Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748

José Estuardo García Barrios
Asesorado por el Ing. Sergio Iván Galindo Morán

Guatemala, noviembre de 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ ESTUARDO GARCÍA BARRIOS
ASESORADO POR EL ING. SERGIO IVÁN GALINDO MORÁN

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, NOVIEMBRE DE 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de León Bran
VOCAL V	Br. Kevin Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Dr. Juan Carlos Córdova Zeceña
EXAMINADOR	Ing. Gustavo Benigno Orozco Godínez
EXAMINADOR	Ing. Edgardo Loukota Castellanos
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 13 de febrero de 2018.

José Estuardo García Barrios

Guatemala, 17 de febrero de 2020

Ing. Armando Rivera
Director Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Respetable Ingeniero Rivera

Por medio de la presente informo a usted que, como asesor del Trabajo de Graduación del estudiante universitario José Estuardo García Barrios, quien se identifica con carné universitario No. 1995-20714, procedí a hacer la revisión de la tesis que contiene cuatro capítulos, cuyo título es: "**IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748.",** el cual encuentro como satisfactorio.

En tal virtud, LA DOY POR APROBADA, solicitándole dar el trámite correspondiente.

Atentamente

Ing. Sergio Iván Galindo Morán
Colegiado 5921

Sergio Iván Galindo Morán
Ingeniero Electrónico
Colegiado 5921

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 4 de marzo de 2020

Señor Director
Armando Alonso Rivera Carrillo
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **IDENTIFICACIÓN DE SISTEMAS DE AUDO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748,** desarrollado por el estudiante **José Estuardo García Barrios**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

REF. EIME 16. 2020.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación del estudiante: JOSÉ ESTUARDO GARCÍA BARRIOS titulado; IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748, procede a la autorización del mismo.

Ing. Armando Alonso Rivera Carrillo

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 9 DE MARZO 2020.

Decanato
Facultad de Ingeniería
24189101- 24189102

DTG. 360.2020.

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Eléctrica, al Trabajo de Graduación titulado: **IDENTIFICACIÓN DE SISTEMAS DE AUDIO EN UN AMBIENTE REVERBERANTE Y SU IMPLEMENTACIÓN EN EL PROCESADOR DIGITAL DE SEÑALES TMS320C6748,** presentado por el estudiante universitario: **José Estuardo García Barrios,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Anabela Cordova Estrada
Decana

Guatemala, noviembre de 2020

AACE/asga

ACTO QUE DEDICO A:

Dios	Gracias por absolutamente todo.
Mi madre	Eufemia Magnolia Barrios de León, gracias por ser mi ángel de la guarda terrenal.
Mi padre	Augusto Albino García Gómez, (q. e. p. d.), gracias por su dedicación y ejemplo, siempre lo llevo en mí.
Mi esposa	Ana Portales, gracias por aparecer en mi camino, me enseñaste a vivir.
Mis hijas	Sarah y Abi García Portales, ustedes son la luz de mi vida. Nunca imaginé que se podía sentir un amor así de grande.
Mis hermanas	Mari y Mirza García, gracias por quererme aún después de que no les hice su infancia fácil. Gracias Mari por tu amistad. Gracias Michi por ser esa persona tan determinada en la que siempre puedo confiar.

Mis amigos	Si menciono a uno dejo fuera a muchos, pero si te tomaste la molestia de leer esta página y te sentiste aludido en esta línea, seguro que sos cercano y gracias por cada momento compartido.
Guatemala	Gracias por cada día templado y por la gente que te habita, siempre que estoy fuera te extraño.
Planeta tierra	Gracias por ser nuestro hogar, mi sueño es que como humanidad aprendamos a cuidarte.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ir y enseñar a todos, incluyéndome.
Facultad de Ingeniería	Por darme una base cuantitativa sólida.
Mis amigos de la Facultad	Por el compañerismo, amistad, entusiasmo y por el apoyo demostrado, en especial Sergio Galindo y Ariel Chitay.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
MHz	Velocidad del reloj en mega Hertz

GLOSARIO

ALU	Unidad lógica aritmética (arithmetic logic unit).
CPU	Unidad de procesamiento central (Central processing unit).
DMA	Acceso directo a memoria (Direct Memory Access,). Forma de direccionar memoria que permite su acceso de manera más inmediata.
DRAM	Memoria de acceso aleatorio y dinámico (Dynamic Random Access Memory). Memoria de rápido acceso rápido ya que no interviene un registro intermedio para su recuperación.
DSP	Procesador digital de señales (Digital Sigal Processor), tipo especializado de controlador programable que permite el manejo especializado de señales digitales.
EMIF	Interfaz externa de memoria (External Memory Interface). Interfaz que permite expandir la memoria de un procesador.
EPROM	erasable programmable read-only memory.

FIR	Filtros de respuesta finita (Finite Impulse Response).
Hardware	Parte física de una computadora.
HPI	Interfaz de puerto controlador (Host Port Interface).
IIR	Filtro de respuesta infinita (Infinite Impulse Response).
LCDK	equipo de desarrollo de bajo costo o (low cost development kit).
LMS	Mínimos cuadrados (Least Minimum Squares). Algoritmo matemático que permite encontrar el punto mínimo de error, definiéndose el error como la diferencia entre una señal de salida y una función aplicada a la señal de entrada. El objetivo es encontrar los parámetros de la señal que transforma la señal de entrada.
McBSP	Puerto Serial Multi-canal con búfer (Multichannel Buffered Serial Port).
MFLOPS	Millones de operaciones de punto flotante por segundo (millinos of floating-point operations per second).

MIPS	Millones de instrucciones por segundo (Millions of instruccions per second).
MMACS	Millones de operaciones de multiplicación y acumulación por segundo (Millions of multiply and accumulate per second).
MOP	Millones de operaciones por segundo (Millions of operations per second).
MSE	Error cuadráticos medios (Mínimum squared error).
Pipeline	Técnica para la implementación de procesamiento en paralelos dentro de un procesador.
SBSRAM	Memoria de acceso aleatorio y sincronía de ráfaga estática (Synchronous Burst Static Random Access Memory).
SDRAM	Memoria de acceso aleatorio y sincronía dinámica, (Synchronous Dynamic Random Access Memory).
Software	El conjunto de comandos e instrucciones dados a una computadora para que ésta funcione.
SQNR	La razón de señal a ruido de cuantificación (Signal to quantification noise ratio).

SRAM	Memoria de acceso directo y estática (Static Random Access Memory).
VLIW	Palabra de instrucción muy larga (Very long instruction word).

RESUMEN

El presente trabajo de graduación explora la aplicación del procesamiento digital a una aplicación específica en un chip especializado que es la identificación de un sistema tomando en cuenta las consideraciones de diseño para lograr un procesamiento en tiempo real.

El procesamiento digital de señales (DSP, por sus siglas en inglés), trata sobre la representación de señales en forma digital, el procesamiento de estas señales, así como de la información que contienen.

El DSP ofrece significativas ventajas sobre el procesamiento analógico; por ejemplo: mayor confiabilidad, mayor precisión, mejor estabilidad y facilidad de almacenamiento.

El tema se desarrolla en tres partes, en el primer capítulo se hace un repaso de las ideas fundamentales del procesamiento digital, en el segundo capítulo se detalla el procesador digital utilizado en este trabajo y por último, en el capítulo tres, se presenta la aplicación específica.

OBJETIVOS

General

Explicar las consideraciones de diseño para la implementación de algoritmo de procesamiento digital.

Específicos

1. Exponer las bases teóricas de filtros adaptativos y su aplicación para la implantación de identificación de sistemas.

2. Formular la implementación de un algoritmo de procesamiento digital en un procesador digital específico, el procesador de Texas Instruments TMS320C6748.

INTRODUCCIÓN

El procesamiento digital de señales (DSP, por sus siglas en inglés), trata sobre la representación de señales en forma digital, el procesamiento de estas señales, así como de la información que contienen.

Durante las últimas décadas las aplicaciones del procesamiento digital han incrementado sustancialmente debido a la reducción del costo y aumento del poder computacional de los procesadores digitales. Asimismo, este aumento también se debe a que el DSP ofrece significativas ventajas sobre el procesamiento analógico; por ejemplo: mayor confiabilidad, mayor precisión, mejor estabilidad y facilidad de almacenamiento.

Para hacer uso de los sistemas en tiempo discreto con señales analógicas, como suele ser el caso en muchas aplicaciones, las señales analógicas de entrada se convierten en digitales por medio de convertidores analógico-digitales. Después, el sistema en tiempo discreto recibe muestras de la señal de entrada con cierta frecuencia, denominada frecuencia de muestreo. La magnitud de la señal al ser digitalizada sólo puede tomar un número finito de valores. El sistema de procesamiento en tiempo discreto es ahora, entonces, un sistema de procesamiento digital de señales. Una vez completado el procesamiento, la señal digital y de tiempo discreto de salida, es convertida nuevamente a una señal analógica por medio de convertidores digital-analógicos.

Texas Instruments, compañía líder en el sector de semiconductores, fabrica chips DSP que permiten compilar programas escritos en el lenguaje C. Dicha empresa incluso desarrolla la arquitectura de sus chips DSP tomando en cuenta

el compilador C que proveen con el mismo. Este enfoque permite que el código generado desde C sea bastante óptimo y que sólo sea necesario recurrir al lenguaje ensamblador cuando los requerimientos de eficiencia sean muy exigentes. Otra ventaja de utilizar un lenguaje de alto nivel como C, es que una aplicación se puede transportar a otro procesador de manera más fácil que si fuese hecha con lenguaje ensamblador.

La elección de utilizar un procesador fabricado por Texas Instruments (TMS320C6748), para el presente trabajo no es casual. Conforme datos de la empresa Forward Concepts, que se dedica realizar estudios de mercado de componentes y dispositivos electrónicos, el mercado global de procesadores digitales fue USD 7,8 millardos durante 2007, en cual Texas Instruments participó con un 65 %.

El presente trabajo, propuesto con el tema “Identificación de sistemas de audio en un ambiente reverberante y su implementación en el procesador digital de señales TMS320C6748”, tiene como objetivo exponer las consideraciones para pasar de la teoría a la práctica en este tema de la electrónica.

1. SISTEMAS DE PROCESAMIENTO DIGITAL DE SEÑALES

1.1. Introducción

Una señal es cualquier variable que se pueda medir, ésta puede ser temperatura, precios de cierre de una acción un mercado de valores, frecuencia cardíaca de un paciente, amplitud de una onda de sonido, entre otros.

Las señales pueden ser clasificadas en tres categorías: señales en tiempo continuo, es decir señales analógicas, señales en tiempo discreto y señales digitales. Las señales con las que se convive día a día son señales analógicas, estas señales están definidas continuamente en el tiempo y tienen una resolución infinita en su amplitud, para procesar estas señales se necesitan circuitos electrónicos que contienen tanto elementos pasivos como activos.

Las señales en tiempo discreto existen sólo en determinados instantes entonces pueden ser representadas como una secuencia de números que tienen un rango continuo de valores.

Las señales digitales son discretas tanto en tiempo como en amplitud, por lo tanto, pueden ser procesadas por computadoras.

El procesamiento digital de señales (DSP, por sus siglas en inglés), se refiere al estudio de la representación de señales digitales y el uso de sistemas digitales para analizar, modificar, almacenar, transmitir o extraer información de estas señales. Las tecnologías DSP se utilizan no solamente en áreas donde en

el pasado se utilizaban técnicas analógicas si no también donde las técnicas analógicas son muy difíciles o imposibles de aplicar.

Dentro de las muchas ventajas del uso de procesadores digitales de señales en comparación con sus contrapartes analógicas se cuenta con las siguientes:

- Flexibilidad: las operaciones que un sistema DSP aplica pueden ser fácilmente modificadas y actualizadas con software que implementa algoritmos específicos, no hace falta modificar un circuito o intercambiar componentes electrónicos, lo cual hace que un sistema DSP pueda ser actualizado aun cuando éste ya está en uso mediante la instalación de un programa en la memoria del dispositivo, esta actualización puede tener el objetivo de cubrir nuevos requerimientos, agregar nuevas funcionalidades o mejorar el rendimiento del sistema.

- Reproducción: la funcionalidad de un sistema DSP puede ser reproducida fielmente de un dispositivo a otro. Asimismo, mediante el uso de técnicas DSP, el sistema digital puede grabar, transferir o reproducir una señal muchas veces sin degradar la calidad. Por otro lado, los circuitos analógicos no tendrán exactamente las mismas características aun cuando su construcción tiene la misma especificación debido a que sus componentes analógicos tienen ciertas tolerancias.

- Confiabilidad: la memoria y los programas almacenados en un dispositivo DSP no se deterioran con el uso o el tiempo, por lo tanto, el rendimiento de un sistema DSP no se dañará con el cambio de condiciones ambientales o el desgaste de componentes electrónicos como sucede con los sistemas analógicos.

- Complejidad: la tecnología DSP permite la implementación de sistemas complejos, por ejemplo, el sistema de reconocimiento de voz que utilizan dispositivos portátiles. Además, hay ciertos algoritmos, como compresión y reconocimiento de imágenes, compresión de señales de audio, así como almacenamiento y transmisión de datos, que sólo puede ser realizados utilizando sistemas DSP.

Debido a la rápida evolución de la tecnología de semiconductores, explicada por la premisa propuesta por Gordon Moore que establece que el número de transistores en un microchip se duplica cada dos años, los dispositivos DSP tiene un costo más bajo con respecto a sistemas analógicos para la mayor parte de aplicaciones. Los algoritmos pueden ser desarrollados en lenguajes de alto nivel y hace que los sistemas de DSP sean relativamente fáciles de diseñar, desarrollar, analizar, simular, probar y mantener.

Entre las desventajas asociadas a sistemas DSP está el hecho de que el ancho de banda de un sistema DSP está supeditado a la frecuencia de muestreo. Asimismo, la mayoría de los algoritmos DSP son implementados con restricciones en cuanto a la resolución de la señal ya que se usa un número de bits determinado, y puede introducir errores de precisión aritmética derivados de la conversión analógica digital.

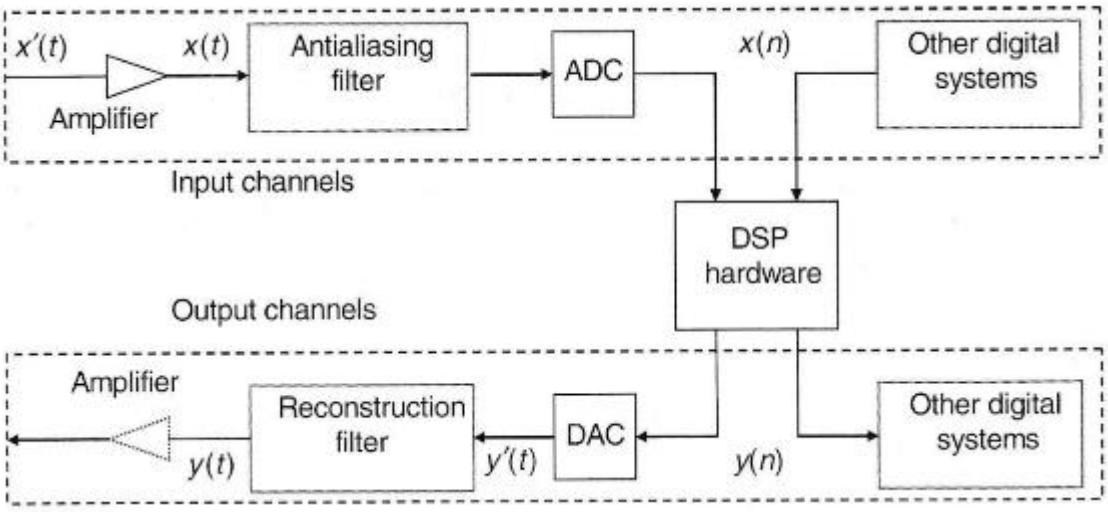
1.2. Elementos básicos de un sistema de procesamiento digital en tiempo real

Las aplicaciones para el procesamiento digital de señales pueden darse o no en tiempo real. Cuando el procesamiento no se realiza en tiempo real, la manipulación de la señal se realiza una vez ésta se ha almacenado. Por otro lado, el procesamiento en tiempo real, en cambio, impone una demanda alta en

términos de hardware y software ya que se deben completar las tareas de manipulación de la señal de entrada en un lapso específico.

Un diagrama de bloques de un procesador digital se muestra en la siguiente figura:

Figura 1. **Convertir analógico digital**



Fuente: SEN, Kuo. *Real-Time digital signal processing*. p. 2.

Conforme lo indica el diagrama, una señal analógica es convertida en señal digital, luego es transformada por el procesador digital y, por último, convertida de nuevo en una señal analógica. Para algunas aplicaciones, la señal puede ya estar en formato digital, por ejemplo, cuando la información es almacenada para su uso posterior. Por otro lado, en otras aplicaciones el requerimiento puede ser que la señal se genere directamente sólo en formato digital como es el caso de los generadores de señales.

1.2.1. Interfaz analógica

El propósito de un convertidor analógico digital (ADC, por sus siglas en inglés), es convertir una señal analógica en una señal digital para que sea procesada por el hardware digital. La señal de entrada es obtenida por un sensor electrónico apropiado, el cual puede medir presión, temperatura, sonido, entre otros. y convierte dicha señal en una señal eléctrica, esta señal es procesada por el hardware DSP y el resultado es una señal $y(n)$ que aún está en formato digital. En muchas aplicaciónes de procesamiento digital, esta señal debe ser convertida de nuevo en una señal analógica $y(t)$ mediante un conversor digital analógico (DAC, por sus siglas en inglés).

Una aplicación de este sistema es un reproductor digital de audio, en este la música es grabada en formato digital y luego el reproductor lee la señal digital codificada a partir de la memoria del dispositivo y luego reconstruye la señal analógica para que ésta pueda ser reproducida. Un sistema en tiempo real hará la conversión AD de forma continua y procesada por el *hardware* DSP a la misma velocidad. Para mantener el procesamiento en tiempo real, el hardware DSP debe realizar todas las operaciones requeridas dentro de un lapso fijo, de manera que pueda procesar y presentar la señal digital al DAC antes de que la siguiente señal llegue al ADC.

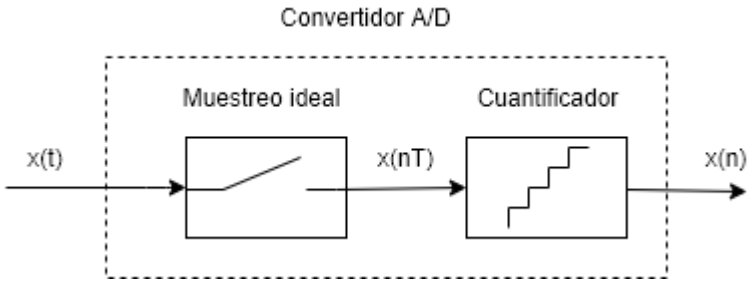
1.2.2. Muestreo

Se percibe el mundo de manera continua, por medio de los sentidos en tiempo analógico. Sin embargo, para poder procesar señales, perceptibles o no para nuestros sentidos, en un procesador digital se necesita tomar muestras y digitalizar la variable en estudio en dos dimensiones. La primera es en cuanto a tiempo, esto se logra por medio de una muestra periódica en intervalos T_s, el

inverso multiplicativo de este intervalo es la frecuencia de muestreo que se denominará $F_s = \frac{1}{T_s}$. Por ejemplo, la señal de sonido que proviene de una grabación estándar en disco compacto o archivo mp3 es de 41 000 muestras por segundo, la transición entre muestra y muestra es imperceptible para el oído humano. Este caso se dice que, usando la notación previamente definida, que $F_s = 41KHz$ y $T_s = 24{,}39\mu s$.

El componente ADC convierte la señal analógica $x(t)$ en una señal digital $x(n)$. La conversión analógica digital, referidad como digitalización, consiste en muestro (digitalización en tiempo) y cuantificación (digitalización en amplitud), este proceso se ilustra en la siguiente figura:

Figura 2. **Convertidor analógico digital**



Fuente: elaboración propia, empleando Draw.io 2000.

Los pasos para digitalizar una señal son los siguientes:

- La señal $x(t)$ es muestreada a intervalos uniformes de duración T, este proceso de muestreo convierte la señal analógica en una señal en tiempo discreto $x(nT)$ con amplitud continua donde n es un número entero.

