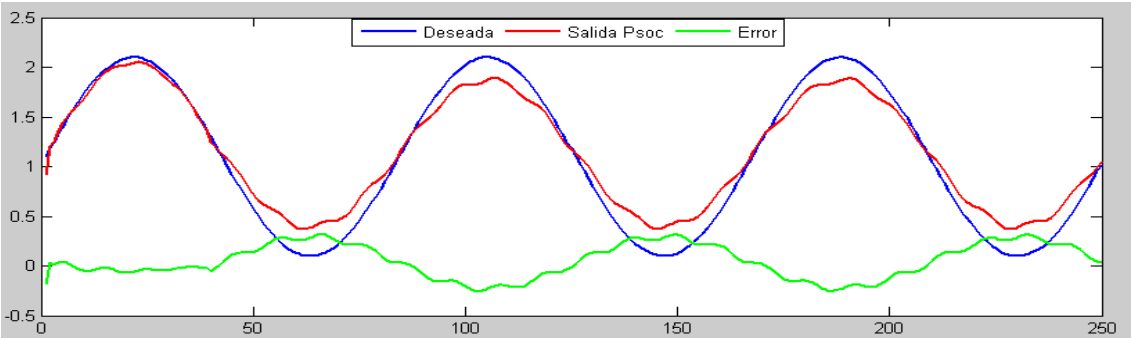
Figura 54. Comparación Filtrado Wiener PSoC y Filtrado por simulación en MATLAB® (Ruido seno).



Fuente: Matlab®.

Ahora, para evaluar el desempeño del filtro se compara la señal deseada con la de salida del filtro. En la Figura 55 se puede observar que la señal filtrada durante todo el proceso de filtrado tiende a seguir la forma de la señal deseada también, además tiene menor amplitud, lo que da lugar a una notable diferencia correspondiente al error en la figura mencionada.

Figura 55. Comparación entre la señal de deseada y la señal de salida en el Filtro Wiener (Ruido seno).

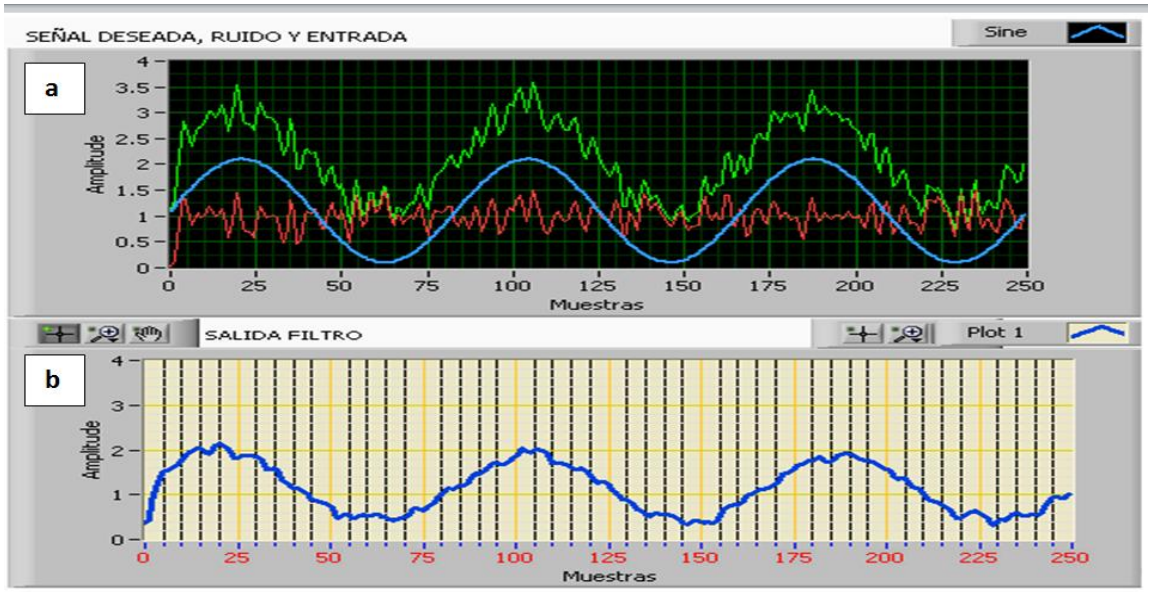


Fuente: Matlab®.

6.1.2 RUIDO BLANCO

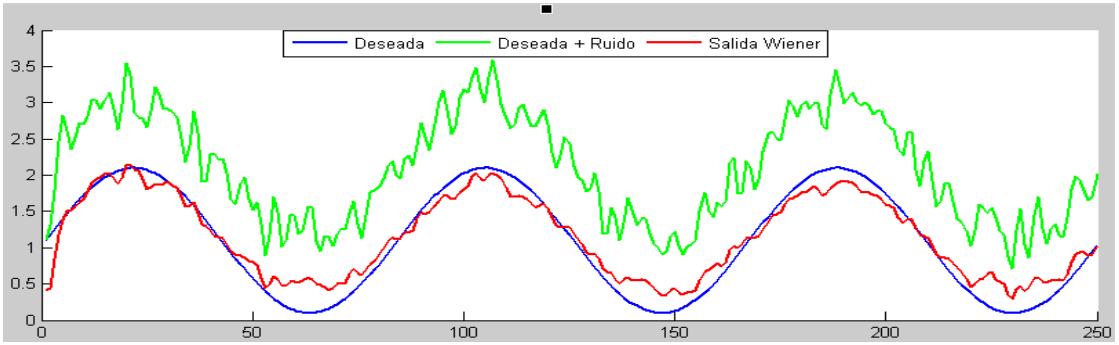
Para una segunda prueba, se mantienen las mismas características de la señal deseada pero esta vez se usa un ruido blanco uniforme de amplitud 0.5, éste corresponde a una señal aleatoria y se caracteriza por poseer todas las frecuencias. Después de realizar el filtrado de la señal de entrada en PSoC, se obtienen los resultados de la Figura 56 y se realizan simulaciones en MATLAB® obteniendo resultados de la Figura 57.

Figura 56. (a) Azul, señal deseada. Rojo, ruido, Verde, señal de entrada (Señal deseada + ruido). (b) Salida Filtro Wiener.



Fuente: LabView™.

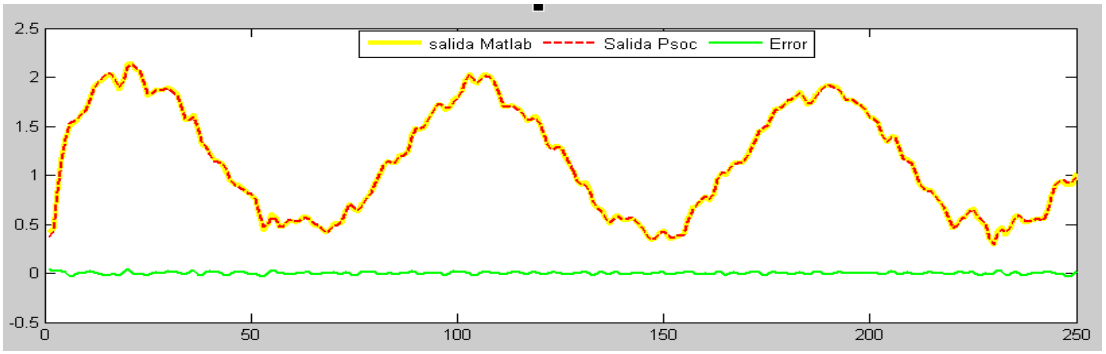
Figura 57. Simulación Filtro Wiener en MATLAB®.



Fuente: Matlab®.

En la Figura 58 se puede observar la salida del filtro obtenido en PSoC y la salida mediante simulación en MATLAB®, así como una medida de comparación correspondiente al error que en este caso tuvo un valor promedio de 0.0103 y con el que es posible verificar la similitud de los valores obtenidos en cada caso.

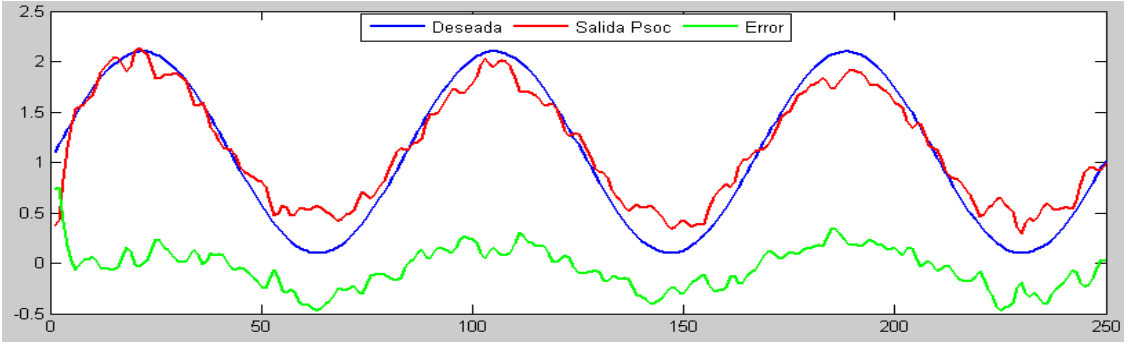
Figura 58. Comparación Filtrado Wiener PSoC y filtrado por simulación en MATLAB®. (Ruido blanco).



Fuente: Matlab®.

Ahora, igual que en la sección anterior se desea comparar la señal deseada con la salida del filtro, por lo que se obtienen los resultados de la Figura 59 con un error bastante notable y similar al de la Figura 55. Es decir, se conserva el mismo tipo de filtrado, con una gran atenuación del ruido propio de la señal de entrada pero con una disminución considerable de amplitud respecto a la señal deseada.

Figura 59. Comparación entre la señal de deseada y la señal de salida en el Filtro Wiener (Ruido blanco).



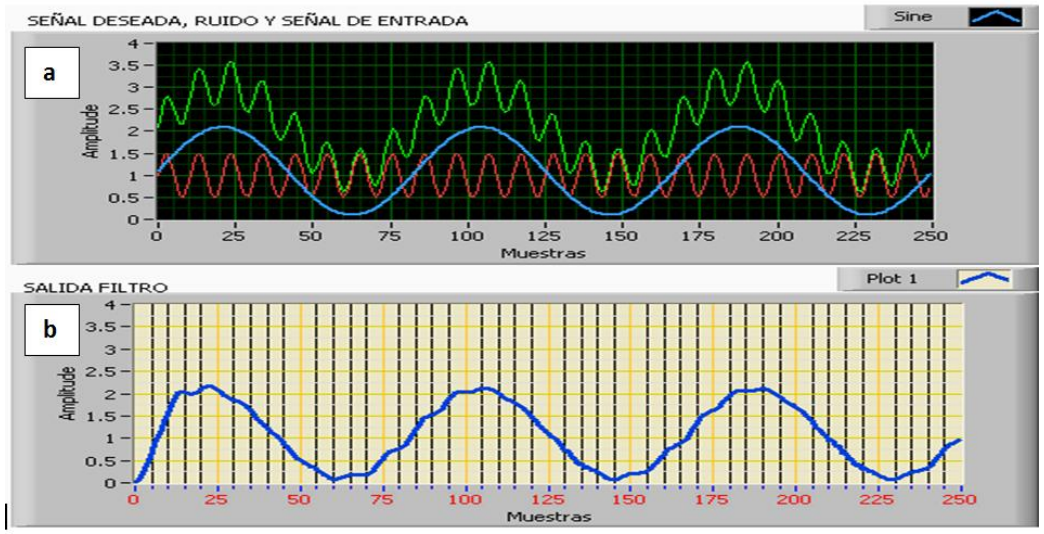
Fuente: Matlab®.

