UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE METODOLOGÍAS DE RECONSTRUCCIÓN DE SEÑALES Y CONTROL DE MEDICIONES EN SISTEMAS EMBEBIDOS DE MEDICIÓN DIGITAL

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JUAN MOISES MAURICIO VILLANUEVA

PROMOCIÓN
2002 – I

LIMA – PERÚ
2006

IMPLEMENTACIÓN DE METODOLOGIAS DE RECONSTRUCCIÓN DE SEÑALES Y CONTROL DE MEDICIONES EN SISTEMAS EMBEBIDOS DE MEDICIÓN DIGITAL

A Dios: sobre todas las cosas.
A mi madre: Ruth Villanueva
A mis hermanos: Milagros, Moises y Carlos

SUMARIO

Actualmente sistemas de medición digital son cada vez más complejos en su arquitectura, compuestos por módulos de sensores, conversión analógica digital, módulos de reconstrucción de señales para eliminar las no linealidades de los sensores y módulos de control en sistemas en la cual se puede actuar sobre el ambiente de forma a controlar las señales de interferencia, obteniendo una mejora en la calidad de medición, modificando la dinámica de medición disminuyendo el tiempo de respuesta de estimación de la medición.

El modulo de reconstrucción de señales usualmente requiere de la implementación de la función inversa (o inversa aproximada) de la función no lineal del sensor. Esas funciones no lineales no pueden ser implementadas directamente debido a las restricciones de cálculo en punto fijo y resolución limitada, característica de arquitectura del procesador embebido de bajo costo, empleados en diversas aplicaciones de medición.

Para resolver el problema del modulo de reconstrucción de señales, se presenta un procedimiento para determinación de aproximación linear por partes de funciones no lineales para sistemas embebidos de bajo costo. Para esto, se desarrollo un algoritmo genético que determinara a posición y número de puntos de quiebra y tamaño mínimo de la tabla de equivalencia (LUT – *Look-Up Table*), para almacenar estos puntos de quiebra dentro de un procesador digital. La función no lineal puede entonces ser aproximada por funciones lineales a partir de los valores de los puntos de quiebra encontrados. El algoritmo desarrollado es probado e implementado en un Procesador Digital de Señales (DSP56811) de punto fijo, para dos casos de estudio: aproximación de la función seno en el primer cuadrante y reconstrucciones de medida de temperatura de un sensor termistor.

ÍNDICE

PROLOGO

Este trabajo esta basado en el estudio y análisis de sistemas de medición digital, según su forma de medición directa o indirecta, implementando metodologías de reconstrucción digital de señales provenientes de sensores con características no lineales. Así también, son estudiados sistemas de medición según su actuación en el ambiente de medición, como sistemas sin realimentación y con realimentación, para este ultimo caso, es utilizado un bloque de control con la finalidad de disminuir las constantes de tiempo de medición y mejorar la calidad de la señal medida. Las estructuras de medición presentadas son las mas generales posibles con la finalidad de que sean empleadas en sistemas embebidos de medición digital.

El bloque de control utilizado en este tipo de sistemas de medición es aplicado en sistemas en la cual es posible la actuación en el ambiente de medición. El modulo de reconstrucción, requiere implementar la función inversa (o inversa aproximada) del sensor, para dar solución a este problema se desarrolla e implementa un algoritmo genético para determinar funciones no lineales por funciones lineales por partes, con el objetivo de encontrar el mínimo número de puntos de quiebra y sus valores para que sean almacenados en una tabla de equivalencia (LUT – *Look-Up Table*) y de esta manera obtener los valores de la función aproximada por funciones lineales por partes.

Para la implementación del bloque de reconstrucción de señales, es necesario el análisis y estudio de las incertezas y propagación de errores que deben ser considerados en los sistemas embebidos de resolución limitada, con la finalidad de determinar los limites superior e inferior de errores para determinar las funciones aproximadas implementadas en sistemas embebidos.

El análisis realizado es basado en sistemas de medición que utilizan básicamente sistemas embebidos de bajo costo como procesadores de resolución limitada (DSP de punto fijo), Microcontroladores, ASICs (circuitos integrados de aplicación específica), etc.