- La amplitud de cada muestra $x(nT)$ es cuantizada en uno de los niveles 2^B, donde B es el número de bits utilizados para representa cada muestra. Cada nivel de amplitud es codificado en un número binario $x(n)$ con B bits.

Estos dos procesos son separados porque introducen dos tipos de distorsiones, por un lado, el muestreo puede introducir solapamiento, mientras que la digitalización de la señal puede introducir ruido de cuantificación. El teorema de Shannon establece que para poder representar una señal analógica $x(t)$ de forma exacta con una señal en tiempo discreto $x(nT)$, la frecuencia de muestreo f_s debe ser al menos el doble del componente de frecuencia máximo f_M de la señal analógica. Es decir,

$$f_s \geq 2f_M$$

f_M también es denominada el ancho de banda de la señal $x(t)$. La frecuencia mínima f_s es denominada frecuencia de Nyquist.

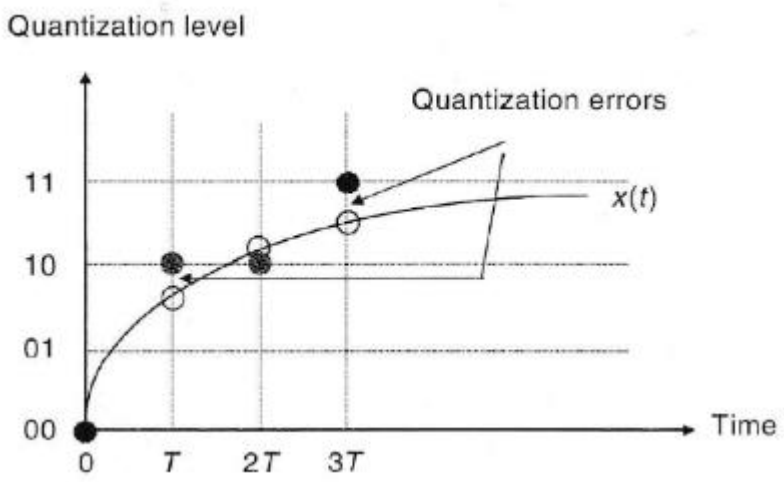
1.2.3. Cuantificación y codificación

La otra dimensión en la cual se digitaliza la señal para su procesamiento es en cuanto a amplitud, porque un procesador digital representa cualquier señal, internamente un número binario. Cuando esta señal se digitaliza se tendrá que hacer una elección de cuantos bits se utilizarán para representarla, se utilizan 2 bits, los posibles valores serían: 00 01 10 u 11, sólo cuatro niveles diferentes. En general si una señal se representa con N bits, los diferentes niveles de digitalización son 2^N. Con cada bit que se agrega, la señal tendrá más resolución.

Si $x(nT)$ se encuentra dentro de dos niveles de cuantificación, la señal será truncada o aproximada para producir x(n). El método más utilizado, por lo

general, es el de aproximación, de manera que el proceso de cuantificación representa una señal con valor continuo $x(nT)$ con su valor más cercano a la señal digital $x(n)$. Como se muestra en la siguiente figura.

Figura 3. **Error de cuantificación**



Fuente: Fuente: SEN, Kuo. *Real-Time digital signal processing*. p. 7.

El error de cuantificación es la diferencia entre el número cuantificado y el valor original, esta diferencia aparece como ruido en la salida del convertidor. Este error se denomina error de cuantificación y puede ser caracterizado por la razón de señal a ruido de cuantificación (SQNR, por sus siglas en inglés):

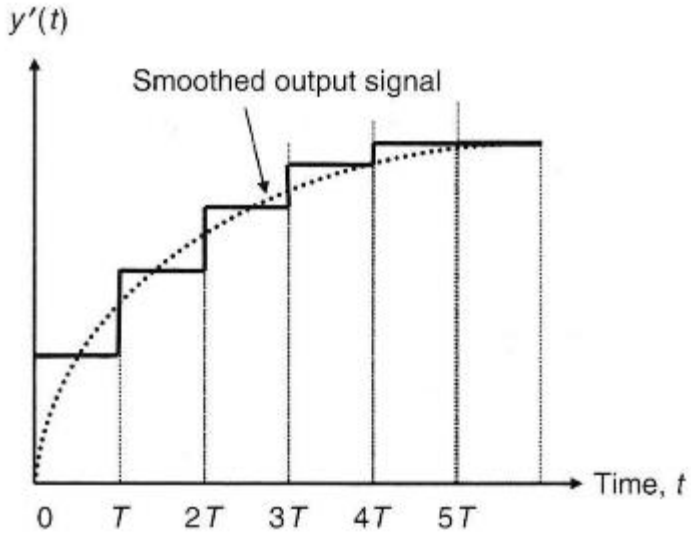
$$SQNR \approx 6BdB$$

Donde B es el número de bits. Consecuentemente, mientras más bits se utilicen, se tendrán más niveles de cuantificación y la razón de señal a ruido de cuantificación (SQNR), será más alta.

1.2.4. Filtro de suavizado

Los convertidores digital-analógico producen una onda con forma de escalera como se muestra en la gráfica de línea solida siguiente.

Figura 4. **Filtro de suavizado**



Fuente: SEN, Kuo. *Real-Time digital signal processing.* p. 9.

La entrada es convertida a un número binario correspondiente y luego mantenido por T segundos. Esta señal de salida de escalera contiene muchos componentes de alta frecuencia debido al cambio abrupto en el nivel de la señal. Un filtro de suavizado se utiliza para suavizar la señal con forma de escalera producida por el circuito DAC. Este filtro tiene el efecto de redondear las esquinas de la señal de escalera.

1.2.5. Convertidores de datos

Existen dos métodos para conectar un ADC/ADC para un procesador digital: serial y paralelo. Un convertidor paralelo transmite todos los bits a la vez, mientras que un convertir serial recibe o transmite los bits uno a la vez. Los convertidores paralelos se conectan a los buses de datos del procesador digital. Los convertidores serial se conectan directamente a los puertos seriales del procesador digital. Debido a que los convertidores serial requieren menos puntos de conexión, muchos sistemas DSP utilizan el sistema serial de convertidores ADC/ADC.

Muchas aplicaciones utilizan un ADC/ADC embebido en mismo chip de interfaz analógica (AIC, por sus siglas en inglés). También denominado códec (del inglés *coder/decoder*), que integra en un sólo circuito integrado un filtro anti solapamiento, un ADC, un DAC y un filtro de reconstrucción.

En este trabajo, se utilizará el equipo de Texas Instruments OMAP-l38 DSP+ARM9 que contiene un chip TLV320AIC3106 que es una interfaz analógica o códec. Las aplicaciones típicas de un códec incluyen sistemas de procesamiento de voz, de audio y controladores industriales. El chip TLV320AIC3106 está diseñado para aplicaciones de bajo consumo de potencia como teléfonos inteligentes y cámaras digitales, además soporta datos de 16, 20, 24 y 32 bits con frecuencias de muestreo de 8kHz a 96 kHz.

1.3. Hardware DSP

La gran mayoría de sistemas DSP requieren la realización de operaciones aritméticas intensivas como repetidas multiplicaciones y sumas. Estas operaciones pueden ser implementadas utilizando hardware digital como

microprocesadores, microcontroladores, procesadores digitales de señales o circuitos integrados de propósito específico. La selección del hardware apropiado puede determinarse para una aplicación, basándose en los criterios de rendimiento, costo o consumo de potencia.

1.3.1. Opciones de hardware para sistemas DSP

El procesamiento digital de una señal $x(n)$ es realizado utilizando hardware DSP, aunque es posible implementar un algoritmo DSP en diferente hardware digital, la aplicación específica determina la plataforma de hardware adecuada. Las opciones de hardware para aplicaciones DSP son las siguientes:

- Circuitos integrados para aplicación específica (ASIC, por sus siglas en inglés).
- Compuertas lógicas programables en campo (FPGAs, por sus siglas en inglés).
- Microprocesadores (μP) y microcontroladores (μC) de propósito general.
- Procesadores digitales de señales de propósito general.
- Procesadores digitales de señales con aceleradores de hardware de aplicación específica.

La siguiente tabla muestra un resumen de las características comparativas de estas opciones:

Tabla I. **Resumen de implementaciones de hardware de procesador digital de señales**

	ASIC	**FPGA**	$\mu P, \mu C$	**DSP**	**DSP con aceleración**
Flexibilidad	no	limito	alto	alto	medio
Tiempo de diseño	largo	medio	corto	corto	corto
Consumo de potencia	bajo	bajo-medio	medio-alto	bajo-medio	bajo-medio
Rendimiento	alto	alto	bajo-medio	medio-alto	alto
Costo de desarrollo	alto	medio	bajo	bajo	bajo
Costo de producción	bajo	bajo-medio	medio-alto	bajo-medio	medio

Fuente: SEN, Kuo. *Real-Time digital signal processing*. p. 11.

Un procesador digital de señales es básicamente un microprocesador con una arquitectura y un conjunto de instrucciones diseñadas específicamente para aplicaciones de DSP. En comparación con soluciones basadas en ASIC y FPGA, el procesador digital de señales tiene la ventaja de fácil desarrollo y programación en el campo lo que facilita la actualización de funcionalidades o solución de problemas. En comparación con hardware ASIC y FPGA son más económicos y en relación con los microprocesadores generales, los procesadores digitales de señales tienen mejor velocidad y manejo eficiente de la energía y menor costo en muchas aplicaciones de DSP.

Los procesadores digitales de señales se han convertido en la pieza fundamental de muchas aplicaciones como control de motores, sistemas automotrices, electrodomésticos, electrónicos de consumo, dispositivos para uso médico, y un rango vasto de aplicaciones en sistemas y equipos de comunicación y transmisión. Los procesadores digitales son programados con herramientas

para compilar, ensamblar, optimizar, enlazar, depurar, simular y emular. En este trabajo se utilizará el chip OMAP-l38 DSP+ARM9 que contiene dos núcleos: un microprocesador de uso general basado en la arquitectura ARM y un DSP basado en la arquitectura C674x.

1.3.2. Procesadores digitales de señales

Intel introdujo el procesador digital 2920, un procesador de enteros de 25 bits y un ciclo de instrucción de 400 ns, en 1979 para aplicaciones de DSP. En 1982, Texas Instruments introdujo el chip TMS32010, un procesador de punto fijo de 16 bits. Estos desarrollos fueron seguidos por productos más rápidos y también de punto flotante.

El hardware de un procesador digital incluye multiplicadores, ejecución de una instrucción por ciclo de reloj y el uso de instrucciones complejas que realizan múltiples operaciones como multiplicar, acumular y actualizar dirección de punteros de memoria.

Los chips DSP tiene las siguientes características que son comunes a algoritmos DSP:

- Unidades rápidas de multiplicar acumular: la operación multiplicar sumar o multiplicar acumular (MAC, por sus siglas en inglés), son necesarias en la mayoría de las funciones de DSP incluyendo el filtrado, transformada de Fourier y la función de correlación. Para poder efectuar una operación MAC eficientemente, los procesadores digitales de señales integran el multiplicador y el acumulador en la misma ruta de datos y completan una operación MAC en un solo ciclo de instrucción.

- Accesos múltiples a la memoria: la mayoría de DSP adoptan arquitecturas Harvard modificadas para mantener la memoria de instrucción y de datos separada para permitir la obtención simultánea de instrucciones y datos.

- Modos especiales de direccionamiento: los DSP frecuentemente incorporan unidades de generación de dirección para generar direcciones de datos en paralelo con la ejecución de instrucciones. Estas unidades usualmente soportan direccionamiento circular y direccionamiento con reversión de bits, los cuales son usualmente requeridos para aplicaciones de DSP.

- Conjunto optimizado de instrucciones: los procesadores digitales de señales soportan instrucciones especiales que permiten calcular algoritmos de DSP que son computacionalmente intensos.

- Interfaz de periféricos: los procesadores digitales de señales usualmente incorporan puertos de interfaz de seriales y paralelos para datos seriales y paralelos.

Estos procesadores digitales utilizan hardware especializado e instrucciones complejas para permitir operaciones sean ejecutadas en un ciclo de instrucción. Adicionalmente, los chips DSP alcanzan un gran nivel de paralelismo al contar con ejecución en paralelo de múltiples instrucciones simples a una velocidad más alta de reloj. El dispositivo utilizado en este trabajo es el OMAP-L138 C6000 DSP+ARM utiliza una arquitectura de instrucciones muy larga (VLIW, por sus siglas en inglés).

1.3.3. Procesadores digitales de punto fijo y punto flotante

Los procesadores digitales se distinguen por el formato aritmético que utilizan: estos pueden ser de punto fijo o punto flotante. Los procesadores de punto fijo usualmente son de 16 o 24 bits, mientras que los procesadores de punto flotante usualmente utilizan 32 bits. En este trabajo se utilizará un procesador con un núcleo de la serie C674x de Texas Instruments que es de punto flotante y representa los números con una mantisa de 24 bits y el exponencial con 8 bits. La mantisa representa una fracción en el rango de -1 a +1, mientras que el exponente es un entero que representa el número de puntos binarios que deben ser desplazados hacia la izquierda o derecha con el objetivo de obtener el valor real. La ventaja de un procesador de punto flotante respecto un procesador de punto fijo es que no es necesario manejar factor de escala y otras técnicas para prevenir un desborde aritmético.

Un procesador de punto flotante puede ser requerido en aplicaciones donde los coeficientes varían con el tiempo. Los procesadores de punto flotante suportan el uso de compiladores de alto nivel (por ejemplo, de lenguaje C), lo cual reduce el costo de desarrollo y mantenimiento. Por otro lado, los procesadores de punto flotante utilizan más energía y tienen un costo más alto que los procesadores de punto fijo, pero su ciclo de desarrollo más rápido puede justificar este costo extra.

1.3.4. Restricciones de procesamiento en tiempo-real

El ancho de banda es una de las limitaciones más importantes de un sistema de procesamiento digital en tiempo real. La velocidad de procesamiento determina la frecuencia máxima al que la señal analógica puede ser muestreada. Por ejemplo, en un procesamiento de muestra por muestra, una salida es

generada antes de que la nueva entrada sea presentada al sistema. Debido a esto, el tiempo de retraso entre entrada y salida para un procesamiento de muestra por muestra debe ser menos de un intervalo de muestreo (T segundos). Un sistema de procesamiento en tiempo real impone ciertas demandas en el tiempo de procesamiento $t_p$que debe ser menor al período de muestreo, T_s, tomando en cuenta también el tiempo en los procesos de entrada y salida y las conversiones analógica/digital, t_o, de modo que:

$$T_s \leq t_p + t_o$$

La ecuación anterior impone una restricción en la frecuencia máxima de una señal que puede ser procesada por el sistema DSP utilizando el método de muestra por muestra, ese ancho de banda f_M es dado por:

$$f_M \leq \frac{f_s}{2} < \frac{1}{2(t_p + t_o)}$$

Generalmente el ancho de banda de procesamiento en tiempo real puede ser incrementado utilizando procesadores digitales más veloces, algoritmos de DSP optimizados, y múltiples procesadores o procesadores con núcleos múltiples. Sin embargo, siempre hay una relación costo/beneficio en cuanto al costo de un sistema y su rendimiento.

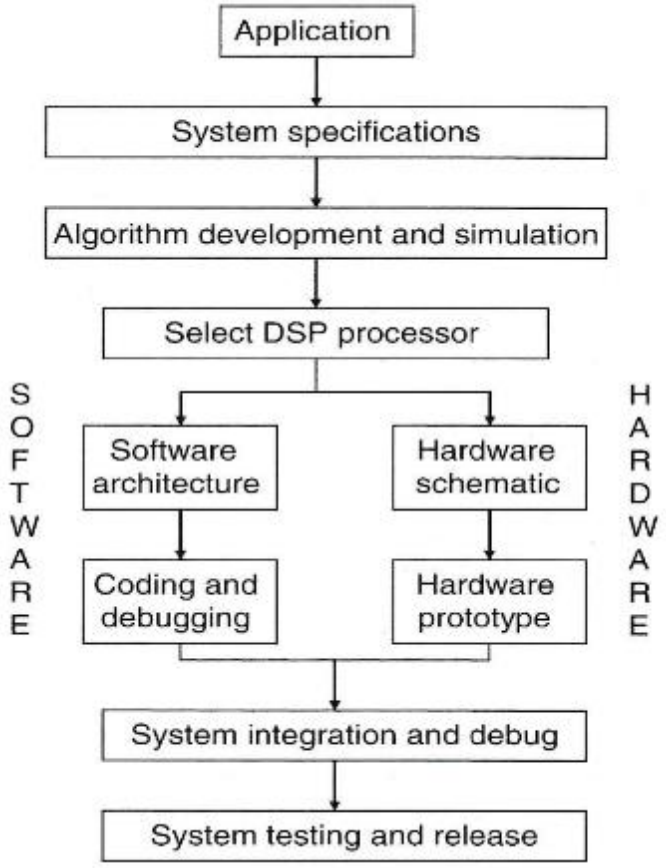
Otro método para incrementar el ancho de banda de un sistema de procesamiento digital es reducir el tiempo de procesamiento de entrada/salida por medio de la utilización del método de bloque por bloque en lugar de muestra por muestra. Con el método de procesamiento por bloque, las operaciones de entrada y salida son usualmente manejadas por controladores de memoria de acceso directo (MDA, por sus siglas en inglés) que coloca la información en una

memoria intermedia. El controlador de MDA interrumpe al procesador cuando la memoria temporal está llena y las señales del bloque están listas para ser procesadas.

1.4. Diseño de sistemas DSP

Un diagrama generalizado del proceso de diseño de sistemas DSP es ilustrado en la siguiente figura.

Figura 5. **Diagrama de flujo simplificado del diseño de un sistema DSP**



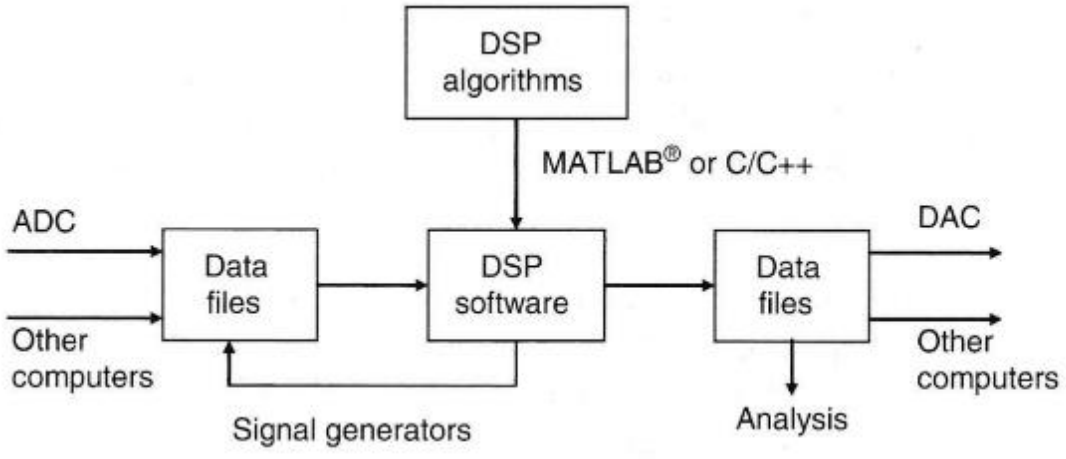
Fuente: SEN, Kuo. *Real-Time digital signal processing.* p. 17.

1.4.1. Desarrollo del algoritmo

Un sistema DSP es frecuentemente caracterizado por un algoritmo embebido, que especifica las operaciones aritméticas a ser realizadas. El algoritmo para una aplicación dada es inicialmente descrito utilizando ecuaciones de diferencia y diagramas de flujo con nombres simbólicos para las entradas y salidas. La siguiente etapa del proceso es proveer más detalles en las secuencias de operación que deben ser realizados a manera de obtener la salida.

Para el desarrollo del algoritmo es muchas veces más sencillo utilizar lenguajes de alto nivel (como Python, Matlab o C/C++), para realizar las simulaciones. Un algoritmo DSP puede ser simulado utilizando una computadora de uso general y así analizar su rendimiento, este proceso se ilustra en el diagrama siguiente:

Figura 6. **Diagrama de flujo del desarrollo de un algoritmo DSP desde una computadora de propósito general**



Fuente: SEN, Kuo. *Real-Time digital signal processing*. p. 17.