6.2 FILTRO LMS

6.2.1 RUIDO SENO

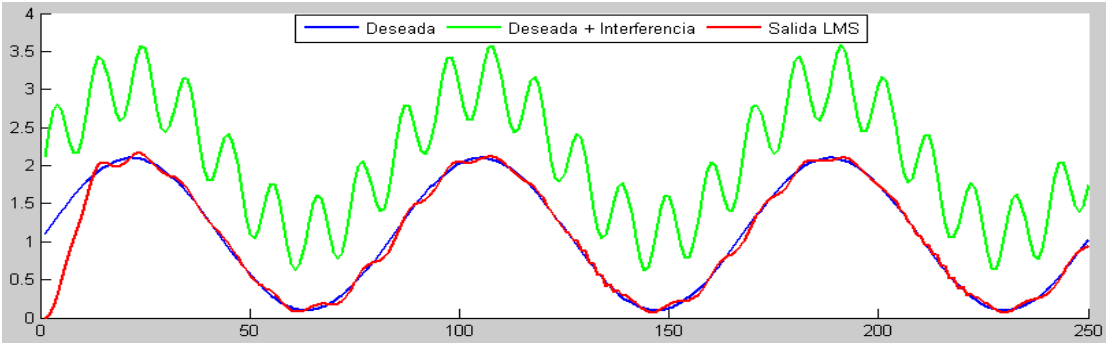
Las características de la señal de entrada y del ruido correspondiente a una señal seno son las mismas utilizadas en el filtro Wiener Figura 51. Los resultados obtenidos se pueden ver en la Figura 60.

Figura 60. (a) Azul, señal deseada. Rojo, ruido, Verde, señal de entrada (Señal deseada + ruido). (b) Salida Filtro LMS.



Fuente: LabView™.

Figura 61. Simulación Filtro LMS en MATLAB®.

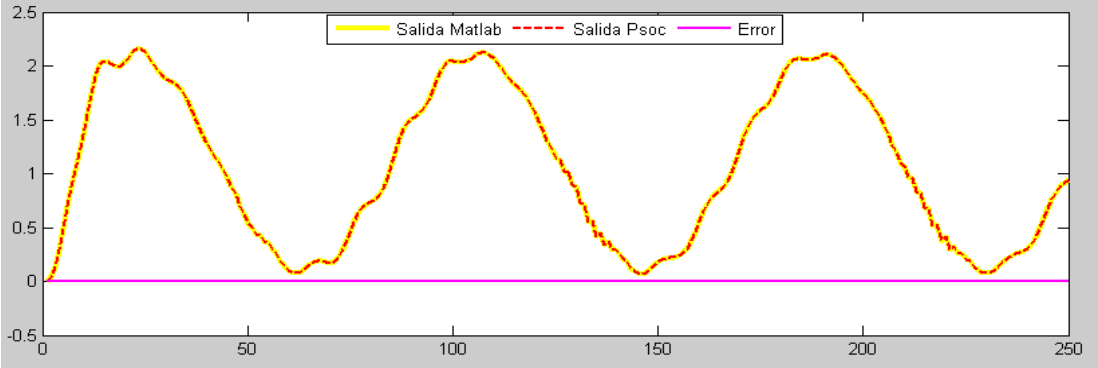


Fuente: Matlab®.

De la misma manera que en el Filtro Wiener, se realizan las simulaciones en MATLAB® para poder comparar los resultados obtenidos.

Al comparar la salida obtenida con PSoC y la obtenida mediante simulación en MATLAB®. (Figura 62), se observa que no hay mayor diferencia entre los valores de las dos señales, entonces el promedio del error es de 2.4473e-04.

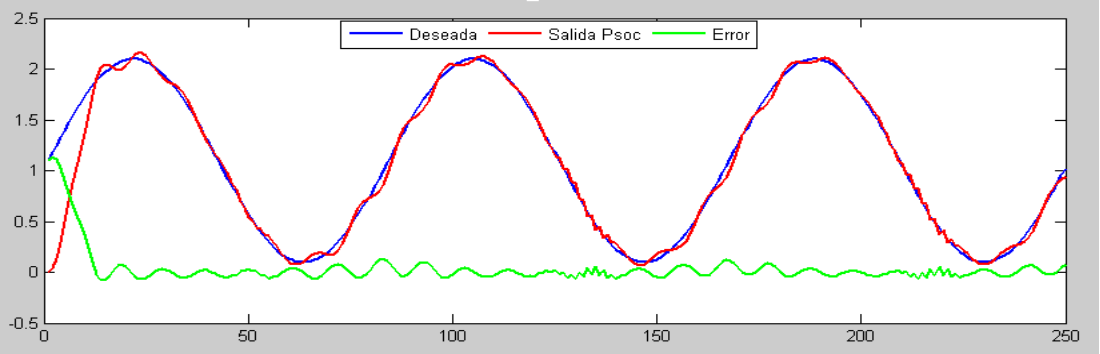
Figura 62. Comparación Filtrado LMS PSoC y filtrado por simulación en MATLAB® (Ruido seno).



Fuente: Matlab®.

Ahora, al comparar la salida del filtro con la señal deseada Figura 67, se puede observar que el error tiene su valor máximo en las primeras muestras debido a que el valor inicial del vector de pesos del filtro LMS w(0) es cero, como se explicó el sección 2.2. Más adelante la salida del filtro se asemeja a la señal deseada en amplitud y forma.

Figura 63. Comparación entre la señal de deseada y la señal de salida en el Filtro LMS (Ruido Seno).

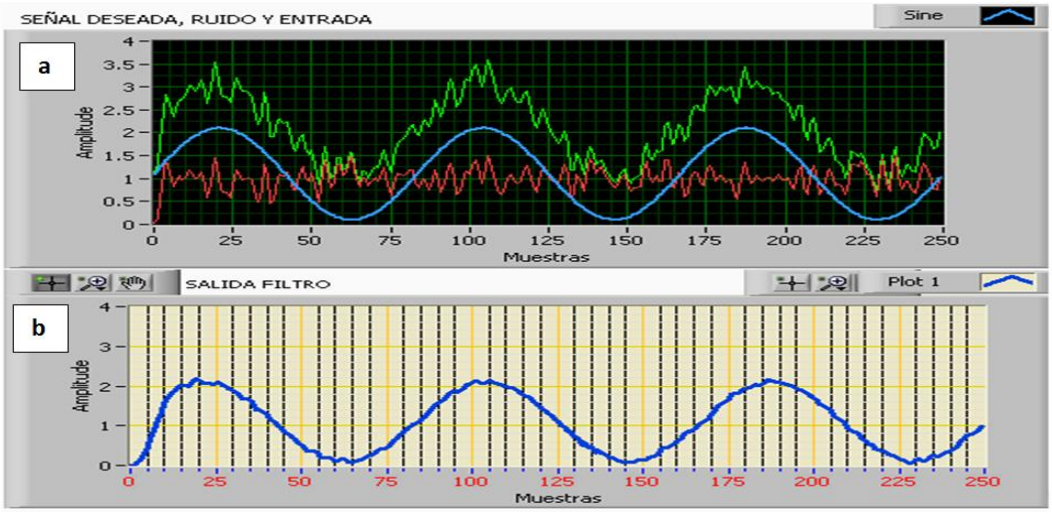


Fuente: Matlab®.

6.2.2 RUIDO BLANCO

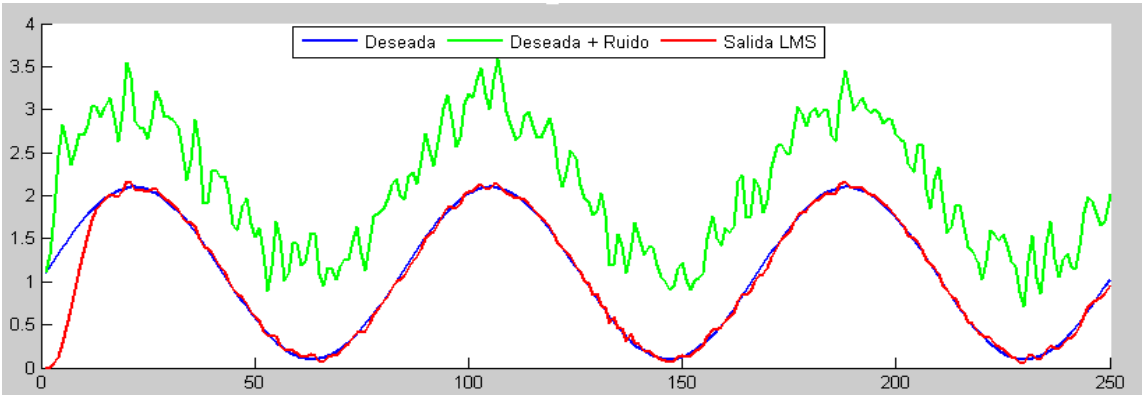
La segunda prueba del filtro se realiza bajo las mismas condiciones de la prueba realizada en el numeral 4.1.2, es decir, cambiando el tipo de ruido por un ruido blanco uniforme de amplitud 0.5. En la Figura 64 y la Figura 65 se observan los resultados obtenidos mediante la simulación de MATLAB®.

Figura 64. (a) Azul, señal deseada. Rojo, ruido, Verde, señal de entrada (Señal deseada + ruido). (b) Salida Filtro LMS.



Fuente: LabView™.

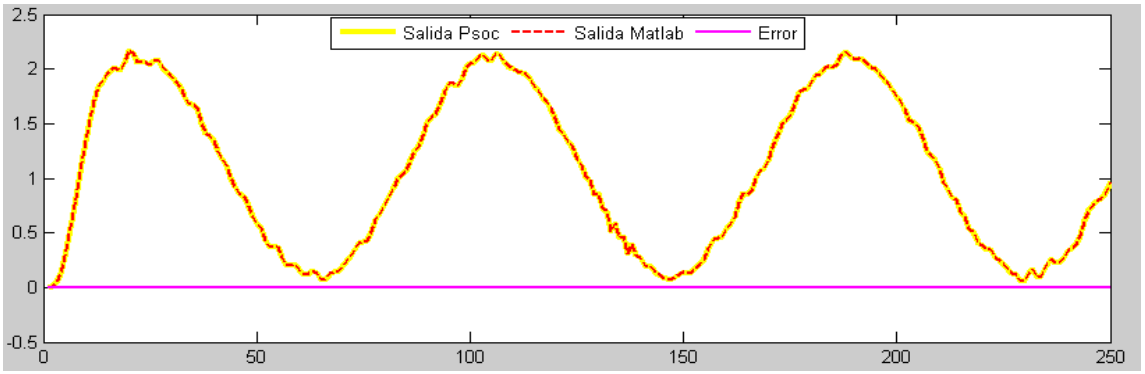
Figura 65. Simulación Filtro LMS en MATLAB®.



Fuente: Matlab®.

De nuevo se comparan las salidas de los filtros de PSoC y de MATLAB®. (Figura 66) hallando un error promedio de 2.4451e-04.

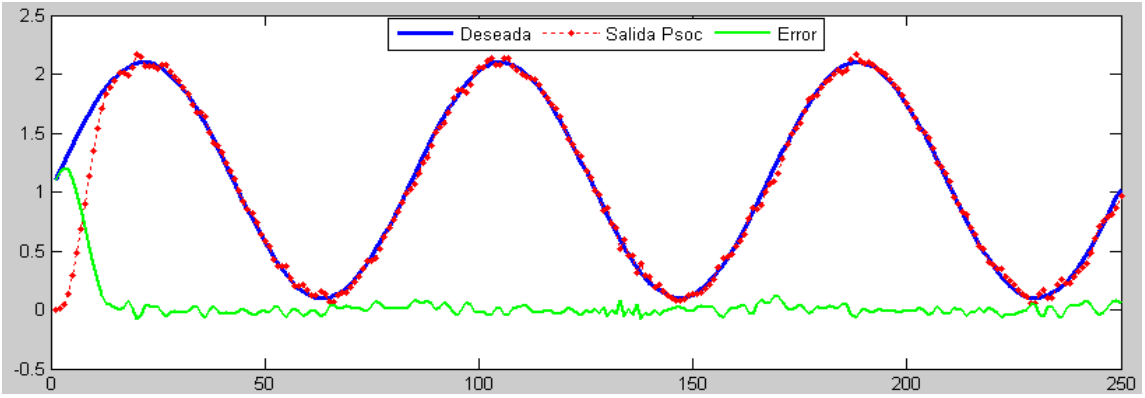
Figura 66. Comparación Filtrado LMS PSoC y filtrado por simulación en MATLAB®. (Ruido Blanco).