Este trabajo está dividido en seis capítulos, referencias bibliográficas y un apéndice.

- En el Capitulo 1, se presentan aspectos generales de forma a presentar las definiciones y conceptos básicos, tales como: sistemas de medición, clasificaciones de sistemas de medida, sensores, transductores, y sistemas embebidos de tiempo real, a ser utilizados durante la realización del trabajo.
- En el Capitulo2, se presenta las definiciones de una estructura general de un sistema de medición, considerando sistemas que actúan en el medio de medición, como sistema con realimentación, así como los componente que constituyen el sistema de medición, tales

como: sensores, modulo de acondicionamiento de señales, conversores analógico-digital (AD) y conversores digital – analógico (DA).

- En el Capitulo 3, se presenta las definiciones para el análisis e implementación del bloque de control y reconstrucción de señales en un sistema de medición. Se presenta un procedimiento para el diseño del controlador con la finalidad de disminuir las constantes de tiempo de medición. Así también, es presentado un procedimiento para la implementación del bloque de reconstrucción utilizando aproximación de funciones no lineales por funciones lineales por partes.
- En el Capitulo 4, se presentan definiciones sobre la implementación de tablas de equivalencia (LUT – *Look-Up Table*) para aproximación de funciones no lineales, utilizadas para la implementación del bloque de reconstrucción de señales. Se realiza un análisis de incertezas y propagación de errores para aproximar funciones no lineales considerando su implementación en sistemas embebidos de resolución limitada y de bajo costo. Se analiza el caso de estudio de la aproximación de la función seno en el primer cuadrante.
- En el Capitulo 5, se presenta un procedimiento para la determinación de aproximación de funciones no lineales por funciones lineales por partes utilizando tablas de equivalencia (LUT). Para resolver este problema es utilizado un algoritmo genético, siendo formulado de manera adecuada para dar solución al problema de aproximación de funciones.
- En el Capitulo 6, son presentadas dos aplicaciones de las definiciones y conceptos presentados en los capítulos anteriores. La primera, consiste en la implementación de la función seno aproximada en el primer cuadrante por funciones lineales por partes. La segunda aplicación, consiste en reconstrucción de la medida de medición de temperatura de un sensor no lineal termistor. Ambas aplicaciones fueron implementadas utilizando el Procesador Digital de Señales DSP56811 de punto fijo.

Finalmente, agradezco a todas aquellas personas que directa o indirectamente, contribuyeron para la elaboración de este trabajo, y de modo especial:

A mi madre Ruth Villanueva y mis hermanos, Milagros, Moises y Carlos, por su gran apoyo y cariño que siempre me han dado.

Al Dr. Jorge Del Carpio Salinas director del Instituto de Investigación de la Facultad Eléctrica y Electrónica, por su paciencia y confianza depositada en mi persona y por la oportunidad de poder realizar este trabajo, y sobre todo por la amistad.

A Shirley Guillen y a los amigos del Instituto de Investigación de la Facultad Eléctrica y Electrónica que sin su apoyo no hubiera sido posible realizar este trabajo.

A Dios por no dejarme desistir, delante de las dificultades.

CAPITULO I
ASPECTOS GENERALES

1.1 Sistemas en Tiempo Real

Los sistemas de tiempo real están presentes en un gran número de aplicaciones del mundo actual, tales como: un sistema de control de vuelo de un avión, un sistema de procesamiento de señales de un satélite de comunicaciones, un sistema de procesamiento de audio y vídeo en un equipo de televisión digital, etc. Una de las definiciones de un sistema en tiempo real es aquella cuyo correcto funcionamiento depende no solo de la corrección lógica de los resultados que produce, sino también de que los resultados sean producidos en un plazo de tiempo acotado y determinado. Por ejemplo, un sistema de control de temperatura de una planta nuclear, funcionará adecuadamente no solo si es capaz de detectar los valores de temperatura de los reactores y de tomar las decisiones adecuadas ante estos valores, sino que debe ser capaz de hacer todo esto dentro de un plazo de tiempo determinado. Caso contrario, su funcionamiento no será correcto y además puede tener consecuencias intolerables como el costo de vidas humanas y/o perdidas económicas ocasionadas a la empresa.