La señal de entrada usualmente puede ser generada por un generador de señal, a partir de un experimento o basada en un ambiente real para una aplicación específica. El programa de simulación utiliza las muestras de la señal grabadas en un archivo de datos como entrada para producir la señal de salida que es grabada como un archivo para análisis subsiguiente.

1.4.2. Selección del procesador DSP

El objetivo de la selección de un procesador DSP es escoger un procesador que cumpla con los requerimientos del proyecto de la manera más efectiva tomando en cuenta el costo. Para aplicaciones de DSP en tiempo real, la eficiencia del flujo hacia y desde el procesador también es crítica. Algunos parámetros para evaluar la velocidad de un DSP son los siguientes:

- MIPS: Millones de instrucciones por segundo.
- MOP: Millones de operaciones por segundo.
- MFLOPS: Millones de operaciones de punto flotante por segundo.
- MHz: velocidad del reloj en mega Hertz.
- MMACS: Millones de operaciones de multiplicación y acumulación por segundo.

El costo del hardware y la integración en el proceso de manufactura son importantes para aplicaciones de mucho volumen. Para dispositivos portátiles, el consumo de energía es importante. Para aplicaciones de volumen bajo a medio, habrá una solución intermedia entre tiempo de desarrollo y el costo del hardware. Cuando la velocidad de procesamiento es la prioridad, la comparación más importante entre procesadores es por medio del perfilamiento del algoritmo. En este caso, el código se escribe y optimiza para todos los potenciales procesadores y luego se compara el tiempo de ejecución entre ellos. Otros

factores importantes son el uso de memoria, dispositivos periféricos como convertidores e interfaces de entrada y salida. Además, es importante tener un conjunto de herramientas para el desarrollo de software, entre los cuales se destacan los siguientes:

- Herramientas de desarrollo de software como compiladores, ensambladores, enlazadores, depuradores y simuladores.
- Tarjetas de DSP disponibles comerciales para desarrollo y pruebas antes de que el hardware DSP esté disponible.
- Herramientas para pruebas como emuladores y analizadores lógicos.
- Acceso a la documentación de las librerías de programación DSP, material de referencia, fichas de datos de los dispositivos y material similar.

1.4.3. Desarrollo de software

Entre los criterios para categorizar un software se tienen la confiabilidad, la facilidad de mantenimiento, flexibilidad y eficiencia. Un programa confiable falla muy poco frecuentemente o no falla. Cuando un programa falla, si es fácil de mantener se podrá aplicar una corrección fácilmente. Idealmente esta corrección se podrá realizar por una persona distinta de la que escribió el programa originalmente. Un programa flexible es uno que es fácil de modificar cuando los requerimientos cambian. Un buen programa de DSP contiene funciones pequeñas que cumplen un solo propósito que puede ser fácilmente en otros programas para propósitos distintos.

El diseño de hardware y software se puede realizar al mismo tiempo para una aplicación de DSP. Puesto que hay muchos factores interdependientes entre hardware y software, un diseñador DSP entenderá tantos los aspectos de hardware como de software. El costo del hardware se ha reducido drásticamente

en los últimos años, el mayor costo de las soluciones DSP ahora reside en el desarrollo de software.

El ciclo de vida del software implica la finalización del proyecto de software incluye la definición del proyecto, la creación de especificaciones detalladas, la escritura del código y pruebas modulares, la integración de con otros sistemas, pruebas, y el mantenimiento del producto de software. El mantenimiento del software es una parte importante del costo de un sistema DSP. El mantenimiento incluye la mejora de las funcionalidades del software, la reparación de los errores identificados a medida que se usa el software.

Es importante utilizar los nombres de variables significativas en el código fuente y documentar los programas detalladamente con títulos y comentarios porque simplifica enormemente la tarea de mantenimiento del software. Los trucos de programación deben ser evitados a toda costa, porque no son fiables y son difícil de entender para otra persona incluso con documentación.

Las buenas técnicas de programación juegan un papel esencial en un proyecto de DSP exitoso. Se debe iniciar un enfoque de programación estructurado y bien documentado desde el principio, es importante desarrollar las especificaciones generales para las tareas de procesamiento de señales antes de escribir cualquier programa. Las especificaciones incluyen el algoritmo básico y la tarea descripción, requisitos de memoria, restricciones en el tamaño del programa y tiempo de ejecución.

Después de un análisis meticuloso de las especificaciones es posible detectar errores incluso antes de escribir el código y así prevenir posibles cambios de código en la etapa de integración de sistemas. Un diagrama de flujo es una herramienta de diseño muy útil para adoptar en esta etapa. Escribir y

probar el código DSP es un proceso altamente interactivo. Con el uso de herramientas de desarrollo de software que incluyen simuladores o placas de evaluación como la Texas Instruments OMAP-L138 que se utilizará en este trabajo, el código se puede probar regularmente conforme se escribe. Escribir código en módulos o secciones puede ayudar en este proceso, y cada módulo se puede probar de forma individual, lo que aumenta la probabilidad de que todo el sistema funcione durante la etapa de integración del sistema.

Hay dos lenguajes de programación comúnmente utilizados en el desarrollo de software DSP: C y lenguaje ensamblador. El lenguaje ensamblador es similar al código máquina realmente usado por el procesador y utiliza directamente el conjunto de instrucciones que provee el fabricante del procesador. La programación en lenguaje ensamblador brinda a los ingenieros el control total del procesador funciones y recursos, lo que resulta en el programa más eficiente. Sin embargo, esta es una tarea muy lenta y laboriosa, especialmente para las arquitecturas de procesadores altamente paralelas y los algoritmos DSP más complejos.

Por otro lado, el lenguaje C es más fácil para el desarrollo de software, así como su actualización mantenimiento. La desventaja de utilizar C es que el código de máquina generado por el compilador de C a menudo es ineficiente tanto en velocidad de procesamiento como en uso de la memoria.

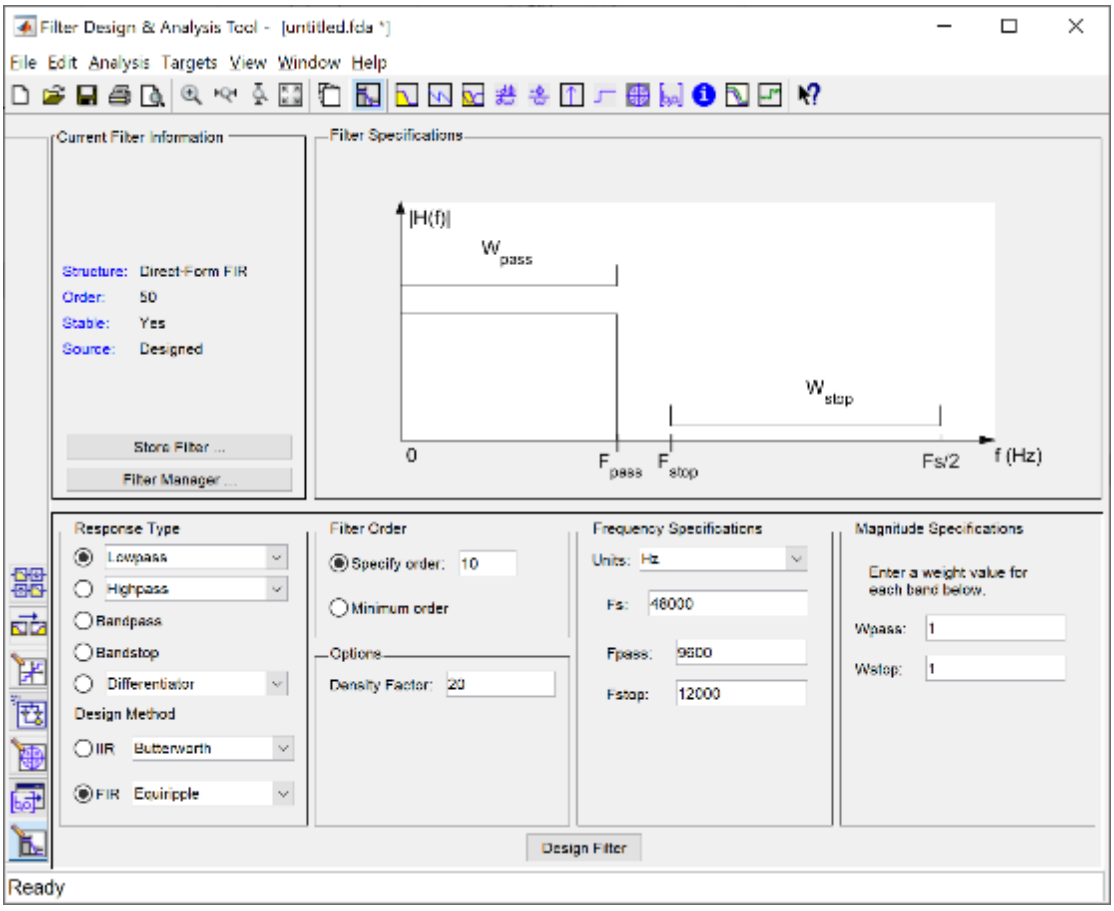
Regularmente, la mejor solución es utilizar una mezcla de C y código ensamblador. El programa general es escrito usando C, pero los bucles y procedimientos críticos de tiempo de ejecución son reemplazados por código ensamblador. En un entorno de programación mixta, se puede llamar a una rutina de ensamblaje como una función o desde el mismo programa C como una llamada en línea.

1.4.4. Herramientas de desarrollo de software

La mayoría de las operaciones DSP se pueden categorizar como de análisis de señal o filtrado. El análisis de señal se ocupa de la medición de las propiedades de la señal. Matlab es una poderosa herramienta para análisis de la señal y visualización, que son componentes críticos en la comprensión y desarrollo de sistemas DSP. C es una herramienta eficiente para realizar procesamiento de señales y su implementación en diferentes plataformas de DSP.

MATLAB es un entorno interactivo para el desarrollo de software técnico y científico que permite realizar análisis numérico, cálculos complejos y visualización de resultados. Su fortaleza radica en el hecho de aplicado a problemas complejos puede resolverlos en una fracción del tiempo que una solución en lenguajes como C o C++ requerirían. Además, esta herramienta provee herramientas que permiten el diseño de filtros (FDATool), y para el procesamiento de señales (SPTool).

Figura 7. **Pantalla de inicio de la herramienta para diseño digital de filtros de Matlab FDATool**

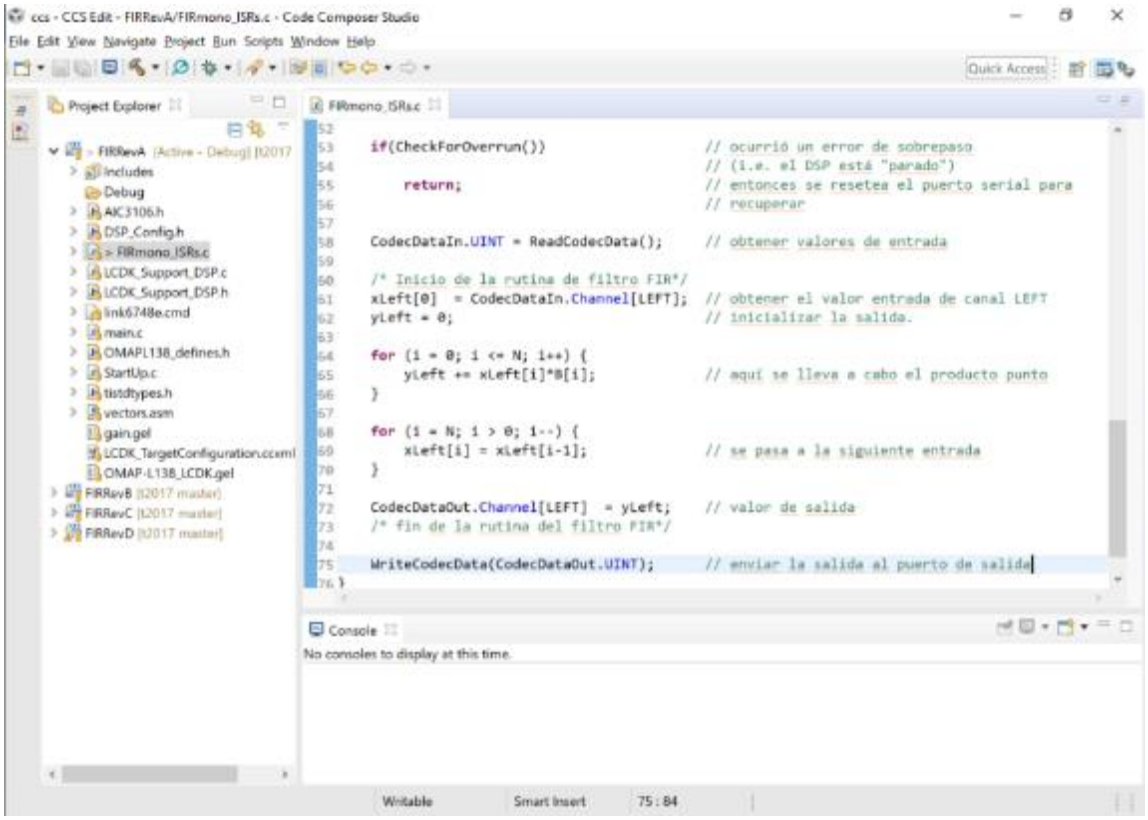


Fuente: The Mathworks. Captura de pantalla de Matlab 2015b.

El propósito de los lenguajes de programación es resolver problemas que implican la manipulación de información. El propósito de los programas DSP es controlar señales para resolver señales específicas. Los programas producidos con lenguajes de alto nivel como C y C++ pueden ser transferidos a otras plataformas de hardware. Debido a que el lenguaje C es de alto nivel, pero también tiene acceso de bajo nivel al procesador, es utilizado frecuentemente para aplicaciones de DSP.

C, se ha convertido en el lenguaje de elección para muchos ingenieros de software DSP, no solamente porque tiene un conjunto poderoso de instrucciones y estructuras de datos, sino también porque puede ser fácilmente portado a diferentes procesadores de señal digital. Un entorno de desarrollo C usualmente incluye un depurador que es muy útil para detectar errores en el código fuente de los programas y permite desplegar los valores de variables intermedias, así como ejecutar el programa línea por línea.

Figura 8. **Ilustración del uso de un entorno de desarrollo integrado para la implementación de algoritmos DSP en tiempo real en lenguaje C**



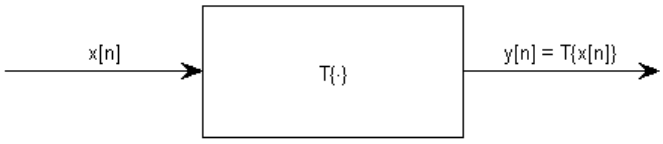
Fuente: Texas Instruments. Captura de pantalla de Code Composer Studio 7.4.

2. SECUENCIAS Y SISTEMAS EN TIEMPO DISCRETO

2.1. Señales en tiempo discreto (secuencias) y su notación

Una señal en tiempo discreto es una señal cuya variable independiente sólo existe en instantes discretos de tiempo, de modo que no es representada de forma continua sino como una secuencia de valores. Un sistema de procesamiento en tiempo discreto se puede visualizar como un operador matemático que mapea o transforma una señal de entrada hacia una señal de salida. El símbolo T {·} se utiliza para representar dicha transformación. Como se puede observar en la figura 9, la señal de entrada x[n] es transformada por el operador T {·} en la salida y[n].

Figura 9. **Representación de un sistema en tiempo discreto, T {·}, que mapea la señal de entrada x[n] hacia la señal de salida y[n]**



Fuente: elaboración propia, empleando Draw.io 2000.

Para hacer uso de los sistemas en tiempo discreto con señales analógicas, como suele ser el caso en muchas aplicaciones, las señales analógicas de entrada se convierten en digitales por medio de convertidores analógico-digitales. Después, el sistema en tiempo discreto recibe muestras de la señal de entrada

con cierta frecuencia, denominada frecuencia de muestreo. La magnitud de la señal al ser digitalizada sólo puede tomar un número finito de valores. El sistema de procesamiento en tiempo discreto es ahora, entonces, un sistema de procesamiento digital de señales. Una vez completado el procesamiento, la señal digital y de tiempo discreto de salida, es convertida nuevamente a una señal analógica por medio de convertidores digital-analógicos.

El esquema de la figura 10 muestra el proceso anteriormente descrito: x(t) es la señal analógica de entrada, luego de muestrear dicha entrada a una frecuencia f_{s_in}, se obtiene la secuencia x[n], que se procesa digitalmente para obtener y[n] = T{x[n]}; finalmente, y[n] se muestrea a una frecuencia f_{s_out}, para obtener la señal analógica de salida y(t). Para definir las frecuencias de muestreo se utiliza el criterio de Nyquist.

Figura 10. **Esquema de un procesador digital de señales utilizado para procesar señales analógicas**

Fuente: elaboración propia, empleando Draw.io 2000.

La figura 10 podría esquematizar una sucesión en la que la señal x[n] ya fue digitalizada y almacenada, de modo que el proceso para llegar a y(t) puede hacerse mucho tiempo después de que la señal de entrada fue generada, como es el caso de la reproducción de música en un disco compacto, por ejemplo. Por otro lado, todo el proceso descrito podría tomar lugar de manera secuencial y simultánea. Si este es el caso, se dice que el sistema funciona en tiempo real. El

término “tiempo real” significa que el sistema responderá con suficiente rapidez para procesar adecuadamente la señal de entrada entre muestra y muestra.

Un algoritmo es una secuencia lógica de pasos que se toman con el objetivo de resolver un problema. Algunos de los algoritmos básicos en el procesamiento digital de señales están relacionados con los filtros de respuesta finita al impulso (FIR, por sus siglas en inglés), los filtros de respuesta infinita al impulso (IIR, por sus siglas en inglés), la convolución, la transformada rápida de Fourier, y la transformada discreta del coseno. Cuando los primeros chips DSP fueron fabricados, estos algoritmos eran implementados en ellos exclusivamente en lenguaje ensamblador. Sin embargo, actualmente muchos fabricantes de chips DSP proveen compiladores de lenguajes de alto nivel para los procesadores que fabrican, permitiendo así que el desarrollo de algoritmos como es el caso de Texas Instruments y el DSP que se utiliza en este trabajo.

2.2. La transformada discreta de Fourier

La transformada discreta de Fourier tiene su origen, como es de esperarse en la transformada de Fourier en el tiempo continuo. Similarmente, esta se utiliza para obtener la respuesta en frecuencia sobre una secuencia:

$$X(\omega) = \sum_{k=0}^{K-1} x\,(n) e^{-j2\pi n\omega/K}$$

En dispositivos DSP se utiliza la transformada rápida de Fourier que reduce significativamente el número de operaciones y que viene, en ciertos chips DSP, implementada por defecto en funciones que se denomina intrínsecas que están optimizadas para poder realizar esta operación de forma rápida.

2.3. Filtros de respuesta finita al impulso (FIR)

Un filtro de respuesta finita es aquel que si es excitado con un impulso unitario da una respuesta finita en el tiempo, donde la salida del filtro sólo hace una transformación de la señal de entrada y sus elementos rezagados. Algebraicamente:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots + b_k x[n-k]$$

Más concisamente:

$$y[n] = \sum_{k=0}^{N} b_k x[n-k]$$

Las principales características de este filtro son:

- El filtro es totalmente descrito por los coeficientes asociados a la señal de entrada y sus elementos rezagados.

- La salida del filtro no lo retroalimenta, y hace que la señal de salida sea 0 una vez se han procesado todos los elementos rezagados de la señal de entrada, haciendo la salida, efectivamente, finita.

- El filtro es estable, mientras la entrada al filtro sea finita, la salida también lo es.

- Lo anterior implica que el filtro no tiene ceros, sólo polos, es decir, el filtro puede tener entradas que hagan la salida cero, pero no infinito.

Uno de los aspectos más interesantes del análisis de los filtros de respuesta finita es la predicción del comportamiento del filtro cuando una muestra sinusoidal de varias frecuencias es aplicada a la entrada, es decir, cómo estimar la respuesta en el dominio de la frecuencia.

- La respuesta al impulso de un filtro es la secuencia de salida cuando una entrada con valor unitario, un impulso, es aplicada.