Fuente: Matlab®.

Luego, se realizó la comparación entre la señal deseada y la salida del filtro (Figura 67) donde los resultados encontrados fueron similares, es decir, el error es mayor en las primeras muestras pero la señal de salida posee cambios suaves entre un valor y otro.

Figura 67. Comparación entre la señal de deseada y la señal de salida en el Filtro LMS (Ruido Blanco).



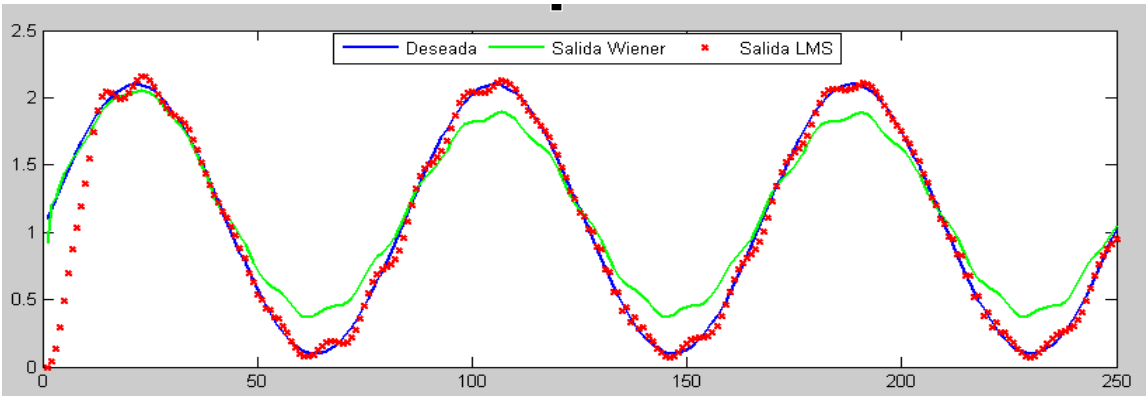
Fuente: Matlab®.

6.3 COMPARACIÓN FILTROS

6.3.1 CÁLCULO DE ERRORES

Por último, se comparan las salidas de los filtros Wiener y LMS en la Figura 68. El tipo de ruido es conocido y corresponde a una señal seno. El filtro Wiener tiene menor amplitud pero su forma se asemeja más a la señal deseada, caso contrario que el filtro LMS, ya que la amplitud es similar pero tiene mayores variaciones en la forma de la señal.

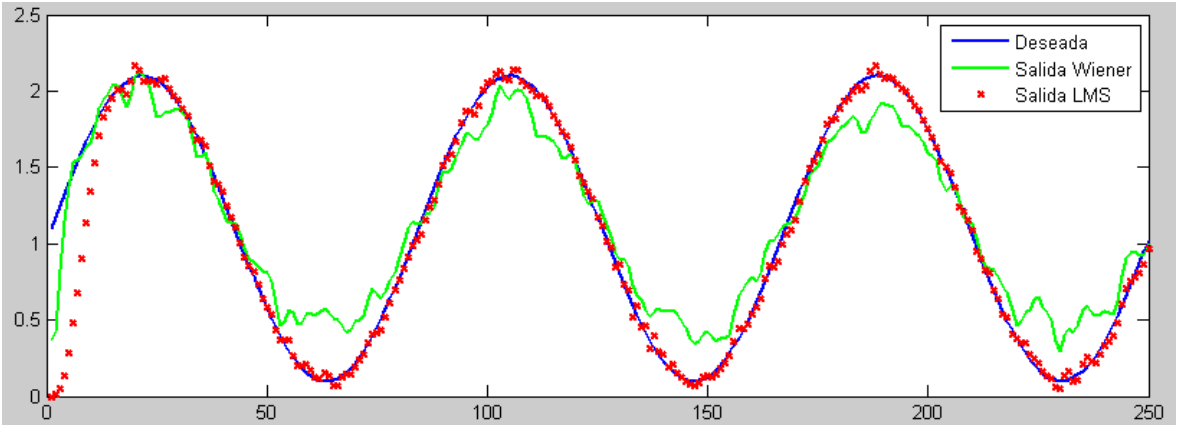
Figura 68. Comparación entre Filtrado Wiener y Filtrado LMS (Ruido Seno).



Fuente: Matlab®.

Por otra parte, al cambiar el tipo de ruido (Figura 69) se obtienen mejores resultados con el filtro LMS y el filtro Wiener parece guardar las mismas proporciones en amplitud pero posee una mayor distorsión.

Figura 69. Comparación entre Filtrado Wiener y Filtrado LMS (Ruido Blanco).



Fuente: Matlab®.

Para medir la eficiencia de los filtros y verificar su rendimiento se toman las siguientes medidas de desempeño, el error de identificación, el error en la estimación y el porcentaje de error [14]:

1. El error de identificación consiste en una medida del error en la salida del sistema adaptable con respecto a la salida deseada en presencia de ruido. Se define por la ecuación (65).

$$error = 10log_{10}\left(\frac{\sum_{n=1}^{N}\left(d(n)-y(n)\right)^2}{\sum_{n=1}^{N}\left(d(n)-r(n)\right)^2}\right)\ [dB] \qquad \textbf{(65)}$$

Donde N, es el número de muestras, es decir 250.

2. El error en la estimación también es una medida de desempeño y como se mencionó en el capítulo 1, está dado por la diferencia entre la señal deseada y la señal estimada [14]. Se define por la ecuación (57).
3. El porcentaje de error (% Error) que está dado por cociente entre el promedio del valor absoluto del error (la diferencia entre la señal deseada y estimada) y la amplitud de la señal deseada.

Los resultados de dicho errores ante el ruido correspondiente a la interferencia seno y al ruido blanco uniforme se pueden observan en las Tabla 2 y Tabla 3, respectivamente.

Tabla 2. Errores de identificación y estimación (Ruido: Señal Seno)

	FILTRO WIENER	**FILTRO LMS**
Error de Identificación	-13.0463 [dB]	-12.1466 [dB]
Promedio del Error de Estimación	-0.0498	0.0350
% Error	14.95	6.73

Fuente: Autores.

Tabla 3. Errores de identificación y estimación (Ruido: Blanco Uniforme)

	FILTRO WIENER	**FILTRO LMS**
Error de Identificación	-11.2411 [dB]	-11.8263 [dB]
Promedio del Error de Estimación	-0.0478	0.0410
% Error	17	6.71

Fuente: Autores.

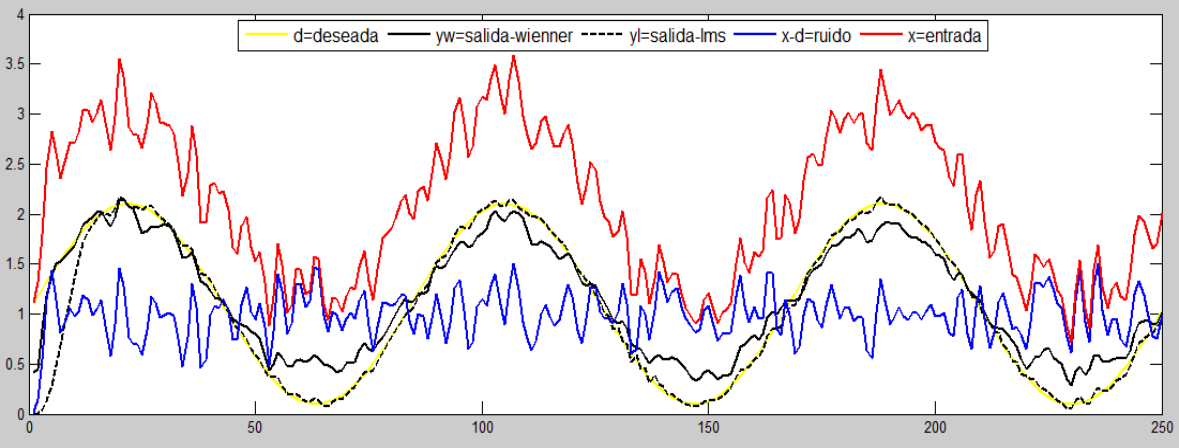
6.3.2 ANÁLISIS EN FRECUENCIA

Además de los cálculos anteriores es posible realizar un estudio frecuencial de las variables involucradas.

- **Ruido Blanco**

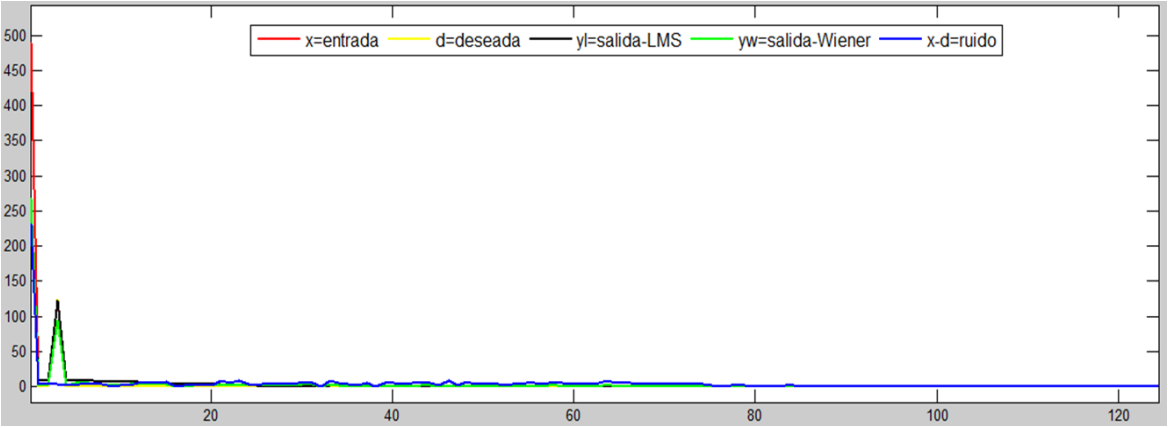
De esta manera y en el caso del filtrado de la señal con ruido blanco, para las señales de la Figura 70 es posible encontrar su equivalente en frecuencia en la Figura 71 y la Figura 72.

Figura 70. Señales en el tiempo involucradas en filtrado con ruido blanco.



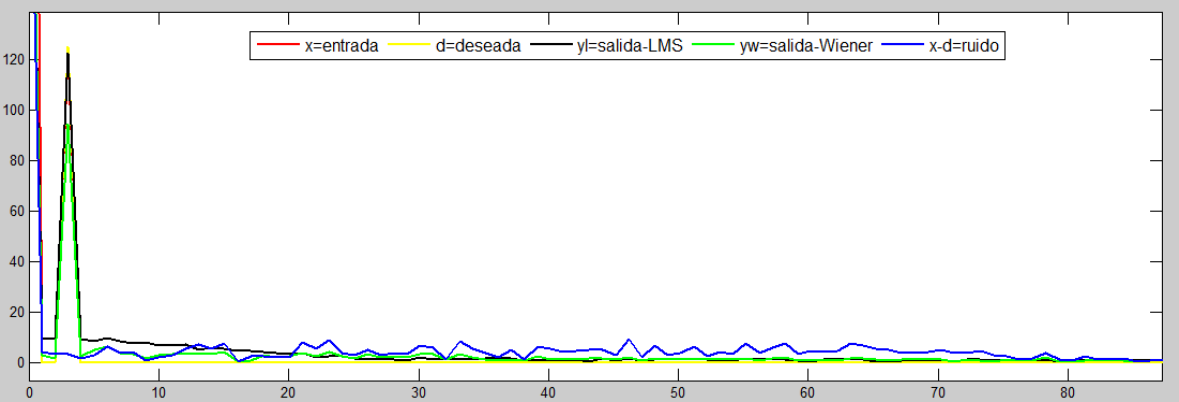
Fuente: Matlab®.

Figura 71. Señales en frecuencia involucradas en filtrado con ruido blanco.



Fuente: Matlab®.