En los últimos años los sistemas de tiempo real se están haciendo cada vez más complejos a la vez que controlan un mayor rango de aplicaciones de la vida moderna. Algunos aspectos importantes de estos sistemas (que añaden complejidad a su desarrollo) son:

- *Distribución*: requerida para controlar y comunicarse con componentes remotos
- *Tolerancia a fallas*: requerida cuando la aplicación que se controla es crítica.

1.2. Sistemas Embebidos

Numerosos sistemas en tiempo real están compuestos por sistemas embebidos que son distintos módulos residentes en diversos procesadores y que necesitan estar comunicados continuamente con el ambiente de operación por medio de sensores, actuadores y funciones de control digital para llevar a cabo un fin común. Las características más frecuentes encontradas en estos sistemas embebidos son:

a) *Funcionalidad específica*: un sistema embebido usualmente ejecuta únicamente un programa, repetidamente.

b) *Restricciones de diseño*: restricciones impuestas por la necesidad de portabilidad, bajo consumo, dimensiones reducidas, desempeño y producción en gran escala (bajo costo), además de las restricciones del ambiente de operación con capacidad de procesamiento en tiempo real.

c) *Comportamiento reactivo en tiempo real*: diversos sistemas embebidos deben reaccionar continuamente a cambios en el ambiente del sistema y deben realizar ciertos cálculos en tiempo real con restricciones de tiempo y sin retardo.

Así también los sistemas embebidos, están compuestos por *hardware* y *software* con procesamiento dedicado para ejecutar sus tareas programadas. Consecuentemente, diversas soluciones de sistemas embebidos son implementadas en microprocesadores o microcontroladores con cierto poder computacional, en la cual los cálculos pueden ser realizados en punto flotante (*float-point*). Sin embargo, diversas aplicaciones embebidas requieren de uso de *hardware* en punto fijo (*fixed-point*). Esto puede ser justificado por motivos tales como reducción de costo, complejidad y aumento de la velocidad de procesamiento. Así los sistemas embebidos basados en *hardware* de bajo costo pueden ser compuestos de: microcontroladores, FPGA (*Field Programmable Gate Array*), DSP (Procesador Digital de Señales) de punto fijo o ASIC (Circuitos Integrados de Aplicación Especifica), en que generalmente los cálculos son realizados en punto fijo y con resolución limitada.

Una particular clase de procesadores digitales es un ASIP ("*Application Specific Instruction Set Processor*"), diseñado para una particular clase de aplicaciones con características comunes, tales como procesamiento digital de señales, control embebido, etc. El diseño de tales procesadores permite optimizar una gran variedad de aplicaciones, adicionando unidades de funcionalidad específica para realizar operaciones comunes, y eliminar otras unidades que no son usadas frecuentemente. Sistemas embebidos basados en ASIP se benefician de flexibilidad, buen desempeño de operación, potencia y tamaño. Procesadores Digitales de Señales (DSP) es una clase común de ASIP, diseñados para realizar operaciones en señales digitales, los cuales son codificados digitalmente a partir de señales analógicas a partir de un proceso de muestreo y adquisición. Entre las operaciones digitales más comunes están: filtrado de señales, transformaciones, o combinación de señales. Tales operaciones requieren de una evaluación matemática intensiva, incluyendo operaciones de multiplicación, adición, desplazamientos binarios, etc. Para soportar tales operaciones, un DSP presenta en su arquitectura una unidad especial llamada: "Unidad Multiplicador Acumulador". Debido a que los DSP manipulan frecuentemente una gran cantidad de datos, un DSP está constituido por un *hardware* especial para localización de memoria secuencial en paralelo con otras operaciones, aumentando la velocidad de ejecución.