- Punto importante: cuando se habla de un filtro de respuesta finita, de los coeficientes del filtro y respuesta al impulso, se está hablando de lo mismo.

- Convolución en el dominio del tiempo es equivalente a la multiplicación en el dominio de la frecuencia.

- Algunas ventajas que ofrecen los filtros de respuesta finita son:

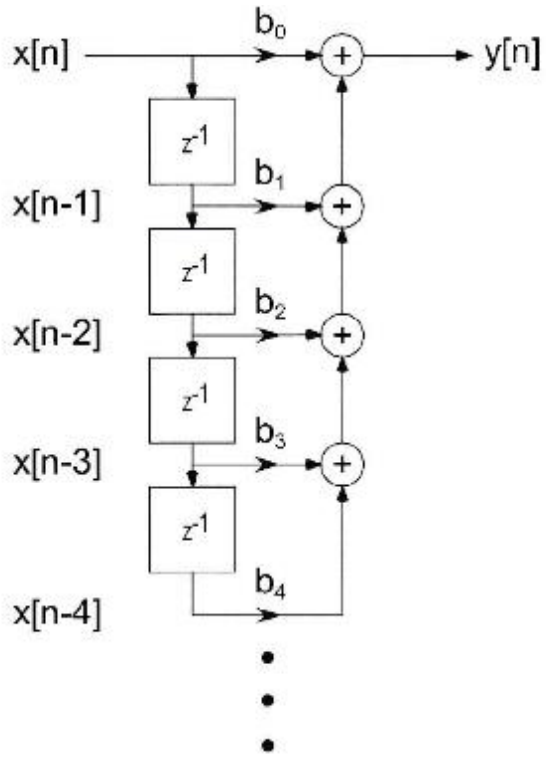
    - Siempre son estables.

    - El diseño de filtros de fase lineal puede ser garantizado.

    - Los errores de precisión debido a la conversión digital son menos severos.

    - Pueden ser implementados eficientemente en la mayoría de los procesadores digitales con hardware optimizado e instrucciones del procesador especializadas.

Figura 11. **Diagrama de bloques de un filtro de respuesta finita (FIR)**



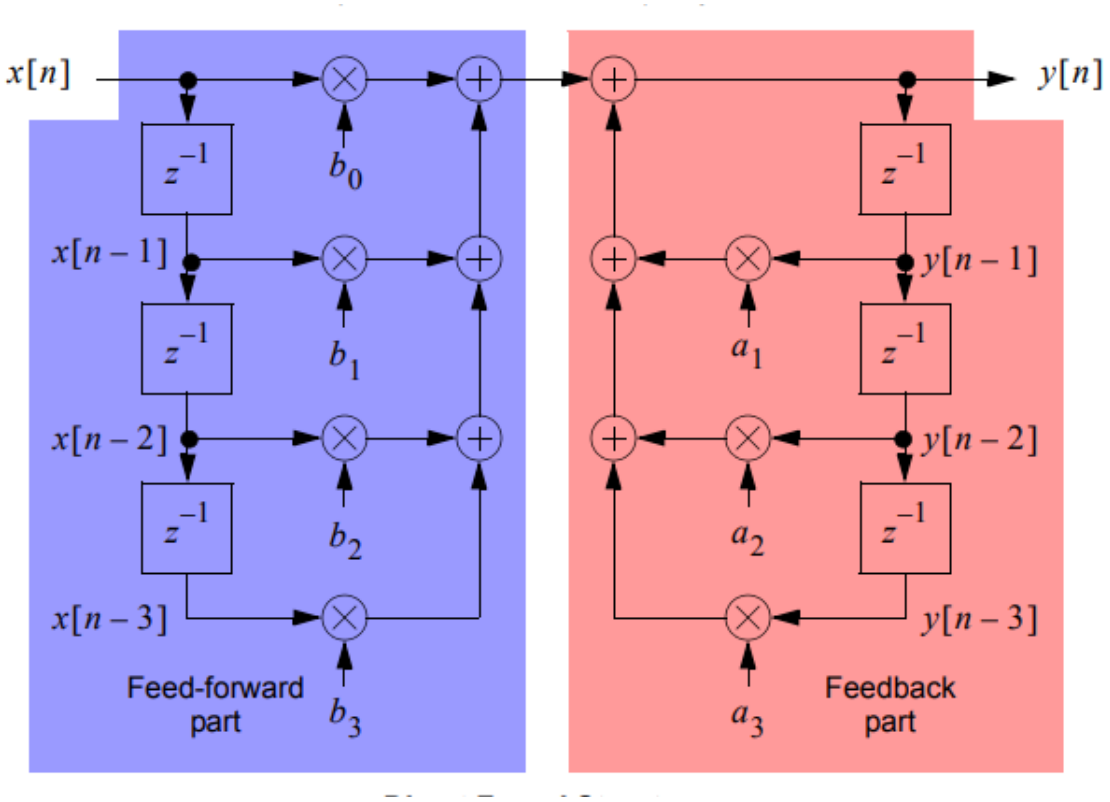
Fuente: WELCH Thad; WRIGHT, Cameron; MORROW, Michael. *Real-Time digital signal processing from MATLAB to C with the TMS320C6x DSPs*. p. 36.

2.4. Filtros de respuesta infinita (IRR)

Los filtros de respuesta infinita se representan en el dominio del tiempo de la siguiente manera:

$$y(n) = \sum_{l=1}^{N} a_l\, y(n-l) + \sum_{k=0}^{M} b_k\, y(n-k)$$

Figura 12. **Diagrama de bloques de un filtro IIR en el que N=M=3**



Fuente: ECE 2610 Signals and Systems. *Direct form 1 structure*. http://www.eas.uccs.edu/~mwickert/ece2610/lecture_notes/ece2610_chap8.pdf. Consulta: junio de 2018.

Nótese que para este tipo de filtros la salida rezagada es parte de la entrada y no solamente la entrada y sus rezagos como es el caso de los filtros FIR, esta situación de retroalimentación hace que los filtros IIR puedan ser más inestables que los filtros FIR de modo que con este tipo de filtros hay dos tipos de problemas a considerar en particular:

- Estabilidad: dado que siempre hay retroalimentación involucrada en un filtro de tipo IIR, el sistema puede volverse inestable. Para sistemas en

tiempo real se puede matemáticamente establecer la estabilidad manteniendo los polos de la función de transferencia dentro el circulo unitario.

- Respuesta de fase: los filtros de tipo FIR muestran una respuesta de fase lineal mientras que los filtros IIR no pueden tener esa respuesta de fase lineal. Tener un rezago de fase constante es crucial para ciertas aplicaciones.

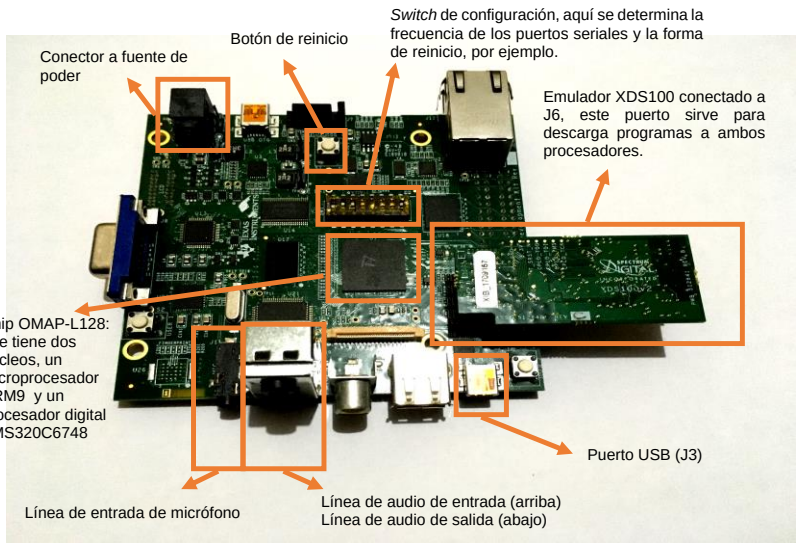
En resumen, IIR filtros pueden cumplir con requerimientos con un menor número de coeficientes que un filtro FIR pero con el precio de ser inestables. Es por esta razón que en este trabajo el diseño de solución toma en cuenta un filtro de tipo FIR a manera de garantizar la estabilidad del sistema.

3. EL PROCESADOR TMS320C6748

3.1. El *kit* de desarrollo OMAP-L138 DSP+ARM9

Para el presente trabajo, se eligió la implementación de un algoritmo de procesamiento digital de señales en el procesador TMS320C6748. Para poder programar dicho procesador el fabricante Texas Instruments también fabrica equipos o *kits* de desarrollo que sirven para fácilmente probar el *hardware* ya que cuentan con un circuito que permite la conexión de accesorios a diferentes puertos de ahí que se define como un equipo de desarrollo de bajo costo o LCDK (del inglés, *low cost development kit*).

Figura 13. ***Kit* de desarrollo OMAP-L138 DSP+ARM9 y la ubicación de los puertos comúnmente utilizados**



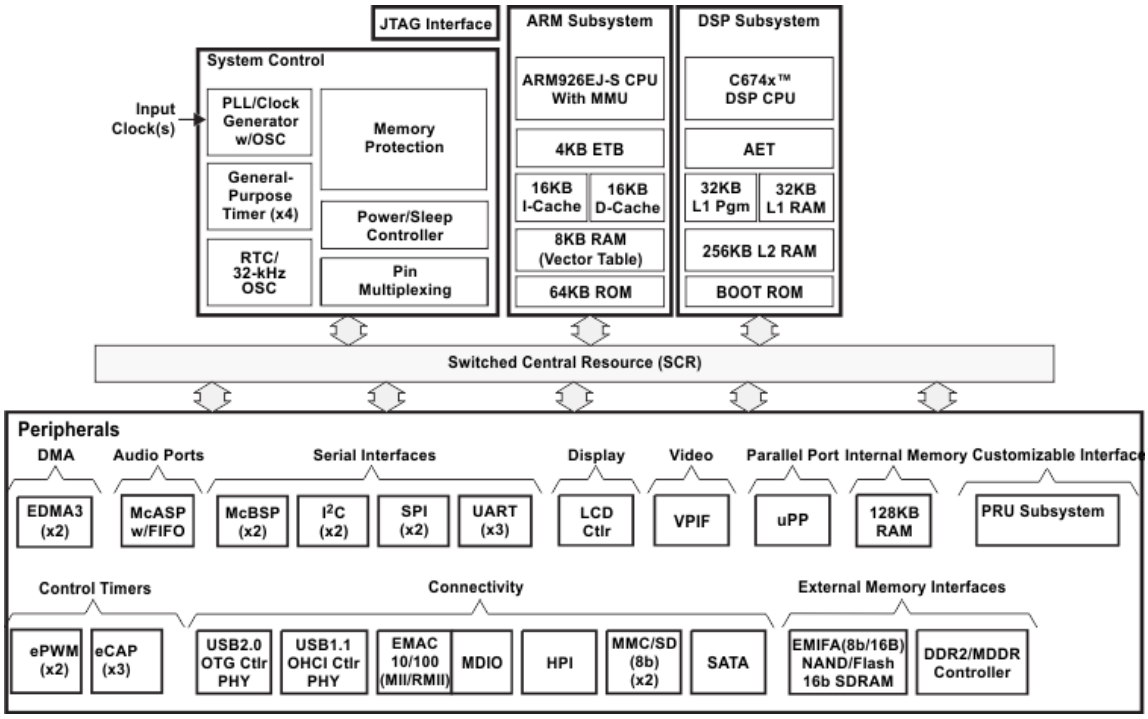
Fuente: elaboración propia, empleando Paint 2019.

El kit OMAP-L138 LCDK (del inglés *L138/C6748 Development Kit*), que es un kit de hardware y software que permiten el desarrollo y experimentación con el chip OMAP-L138 que contiene dos núcleos: Un procesador digital de señales TMS320C6748 y un procesador ARM9 integrado en la misma plataforma. El kit también incluye la herramienta Code Composer que provee un entorno de desarrollo para programar en lenguaje C y en lenguaje ensamblador. Este kit de desarrollo fue elegido para el presente trabajo.

El kit OMAP-L138 LCDK tiene, entre otras, las siguientes características:

- Un chip OMAP-L138 con dos núcleos: Un procesador digital de señales MS320C6748 operando a 456 MHz y un procesador ARM926EJ-S operando a 456 MHz.
- Codificador/decodificador de audio estéreo AC97.
- 2 entradas y 1 salidas de audio estéreo.
- 2 puertos USB.
- Puerto Ethernet a 10/100 Mbps.
- Puertos de entrada y salida de video.
- Conexiones estándar para conectar tarjetas de expansión.
- Conector para tarjeta de memoria SD.

Figura 14. **Diagrama de bloques del procesador OMAP-L138**



Fuente: Functional Block Diagram for OMAP-L138. *Datasheet.* https://www.ti.com/data-sheets/diagram.tsp?genericPartNumber=OMAP-L138&diagramId=SPRS586J. Consulta: junio de 2018.

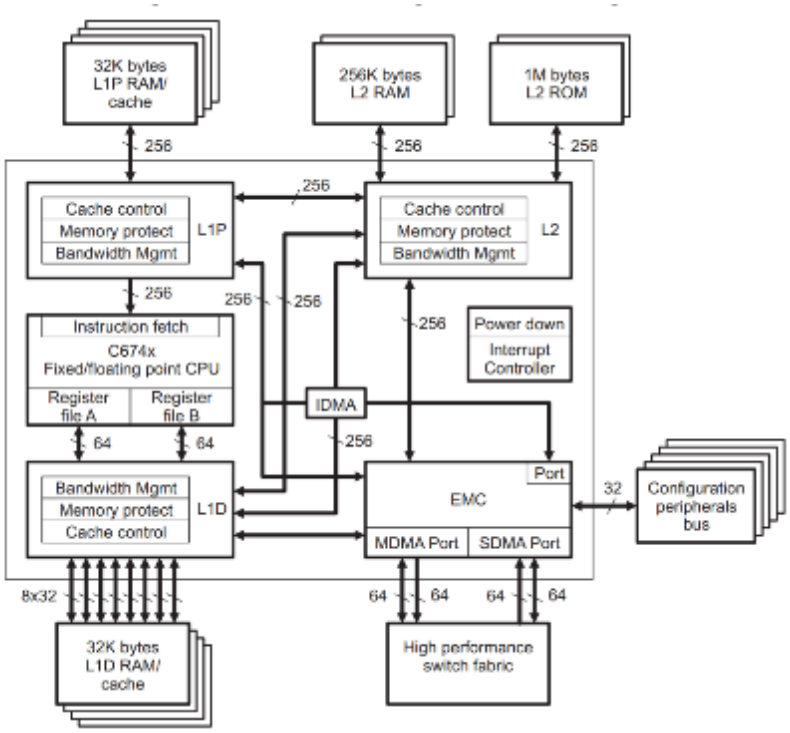
En las siguientes secciones se elabora la arquitectura del procesador digital de señales utilizado en este trabajo.

3.2. Arquitectura del computador

La arquitectura de un procesador se refiere a la interfaz que existe entre el hardware y software al más bajo nivel y que conlleva toda la información necesaria para escribir un programa en lenguaje máquina que se ejecutará correctamente, incluyendo las instrucciones, los registros y el tamaño de memoria entre otros atributos.

Un diagrama de bloques del procesador TMS320C6748 se muestra en la siguiente figura.

Figura 15. **Diagrama de bloques de la familia de procesadores TMS320C674x**



Fuente: OMAP-L138 C6000 DSP+ARM Processor Technical Reference Manual. *Introduction TMS320C674x Megamodule Block Diagram*. p. 93.

La unidad central de procesamiento (CPU, por sus siglas en inglés), contiene registros y las unidades funcionales, todas interconectadas por buses de señalización y secuenciadas por la unidad de tiempo y control. El sistema de memoria y los subsistemas de entrada / salida (E/S), están conectados a la CPU por el bus del sistema. El generador de reloj proporciona la temporización para la operación lógica y el circuito de reinicio aseguran que el procesador comience

desde un estado conocido. Comúnmente, cierto número de dispositivos periféricos se encuentran en el subsistema de E/S de un microprocesador.

- Interfaces de puertos de E/S paralelos a dispositivos externos para control o detección con sensores.

- Los puertos en serie facilitan las comunicaciones con dispositivos locales o distantes utilizando hardware protocolos seriales.

- El contador y los temporizadores se utilizan para establecer intervalos de tiempo precisos, generar formas de onda rectangulares y para contar eventos externos.

Aunque las E/S y la memoria se muestran en el diagrama como externas a la CPU, en la práctica se pueden integrar en el dispositivo procesador.

3.3. Arquitectura del conjunto de instrucciones

La arquitectura del conjunto de instrucciones (ISA, por sus siglas en inglés), se puede utilizar como una forma de diferenciación para los distintos tipos de procesadores. Por un lado, existen los computadores con conjunto de instrucciones complejas (CISC, por sus siglas en inglés), que soporta instrucciones que realizan instrucciones complejas. Por ejemplo, una simple instrucción puede ejecutar un filtrado o hacer una búsqueda en un arreglo. Por otro lado, existen las computadoras con un conjunto de instrucciones reducido (RISC, por sus siglas en inglés), que tienen un número limitado de instrucciones de bajo nivel. Un conjunto de instrucciones RISC sólo permite operaciones en información almacenada en los registros del procesador.

Debido a que una máquina CISC puede ejecutar tareas complejas directamente desde el hardware, usualmente tiene una ventaja en rendimiento en ciertas tareas específicas. Sin embargo, en general, un conjunto de instrucciones complejas hace bastante difícil optimizar el rendimiento de un procesador, de manera que casi todos los microprocesadores de uso general adoptan una arquitectura RISC. La familia ARM de procesadores, como el núcleo ARM926EJ-S disponible en el equipo OMAP-L138 es RISC.

3.3.1. Arquitecturas de registros

Las arquitecturas del procesador también pueden clasificarse sobre la base de las ubicaciones posibles de la fuente y destino de los operandos de una instrucción. En este sentido, existen dos clasificaciones comunes de arquitecturas:

- Registro/memoria: estas arquitecturas que permiten que uno o más operandos de las instrucciones que se encuentran en memoria. Esta arquitectura se usa comúnmente en las máquinas CISC.

- Cargar/grabar: estas arquitecturas requieren que todos los operandos de instrucciones estén en los registros. Solamente muy pocas instrucciones tienen acceso a la memoria y este acceso es típicamente realizado como una simple transferencia de un espacio en memoria a un registro (cargar), o de un registro a un espacio en memoria (grabar). La familia de procesadores DSP de Texas Instruments TMS320C6x utiliza una arquitectura de cargar/grabar.

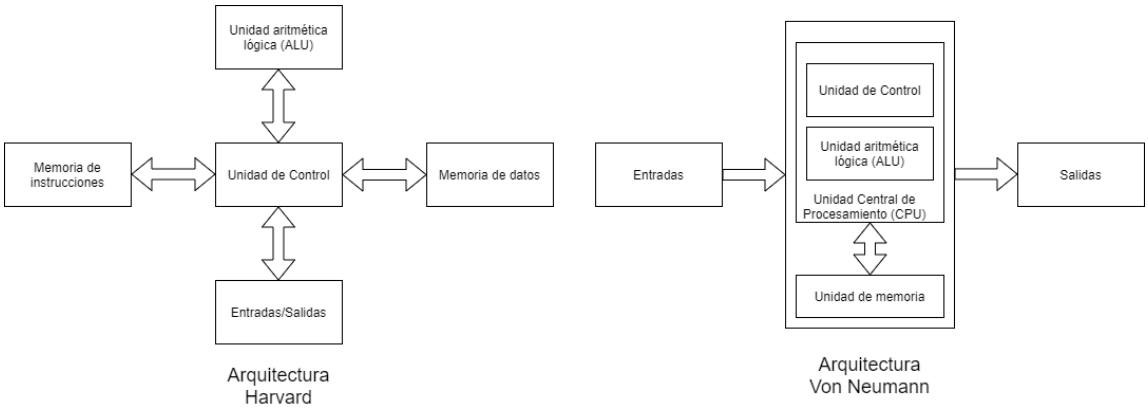
3.3.2. Arquitecturas de memoria

La organización de la estructura de memoria de un procesador es un elemento claro del operativo general del sistema. La información en la memoria de un procesador se puede categorizar en dos distintos tipos:

- Las instrucciones que el procesador ejecutará en el espacio de código.
- La información a la que el procesador tendrá acceso como parte de la ejecución es grabada en el espacio de datos.