Figura 72. Señales en frecuencia involucradas en filtrado con ruido blanco (Zoom).



Fuente: Matlab®.

En las gráficas de las señales en el dominio de la frecuencia es posible observar el máximo valor de la señal deseada en la frecuencia de 3 Hz. En la Tabla 4 es posible conocer el valor de la señal de entrada y las salidas de los dos tipos de filtrado respecto a la señal deseada en esta frecuencia. Además, se muestran los porcentajes de distorsión armónica total para los dos tipos de filtrado.

Tabla 4. Características en frecuencia de señales involucradas en filtrado con ruido blanco.

	Deseada	Entrada	Salida LMS	Salida Wiener
Valor del armónico en F=3 Hz	125	121.9	122.7	94.37
Porcentaje respecto a la señal deseada			98.16%	75.5%
% THD			3.48	11.59

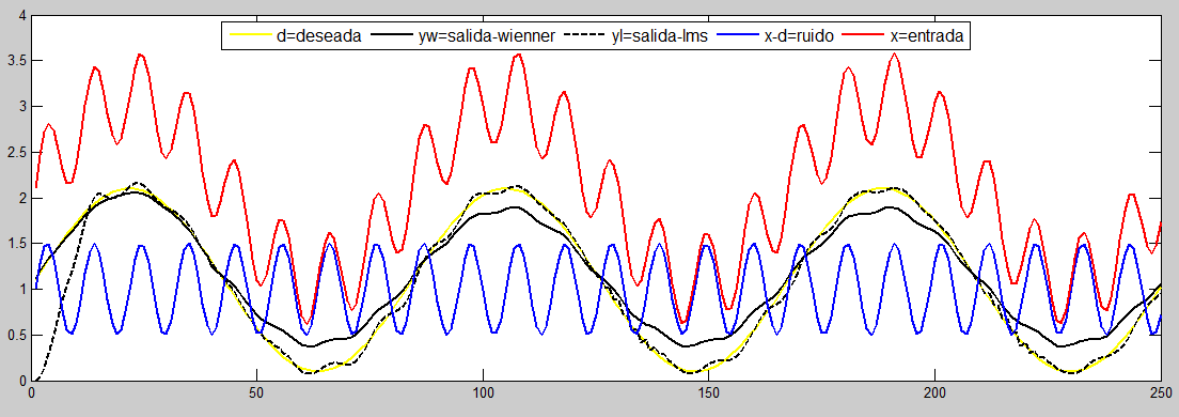
Fuente: Autores.

Además de lo anterior, en la Figura 72 es posible apreciar los valores del ruido blanco así como de valores más bajos de las señales de salida en todo el espectro de frecuencia y en la Figura 71 se observan los valores de continua de todas las señales.

- **Ruido Seno**

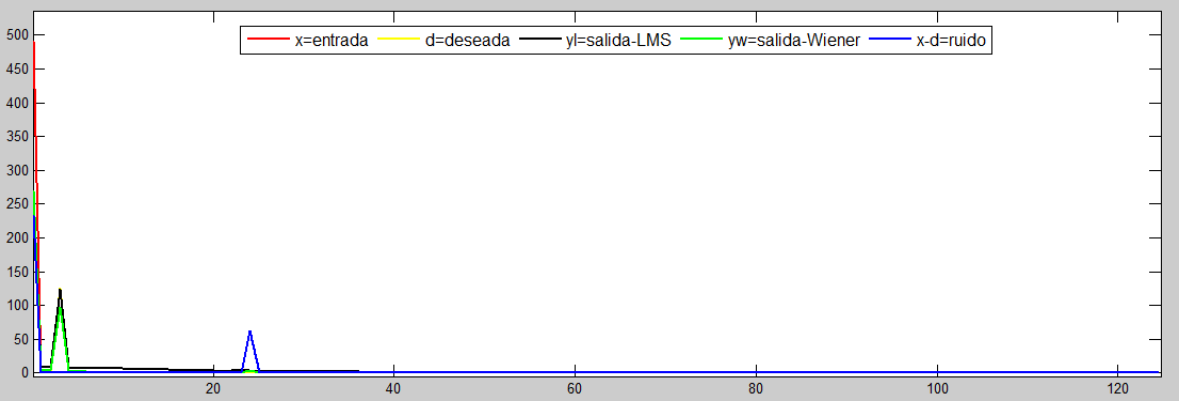
Finalmente en el caso del filtrado de la señal con ruido seno y para las señales de la Figura 73 es posible encontrar su equivalente en frecuencia en la Figura 74 y Figura 75.

Figura 73. Señales en el tiempo involucradas en filtrado con ruido seno.



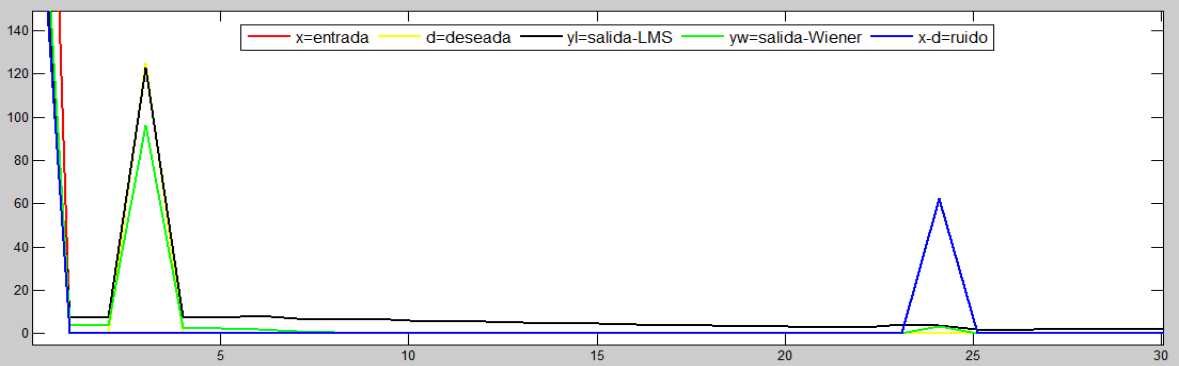
Fuente: Matlab®.

Figura 74. Señales en frecuencia involucradas en filtrado con ruido seno.



Fuente: Matlab®.

Figura 75. Señales en frecuencia involucradas en filtrado con ruido seno (Zoom).



Fuente: Matlab®.

En las gráficas de las señales en el dominio de la frecuencia es posible observar el máximo valor de la señal deseada en una frecuencia de 3 Hz y el máximo valor de la señal de ruido seno en una frecuencia mayor correspondiente a 24 Hz, lo que corresponde con sus características en el tiempo. En la Tabla 5 es posible conocer el valor de la señal de entrada y las salidas de los dos tipos de filtrado respecto a la señal deseada en el armónico de la señal deseada es la frecuencia de 3 Hz, así como el valor de la señal ruido y las salidas de los filtros respecto a ella en la frecuencia del armónico de la señal de ruido, es decir 24 Hz. Además, se muestran los porcentajes de distorsión armónica total para los dos tipos de filtrado.

Tabla 5. Características en frecuencia de señales involucradas en filtrado con ruido seno.

	Deseada	Entrada	Salida LMS	Salida Wiener	RB
Valor del armónico en F = 3Hz	125	125	123.1	96.23	0
Porcentaje respecto a la señal deseada			98.48 %	76.98 %	
Valor del armónico en F = 24 Hz	0	62.5	3.99	3.55	62.5
Porcentaje respecto al ruido seno			6.39%	5.68%	
% THD			5.58	4.23	

Fuente: Autores.

6.3.3 TIEMPOS DE CÓMPUTO

Finalmente se realizaron una serie de pruebas para encontrar el tiempo que cada filtro usa para procesar y obtener una salida adaptable. Los resultados pueden observarse en la Tabla 6.

Tabla 6. Tiempos de cómputo.

Filtro Wiener		**Filtro LMS**
Función	**Tiempo [ms]**	**Tiempo [ms]**
Autocorrelación	217	207
Matriz toeplitz	1	
Correlación	64	
Cholesky	88	
F. Inversa	402	
Coeficientes	12	
FirWiener	141	
Total	925	207

Fuente: Autores.

7 CONCLUSIONES

- Se realizó la programación de los filtros adaptativos Wiener y LMS en la herramienta de software PSoC *creator* por medio del uso del lenguaje c y empleando diferentes tipos de funciones diseñadas. Además de dicha programación también se programó el proceso de envío y recepción de datos por medio de algunos bloques funcionales del sistema de desarrollo PSoC.

- El programa realizado en LabVIEW™ permite la generación de señales conocidas a través de la selección de ciertos parámetros como Amplitud, fase, número de muestras, nivel de continua, dichas señales actúan como entradas al sistema permitiendo realizar el filtrado adaptativo ante la presencia de diferentes tipo de señales, en unos casos con una señal de ruido blanco y en otros, con la presencia de una señal seno de mayor frecuencia como interferencia.

- El programa realizado en LabVIEW™ brinda una interfaz amigable, permitiendo al usuario visualizar los datos de entrada y salida de los filtros de forma gráfica, realizar el almacenamiento de las señales requeridas para su análisis y seleccionar el tipo de filtrado adaptativo requerido (Wiener o LMS).

- El software MATLAB® representó una importante herramienta de apoyo no solo para el diseño de los filtros adaptativos sino para el análisis de las señales obtenidas a la salida de los filtros. Además de esta herramienta fue útil el uso del software DEV-C++ pues fue evidente la necesidad del uso del lenguaje c como requisitos para adecuar los diseño en el sistema de desarrollo PSoC.

- La principal característica del algoritmo del filtro adaptativo LMS encontrada fue su sencillez y simplicidad, tanto en los requisitos de memoria como en la capacidad computacional ya que no requiere medida de las funciones de

correlación, inversión de matrices, entre otras. Pero uno de sus aspectos determinantes a tener muy en cuenta consistió en la dependencia del mismo de la velocidad de convergencia (μ), por lo que fue necesario fijar su valor teniendo en cuenta el error de salida del filtro y su rapidez de adaptación y de esta manera elegir un valor óptimo y eficiente.

- Se valoró el desempeño de los filtros adaptativos Wiener y LMS en la eliminación de ruido de señales conocidas de baja frecuencia, realizando pruebas con dos tipos diferentes de ruido frente a una señal seno de amplitud y frecuencia conocida. Dichos procesos de filtrado lograron cumplir satisfactoriamente con los objetivos propuestos al reducir el ruido de la señal de entrada y así disminuir la señal de error.

- El error de identificación y el promedio del error de estimación del filtro Wiener son mayores a los del filtro LMS tanto en las pruebas realizadas con el ruido blanco como en las pruebas con el ruido seno de mayor frecuencia. Lo que lleva a concluir que el filtrado LMS además de tener menores tiempos de cómputo (casi 5 veces menor al filtro Wiener), arrojó mejores resultados y su señal de salida se acercó un poco más a la señal de referencia, pues aunque la salida del filtro Wiener se caracterizó por una más curva suave y con menores sobresaltos que la señal de salida LMS, está ultima se acercó más en amplitud a la señal deseada.

- Los resultados obtenidos también fueron comparados en el dominio de la frecuencia, mediante este análisis se obtuvo que: En el caso del ruido blanco y observando los valores de las señales en la frecuencia correspondiente a 3 Hz (frecuencia de la señal deseada), la salida del filtro LMS correspondió al 98.16% de la señal deseada, mientras que la salida del filtro Wiener solo llegó al 75.5%. Ahora, en el caso del ruido seno, observando inicialmente los valores de las señales correspondientes a la frecuencia de 3 Hz, la salida del filtro LMS correspondió al 98.48% de la señal deseada, mientras que la salida del filtro

Wiener solo llegó al 76.98% y observando los valores de las señales en la frecuencia correspondiente a 24 Hz (frecuencia de la señal de ruido: seno) la salida del filtro LMS correspondió al 6.39% de la señal de ruido, mientras que la salida del filtro Wiener bajó al 5.68%, lo que muestra que aunque el filtro LMS alcanzó un valor más cercano respecto al armónico de la señal deseada, también tuvo una menor eliminación del armónico de la señal de ruido.