1.3. Sistemas de Medición

En diversas áreas de ciencia y tecnología, mediciones son utilizadas para obtener informaciones sobre mediciones físicas, para monitoreamiento, regulación o control de procesos industriales, para evaluación experimental de modelos teóricos, entre otros. En estas mediciones se puede obtener informaciones importantes para el control automático de procesos y tomar decisiones con la finalidad de garantizar la calidad del producto final. Tales sistemas son constituidos

básicamente de sensores, transductores, circuitos de acondicionamiento, tales como amplificadores, filtros, módulos para presentación (*display*), almacenamiento (memoria), procesamiento y/o control a partir de los datos adquiridos.

Con el avance de la tecnología de procesadores digitales fue posible la utilización de herramientas para el tratamiento digital de señales, que antes no existían o requerían un alto costo computacional. Para esto se requiere que las señales analógicas sean acondicionadas y digitalizadas para que sean conectadas al sistema de adquisición digital de datos. Con el surgimiento de estas nuevas tecnologías y herramientas, fue necesaria la unificación entre los sistemas de medición y reconstrucción de señales, teniendo como finalidad un mejor entendimiento del proceso de medición, sus efectos e interferencias para obtener mejores resultados.

Actualmente los sistemas de medición integran parcial o totalmente los sensores, bloques de acondicionamiento, conversores analógico – digital (A/D) y unidades de procesamiento digital, siendo una tendencia actual en sistemas de instrumentación, que tiene como ventaja la disminución del tamaño del producto final, reducción de conexiones, mayor confiabilidad, reducción de costos, entre otras.

Así también los sistemas de medición pueden poseer una inteligencia asociada, que no es visible al usuario, estos sistemas son llamados de sensores inteligentes. Esa inteligencia puede ser disponible en la parte analógica o en la parte digital con la utilización de Microcontroladores, Procesadores Digital de Señales (DSP), o circuitos integrados para aplicaciones especificas (ASIC).

1.4. Clasificación de Sistemas de Medición

Los sistemas de medición pueden ser clasificados en dos grupos: según la disponibilidad de la señal de interés el sistema de medición puede ser directa o indirecta y según la actuación en el medio de medición (ambiente), utilizando un lazo de realimentación y control.

1.4.1. Medición según la Disponibilidad de la Señal

En la *medición directa*, la señal proveniente del sensor que se encuentra disponible, representa directamente la medida principal. Esta señal puede ser acondicionado utilizando circuitos analógicos, y ser utilizados por sistemas de medición digital, muestreando y convirtiéndola para la forma digital utilizando un conversor analógico – digital (A/D).

En la *medición indirecta*, la señal principal no es directamente convertida en señal eléctrica por el sensor, pero actúa sobre otra señal secundaria, que puede ser medida acondicionada y convertida para la forma digital y sus valores utilizados para estimar la señal de interés de forma digital.

1.4.2. Medición según la Actuación en el Medio

- *Sistemas de medición sin realimentación*, no actúan en el medio de medición sin modificar la señal principal o la señal secundaria.

- *Sistemas de medición con realimentación*, al contrario, actúan en el medio de medición utilizando una malla de realimentación y control, de forma a modificar los valores de la señal secundaria.

Ejemplos de sistemas con realimentación son instrumentos que utilizan el método de oposición, el método de instrumento nulo, para medición. En estos sistemas, se ejerce una actuación de forma opuesta al efecto de la señal de interés en el medio de medición. La actuación es aplicada al medio de medición por una malla de realimentación y es controlada hasta que se obtenga un balanceamiento. Los valores de la señal principal son obtenidos directamente a partir de los valores de la señal aplicada al medio de medición.

1.5. Sensores y Transductores

Un sensor sugiere la conexión con los sentidos humano y nos puede proveer informaciones de señales físicas y químicas que de otra forma, no podrían ser percibidos o cuantificados por nuestro sentido. Un transductor es un dispositivo que convierte energía de un dominio para otro, ajustado para minimizar los errores en el proceso de conversión y un sensor es un dispositivo que entrega una señal útil en función de determinado objeto de medición. El sensor es un elemento básico de un transductor, mas puede también referirse a la detección de tensión o corriente en un régimen eléctrico que no requiere conversión.