En el diseño del procesador, estos dos espacios se pueden colocar en espacios de memoria físicamente independientes y separados, o bien pueden existir en el mismo espacio, como se muestra en el siguiente diagrama.

Figura 16. **Comparación de arquitectura Harvard versus Von Neumann**



Fuente: elaboración propia, empleando Draw.io 2000.

La arquitectura con el espacio de código y datos separado se denomina arquitectura Harvard. La ventaja principal de esta arquitectura es que pueden existir operaciones simultáneas en los espacios de código y memoria, lo que

aumenta el ancho de banda de la memoria. Además, dado que el código y los espacios de datos tienen diferentes buses hacia el procesador, sus tamaños pueden ser diferentes a fin de optimizar cada uno, y el código puede estar absolutamente protegido contra la corrupción inadvertida debido a las operaciones de datos.

La principal desventaja de esta arquitectura es que el código y los datos sólo se puede colocar en sus respectivos espacios, así que la memoria libre en el espacio de código no se puede utilizar para datos y viceversa. La arquitectura Harvard se usa comúnmente en microcontroladores de tipo espacial, incluyendo las familias de DSP de Texas Instruments TMS320c2x y TMS320c5x.

Los procesadores donde código y los datos existen en el mismo espacio de memoria se refirieren como las arquitecturas de Von Neumann. La ventaja principal de esta arquitectura es que toda la memoria la memoria está disponible para código o datos en cualquier proporción, lo que le da una flexibilidad mucho mayor a la asignación de memoria.

Las desventajas incluyen el hecho no se puede hacer un acceso simultáneo al código y a los datos, y que existe la posibilidad de corrupción entre los espacios de código y datos. Esta arquitectura es utilizada por casi todos los microcontroladores de uso general y por un buen número de procesadores especializados, debido principalmente a su flexibilidad. Los procesadores DSP de la familia TMS320C6x de Texas Instruments utilizan una arquitectura de memoria Von Neumann para la memoria externa.

Además de sus memorias principales, los sistemas de procesador modernos usualmente tienen memoria caché. Estas son memorias locales muy rápidas de un tamaño limitado que están dentro del procesador y normalmente

funcionan sin ningún control explícito por parte del procesador. El propósito de la memoria caché es retener una copia de un subconjunto de instrucciones con el propósito de que estén listas cuando el procesador las necesite y de esta manera éste no tenga que esperar a hacer la solicitud a la memoria principal que puede ser más lenta. La razón por la que esta técnica es muy efectiva es que los programas tienden a tener un comportamiento que puede ser predicho lo cual puede ser aprovechado. En particular, la mayoría de las memorias caché están optimizadas para aprovechar las siguientes propiedades:

- Localidad temporal: esta es una propiedad por medio de la cual el código y los datos que son utilizados tienen una alta probabilidad de ser usados otra vez. Si se mantiene el código y los datos recientes en la memoria caché, la probabilidad de volverlos a usar es alta.

- Localidad espacial: esta es una propiedad por medio de la cual el código y los datos que están cerca a la ubicación en memoria que fue utilizada puede ser requeridos con más probabilidad que otras ubicaciones en memoria más distantes. Para aprovechar la localidad espacial, la mayoría de memorias cachés automáticamente obtienen un bloque de datos más grande de la memoria principal cuando cualquier ubicación dentro del bloque es utilizada.

Como las memorias caché tienen tamaño limitado, varios algoritmos son utilizados para tratar de mantener la información que es más probable que sea necesaria en la memoria caché. Este tipo de memorias puede proveer una mejora significativa en el rendimiento. De hecho, muchos procesadores modernos de uso general dependen en jerarquías multi-nivel de memoria caché para lograr un buen desempeño. Una desventaja de este tipo de memoria es que hacen difícil

predecir la ejecución del programa porque la memoria caché es asignada con algoritmos probabilísticos.

- Ciclo de instrucción

En su forma más simple, un procesador opera mediante la lectura de una instrucción de memoria para luego ejecutarla. Para poder ejecutar la instrucción, el procesador primero debe decodificar la instrucción para determinar qué es lo que debe hacer, leer los operandos necesarios luego realizar la acción requerida, para luego escribir el resultado en la ubicación de memoria adecuada. Este comportamiento secuencial resulta en un diseño simple, sin embargo, el rendimiento se ve afectado. Por ejemplo, cuando la instrucción es obtenida, el bus que codifica la memoria estará en reposo hasta que la instrucción se ejecuta completamente. Similarmente, las unidades funcionales que, en efecto, ejecutan la tarea (sumador, multiplicador, por ejemplo) estarán en reposo mientras se obtiene la instrucción, se decodifica, y mientras los resultados se escriben hacia su respectiva salida. Este modelo secuencial se podría mejorar si todas las partes del sistema pudiese trabajar simultáneamente.

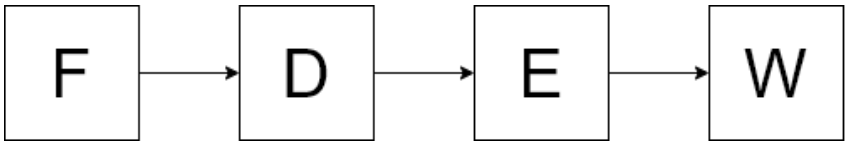
- Paralelización o *Pipelining*

La paralelización del computador permite ejecutar varias instrucciones simultáneamente, éste se lleva a cabo por medio de un *pipeline* que divide la operación del procesador en varias etapas. Con el fin mejorar la utilización de recursos del hardware del procesador, el procesamiento que éste realiza se divide en etapas donde cada una maneja una porción distinta del procesamiento total de una instrucción. En cuanto una instrucción se procesa en una etapa del *pipeline*, ésta se pasa a la siguiente etapa. Debido a que el *pipeline* puede ir tan

rápido como la etapa más lenta, está diseñada para que sea equilibrada donde cada etapa tenga un tiempo similar.

Una canalización o *pipeline* representativa de cuatro etapas (las *pipelines* de un procesador moderno pueden tener más de veinte etapas), se muestra en la figura siguiente.

Figura 17. **Esquema de *pipeline***



Fuente: elaboración propia.

F (obtener o *fetch*), es responsable de leer la próxima instrucción de memoria D (decodificar) esta etapa decodifica la instrucción para determinar qué acción y operandos se requieren. E (ejecutar), esta etapa carga los operandos si hay alguno y luego realiza las operaciones requeridas. W (escribir o *write-back*) esta operación graba el resultado de las operaciones en el destino.

Si se asume que cada etapa de la *pipeline* tiene 10 ns de retraso, entonces si cada instrucción se completa de forma secuencia, tomaría 40 ns completar cada instrucción y el procesador podría ejecutar 25 millones de instrucciones por segundo (25 MIPS). Utilizando *pipeline*, cada etapa operaría independientemente en una instrucción diferente y siempre tomaría 40 ns ejecutar cada instrucción con la diferencia que las instrucciones podría ser ejecutadas en paralelo, eso tiene el efecto de que cada instrucción ahora termina su viaje a lo largo de la *pipeline* cada 10 ns, de manera que el procesador ahora opera a 100 MIPS, una mejora del 400 %. En este caso, el procesador aún tiene cuatro instrucciones

siendo ejecutadas a la vez. Esta mejora en el rendimiento puede alcanzarse sólo si el pipeline recibe un flujo contante de instrucciones y tiene acceso libre de restricciones a los operandos requeridos como entrada y salida de las instrucciones.

3.3.3. Emisión simple versus emisión múltiple

Hasta el momento, se han descrito únicamente procesadores de emisión simple, es decir que ejecutan una operación a la vez. A pesar de que la *pipeline* introduce una forma de paralelismo, la ejecución de instrucciones sigue siendo secuencial. Por ejemplo, si hay una serie de tres instrucciones ADD, las operaciones de suma ocurren en la etapa *ejecutar* de la *pipeline* secuencialmente. Para mejorar la velocidad de ejecución aún más, existe el procesador de emisión múltiple para ejecutar más de una operación en paralelo. Los procesadores de emisión múltiple requieren cierto número de consideraciones:

- Para ejecutar más de una operación, el procesador requiere instancias múltiples de cierto hardware. Ejemplos de estas unidades funcionales son: Unidad lógica aritmética (ALU, por sus siglas en inglés), y sumadores y multiplicadores especializados.

- Cuando el procesador ejecuta instrucciones en paralelo, debe determinar si las mismas pueden ser ejecutadas en cualquier momento dado. Este problema se resuelve con la planificación del tiempo del procesador o *scheduling*.

- Debe haber un mecanismo para que el procesador recupere el orden adecuado en los resultados para preservar sus interdependencias debido a que las unidades funcionales tendrán distintos retardos, y aún si las instrucciones son ejecutadas en orden es posible que las operaciones sean completadas fuera de orden, y hace que el estado del procesador sea difícil de determinar en un instante dado.

3.3.4. Planificación del tiempo el procesador

En procesador de emisión múltiple la planificación del tiempo del procesador o *scheduling* es necesaria para determinar en orden en el cual las instrucciones pueden ser ejecutadas de manera segura. El *scheduling* debe asegurar que las dependencias entre instrucciones son preservadas al ser ejecutadas y sus resultados se graban. El *scheduling* puede ser completado en tiempo real en el hardware del procesador (*scheduling* dinámico), o puede ser hecho por anticipado mediante las herramientas de generación de código (*scheduling* estático). Esta elección lleva a arquitecturas fundamentalmente diferentes siendo cada una adecuada para diferentes entornos.

La planificación dinámica se utiliza sobre todo con código que no fue escrito para ser intrínsecamente paralelo. El procesador intenta descubrir el paralelismo en el código (paralelismo a nivel de instrucción, ILP por sus siglas en inglés), al examinar las interdependencias entre instrucciones, administrando asignaciones de unidades funcionales y garantizando que los resultados se almacenen en el orden adecuado. Esto le da a la planificación dinámica una gran ventaja, y puede explotar paralelismos en el código que no fue escrito de forma paralela, lo que permite que se ejecute sin cambios de código de serie en una arquitectura paralela (la programación dinámica se utiliza en casi todos los procesadores de las computadoras de escritorio).

La principal desventaja es que la programación es un proceso no trivial por lo que hacerlo en tiempo real requiere una cantidad adicional significativa de hardware con el costo asociado y el consumo de energía. Debido a los requisitos de hardware, el hardware típico de planificación dinámica sólo es capaz de mirar a un conjunto de solamente varios cientos de instrucciones, lo que limita su capacidad para encontrar el paralelismo. Los procesadores superescalares usan múltiples unidades funcionales y planificación dinámica, y el procesador impone todas las dependencias entre instrucciones. El orden de ejecución exacto no es conocido hasta el tiempo de ejecución, pero se garantiza que la ejecución producirá los mismos resultados que se obtendrían con una ejecución secuencial del código.

La planificación estática elimina la carga de la planificación del procesador y en su lugar requiere un código que especifica explícitamente qué instrucciones se pueden ejecutar en paralelo. En este caso, el compilador es responsable de determinar qué instrucciones pueden ser ejecutadas en paralelo y para asegurar que el resultado de una operación estará listo antes de que se use en otra instrucción. Las principales ventajas de la planificación estática son que el compilador puede buscar el paralelismo en todo el programa y así determinar un orden de ejecución más eficiente, y que elimina la necesidad del hardware de planificación dinámica, lo cual representa un ahorro significativo de energía y costos.

La principal desventaja de la planificación estática es que el código que se genera depende mucho de la máquina y puede ser menos adaptable a la dinámica cambiante del sistema.

La arquitectura estática de palabra de instrucción muy larga (del inglés *Very Large Integration Word*, VLIW), es utilizada en el procesador TMS320C6x y

obtiene ocho instrucciones en paralelo (un paquete de obtención) para pasar simultáneamente a sus ocho unidades funcionales. Si una unidad funcional no se utiliza, entonces se pasa una instrucción de no operación (NOP). Los compiladores de lenguajes de alto nivel hacen toda la programación de instrucciones y hacen cumplir las dependencias. La escritura de código en lenguaje de ensamblador para esta arquitectura es difícil, pero por lo general solo se hace para el código muy crítico con el fin de maximizar la utilización de la unidad funcional y reducir el tiempo de ejecución.

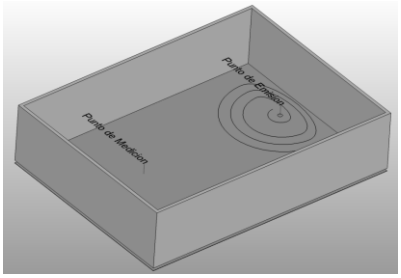
4. IDENTIFICACIÓN DE SISTEMA DE AUDIO E IMPLEMENTACIÓN EN EL PROCESADOR TMS320C6748 EN UN AMBIENTE REVERBERANTE

En esta sección del trabajo se introduce el problema a resolver y el algoritmo de identificación de sistemas denominado mínimos cuadrados (LMS, por sus siglas en inglés). Una vez el sistema ha sido identificado, se procede a su implementación en el procesador TMS320C6748 que está incluido en el equipo de evaluación OMAP-L138+ARM9. Por último, se presentan los resultados.

4.1. Descripción del problema: identificación de sistemas de audio en un ambiente reverberante

En un ambiente reverberante, como se ilustra en la siguiente figura, una fuente de audio genera ondas de sonido que viajan en el espacio pero que también regresan con rezagos debido al eco característico del lugar.

Figura 18. **Ilustración de un ambiente reverberante**

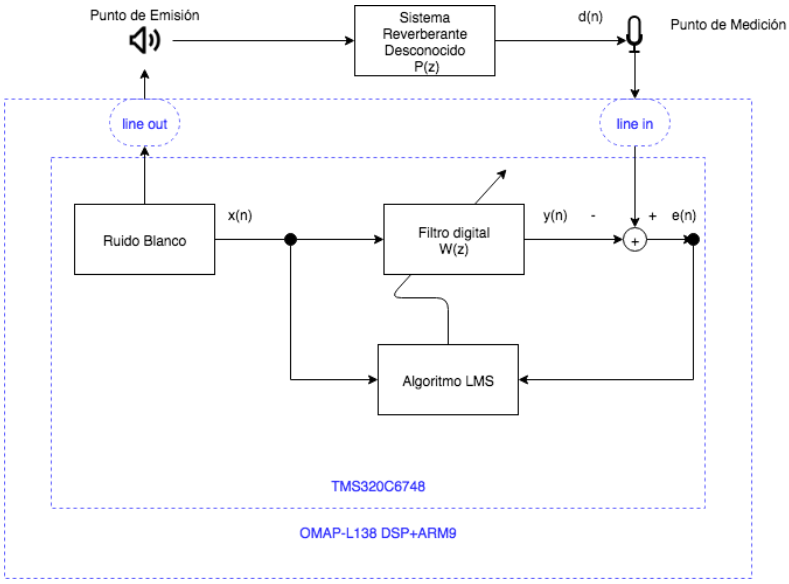


Fuente: elaboración propia, empleando FreeCAD 2016.

El problema que se plantea en este trabajo es la identificación del sistema que genera el ambiente reverberante. En el punto de medición la señal de audio que se escucha es la suma de las ondas de sonido que llegan de forma directa y las que llegan después de percutir con las paredes del ambiente, ya que la geometría del lugar causa que las ondas reboten y llegue con rezago al punto de medición. Uno puede visualizar la suma de todas las señales de audio en el punto de medición como la salida de un filtro de respuesta finita (el eco para después de cierto tiempo). A través de las técnicas de identificación de sistemas descritas en secciones posteriores, este sistema puede ser identificado.

La configuración de la identificación del sistema se ilustra en la siguiente figura, donde se utiliza la salida de audio del equipo OMAP-L138 para transmitir ruido blanco, luego este ruido blanco se transmite por el sistema reverberante y es medido por un micrófono que está conectado a la línea de entrada del equipo OMAP-L138. Tanto el ruido blanco como la señal captada por el micrófono es convertida analógico/digital y digital/analógico, respectivamente, utilizando el códec TLB320AIC3106.

Figura 19. **Diagrama de bloques del problema a resolver**



Fuente: elaboración propia, empleando Draw.io 2000.

En una sección posterior se describe la implementación del algoritmo de identificación de sistemas en el procesador digital TMS320C6748.

4.2. Filtros adaptativos

En ciertas aplicaciones de filtros digitales, el ambiente cambia constantemente, un filtro óptimo en un conjunto de condiciones ambientales puede que no sea óptimo bajo otras condiciones. Puede que los parámetros del filtro necesiten cambiarse dependiendo de las condiciones ambientales actuales. Un filtro que se ajusta a las condiciones cambiantes del ambiente se conoce como un filtro adaptativo.

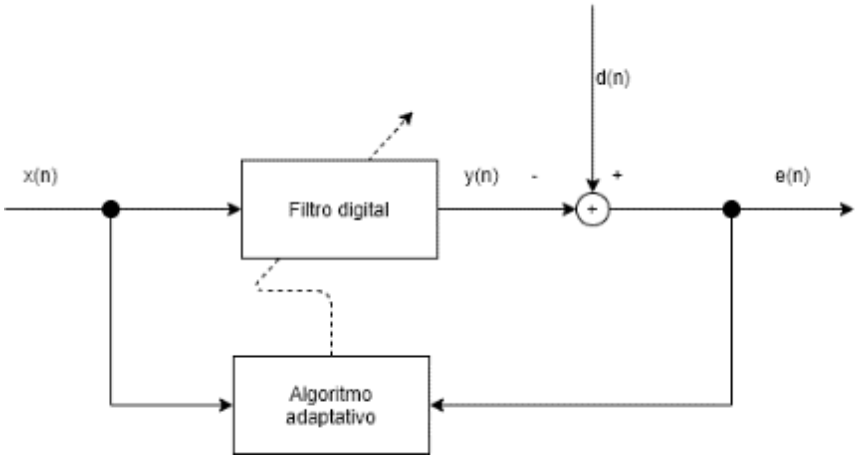
Los filtros de respuesta finita (FIR), y de respuesta infinita (IIR), operan con coeficientes fijos, estos coeficientes se de determinan a partir de un diseño del

filtro. Para algunas aplicaciones, este diseño es predeterminado, pero para otras, puede que un diseño predeterminado, no trabaje adecuadamente y las características del sistema pueden ser cambiantes, para estos casos, los filtros adaptativos pueden ser usados porque sus coeficientes se ajustan automáticamente mediante algoritmos que se adaptan a las señales y sistemas cambiantes.

Hay cuatro áreas de aplicación de los filtros adaptativos: identificación de sistema, predicción, cancelación de ruido y modelado inverso. La diferencia principal entre estas aplicaciones es la configuración de señal de entrada, la señal de salida, la señal deseada y la señal de error.

Un filtro adaptativo consiste en dos partes distintas: Un filtro digital que realiza el filtrado deseado y un algoritmo adaptativo que automáticamente actualiza los coeficientes del filtro. En general se pueden representar con la siguiente figura.

Figura 20. **Diagrama de bloques de un filtro adaptativo**



Fuente: elaboración propia, empleando Draw.io 2000.

En el diagrama de bloques del filtro adaptativo anterior, el filtro procesa la señal de entrada $x(n)$ y produce una señal de salida $y(n)$, la salida $y(n)$ es comparada con una señal deseada $d(n)$ y una señal de error $e(n)$ es calculada como la diferencia de ambas, es decir $d(n) - y(n)$, esta señal de error es utilizada para cambiar los coeficientes del filtro con el objetivo de hacer la salida más semejante a la señal deseada. La fuente de las señales $x(n)$ y $d(n)$, y las características de las señales $y(n)$ y $e(n)$ depende de la aplicación. El algoritmo adaptativo ajusta los filtros de los coeficientes para minimizar la función de costo de la señal dada $e(n)$, de manera que los coeficientes del filtro se actualizan a modo de que la diferencia entre la señal deseada y la señal de salida sea minimizada progresivamente.

Los filtros adaptativos pueden usar filtros FIR o IIR, pero debido a que los filtros IIR pueden tener polos fuera del círculo unitario y por lo tanto ser inestable, los filtros FIR se utilizan ampliamente para realizar aplicaciones prácticas y éste es el caso en el presente trabajo.