8 OBSERVACIONES Y RECOMENDACIONES

- El presente proyecto estuvo orientado al desarrollo de un proyecto de posgrado en el que el filtrado adaptativo Wiener y LMS esta vez sería realizado a señales reales de baja frecuencia, en concreto a de señales fisiológicas como lo son la señal electrocardiográfica y la de presión arterial por lo que se sugiere continuar con la presente investigación, no solo empleando los algoritmos que se diseñaron sino también mediante la implementación de otros tipos de filtros como el filtro NLMS y Kalman.

- Para posteriores pruebas y en el caso del filtro LMS es necesario redefinir el valor de la velocidad de convergencia μ, siendo este uno de los parámetros determinantes en el desempeño del filtro.

- Se sugiere la implementación de los filtros adaptativos en otros microcontroladores, para comparar sus ventajas o desventajas frente al sistema de desarrollo utilizado PSoC en cuanto a parámetros como tiempo de cómputo y costo.

- Es posible realizar el diseño en PSoC de un generador de señales, así como la adquisición de las mismas a través de los conversores DAC (conversor digital-analógico) y ADC (conversor analógico-digital) encontrados como bloques funcionales del sistema de desarrollo.

BIBLIOGRAFÍA

[1] Alan V. Oppenheim, Ronald W. Schafer y John R. Buck, "the convolution theorem," in Discrete-Time signal processing, 2nd ed. Prentice-Hall, Upper Saddle River, N.J.:,1997, cap. 2.9.6, pp. 60-61.

[2] A. Ariza, I. Galindo, G. Giraldo. (2012). Mezclador digital de audio de 4 canales. [Online]. Available: http://linuxencaja.net/wiki/Proyecto_de_Grado:_Mezclador_Digital_de_Audio_de_4_canales. [Accessed: 16-Jul-2013].

[3] C. A. Durán. "Implementación del Algoritmo NLMS en un DSP," M.S. thesis, Instituto Politécnico Nacional. Escuela Superior de Ingeniería Mecánica y Eléctrica. México, May. 2010.

[4] E. F. Alba Blanco. J. Ruiz. (Volumen 10 No. 2 Jun-Aug 2009). "Implementación de filtros digitales en FPGA". Instituto Superior Politécnico José A. Echeverría (ISPJAE). Cuba [Online]. Available: http://www.bvs.sld.cu/revistas/bfm2/Volumenes%20anteriores.pdf/vol10/no2/icid02209.pdf. [Accessed: 16-Jul-2013].

[5] Introducción a la teoría de procesamiento Digital de Señales de Audio. Universidad de la República. Escuela Universitaria de Música. Uruguay. (2011) [Online]. Available: http://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/clase10.pdf. [Accessed: 16-Jul-2013].

[6] John G. Proakis y Dimitris G. Manolakis, "correlation of Discrete-Time signal," in Digital signal processing, 3rd ed. Prentice-Hall, Upper Saddle River, N.J.:,1996, cap 2.3, pp. 118-122.

[7] J. G. Ávalos Ochoa. "Algoritmo LMS con Error Codificado Usando un DSP," M.S. thesis, Instituto Politécnico Nacional. México, 2008.

[8] J. Narváez, D. Gamboa, A. Lara, V. Tapia y D. Parreño. (2011, Jan)."Filtros Digitales FIR e IIR (Conceptos de funcionamientos y diseño por ventanas)".Escuela Superior Politécnica de Chimborazo, Ecuador [Online]. Available: http://es.scribd.com/doc/47265461/filtros-paper. [Accessed: 16-Jul-2013].

[9] J. O. Ruiz. (2010). "Implementación de filtros Adaptativos en tecnologías de Lógica Reconfigurable. Revista Colombiana de Tecnología de Avanzada. Volumen 2, numero 16. [Online] Available: http://www.unipamplona.edu.co/unipamplona/portalIG/home_40/recursos/03_v13_18/revista_17/03122011/14.pdf. [Accessed: 16-Jul-2013].

[10] J. Velazquez Lopez, J. C. Sanchez Garcia, H. Perez Meana, “Adaptive filters with codified error LMS Algorithm”, International Journal Electromagnetic Waves and Electronic Systems, Jul. 2006, pp. 23 - 28.

[11] L. Caiza, S. Asadovay, M. Tixi y L. Martínez. (2010, Dec). "Filtro Wiener". Escuela Superior Politécnica de Chimborazo, Ecuador [Online]. Available:http://es.scribd.com/doc/45811181/Filtro-Wiener-doc. [Accessed: 15-Jul-2013].

[12] Monson H. Hayes, "The cholesky descomposition," in Statistical Digital Signal Processing and Modeling, 1st ed. John Wiley & Sons, Inc, New York:,1996, cap 5.2.7, pp. 250-254.

[13] M. E. Iglesias, “Técnicas de filtrado adaptativo y estadística de orden superior para la implementación de algoritmos de cancelación de ruido sobre una arquitectura reconfigurable,” M.S. thesis, Instituto superior politécnico Jose Antonio Echeverria (CUJAE), La Habana, Cuba, 2011.

[14] M. García Hernández, “Propuesta y evaluación de un algoritmo de filtrado adaptable,” M.S. thesis, Universidad Autónoma Metropolitana Iztapalapa, México, 2008.

[15] Physionet. "Algoritmo LMS (Least-Mean-Square)". [Online]. Available:http://physionet.cps.unizar.es/~eduardo/docencia/tds/librohtml/lms1.htm. [Accessed: 16-Jul-2013].

[16] Physionet, "Filtrado de Wiener FIR". [Online]. Available: http://physionet.cps.unizar.es/~eduardo/docencia/tds/librohtml/fir1.htm. [Accessed: 15-Jul-2013].

[17] Physionet, "Filtrado lineal óptimo: Filtrado de Wiener".[Online]. Available: http://physionet.cps.unizar.es/~eduardo/docencia/tds/librohtml/wiener1.htm#filtros. [Accessed: 15-Jul-2013].

[18] PSoC®-Programmable System-on-Chip. PSoC Portfolio, Technology, Software, Kits, Applications. [Online]. Available: http://www.cypress.com/psoc/. [Accessed: 16-Jul-2013].

[19] Robert M. Gray. (2000, March). "Toeplitz and Circulant Matrices: A review". Stanford University [Online]. Available: http://ee.stanford.edu/~gray/toeplitz.pdf. [Accessed: 15-Jul-2013].

[20] Steven C. Chapra y Raymond P. Canele, "Descomposición LU e inversión de matrices," in Métodos Numéricos para ingenieros, 5ta ed. McGraw Hill, México:, 2007, cap 10, pp. 284-299.

[21] U.N.S. "Diseño de filtros digitales," in Procesamiento Digital de Señales. Universidad nacional del Sur U.N.S... 2011. Available: http://www.ingelec.uns.edu.ar/pds2803/Materiales/Cap07/07-Cap07.pdf.[Accessed: 16-Jul-2013].

[22] Y. Vera M. "Cancelación Activa de ruido," M.S. thesis, Universidad de los Andes. Facultad de Ingeniería. Escuela de Ingeniería eléctrica. Venezuela, 2008, Apr.

ANEXOS

ANEXO A. CÓDIGO MATLAB®

```
%close all
clear all
clc

P=8;        %grado filtro
N=100;      %muestras

%deseada
d=[1.000  1.187  1.368  1.536  1.685  1.809  1.905  1.969  1.998  1.992  1.951
1.876  1.771  1.637  1.482  1.309  1.125  0.937  0.751  0.574  0.412  0.271  0.156
0.070  0.018  0.000  0.018  0.070  0.156  0.271  0.412  0.574  0.751  0.937  1.125
1.309  1.482  1.637  1.771  1.876  1.951000  1.992  1.998  1.969  1.905  1.809
1.685  1.536  1.368  1.187  1.000  0.813  0.632  0.464  0.315  0.191  0.095  0.031
0.002  0.008  0.049  0.124  0.229  0.363  0.518  0.691  0.875  1.063  1.249  1.426
1.588  1.729  1.844  1.930  1.982  2.000  1.982  1.930  1.844  1.729  1.588  1.426
1.249  1.063  0.875  0.691  0.518  0.363  0.229  0.124  0.049  0.008  0.002  0.031
0.095  0.191  0.315  0.464  0.632  0.813]';
%entrada
x=[2.000  2.686  2.431  2.045  2.560  3.285  3.089  2.516  2.757  3.414  3.245
2.491  2.428  2.980  2.867  2.015  1.703  2.178  2.204  1.390  0.937  1.395  1.647
1.008  0.519  1.000  1.517  1.133  0.665  1.147  1.888  1.758  1.299  1.696  2.547
2.603  2.096  2.295  3.113  3.262  2.657  2.570  3.239  3.421  2.721  2.333  2.809
3.027  2.305  1.688  2.000  2.312  1.695  0.973  1.191  1.667  1.279  0.579  0.761
1.430  1.343  0.738  0.887  1.705  1.904  1.397  1.453  2.304  2.701  2.242  2.112
2.853  3.335  2.867  2.483  3.000  3.481  2.992  2.353  2.605  3.063  2.610  1.796
1.822  2.297  1.985  1.133  1.020  1.572  1.509  0.755  0.586  1.243  1.484  0.911
0.715  1.440  1.955  1.569  1.314]';

% FILTRO WIENER
r=xcorr(x);
R=toeplitz(r(N:N+P-1));
ptemp=xcorr(x,d);
p=ptemp(N:N+P-1);
w=R\p;    %% coeficientes del  filtro
xrec=filter(w,1,x);
error=d-xrec;

% FILTRO LMS
mu_step=0.01;
y = zeros(N,1); % initialize filter output vector
w = zeros(P,1); % initialize filter coefficient vector
e = zeros(N,1); % initialize error vector
W = zeros(P,N); % filter coefficient matrix for coeff. history

for n = 1:N
  if n <= P % assume zero-samples for delayed data that isn't available
      k = n:-1:1;
```

```
      x1 = [x(k); zeros(P-n,1)];
  else
      x1 = x(n:-1:n-P+1); % M samples of x in reverse order
  end
  y(n) = w'*x1; % filter output
  e(n) = d(n) - y(n); % error
  w = w + mu_step*e(n)'*x1; % update filter coefficients
end
toc;

%%
figure
plot(d,'y','LineWidth',1); hold on  %deseada
plot(x,'r','LineWidth',1);          %entrada
plot(xrec,'k','LineWidth',1);          %salida
plot(error,'b','LineWidth',1);          %error

plot(y,'--k','LineWidth',1);          %salida
plot(e,'m','LineWidth',1);          %error

legend('d=ruido','x=entrada','xrec=salida_wienner','error_wienner=deseada
-salida','y=salida_lms','e=error_lms');
```