Los sensores pueden ser clasificados según su característica de transferencia de la señal de interés de entrada con la señal eléctrica generada en lineales o no lineales. En el caso de sensores no lineales, el efecto de la no linealidad en la señal eléctrica debe ser compensado (reconstrucción de señales) por el sistema de medición.

1.6. Errores en Sistemas de Medición

Existen varias fuentes de errores en un sistema de medición, tales como: ruidos, perturbaciones, errores numéricos, etc. Se definen tres categorías de errores: errores groseros, errores sistemáticos y errores aleatorios.

- **Errores groseros**: en gran parte cometido por errores humanos, con lecturas incorrectas, ajustes y aplicaciones incorrectas de instrumentos y errores computacionales.
- **Errores sistemáticos**: es una cantidad fija y constante, la cual permanece igual para todas las observaciones en un experimento, o sea, influenciara los datos por una cantidad constante. Cuando el origen es conocido, el error puede ser corregido. Los errores sistemáticos tienen origen en fallas de los instrumentos, como aquellas debidas a componentes defectuosos o desgastados y efectos ambientales sobre el instrumento o usuario. El error sistemático puede ser corregido con un nuevo ajuste del instrumento.
- **Error aleatorio**: es una cantidad estadística y muestra la dispersión de los datos. Ellos son provocados por fenómenos que no pueden ser identificados o controlados por ser de naturaleza aleatoria.

1.7. Conceptos y Definiciones en Sistemas de Medición

Algunas veces definiciones tales como exactitud y precisión, son confundidas y usadas con el mismo significado. Para aclarar los conceptos utilizados, algunas definiciones son presentadas a continuación:

- **Exactitud**: es la medida del grado de concordancia entre la indicación de un instrumento y el valor verdadero de la variable sobre medida.
- **Precisión**: es la medida del grado de reproducibilidad de la medida, es decir, para un determinado valor de la variable, la precisión es la medida del grado de alejamiento entre varias medidas sucesivas.
- **Sensibilidad**: es la relación entre la variación de la señal de salida y la variación del valor de la señal sobre medida (ejemplo: Relación de la variación de la señal de salida de un puente *Wheatstone* y la variación de temperatura).
- **Resolución**: es la menor variación de la señal sobre medida que puede ser indicada por el instrumento.
- **Error**: es la medida del desvió entre el valor medido y el valor verdadero.
- **Calibración y Sensibilidad**: La relación entre la variable de entrada de la medida física y la salida de la variable de señal para un sensor específico es conocida como calibración del sensor. Típicamente, un sensor (o un sistema de instrumentación completo), es calibrado para proveer un conocimiento de la entrada física para el sistema y almacenamiento de la salida. Los datos son graficados sobre una curva de calibración, como presentado en la Figura 1.1. En este ejemplo, el sensor tiene una respuesta lineal para valores de la entrada física menores que x_0. La sensibilidad del dispositivo es determinado por el alcance de la curva de calibración. En este ejemplo, para valores de la entrada física mayores que x_0, la curva de calibración se inicia con una sensibilidad menor hasta alcanzar un valor límite de la señal de salida. Este comportamiento es conocido como saturación, y el sensor no puede ser utilizado para mediciones para valores mayores a la saturación. En algunos casos, el sensor no responderá para valores pequeños de la variable de entrada física. La diferencia entre el menor y mayor valor de la variable física que puede ser confiablemente medida por un instrumento es determinado como el rango dinámico del dispositivo.