Un filtro digital FIR tiene coeficientes fijos, los coeficientes también pueden ser visualizados como pesos o ponderaciones w_k:

$$y(n) = \sum_{k=0}^{K-1} w_k x(n-k)$$

La salida de un filtro digital adaptativo de respuesta finita está dada por:

$$y(n) = \sum_{k=0}^{K-1} w_k(n) x(n-k)$$

Como se puede observar, la diferencia entre ambos puede pasar desapercibida con una inspección inicial, pero es grande, en el caso de los filtros adaptativos, los coeficientes dependen del tiempo, no son constantes si no que en función de n, es decir, $w_k(n)$. La ecuación para filtros adaptativos se puede escribir de forma más conveniente utilizando la notación vectorial con negrillas. $\mathbf{w}$ es el vector de pesos, la T significa transposición de un vector columna a un vector fila. $\mathbf{x}$ es la señal de entrada que no necesita ser transpuesta ya que por defecto el vector está en una columna.

Los coeficientes $w_k(n)$ están en función del tiempo y se actualizarán con el algortimo adaptativo. La entrada al sistema se define com el vector $\mathbf{x}(n)$:

$$\mathbf{x}(n) \equiv \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-(K-1)) \end{bmatrix}$$

y los coeficientes del filtro se representan por el vector $\mathbf{w}(n)$:

$$\mathbf{w}(n) \equiv \begin{bmatrix} w_0(n) \\ w_1(n) \\ \vdots \\ w_{K-1}(n) \end{bmatrix}$$

de manera que la salida del filtro FIR adaptativo se puede reescribir como:

$$y(n) = [w_0(n) w_1(n) \dots w_{K-1}(n)] \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-K+1) \end{bmatrix}$$

$$y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$

la salida del filtro se compara entonces con la señal deseada $d(n)$ para obtener la señal de error $e(n)$:

$$e(n) = d(n) - y(n) = d(n) - \mathbf{w}^T(n)\mathbf{x}(n)$$

Esta señal de error es la que se utilizará para definir la función de costo del problema de optimización que representa la determinación de los coeficientes $\mathbf{w}(n)$ del filtro adaptativo tal que la señal de error sea mínima.

4.2.1. La función de correlación

La función de correlación en una variable aleatoria determina la dependencia de una variable aleatoria, y está definida por:

$$r_{xx}(n,k) = E[x(n)x(k)]$$

La correlación es una herramienta útil para detectar las señales que están afectadas por ruido aleatorio, para estimar la respuesta al impulso de un sistema desconocido. Un ejemplo de aplicación es el sonar y el radar donde la señal que es reflejada, es la señal retrasada que se emitió. Mediante la medición del retraso del viaje de ida y vuelta utilizando una medida apropiada de la función de correlación, el radar y el sonar pueden determinar la distancia del objetivo.

4.2.2. La función de costo

La función de costo se utiliza para definir qué tan bien se ajustan los coeficientes del filtro para minimizar el error $e(n)$, una función comúnmente utilizada en el campo de filtros adaptativos, en particular, y en general, en los algoritmos de aprendizaje automático, es la función de errores cuadráticos

medios (MSE, por sus siglas en inglés) que es el valor esperado de la diferencia al cuadrado de la señal deseada y la salida del filtro adaptativo:

$$\xi \equiv E[e^2(n)]$$

.

Algebraicamente, esta expresión se puede representar de manera distinta pero equivalente, utilizando la relación $e(n) = d(n) - y(n) = \mathbf{w}^T(n)$.

$$\begin{aligned}
\xi &= E[d(n) - y(n)]^2 \\
\xi &= E[d^2(n) - 2d(n)y(n) + y^2(n)] \\
\xi &= E[d^2(n)] - 2E[d(n)y(n)] + E[y^2(n)] \\
\xi &= E[d^2(n)] - 2E[d(n)\mathbf{w}^T(n)\mathbf{x}(n)] + E[\mathbf{w}^T(n)\mathbf{x}(n)]^2 \\
\xi &= E[d^2(n)] - 2E[d(n)\mathbf{x}^T(n)\mathbf{w}(n)] + E[\mathbf{w}^T(n)\mathbf{x}(n)\mathbf{w}^T(n)\mathbf{x}(n)] \\
\xi &= E[d^2(n)] - 2E[d(n)\mathbf{x}^T(n)]\mathbf{w}(n) + E[\mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)] \\
\xi &= E[d^2(n)] - 2\mathbf{p}\mathbf{w}(n) + \mathbf{w}^T(n)E[\mathbf{x}(n)\mathbf{x}^T(n)]\mathbf{w}(n) \\
\xi &= E[d^2(n)] - 2\mathbf{p}\mathbf{w}(n) + \mathbf{w}^T(n)\mathbf{R}\mathbf{w}(n)
\end{aligned}$$

El error cuadrático medio se puede calcular como el valor esperado de la diferencia entre la salida del filtro y la señal deseada. Esta diferencia puede ser expresada en términos del vector de correlación cruzada $\mathbf{p}$, que es la correlación entre $d(n)$ y $x(n)$, y de la matriz de auto correlación $\mathbf{R}$, que es la auto correlación del vector de entrada $\mathbf{x}(n)$.

El vector de correlación cruzada se define como:

$$\begin{aligned}
\mathbf{p} &\equiv E[d(n)\mathbf{x}(n)] \\
\mathbf{p} &= [r_{dx}(0)r_{dx}1 \dots r_{dx}K - 1]^T
\end{aligned}$$

y $r_{dx}(k) \equiv E[d(n+k)x(n)]$ es la función de correlación cruzada entre $d(n)$ y $x(n)$.

Asimismo, la autocorrelación de la señal de salida $\mathbf{R}$ se define como:

$$\begin{aligned}\mathbf{R} &\equiv E[\mathbf{x}(n)\mathbf{x}^T(n)] \\ \mathbf{R} &= \begin{bmatrix} r_{11}(0) & x_{xx}(1) & \dots & x_{xx}(K-1) \\ r_{21}(1) & x_{xx}(0) & \dots & x_{xx}(K-2) \\ \vdots & & & \vdots \\ r_{xx}(K-1) & x_{xx}(K-2) & \dots & x_{xx}(0) \end{bmatrix}\end{aligned}$$

donde r_{xx} es la función de auto correlacióin de x(n). La matriz $\mathbf{R}$ es simétrica ya que todos los elementos de la diagonal principal son iguales y las diagonales paralelas a estas también son iguales.

4.2.3. Optimización de la función de costo

MSE es minimizado cuando su derivada respecto al tiempo es cero:

$$\begin{aligned}\xi &= E[d(n) - y(n)]^2 \\ \xi &= E[d^2(n)] - 2\mathbf{p}\mathbf{w}(n) + \mathbf{w}^T(n)\mathbf{R}\mathbf{w}(n) \\ \xi &= E[d^2(n)] - 2\mathbf{p}\mathbf{w}(n) + \mathbf{R}\mathbf{w}^2(n) \\ \frac{d\xi}{dn} &= 0 - 2\mathbf{p}\frac{\mathbf{w}(n)}{dn} + \mathbf{R}[2\mathbf{w}(n)\frac{\mathbf{w}(n)}{dn}]\end{aligned}$$

Para minimizar ξ se requiere que $= 0 $, esto ocurre cuando $\mathbf{R}\mathbf{w}(n) = \mathbf{p}$, de otra forma:

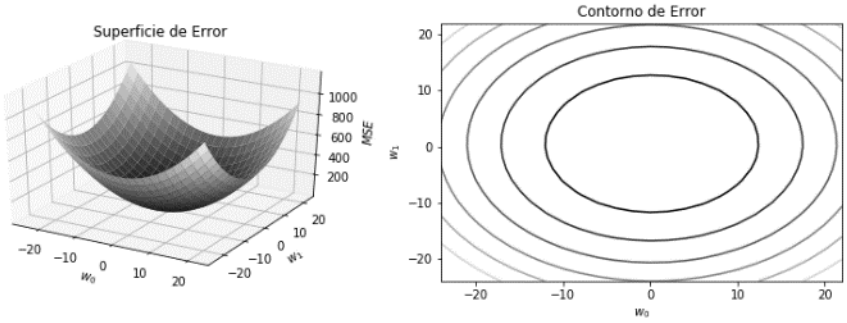
$$\mathbf{w}(\mathbf{n}) = \mathbf{R}^{-1}\mathbf{p}$$

Debido a que la solución requiere calcular una matriz inversa, en la práctica no se utiliza esta solución si no que se utiliza una aproximación, la cual se cubre en la siguiente sección.

La función de costo $\xi \equiv E[e^2(n)]$ es cuadrática para un filtro adaptativo de tipo FIR con respecto al vector de coeficientes $\mathbf{w}(n)$. De manera que los valores de MSE asociados con un vector $\mathbf{w}(n)$ forma un espacio de dimensión $K+1$, este espacio es usualmente llamado superficie de error.

Por ejemplo, si $K=2$ la superficie de error forma un espacio tridimensional con la forma de un paraboloide elíptico, si este paraboloide se corta en planos sobre el punto mínimo ξ_{min} que es paralelo al plano que forman los coeficientes w_0 y w_1, se obtienen elipses concéntricas con valores constantes de la función MSE. A estas elipses se les denomina el contorno de error, como se muestra en la siguiente figura.

Figura 21. **Gráfica de la superficie de error de la función MSE y su respectivo contorno**



Fuente: elaboración propia, empleando Matlab R2015b.

Como se ilustra en la figura anterior, la función MSE es una función cuadrática con un punto mínimo global. Encontrar los coeficientes que minimicen esta función implica descender por la superficie hasta alcanzar el punto más bajo, donde el gradiente de esta función es cero. Los métodos que utilizan el gradiente para encontrar la solución se basan en la idea de que el gradiente de la función

MSE indica la dirección que más rápido encontraría este mínimo. El método de pendiente máxima utiliza el gradiente para encontrar la dirección de búsqueda en que la superficie de error tiene la mayor tasa de cambio decreciente.

El método de pendiente máxima es una técnica iterativa que inicia en un punto arbitrario de la superficie con un vector inicial $\mathbf{w}(0)$ y desciente hasta el fondo de esta en la dirección negativa del gradiente en ese punto. El gradiente de la superficie se denota como $\nabla\xi(n)$, el algoritmo recursivo se puede implementar como indica la siguiente ecuación:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \frac{\mu}{2}\nabla\xi(n)$$

μ se denomina factor de convergencia y es un parámetro del algoritmo que determina la velocidad con que se requiere alcanzar la convergencia, si el factor es muy pequeño (cercano a cero) el algoritmo convergerá más lentamente que si es mucho mayor que cero, sin embargo, un parámetro muy alto de factor de convergencia puede volver al algoritmo inestable, por lo tanto, este parámetro se calibra para que haya un buen compromiso entre estabilidad del algoritmo y velocidad de convergencia.

Las correcciones sucesivas, entonces, logran que el vector alcance el valor óptimo $\mathbf{w}_{optimo}$ para hacer que la superficie de error sea mínima $\xi_{mín}$, en dicho punto el gradiente es cero y el proceso de adaptación ha terminado, porque el vector permanece en su solución óptima.

4.2.4. El algoritmo LMS

En muchas aplicaciones prácticas, $d(n)$ y $x(n)$ son funciones desconocidas, debido a esto, la función MSE no puede ser calculada de forma directa por lo

tanto los métodos de gradiente no pueden ser aplicados de manera directa. El método de mínimos cuadrados (LMS, por sus siglas en inglés) estima la función MSE de la forma:

$$\xi(n) = e^2(n)$$

Es decir, utiliza los errores instantáneos en lugar de los valores esperados de la función de error al cuadrado. De esta forma, el gradiente es la derivada parcial de la función de costo respecto al vector de coeficientes:

$$\nabla\xi(n) = 2[\nabla e(n)]e(n)$$

Dado que señal de error está dada por la diferencia entre la señal deseada y la señal de entrada que pasa por el filtro adaptativo:

$$e(n) = d(n) - \mathbf{w}^T(n)\mathbf{x}(n)$$

El gradiente de $e(n)$ es:

$$\nabla e(n) = -\mathbf{x}(n)$$

Si se sustituye este resultado en la función del gradiente estimado, se tiene que:

$$\nabla\xi(n) = 2[\nabla e(n)]e(n) = 2[-\mathbf{x}(n)]e(n) = -2\mathbf{x}(n)e(n)$$

De manera que el algoritmo LMS se reduce a la siguiente ecuación:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \frac{\mu}{2}[-2\mathbf{x}(n)e(n)]$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{x}(n)e(n)$$

Este algoritmo se puede resumir en los siguientes pasos:

- Determinar el tamaño del filtro (K), en un ambiente reverberante esto se determinará por la geometría del lugar, la frecuencia de muestreo y el número de rezagos que se quiere tomar en cuenta. Asimismo, se debe determinar el factor de convergencia μ y el punto inicial $\mathbf{w}_0$. Estos parámetros determinarán el rendimiento del filtro.

- Calcular el valor de salida del filtro adaptativo de tipo FIR.

$$y(n) = \mathbf{w}^T(n)\mathbf{x}(n) = \sum_{k=0}^{K-1} w_k\,(n)x(n-k)$$

- Calcular la señal de error.

$$e(n) = d(n) - y(n)$$

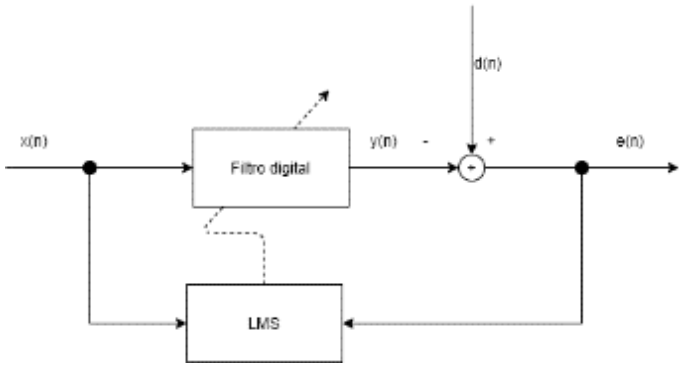
- Actualizar los coeficientes del filtro adaptativo utilizando el algoritmo LMS que se puede reescribir de la forma:

$$w_k(n+1) = w_k(n) + \mu x(n-k)e(n)$$

para $k = 0,1,\dots,K-1$

El diagrama de bloques de este algoritmo está ilustrado en la siguiente figura.

Figura 22. **Diagrama de bloques de algoritmo LMS**

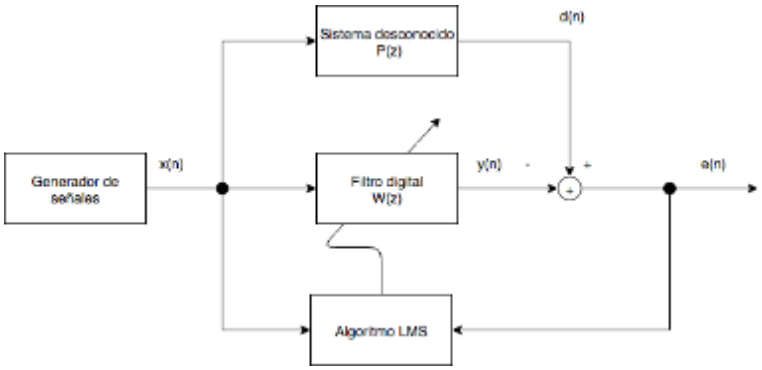


Fuente: elaboración propia, empleando Draw.io 2000.

4.2.5. Identificación de sistemas

La identificación de sistemas es una técnica utilizada para identificar un sistema desconocido. El diagrama de bloques de esta aplicación se ilustra en la figura siguiente.

Figura 23. **Diagrama de bloques de un filtro adaptativo aplicado a la identificación de sistemas**



Fuente: elaboración propia, empleando Draw.io 2000.

$P(z)$ es el sistema desconocido que se requiere identificar con el filtro adaptativo $W(z)$. A ambos bloques se les aplica la señal de entrada $x(n)$ y con la minimización de la señal de diferencia $e(n) = d(n) - y(n)$ se puede identificar las características de $P(z)$.

Si se asume que el sistema desconocido $P(z)$ es un systema FIR de largo K y si se utiliza ruido blanco como señal de entrada, la minimización de $e(n) = d(n) - y(n)$ hará que los coeficientes del filtro $w_k(n)$ se aproximen a $p(k)$. En el caso de que la identificación del sistema sea perfecta, $W(z)$ identifica a $P(z)$ después de que el filtro adaptativo haya convergido y la señal de error $e(n)$ converja a cero.

En esta aplicación, si el sistema es variante con el tiempo, el algoritmo rastrea continuamente las variaciones del sistema desconocido y minimiza el error, lo cual puede ser útil para aplicaciones de cancelación de eco, por ejemplo:

4.3. Implementación del algoritmo LMS en lenguaje C en el procesador TMS320C6748

A continuación, se presenta el entorno para la implementación del algoritmo.

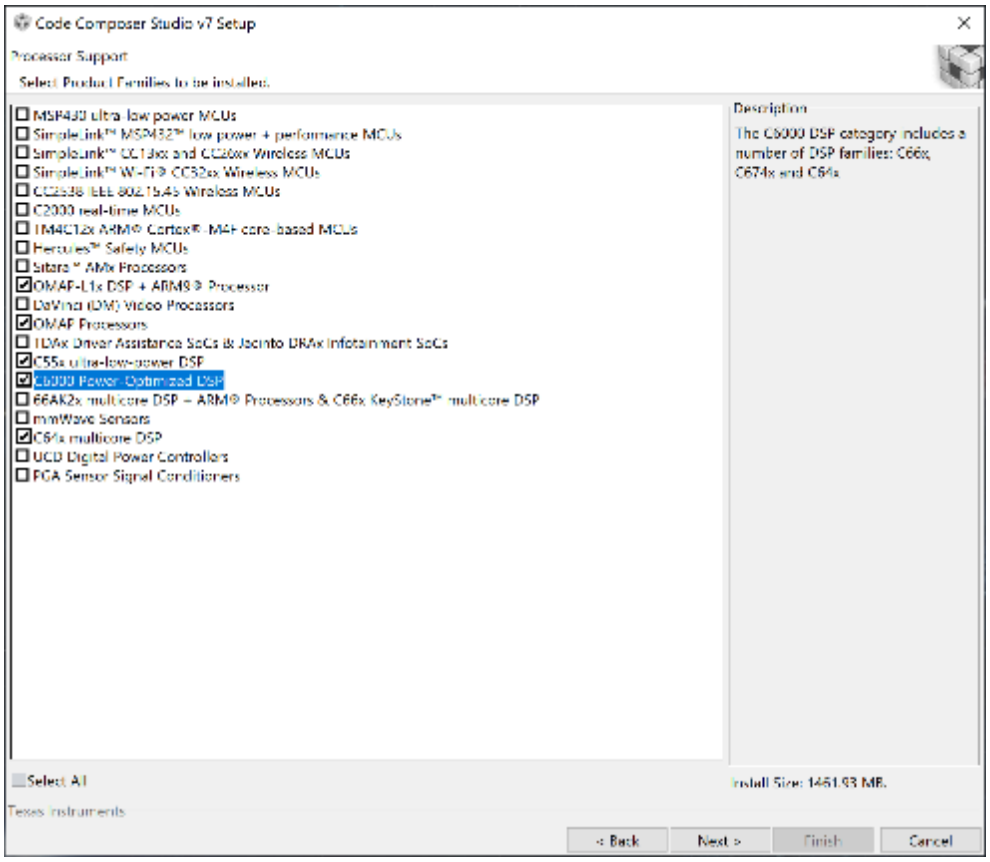
4.3.1. Entorno de desarrollo

Para poder implementar un algoritmo en un chip DSP se necesita un entorno integral de desarrollo, para este trabajo se utilizó Code Composer Studio de Texas Instruments, este entorno de desarrollo está basado en el entorno de desarrollo Eclipse que es de código abierto, el uso de este entorno de Code Composer Studio es libre de licencia y se puede obtener en la página: https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html.

4.3.2. Instalación del entorno de desarrollo

Al instalar el entorno de desarrollo se debe seleccionar la opción que le permite compilar el código en lenguaje C directamente a cada tipo de dispositivo, en este trabajo la opción seleccionada fue: OMAP-L1 DSP + ARM9 ® PRocessors.

Figura 24. **Instalación del entorno de desarrollo Code Composer Studio**



Fuente: elaboración propia, empleando Code Composer Studio 2017.