ANEXO B. CÓDIGO PSOC®

```
#include <device.h>
#define VAL 100000
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define p 8
#define n 100

int16  run1=0;
int16  runfiltro=0;
char CadenaPuerto[15]={'\0'};
float prueba_buffer;
float Sentrada[100]={'\0'};
float Sdeseada[100]={'\0'};
float filtro[100]={'\0'};// p=8
float filtro1[100]={'\0'};// p=8
float auto_cor[(2*100)-1]={'\0'};//(2*n)-1
float toeplitz[8][8];//p=8
float x_cor[8]={'\0'};//p=8
float matriz_L[8][8];//p=8
float matriz_U[8][8];//p=8
float inversa[8][8];// p=8
float co[8]={'\0'};// p=8
int16 contador_e={'\0'};
int16 contador_d={'\0'};
float entrada_imp;
int16 contador_fuera={'\0'};
int16 lleno_b={'\0'};
int16 lleno_a={'\0'};
char mibuffer[5]={'\0'};
int16 ib;
int16 incremento={'\0'};
int16 bandera_a={'\0'};
int16 bandera_b={'\0'};
int16 contador_prueba_8={'\0'};
float prueba_8[8]={'\0'};
float prueba_100[100]={'\0'};
int16 escogefiltro={'\0'};

    void firlms (void);
    void autocorrelacion (void);
    void Mtoeplitz (void);
    void xcorrelacion (void);
    void cholesky  (void);
    void Finversa (void);
    void coeficientes (void);
    void firwiener (void);
```

```
////////////////////////////////////////////////////////////////////////
#include "UART_1.h"
#include "CyLib.h"

CY_ISR(InterruptHandler)
{
//////////////////////////////B U F F E R////////////////////////////////
 if (escogefiltro==0){
                      ib = UART_1_GetChar();
                      if (ib=='w')
                         {
                          escogefiltro=1;//Pin_4_Write(1) ;
                          }
                      else{if (ib=='l')
                               {
                                escogefiltro=2; //Pin_4_Write(1) ;
                                }
                           }
                      }

 else{if ((lleno_b==0) & (escogefiltro!=0))
{//*
         ib = UART_1_GetChar();
         if (ib=='b')
                {                    ////**reconoce_entrada**////
                   bandera_b=1;
                   }
         else if (ib=='a')
                     {               ////**reconoce_deseada**////
                          bandera_a=1;
                          }

 else if (bandera_b==1)
         {                   ////--leer_entrada--////
              if (incremento<4)
                 {
                    mibuffer[incremento] = ib;
              incremento++;
                   }
          else{
                    mibuffer[incremento] = ib;
                    prueba_buffer=atof(mibuffer);
                    if (contador_e <= 99)
                       {      ////++guardar_entrada++////
                          Sentrada[contador_e]=prueba_buffer;
                          contador_e++;
                          }
                    else{
                             lleno_b=1; //breakpoint
                             }
                       incremento=0;
                       bandera_b=0;
                       } //close else
        }
```

```
 else if (bandera_a==1)
         {                  ////--leer_deseada--////
              if (incremento<4)
                 {
                    mibuffer[incremento] = ib;
              incremento++;
                    }
          else{
                     mibuffer[incremento] = ib;
                     prueba_buffer=atof(mibuffer);
                     if (contador_d <= 99)
                        {      ////++guardar_deseada++////
                           Sdeseada[contador_d]=prueba_buffer;
                           contador_d++;
                           }
                     else{
                          lleno_a=1; //breakpoint
                          }
                     incremento=0;
                     bandera_a=0;
                     } //close else
        }
}//*

else if ((lleno_a==0) & (escogefiltro!=0))
{//*
 ib = UART_1_GetChar();
 if (ib=='b'){                      //****entrada****//
                bandera_b=1;
                }
 else if (ib=='a'){                 //****deseada****//
                     bandera_a=1;
                          }
 else if (bandera_b==1)
         {                  ////--leer_entrada--////
          if (incremento<4)
                 {
                    mibuffer[incremento] = ib;
              incremento++;
                    }
          else{
                   mibuffer[incremento] = ib;
                   prueba_buffer=atof(mibuffer);
                   if (contador_e <= 99)
                      {      ////++guardar_entrada++////
                         Sentrada[contador_e]=prueba_buffer;
                           contador_e++;
                           }
                     else{
                              lleno_b=1; //breakpoint
                              }
                     incremento=0;
                     bandera_b=0;
```

```
                    } //close else
          }

else if (bandera_a==1)
         {         //****deseada****//
              if (incremento<4)
                   {
                  mibuffer[incremento] = ib;
                  incremento++;
                  }
          else
                  {
                     mibuffer[incremento] = ib;
                     prueba_buffer=atof(mibuffer);
                     if (contador_d <= 99)
                        {
                           Sdeseada[contador_d]=prueba_buffer;
                           contador_d++;
                           }
                     else{
                            lleno_a=1; //breakpoint
                            }
                     incremento=0;
                     bandera_a=0;
                     } //close else
        }
}//*

else{
     runfiltro=1;
       }
}

////////////////////////////I M P R I M I R/////////////////////////////

//imprimir Sentrada
/*if ((lleno_b==1) & (contador_fuera<=99)){
      entrada_imp=Sentrada[contador_fuera];
      contador_fuera++;
      sprintf(CadenaPuerto,"%5.3f",entrada_imp);
      UART_1_PutString(CadenaPuerto);
*/
//imprimir Sdeseada
/*if ((lleno_a==1) & (contador_fuera<=99)){
      entrada_imp=Sdeseada[contador_fuera];
      contador_fuera++;
      sprintf(CadenaPuerto,"%5.3f",entrada_imp);
      UART_1_PutString(CadenaPuerto);
*/

//imprimir filtro
if ((run1==1) & (contador_fuera<=99)){
      entrada_imp=filtro[contador_fuera];
      contador_fuera++;
```

```
      sprintf(CadenaPuerto,"%5.3f",entrada_imp);
      UART_1_PutString(CadenaPuerto);
                                                                  }
//if(run1==1){Pin_4_Write(1);}

}//close interrupt

////////////////////////////////// M A I N//////////////////////////////
void main()
{
    UART_1_Start() ;
    UART_1_Enable() ;
    isr_1_Start();
    isr_1_SetVector(InterruptHandler);
    isr_1_Enable();
    CyGlobalIntEnable; //Uncomment this line to enable global interrupts

      for(;;)
    {
      if (runfiltro==1)
                      {Pin_2_Write(1) ;
                       if (escogefiltro==1)
                          {
                           autocorrelacion ();
                           Mtoeplitz ();
                           xcorrelacion ();
                           cholesky ();
                           Finversa ();
                           coeficientes ();
                           firwiener ();
                           }
                       else if (escogefiltro==2)
                               {
                                firlms ();
                                }
                       run1=1;
                       //    if (run1=1){Pin_4_Write(1) ;}
                       }
      else{
           Pin_2_Write(0);
           }
      }//close for
}//close main


////////////////////////////////W I E N N E R//////////////////////////////

///////////////////////////////////
/////  auto-correlacion entrada
///////////////////////////////////
void autocorrelacion ()
{
 int i,j;
 float cor;
```

```
 for(i=0;i<=n-1;i++){
                      cor=0;
                      for(j=0;j<=n-i;j++)
                      {
                      cor=cor+(Sentrada[j]*Sentrada[j+i]);
                       }
                      auto_cor[i+n-1]=cor;
                      auto_cor[n-i-1]=cor;
                      }
 return ;
 }

////////////////////////////////
//////// toeplitz
////////////////////////////////
void Mtoeplitz ()
{
 int i,j;
 for(i=0;i<=p-1;i++){
                      for(j=0;j<=p-1;j++)
                                        {
                                         toeplitz[j][i]=auto_cor[i+n-j-1];
                                         matriz_L[j][i]=toeplitz[j][i];
                                         }
                      }
 return;
 }

////////////////////////////////
/////// correlacion cruzada
////////////////////////////////
void xcorrelacion ()
{
 int i,j;
 float cor;
 for(i=0;i<=p-1;i++){
                      cor=0;
                      for(j=0;j<=n-i;j++)
                                        {
                                        cor=cor+(Sentrada[j+i]*Sdeseada[j]);
                                         }
                      x_cor[i]=cor;
                      }
 return;
 }

////////////////////////////////
///////  factorización cholesky
////////////////////////////////
void cholesky  ()
{
 int i,j, h;
 float sum;
 for(h=0;h<=p-1;h++){
```

```
                    for(i=0;i<=h-1;i++){
                                sum=0;
                                for(j=0;j<=i-1;j++)
                                 {
                                 sum=sum+matriz_L[i][j]*matriz_L[h][j];
                                  }
                     matriz_L[h][i]=(matriz_L[h][i]-sum)/matriz_L[i][i];
                                      matriz_U[i][h]=matriz_L[h][i];
                                      }
                    sum=0;
                    for (j=0; j<=h-1; j++)
                         {
                          sum=sum+((matriz_L[h][j])*matriz_L[h][j]);
                          matriz_L[h][h]=sqrt((matriz_L[h][h]-sum));
                          matriz_U[h][h]=matriz_L[h][h];
                          }
 return;
 }

////////////////////////////////
///////  cálculo inversa
////////////////////////////////

void Finversa ()
{
 int i,j, h;
 float sus_ad[p],sus_at[p];
 float sum;
 for (h=0;h<=p-1;h++){
                     for (j=0;j<=p-1;j++){
                                          if (h==j){
                                                    sus_ad[j]=1;
                                                    }
                                          else{
                                               sus_ad[j]=0;
                                               }
                                           }
 //sustitucion adelante
                     sus_ad[0]=sus_ad[0]/matriz_L[0][0];
                     for (i=1;i<=p-1;i++){

                                          sum=sus_ad[i];
                       for (j=0;j<=(i-1);j++){
                              sum=sum-(matriz_L[i][j]*sus_ad[j]);
                              sus_ad[i]=sum/matriz_L[i][i];
//sustitucion atrás
                             sus_at[p-1]=sus_ad[p-1]/matriz_U[p-1][p-1];
                          for (i=p-2;i>=0;i--){
                              sum=0;
                              for (j=i+1;j<=p-1;j++){
                                 sum=sum+matriz_U[i][j]*sus_at[j];
                                                    }
                                sus_at[i]=(sus_ad[i]-sum)/matriz_U[i][i];
```

```
                                        }
                        for (j=0;j<=p-1;j++){
                                          inversa[j][h]=sus_at[j];
                                          }
                      }
}

////////////////////////////////
///////  coeficientes filtro
////////////////////////////////
void coeficientes  ()
{
 int i,j;
 float sum,sum1;
 float co_1[5];
 for (i=0;i<=p-1;i++){
                      sum=0;
                      for (j=0;j<=p-1;j++){
                                           sum1=inversa[i][j]*x_cor[j];
                                           sum=sum+sum1;
                                           co_1[i]=sum1;
                                           }
                      co[i]=sum;
                      sum=0;
                      }
 }

////////////////////////////////
///////   salida filtro
////////////////////////////////
void  firwiener ()
{
 int i,j,h;
 float sum;
 float co1[n];
 for(i=0;i<=n-1;i++){
                     if (i<=p-1){
                                 co1[i]=co[i];
                                 }
                     else{
                          co1[i]=0;
                          }
                     }
 for (i=0;i<=n-1;i++){
                      sum=0;
                      for (j=0;j<=i;j++){
                                         h=i-j;
                                         sum=sum+(co1[j]*Sentrada[h]);
                                          }
                      filtro[i]=sum;
                      }
}
/* [] END OF FILE */
```

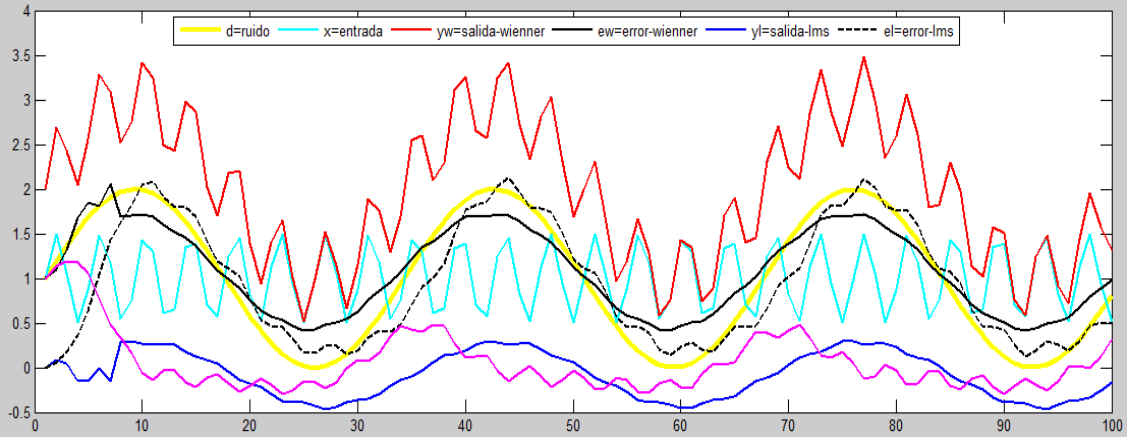
```
////////////////////////////////L  M  S/////////////////////////////////////

void  firlms ()
{
 //int16 mu=0.01;
 int i=0;
 int k=0;
 int j=0;
 int h=0;
 float w[8]={'\0'};
 float e[100]={'\0'};
 float x1[8]={'\0'};
 for (i=0;i<=n-1;i++)
                      {
                       if (i<=p-1)
                                  {
                                   k=i;
                                   for (j=0;j<=i;j++){
                                                      x1[j]=Sentrada[k];
                                                      k= k-1;
                                                       }
                                  }
                       else{
                            k=i;
                            for (h=0;h<=p-1;h++){
                                                 x1[h]=Sentrada[k];
                                                  k= k-1;
                                                  }
                            }
                       for (h=0;h<=p-1;h++){
                            filtro1[i] = filtro1[i] + w[h]*x1[h];
                                                 }
                       e[i] = Sdeseada[i] - filtro1[i];
                       for (h=0;h<=p-1;h++){
                                                 w[h]=w[h]+0.01*e[i]*x1[h];
                                                 }
                       }
 for (i=0;i<=n-1;i++)
                      {
                       filtro[i]=filtro1[i];
                       }
}
/* [] END OF FILE */
```