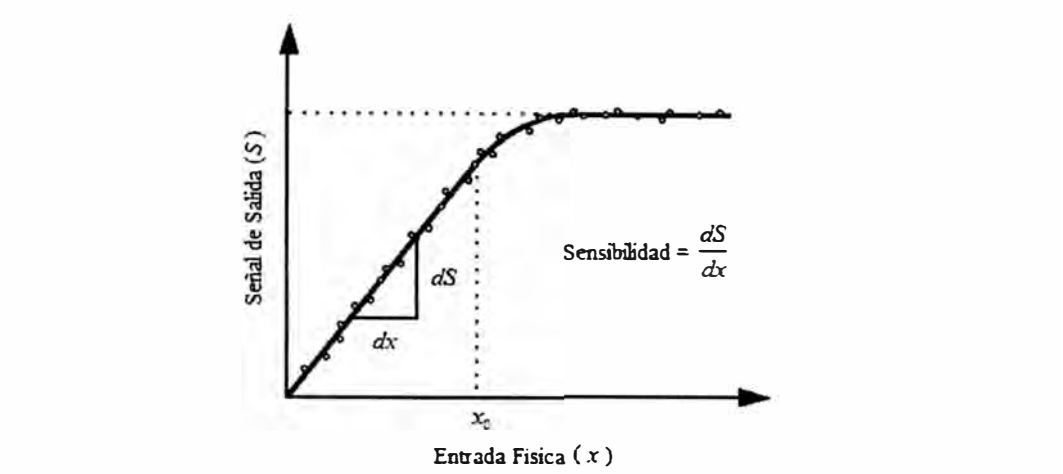


Fig. 1.1. Ejemplo de una curva de calibración.

1.8 Estándares de Medida

En la actualidad existen diversos tipos de estándares, en particular para un "estándar de medidas" aquí es definido como un "estándar de practica o estándar de protocolo" que tienen varios estándares miembros tales como

- ISO (*Internacional Organization for Standardization*)
- IEC (*International Electrotechnical Commission*)
- ANSI (*American National Standards Institute*)
- SCC (*Standard Council of Canada*).

En la Figura 1.2 se presentan los diferentes componentes de los estándares de medida.

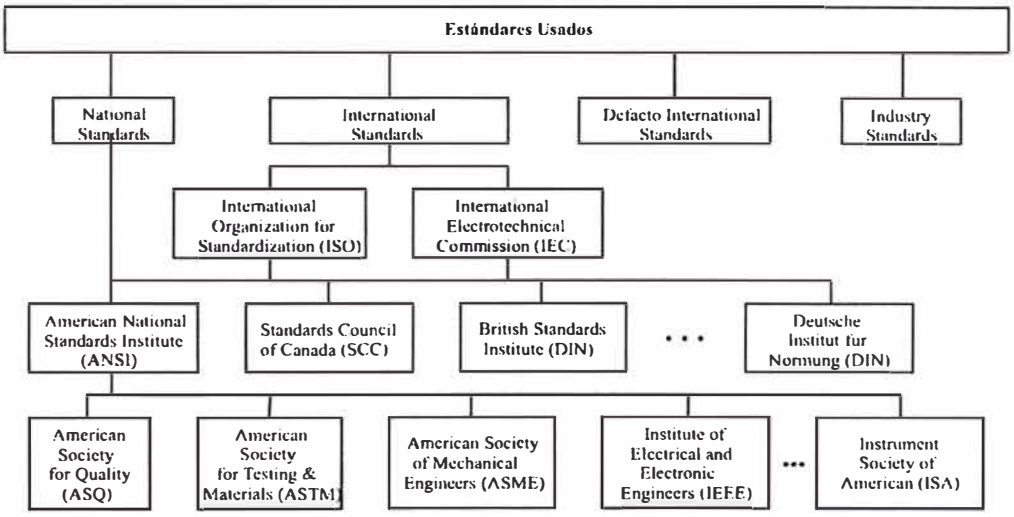


Fig. 1.2. Componentes de los estándares de medida.

Entre los estándares más utilizados están: Estándar de Práctica, Metrología Legal y Metrología Forense.

- *Estándar de Practica (Estándar de Protocolo)*, Tales estándares son definidos a través de documentación específica, describiendo el proceso particular de operación de una tarea o procedimiento. También son llamados de "Protocolos" para evitar equivocaciones con estándares físicos.

- *Metrología Legal,* se refiere a la aplicación de estándares de medida para el control de transacciones diarias en el comercio es conocida como Metrología Legal, o más comúnmente llamado de Pesos y Medidas.
- *Metrología Forense*, se refiere a la aplicación de medidas y estándares de medidas para la solución y prevención del crimen. Su práctica es realizada en laboratorios y agencias internacionales como la INTERPOL (*Internacional Police*), agencia internacional que coordina las actividades policiales de naciones miembros.