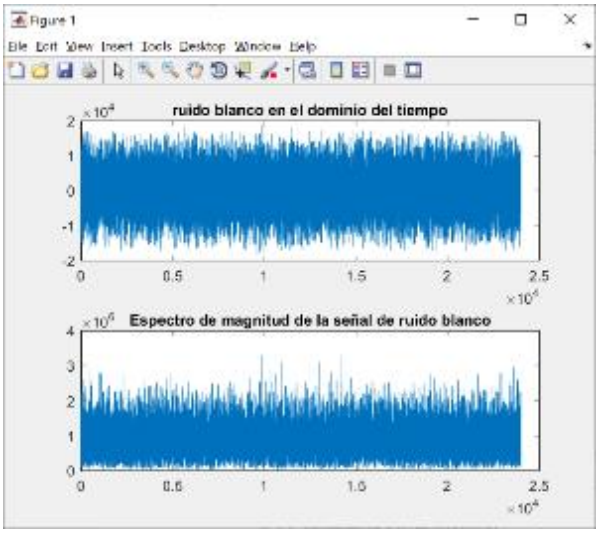
4.3.3. Implementación del algoritmo

A continuación, se presenta la implementación del algoritmo.

4.3.3.1. Señal de entrada

La señal de entrada x(n) para el sistema fue una señal de ruido blanco generada en Matlab. La gráfica de esta señal puede verse en la figura siguiente tanto en el dominio del tiempo como en el dominio de frecuencia en su espectro de magnitud, se puede notar que las señales de ruido blanco tienen una magnitud similar en todos los espectros de frecuencia. Para obtener esta respuesta en frecuencia se utilizó la función de Matlab fft.

Figura 25. **Señal de entrada x(n) tanto en dominio del tiempo como en el dominio de frecuencia**



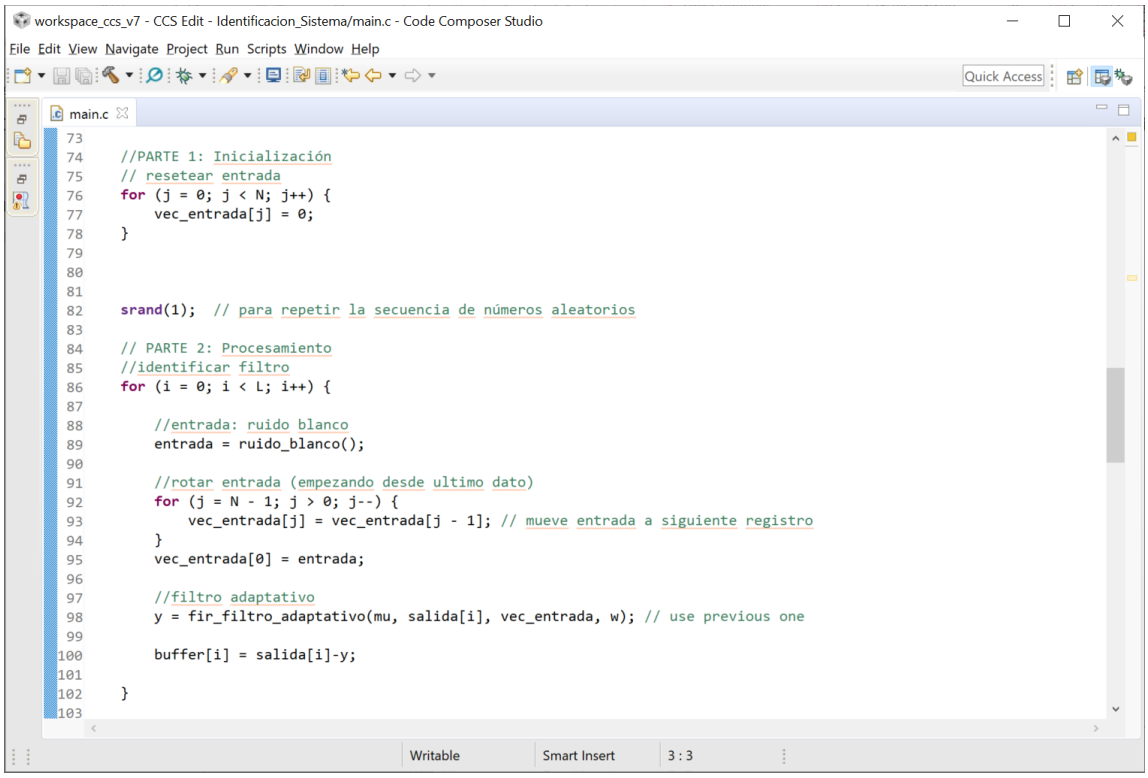
Fuente: elaboración propia, empleando Matlab R2015b.

4.3.4. Implementación del algoritmo

Para la implementación el algoritmo adaptativo se utilizó una función que lo encapsula. El código de la implementación se muestra en las siguientes figuras.

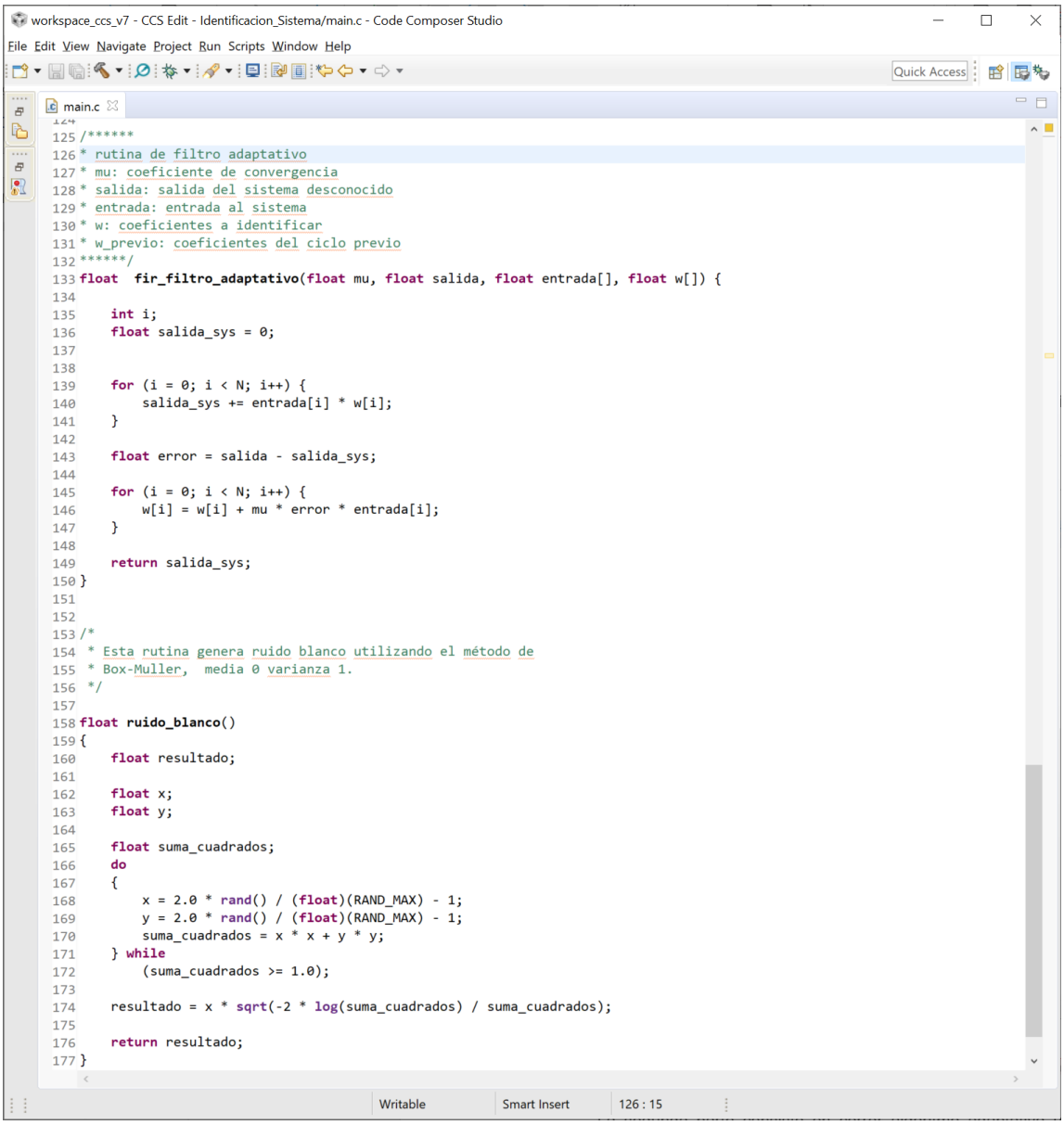
La implementación del algoritmo se divide en tres partes que están comentadas en el código fuente en C. La primera consiste en inicializar los arreglos que representan para cada una de las variables involucradas.

Figura 26. **Parte 1 y 2 del código fuente en lenguaje C para identificación de sistemas**



Fuente: elaboración propia, empleando Code Composer Studio 2017.

Figura 27. **Implementación del algoritmo de identificación del sistema**




```c
/******
* rutina de filtro adaptativo
* mu: coeficiente de convergencia
* salida: salida del sistema desconocido
* entrada: entrada al sistema
* w: coeficientes a identificar
* w_previo: coeficientes del ciclo previo
******/
float  fir_filtro_adaptativo(float mu, float salida, float entrada[], float w[]) {

    int i;
    float salida_sys = 0;


    for (i = 0; i < N; i++) {
        salida_sys += entrada[i] * w[i];
    }

    float error = salida - salida_sys;

    for (i = 0; i < N; i++) {
        w[i] = w[i] + mu * error * entrada[i];
    }

    return salida_sys;
}


/*
 * Esta rutina genera ruido blanco utilizando el método de
 * Box-Muller,  media 0 varianza 1.
 */

float ruido_blanco()
{
    float resultado;

    float x;
    float y;

    float suma_cuadrados;
    do
    {
        x = 2.0 * rand() / (float)(RAND_MAX) - 1;
        y = 2.0 * rand() / (float)(RAND_MAX) - 1;
        suma_cuadrados = x * x + y * y;
    } while
        (suma_cuadrados >= 1.0);

    resultado = x * sqrt(-2 * log(suma_cuadrados) / suma_cuadrados);

    return resultado;
}
```

Fuente: elaboración propia, empleando Code Composer Studio 2017.

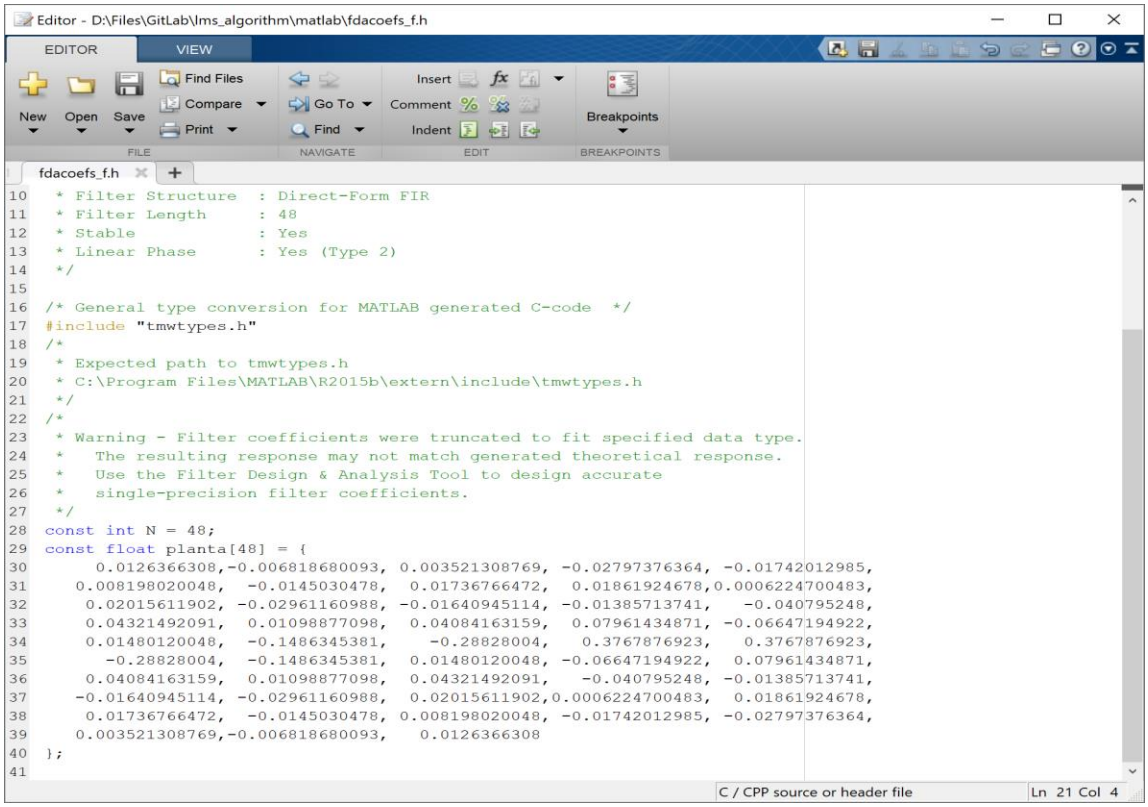
La segunda parte consiste en correr algoritmo adaptativo utilizando la función que encapsula el algoritmo LMS.

La implementación del algoritmo de identificación del sistema se realiza mediante la función fil_filtro_adaptativo que encapsula el algoritmo LMS.

4.3.5. Resultado

Para probar el algoritmo de identificación del sistema se procedió, en primer lugar, con ingresar un filtro FIR que representara el ambiente reverberante. Este filtro se diseñó en Matlab y fue exportado como un archivo *header* de lenguaje C.

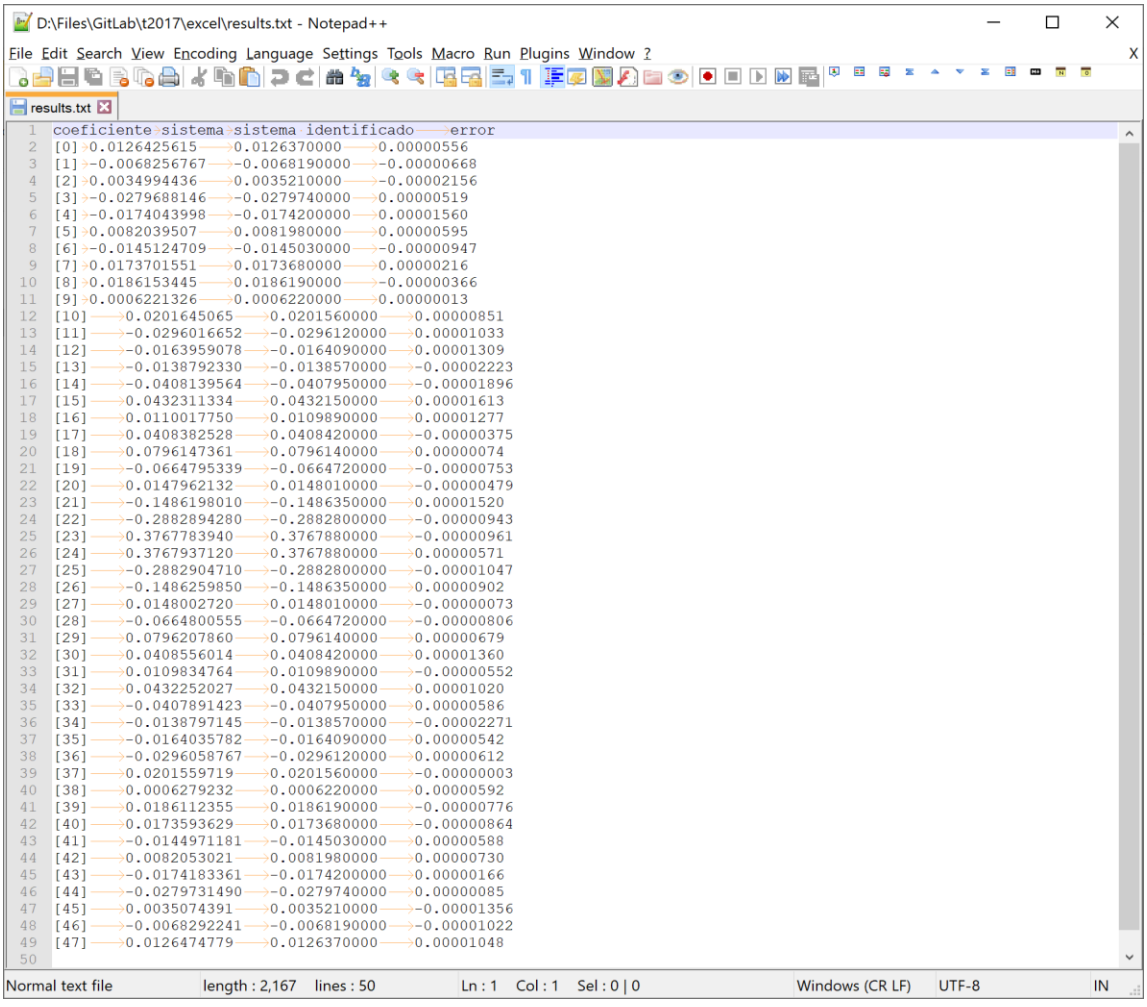
Figura 28. **Coeficientes del filtro FIR generado en Matlab que representa el sistema reverberante a identificar**



Fuente: elaboración propia, empleando Matlab R2015b.

Después de correr el procesador digital dando como entrada ruido blanco y pasando esta señal sobre el filtro diseñado en Matlab que representa el ambiente reverberante, se ve que el algoritmo identificó correctamente los coeficientes del filtro dada por la salida del archivo “results.txt” generada desde la información generada en el dispotivo DSP.

Figura 29. **Resultados del algoritmo LMS para identificación de sistemas**

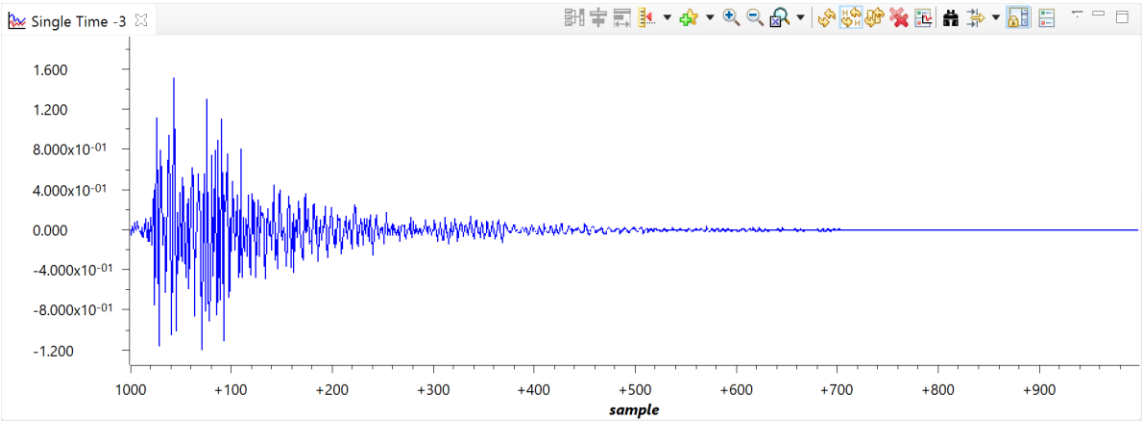


Fuente: elaboración propia, empleando Code Composer Studio 2017.

Se ven ciertos coeficientes que fueron identificados con cierto error, pero la mayoría de los coeficientes fueron identificados correctamente.

La siguiente gráfica muestra cómo el error del algoritmo se va reduciendo mientras más muestras se van procesando.

Figura 30. **Señal de error de salida del sistema desconocido respecto al sistema identificado**



Fuente: elaboración propia, empleando Code Composer Studio 2017.

En la siguiente sección se verán los factores que inciden en la eficiencia del filtro adaptativo ya que no siempre es posible identificar al sistema.

4.4. Evaluación del desempeño de un filtro adaptativo para identificación de sistemas

En esa sección se identificarán los factores que afectan el rendimiento de un filtro adaptativo con relación a la identificación de un sistema digital. Para poder llevarlo a cabo con facilidad, se replicó el algoritmo LMS en hojas de cálculo

y así poder tener control sobre tanto de las señales a ser identificadas como alteraciones a las señales involucradas, inclusión de ruido y cambio en el factor de convergencia.