ANEXO C. OTRAS PRUEBAS Y RESULTADOS

1. PRUEBAS CON N=100 Y P=8

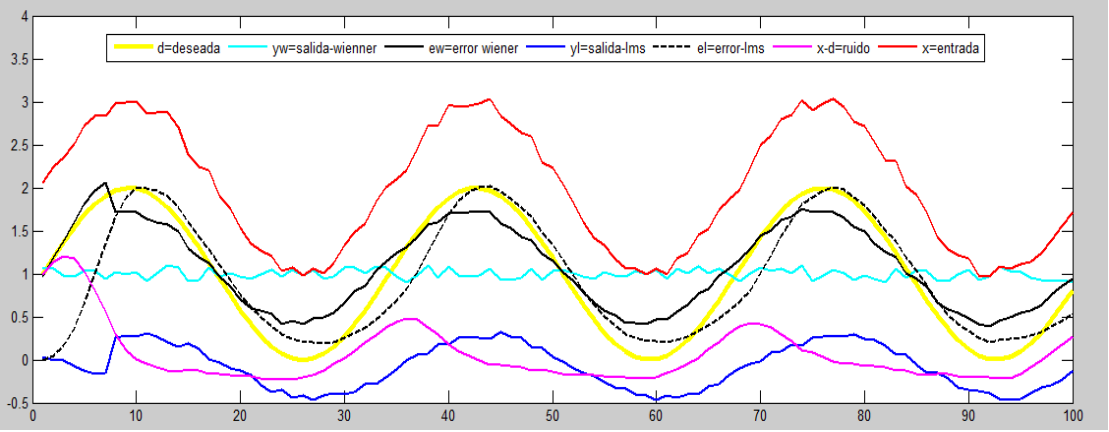
Se realizaron algunas pruebas tomando 100 muestras y un orden de filtro 8 para una señal de interferencia seno de 24 Hz y una señal de ruido blanco uniforme con amplitud de 0.1. Los resultados pueden observarse en la Figura 76 y Figura 77.

Figura 76. Filtrado adaptativo con interferencia seno de 24 Hz (N=100, P=8).



Fuente: Matlab®.

Figura 77. Filtrado adaptativo con ruido blanco uniforme (N=100, P=8).



Fuente: Matlab®.

En la Tabla 7 y la Tabla 8 es posible observar los errores de las diferentes salidas.

Tabla 7. Errores de identificación y estimación (Ruido: Señal Seno) (N=100, P=8).

	FILTRO WIENER	FILTRO LMS
Error de Identificación	-9.4816 [dB]	-7.4119 [dB]
Promedio del Error de Estimación	-0.0790	0.0576
% Error	23.49	23.43

Fuente: Autores®.

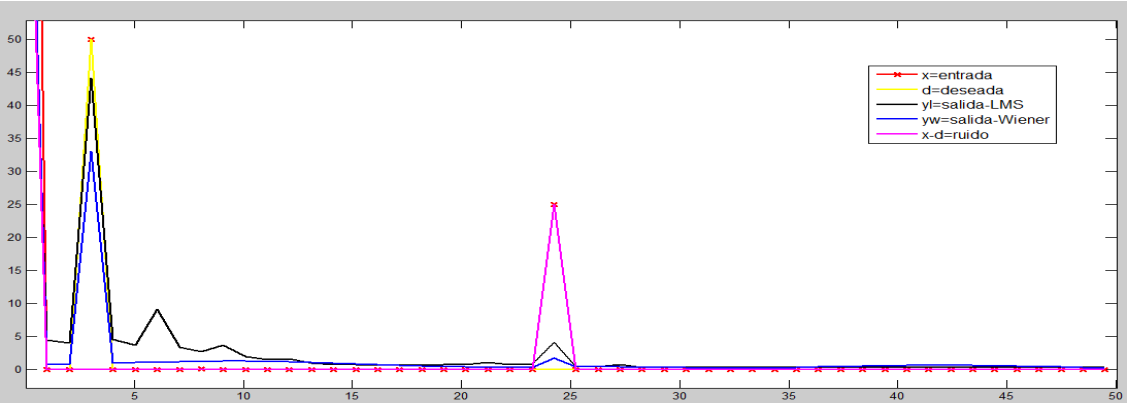
Tabla 8. Errores de identificación y estimación (Ruido: Blanco Uniforme) (N=100, P=8).

	FILTRO WIENER	FILTRO LMS
Error de Identificación	-8.6403 [dB]	-6.6227 [dB]
Promedio del Error de Estimación	-0.0782	0.0568
% Error	23.25	22.86

Fuente: Autores.

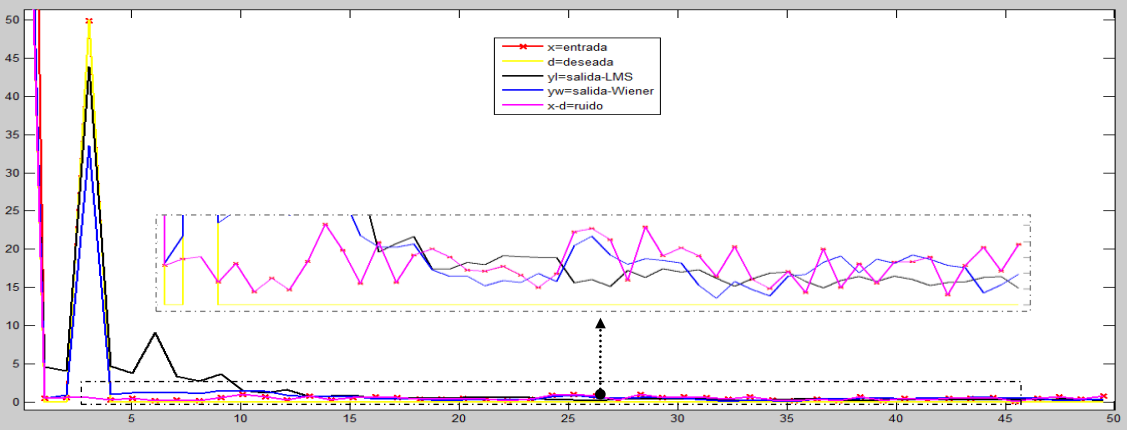
También se obtuvieron los resultados en frecuencia de las salidas de los filtros, los resultados de muestran en la Figura 78, Figura 79, Tabla 9 y Tabla 10.

Figura 78. Señales en frecuencia involucradas en filtrado con interferencia seno de 24 Hz (N=100, P=8).



Fuente: Matlab®.

Figura 79. Señales en frecuencia involucradas en filtrado con ruido blanco (N=100, P=8).



Fuente: Matlab®.

Tabla 9. Características en frecuencia de señales involucradas en filtrado con ruido seno (N=100, P=8).

	Deseada	Entrada	Salida LMS	Salida Wiener	RB
Valor del armónico en F = 3Hz	50	50	44.2	33.07	0
Porcentaje respecto a la señal deseada			88.4%	66.14%	
Valor del armónico en F = 24 Hz	0	25	4.116	1.644	25
Porcentaje respecto al ruido seno			16.464%	6.576%	
% THD			17.34	3.89	

Fuente: Autores.

Tabla 10. Características en frecuencia de señales involucradas en filtrado con ruido blanco (N=100, P=8).

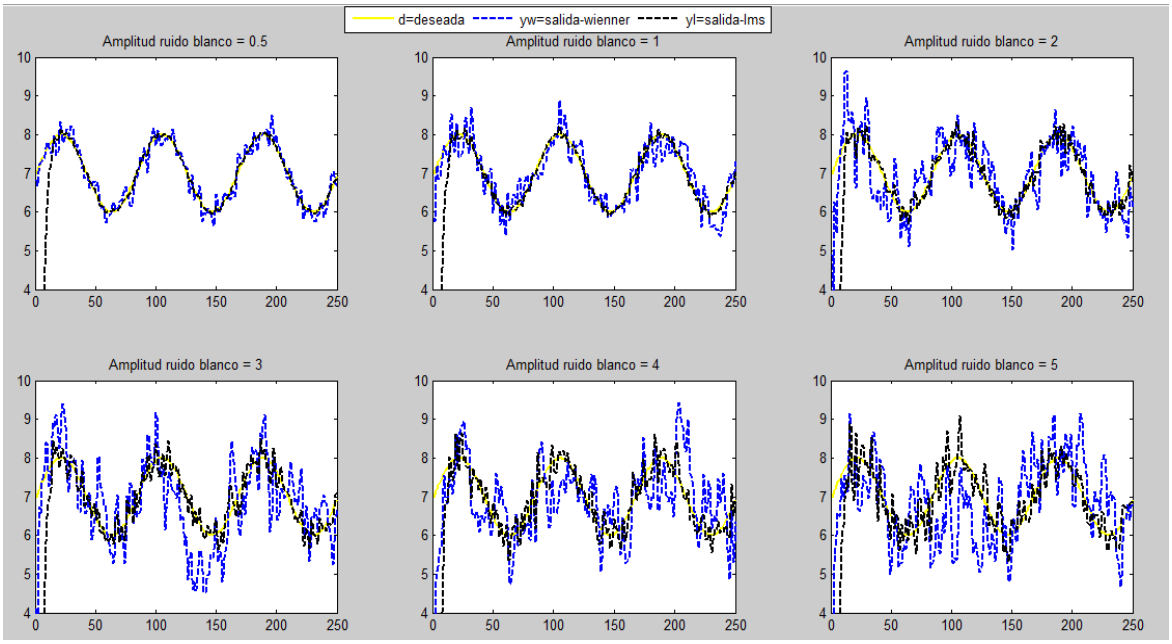
	Deseada	Entrada	Salida LMS	Salida Wiener
Valor del armónico en F=3 Hz	50	50	43.87	33.87
Porcentaje respecto a la señal deseada			87.74%	67.74%
% THD			14.14	4.97

Fuente: Autores.

2. PRUEBAS VARIANDO LA POTENCIA DE LA SEÑAL DE RUIDO BLANCO.

Para averiguar hasta que punto los filtros adaptativos lograban extraer la señal deseada, se realizaron una serie de pruebas aumentando la potencia de la señal de ruido blanco uniforme, los resultados pueden observarse en la Figura 80. Cabe notar que, hubo que modificar el valor de la velocidad de convergencia μ y su valor correspondió a 0.0006.

Figura 80. Salidas de filtro Wiener y LMS variando la amplitud del ruido blanco uniforme.



Fuente: Matlab®.

Para analizar las salidas obtenidas se halló la distorsión armónica total para cada caso, los resultados pueden observarse en la Tabla 11.

Tabla 11. Distorsión armónica total aumentando la potencia del ruido.

Amplitud de ruido blanco	%THD	
	Wiener	LMS
0.5	25.79	5.93
1	39.84	9.45
2	65.74	18.33
3	68.66	31.21
4	139.5	35.91
5	147.87	54.28

Fuente: Autores.