CAPITULO II
DEFINICIÓN DEL PROBLEMA

2.1. Introducción

En sistemas digitales de medición, se puede clasificar el proceso según la medición realizada, con relación a la disponibilidad de la señal eléctrica que represente la señal principal en dos tipos: directa o indirecta.

En la medición directa, la señal de interés es directamente disponible en forma de señal eléctrica en la salida del sensor. Esta señal puede ser acondicionada por un circuito analógico, seguidamente es convertido para la forma digital a través un conversor analógico – digital (A/D), enseguida la señal digital es procesada utilizando microcontroladores (MCU) o procesadores digitales de señales (DSP) de forma a estimar la señal de interés con máxima precisión, como se presenta en la Figura 2.1.

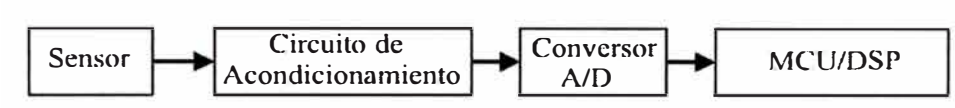


Fig. 2.1. Proceso de Medición Directa

En la medición indirecta, la señal de interés no es directamente convertida en señal eléctrica por el sensor, mas ejerce influencia sobre otra señal secundaria, que puede ser medida en forma digital y sus valores utilizados para estimar la señal principal.

La medición indirecta puede causar degradación en la calidad de la medición, siendo los principales problemas descritos a seguir:

- El tiempo de respuesta para la estimación de la señal principal, a partir de los valores de la señal secundaria, es función de las constantes de tiempo (dinámica) del medio de medición, que pueden ser grandes para algunos procesos físicos.
- La estimación de la señal principal implica la utilización de una función inversa (o inversa aproximada) de la función característica del medio de medición.

Estos problemas relacionados a la medición indirecta (sin realimentación) pueden ser reducidos utilizando una configuración de medición con realimentación. En esta configuración, el sistema de medición realiza básicamente las siguientes funciones:

- Medición y control de la señal secundaria. La medición es realizada utilizando un sensor apropiado. El control es realizado a través de un lazo de realimentación utilizando un actuador adecuado, que puede ser el propio sensor de medición;

- Estimación de la señal principal a partir de los valores de control y de los valores actuales de la señal secundaria.

Consecuentemente, el tiempo de respuesta de estimación de la medida principal puede ser disminuido debido a la modificación de la dinámica del proceso de medición. Además, cuando el sistema de medición se encuentra en estado estacionario, la función de reconstrucción de la señal de interés puede ser simplificada y se puede utilizar apenas los valores instantáneos de la señal secundaria, que generalmente permite obtener resultados de estimación de los valores de la señal de interés con un mejor acondicionamiento.

Consecuentemente se puede concluir que la utilización de un lazo de control y realimentación en la configuración de medición indirecta, se justifica siempre que se desee mejorar la calidad de la medición con respecto al tiempo de respuesta y a mejorar el acondicionamiento en la estimación de la señal principal.

2.2. Estructura General de un Sistema de Medición

En diversas referencias son presentadas varias estructuras de modelos de medición, siendo que la principal característica de estos sistemas, es la de soportar el intercambio de información entre los diversos componentes del sistema de medición. En la Figura 2.2 se presenta el modelo de un sistema de medida que relaciona los bloques de procesamiento digital de señales en el sistema de medición.

A continuación se inicia la descripción del sistema de medición presentado en la Figura 2.2. El objeto de medida es caracterizado por dos variables, una variable de medida X y una medida de interferencia que se encuentra presente en el medio definido por V. Esta medida de interferencia debe ser diferenciada de las señales de perturbación, pues en contraste con las señales de perturbación, la señal de influencia V es medida o controlada y puede ser tomada en cuenta en la evaluación final de la variable principal X.