En este ejemplo se analiza un filtro desconocido con 5 coeficientes, es decir, K=5.

$$z(n) = \sum_{k=0}^{K-1} u_k \, x(n-l)$$

Con coeficientes conocidos y arbitrariamente elegidos como $u_0 = 0{,}5$, $u_0 = 1{,}33$, $u_2 = 0{,}25$, $u_3 = 0{,}2$, $u_4 = 0{,}17$

El filtro adaptativo también es modelado como un filtro FIR.

$$y(n) = \sum_{k=0}^{K-1} a_k \, (n) x(n-l)$$

Con todos los coeficientes inicialmente en cero y el factor de convergencia elegido arbitrariamente para ser inicialmente μ = 0,1.

La señal de entrada es ruido blanco que representa una señal de audio $x(n)$, como se muestra en las siguientes dos figuras.

A esta hoja de cálculo se le pueden hacer 4 tipos de modificación al cambiar el campo amplitud (0 para remover el efecto):

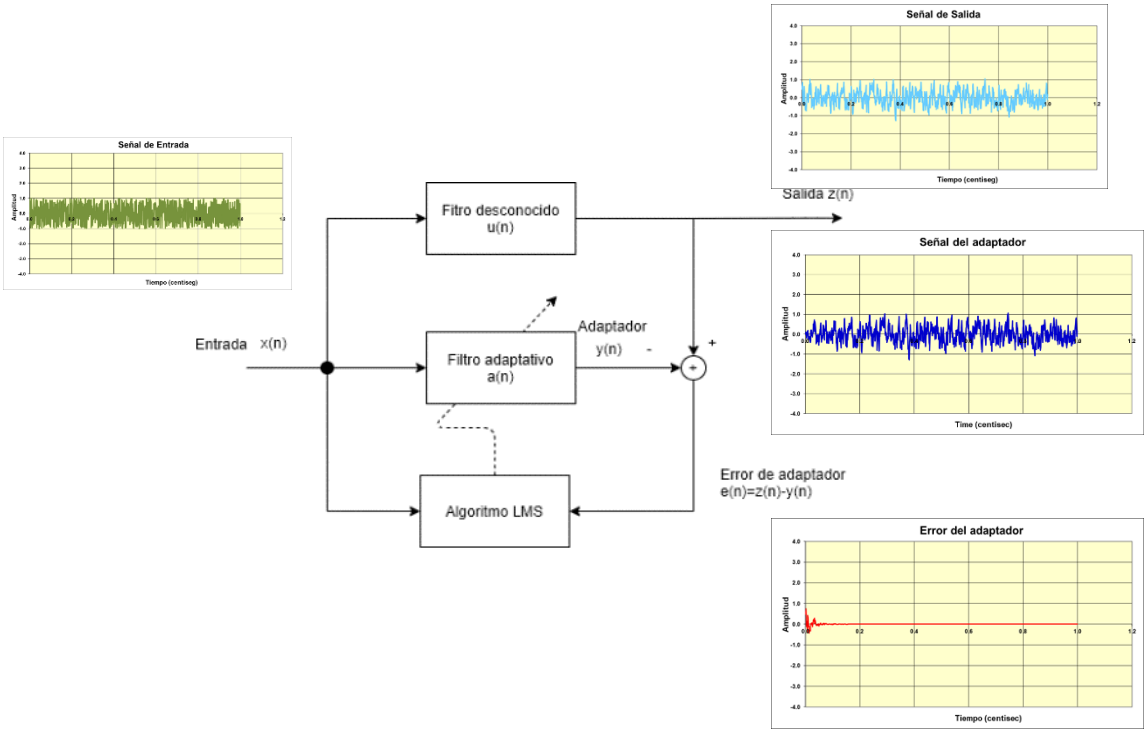
- Agregar 2 tonos a la señal que funciona como señal portadora
- Agregar ruido no correlacionado a la señal de entrada
- Cambiar el factor de convergencia
- Reducir el número de factores en el filtro adaptativo

Figura 31. **Parámetros de las señales utilizadas para ilustrar los factores que afectan el rendimiento de un filtro adaptativo para identificar sistemas**

		Time Units = centisec		Freq Units = Hecto-Hz		
		SEÑAL DE ENTRADA		Tonos = 6.970 and 16.330 HectoHz		
		Tono 1: $x_1(n)$	Tono 2: $x_2(n)$	ial de vox: u(n)	Ruido: v(n)	Señal x(n)
Δt =	0.001	$A_1\sin(w_1t)$	$A_2\sin(w_2t)$	A_3rand()	A_4rand()	x_1+x_2+u+v
	Amplitud=	0	0	1	0	1
	f =	6.970	16.330			
	ω = 2πf =	43.794	102.604			
	T = 1/f =	0.143	0.061			
	Samples in T=	143	61			
	R (ohm)=	1	1	1	1	1
	I promedio (amps)=	0.000	0.000	0.001	0.000	0.001
	Desviación Estándar. =	0.000	0.000	0.582	0.000	0.582
	Potencia I^2R (watts) =	0.000	0.000	0.338	0.000	0.338

Fuente: elaboración propia.

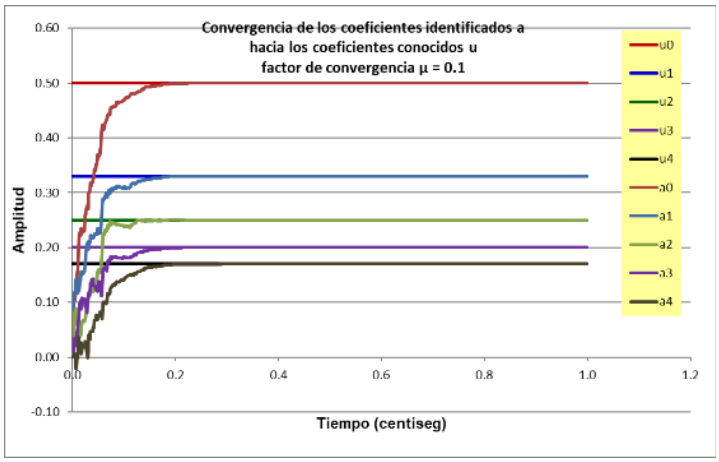
Figura 32. **Diagrama de bloques del filtro adaptativo utilizado para identificación de sistemas en el escenario base**



Fuente: elaboración propia, empleando Draw.io 2000.

En el escenario base, sin ningún cambio a la hoja de cálculo, la siguiente figura muestra que el sistema fue identificado en aproximadamente 10 milisegundos:

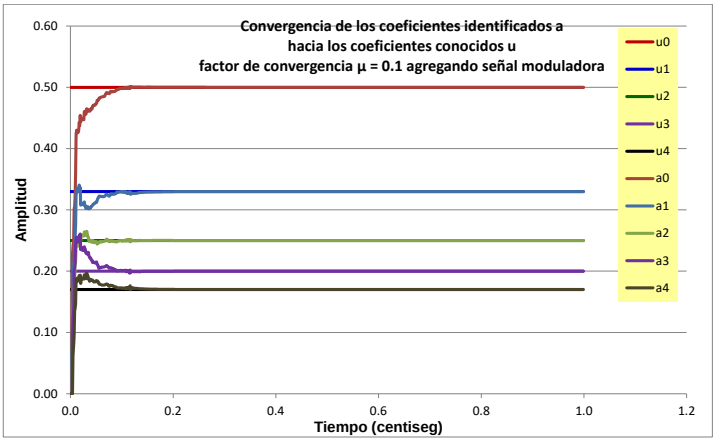
Figura 33. **Convergencia de los factores A a los factores U en el escenario base**



Fuente: elaboración propia.

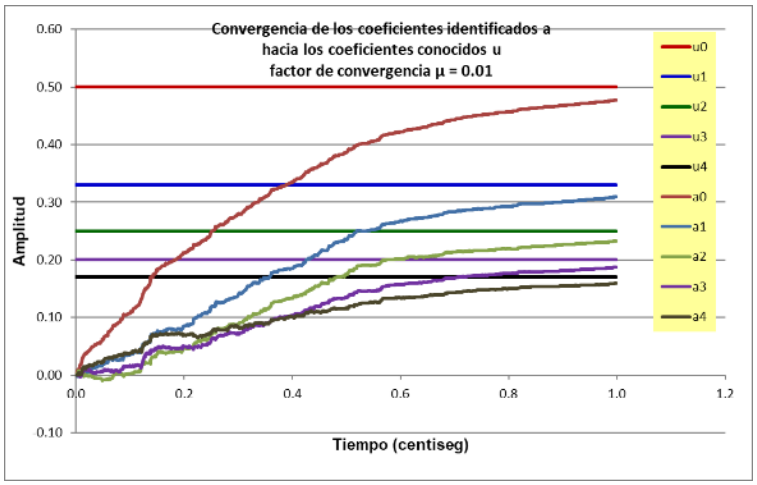
En las siguientes gráficas se ve el efecto de realizar cambios al escenario base.

Figura 34. **Efecto en la convergencia de agregar una señal moduladora**



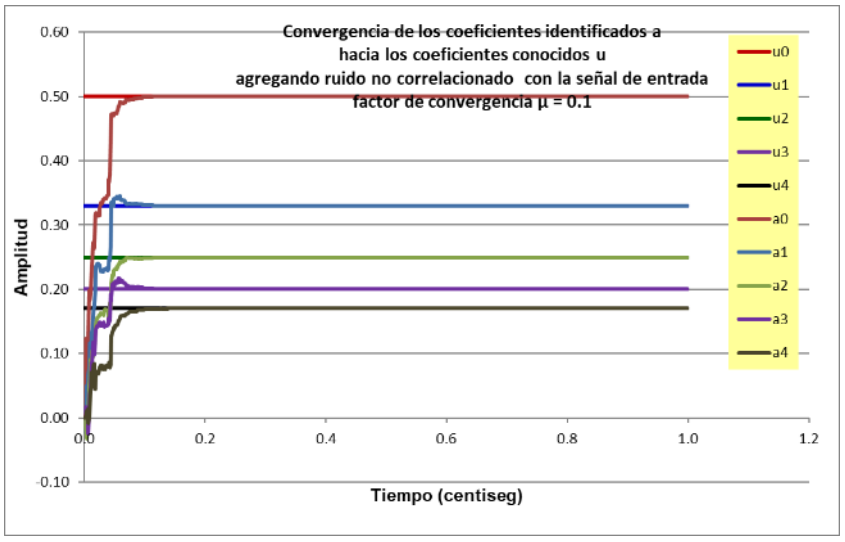
Fuente: elaboración propia.

Figura 35. **Efecto en la convergencia de reducir el factor de convergencia a 0.01**



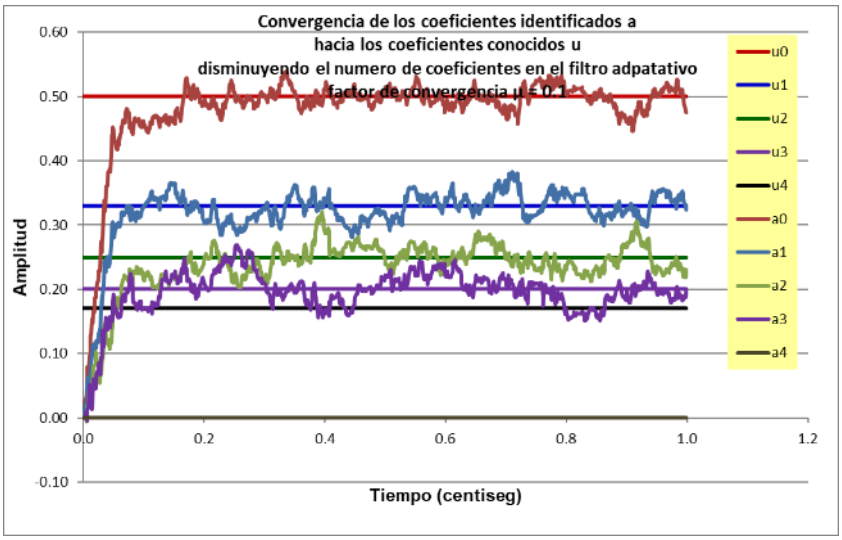
Fuente: elaboración propia.

Figura 36. **Efecto en la convergencia de agregar ruido no correlacionado a la señal de entrada**



Fuente: elaboración propia.

Figura 37. **Efecto en la convergencia de reducir el número de coeficientes del filtro adaptativo**



Fuente: elaboración propia.

De la observación de cada escenario generado a partir del escenario base se puede concluir que:

- Agregar una señal portadora a un filtro adaptativo, en este caso, no cambia significativamente el tiempo de convergencia.

- Reducir el factor de convergencia en el algoritmo LMS incrementa sustancialmente el tiempo de convergencia y por tanto el tiempo que toma identificar el sistema se incrementa.

- Agregar un ruido no correlacionado a la señal de entrada puede reducir el tiempo de convergencia debido a que la señal de error puede influenciar más en las iteraciones del algoritmo LMS.

- La reducción del número de coeficientes puede afectar drásticamente la convergencia de los factores, por tanto, se debe tener al menos el mismo nivel de coeficientes en el filtro adaptativo FIR que los coeficientes que tiene el sistema FIR a identificar.

CONCLUSIONES

1. Una de las restricciones principales del procesamiento en tiempo real es que el algoritmo de procesamiento digital pueda ejecutarse entre muestra y muestra por lo que la elección de hardware y las optimizaciones de software que se realicen son muy importantes para lograr un procesamiento efectivo.

2. Un filtro FIR tiene la ventaja sobre un filtro IIR de que es siempre es estable, sin embargo, para un problema de identificación de sistemas al utilizar un filtro FIR es muy importante conocer el máximo número de coeficientes a utilizar, esto es difícil de determinar a priori en una implementación. Si no se determina el número de coeficientes FIR en al menos el mismo número de coeficientes que el sistema a identificar, es muy probable que el algoritmo LMS no converja.

3. El desempeño del filtro adaptativo aplicado a la identificación de sistemas depende, entre otros factores, del coeficiente de convergencia, del número de coeficientes del filtro digital y de la cantidad de ruido existente en la señal deseada, por lo cual estos elementos deben tomarse en cuenta al calibrar el algoritmo LMS de identificación de sistemas.

RECOMENDACIONES

1. Para la implementación de un filtro digital en tiempo real se aconseja tomar muy en cuenta el tiempo de procesamiento de todo el sistema para que se pueda procesar la señal entre muestra y muestra.

2. Si bien un sistema DSP en tiempo real puede utilizarse para implementar algoritmos aprendizaje automático o *machine learning*, como lo es el algoritmo LMS aplicado a la identificación de sistemas, hay técnicas que, con el poder de cómputo disponible hoy en día, permiten el aprendizaje automático sin conocimiento a priori del sistema. Estas técnicas involucran redes neuronales de múltiples niveles que encuentran su aplicación en tecnologías actuales como lo son los automóviles autodirigidos fabricados por la compañía Tesla. Es aconsejable, para un trabajo posterior, investigar otras áreas de aplicación de aprendizaje automático utilizando redes neuronales aplicado a la identificación de sistemas o a otra problemática donde un conocimiento a priori de un sistema no es posible debido al constante cambio de las variables involucradas.

BIBLIOGRAFÍA

1. BRACEWELL, Ronald. *The fourier transform and its applications*. 3a ed. EEUU: McGraw-Hill, 2000. 618 p.

2. BRUNEAU, Michel. *Fundamentals of acoustics.* Londres, Inglaterra: ISTE Ltd., 2006. 636 p.

3. CHASSAING, Rulph. *Digital signal processing and applications with the C6713 and C6416 DSK*. Nueva York, EEUU: Wiley, 2005. 518 p.

4. CLAUDE, Shannon. *A mathematical theory of communication.* Vol. 27. EEUU: The Bell System Technical Journal, 1948. 55 p.

5. COOLEY, James; TUKEY, John. Mathematics of computation. *An algorithm for the machine calculation of complex fourier series.* Vol. 19. No. 90, EEUU: American Mathematical Society, 1965. 6 p.

6. ECE 2610 Signals and Systems. *Direct form 1 structure*. [en línea]. <http://www.eas.uccs.edu/~mwickert/ece2610/lecture_notes/ece2610_chap8.pdf>. [Consulta: junio de 2018].

7. EMBREE, Paul. C *Algorithms for Real-Time DSP*. New Jersey, EEUU: Prentice Hall PTR, 1995. 245 p.

8. Functional Block Diagram for OMAP-L138. *Datasheet.* [en línea]. <https://www.ti.com/datasheets/diagram.tsp?genericPartNumber=OMAP-L138&diagramId=SPRS586J>. [Consulta: junio de 2018].

9. HAYES, Monson. *Schaum's outline of theory and problems of digital signal processing*. EEUU: McGraw-Hill, 1999. 432 p.

10. JONT, Allen; BERKELEY, David. Allen. *Image method for efficiently simulating small-room acoustics*. New Jersey: Acoustics Research Department, Bell Laboratories, Murray Hill, 1978. 8 p.

11. KEHTARNAVAZ, Nasser; SIMSEK, Burc. *C6x-Based digital signal processing*. Upper Saddle River, NJ, EEUU: Prentice Hall, 2000. 164 p.

12. KERNIGHAN, Brian; RITCHIE, Dennis. *The C programming language*. 2a ed. Englewood Cliffs, EEUU: Prentice Hall, 1988. 272 p.

13. LYONS, Richard. *Understanding digital signal processing*. Englewood Cliffs, EEUU: Prentice Hall, 2011. 954 p.

14. MORENO BILBAO, Asunción; RODRÍGUEZ FONOLLOSA, José Adrián; VALLVERDÚ BAYÉS, Francesc; MARIÑO ACEBAL, José Bernardo. *Tratamiento digital de la señal*. Una introducción experimental. 2a ed. Barcelona, España: Alfaomega, 1999. 400 p.

15. OPPENHEIM, Alan; SCHAFER, Ronald; BUCK, John. *Discrete-Time signal processing.* 2a ed. Englewood Cliffs, EEUU: Prentice Hall, 1999. 888 p.

16. PROAKIS, John; INGLE, Vinay. Digital *signal processing using matlab V.4.* Boston, EEUU: PWS Publishing Company, 1997. 420 p.

17. SANJIT KUMAR, Mitra. *Digital signal processing laboratory using matlab*. Boston, EEUU: WCB McGraw-Hill, 1999. 230 p.

18. _____. *Digital signal processing: a computer-based approach.* 2a ed. Boston, EEUU: McGraw Hill, 2001. 866 p.

19. SEN, Kuo. *Real-Time Digital signal processing implementations and applications*. 2a ed. Nueva York, EEUU: Wiley, 2006. 646 p.

20. SMITH, Julius. *Introduction to digital filters with audio applications*. EEUU: W3K Publishing, 2007. 460 p.

21. System Identification Toolbox. *Numeric Models*. EEUU: The Mathworks, 2015. 968 p.

22. TAUB, Herbert; SCHILLING, Donald. *Principles of communication systems.* 2a ed. Nueva York, EEUU: McGraw-Hill, 1986. 759 p.

23. TLV320AIC3106 Low-Power Stereo Audio CODEC for Portable Audio/Telephony. *Description*. Dallas, TX: Texas Instruments, 2014. 105 p.

24. TMS320C6000 Assembly language tools. User's guide, SPRU186K. *Software Development Tools Overview.* Dallas TX: Texas Instruments, 2002. 399 p.

25. TMS320C6000 CPU and instruction set SPRU189F. *TMS320C62x/C64x/C67x Architecture.* Dallas, TX: Texas Instruments, 2000. 685 p.

26. TMS320C6000 Optimizing C Compiler User's Guide, SPRU187U. *Run-Time environment*. Dallas TX: Texas Instruments, 2002. 278 p.

27. TMS320C6000 Peripherals, SPRU190D. *TMS320C620x/C670x Internal program and data memory.* Dallas TX: Texas Instruments, 2001. 828 p.

28. TMS320C6000 Programmer's Guide, SPRU198G. *Optimizing C/C++ code*. Dallas TX: Texas Instruments, 2002. 382 p.

29. WELCH Thad; WRIGHT, Cameron; MORROW, Michael*. Real-Time digital signal processing from matlab to C with the TMS320C6x DSPs.* 3a ed. EEUU: CRC Press, 2017. 480 p.