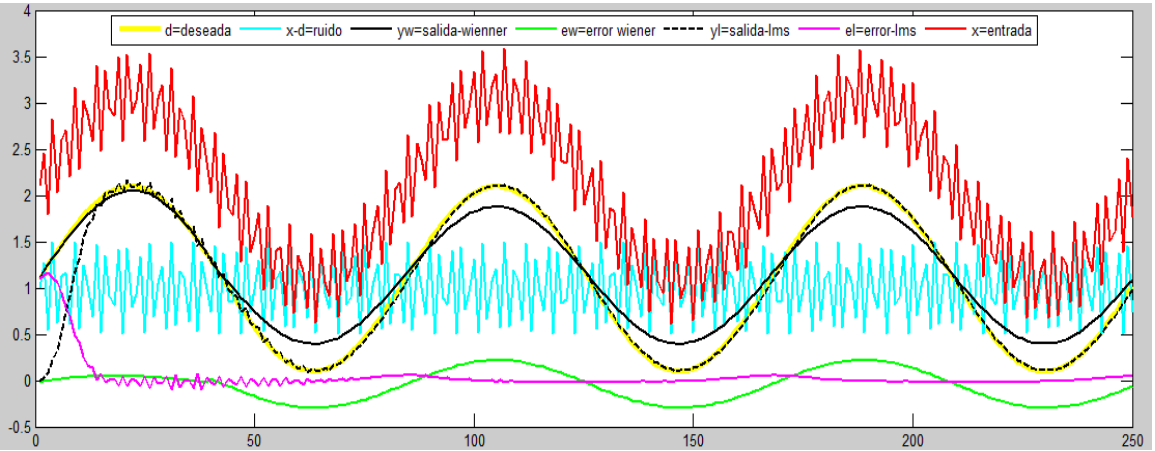
Por los resultados encontrados es notable que el filtrado LMS arrojó mejores resultados ante un ruido blanco uniforme, pues excede el 50% de distorsión cuando la amplitud del ruido es de 5, mientras que el filtrado Wiener lo excede cuando esta amplitud es 2.

3. PRUEBAS CON SEÑAL DE INTERFERENCIA DE ALTA FRECUENCIA.

- **Frecuencia de interferencia = 102 Hz**

Para estas pruebas inicialmente se aumentó la frecuencia de la señal de interferencia a 102 Hz. Las salidas de los filtros pueden observarse en la Figura 81.

Figura 81. Filtro Wiener y LMS (Frecuencia de interferencia=102Hz).



Fuente: Matlab®.

En la Tabla 12 se pueden apreciar el porcentaje de error y la distorsión armónica total de las salidas de los filtros.

Tabla 12. Error y THD de salidas (Frecuencia de interferencia=102Hz)

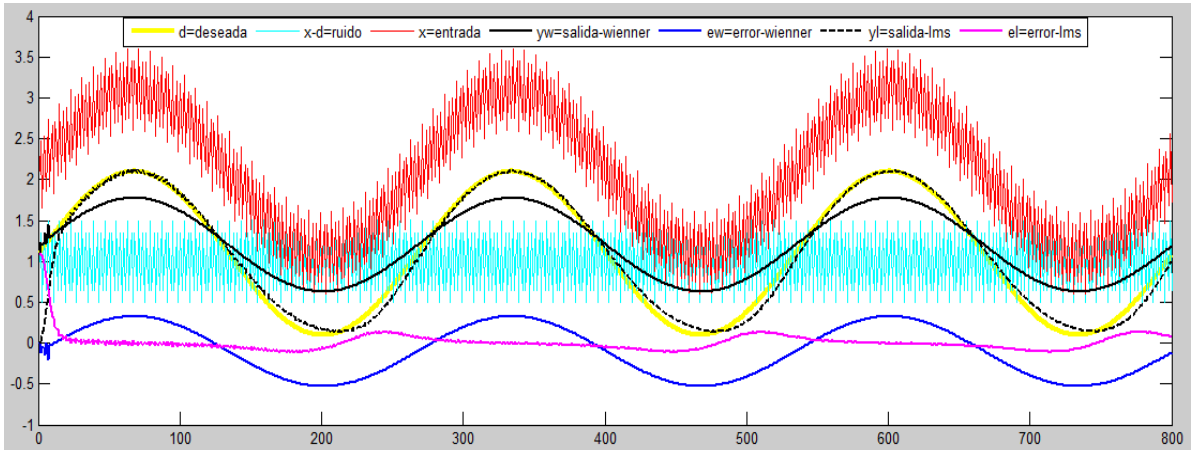
	FILTRO WIENER	FILTRO LMS
% Error	14.92	6.03
% THD	0.68	1.45

Fuente: Autores.

- **Frecuencia de interferencia = 300 Hz**

Para estas pruebas inicialmente se aumentó la frecuencia de la señal de interferencia a 300 Hz, para esto fue necesario aumentar también el número de muestras a 800 para cumplir el teorema de Nyquist. Las salidas de los filtros pueden observarse en la Figura 82.

Figura 82. Filtro Wiener y LMS (Frecuencia de interferencia=300Hz).



Fuente: Matlab®.

En la Tabla 13 se pueden apreciar el porcentaje de error y la distorsión armónica total de las salidas de los filtros.

Tabla 13. Error y THD (Frecuencia de interferencia=300Hz)

	FILTRO WIENER	FILTRO LMS
% Error	28	6.33
% THD	0.17	1.12

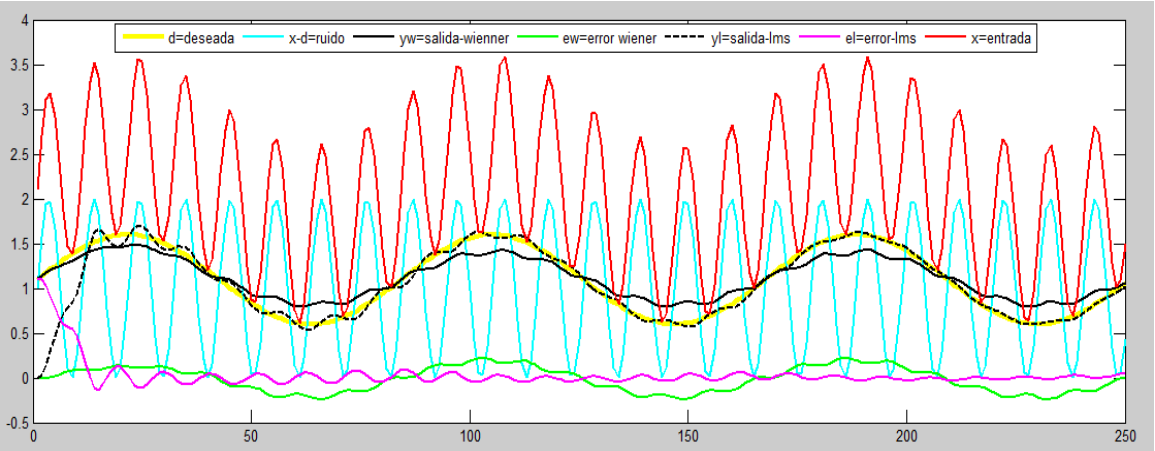
Fuente: Autores.

4. PRUEBAS CON AMPLITUD DE INTERFERENCIA SENO MAYOR QUE AMPLITUD DE SEÑAL DESEADA.

- **Amplitud de interferencia (1) mayor a amplitud de deseada (0.5)**

Ahora, para evaluar el desempeño de los filtros se invirtieron las amplitudes de la señal deseada y la señal de interferencia, es decir, la amplitud de la señal de interferencia mayor a la señal deseada. Las salidas de los filtros pueden observarse en la Figura 83.

Figura 83. Filtro Wiener y LMS (Amplitud interferencia=1, deseada=0.5).



Fuente: Matlab®.

En la Tabla 14 se pueden apreciar el porcentaje de error y la distorsión armónica total de las salidas de los filtros.

Tabla 14. Error y THD (Amplitud interferencia=1, deseada=0.5)

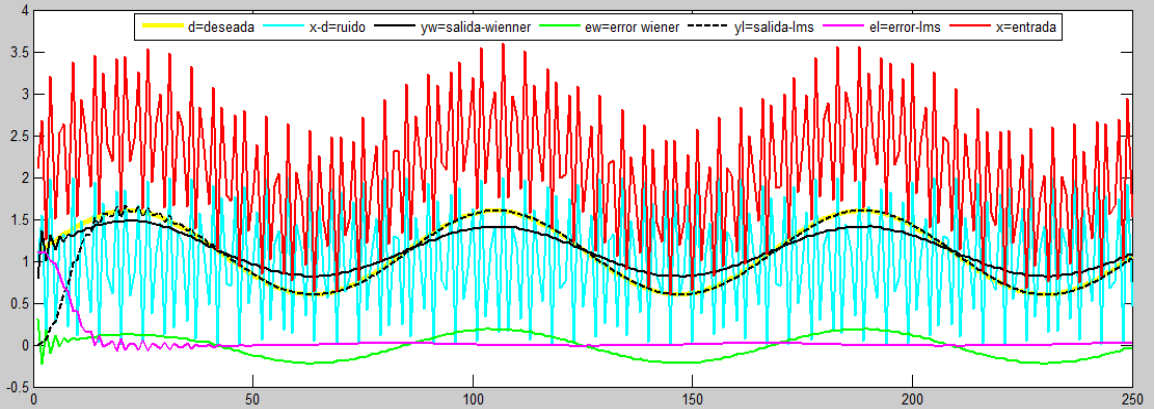
	FILTRO WIENER	FILTRO LMS
% Error	24.70	12.41
% THD	10.11	12.3

Fuente: Autores.

- **Amplitud de interferencia (1) mayor a amplitud de deseada (0.5) con frecuencia de interferencia de 102 Hz.**

Ahora, se conservó las mismas amplitudes, pero se aumentó la frecuencia de la señal de interferencia. Las salidas de los filtros pueden observarse en la Figura 84.

Figura 84. Filtro Wiener y LMS (Amplitud interferencia=1, deseada=0.5, Frecuencia Interferencia=120 Hz).



Fuente: Matlab®.

En la Tabla 15 se pueden apreciar el porcentaje de error y la distorsión armónica total de las salidas de los filtros.

Tabla 15. Error y THD (Amplitud interferencia=1, deseada=0.5, Frecuencia Interferencia=120 Hz)

	FILTRO WIENER	FILTRO LMS
% Error	25.73	9.55
% THD	0.57	0.82

Fuente: Autores.

5. RESUMEN DE PRUEBAS Y RESULTADOS

En la Tabla 16 es posible apreciar los resultados de todas las pruebas realizadas.

Tabla 16. Error y THD de todas las pruebas realizadas

N	P	Señal deseada		Señal interferencia			Filtro Wiener		Filtro LMS	
		Amp	Fre	Tipo	Amp	Fre	%Error	%THD	%Error	%THD
100	8	1	3Hz	Seno	0.5	24Hz	23.49	3.89	23.43	17.34
100	8	1	3Hz	RB	0.5	24Hz	23.25	4.97	22.86	14.14
250	40	1	3Hz	Seno	0.5	24Hz	14.95	4.23	6.73	5.58
250	40	1	3Hz	RB	0.5	24Hz	17	11.59	6.71	3.48
250	40	1	3Hz	Seno	0.5	102Hz	14.92	0.68	6.03	1.45
800	40	1	3Hz	Seno	0.5	300Hz	28	0.17	6.33	1.12
250	40	0.5	3Hz	Seno	1	24Hz	24.70	10.11	12.41	12.3
250	40	0.5	3Hz	Seno	1	102Hz	25.73	0.57	9.55	0.82
250	40	1	3Hz	RB	0.5	24Hz	16.71	25.79	4.03	5.93
250	40	1	3Hz	RB	1	24Hz	28.64	39.84	7.24	9.45
250	40	1	3Hz	RB	2	24Hz	43.92	65.74	13.89	18.33
250	40	1	3Hz	RB	3	24Hz	65.04	68.66	19.80	31.21
250	40	1	3Hz	RB	4	24Hz	68.52	139.5	23.66	35.91
250	40	1	3Hz	RB	5	24Hz	82.96	147.87	29.34	54.28

Fuente: Autores.