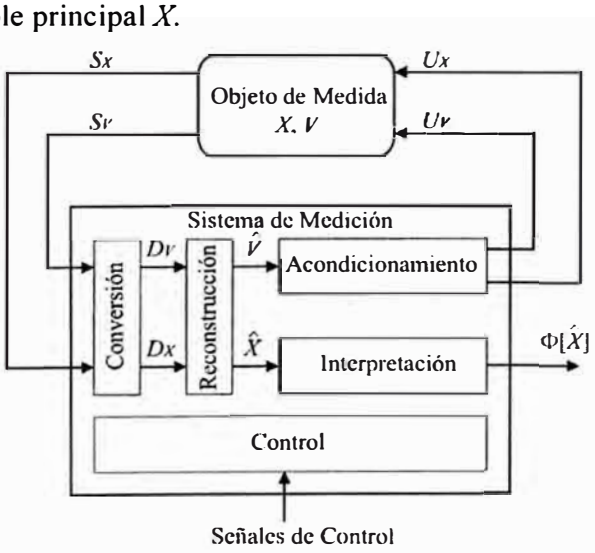


Fig. 2.2. Estructura de un Sistema de Medida.

El sistema general de medida es diseñado para proveer un valor de estimación $\hat{X}$ de la variable de medida X o en la forma más general, una función o una funcional $\Phi[\hat{X}]$ del valor estimado. Para realizar la estimación $\hat{X}$ de la variable medida X, el sistema de medición tiene como entradas las señales, S_x y S_V y como salidas las señales U_x y U_V, siendo sus principales tareas las siguientes:

- La señal S_x lleva información de la variable de medida X;
- La señal S_V lleva información de la cantidad de influencia generalizada V;
- La señal U_x excita el objeto de medida para provocar una manifestación deseable de la medida X.
- La señal U_V controla el objeto de medida para crear un estado deseable de la cantidad de influencia generalizada V;

2.3. Definiciones sobre un Sistema de Medición con Realimentación

En la Figura 2.3 se presenta un esquema general de medición con realimentación, compuesto por dos partes:

- Medio de medición
- Sistema de estimación y control.

La principal función de la parte de control es estimar la señal principal x_P, con la mayor eficiencia posible. Esa señal es correlacionada con otras medidas secundarias presentes en el medio de medición, definido por $P(.)$. El sistema de medición puede sufrir influencia de señales de interferencia, definidas por V que deben ser compensadas en la estimación. Para estimar los valores de la señal principal X, es necesario medir y controlar la señal secundaria asociada al proceso V, utilizando un sensor adecuado. Los elementos sensor, actuador y medio de medición pueden ser físicamente elementos distintos, o simplemente un elemento que realice esas tres funciones. El sistema de medición puede actuar sobre el medio de medición y controlar la señal secundaria directamente a través de la variable U. El sistema de estimación y control presentado en la Figura 2.3 puede ser dividido en tres partes:

- Bloques para la medición de la señal secundaria;
- Subsistema de control y bloque de estimación de la señal de interés;
- Bloque de medición de las variables de interferencia.

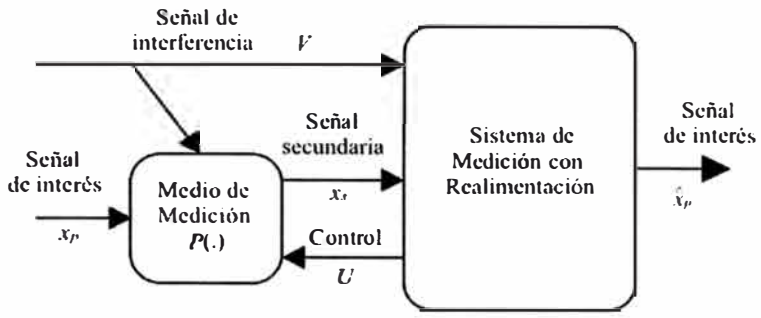


Fig. 2.3 Diagrama general de un sistema de medición con realimentación

En la Figura 2.4, se presenta el diagrama detallado del sistema de medición presentado en la Figura 2.3, inicialmente la señal secundaria x_s es convertida directamente para una señal eléctrica y por un sensor con función de transferencia $f(.)$. Esta señal eléctrica es digitalizada por un conversor analógico – digital (A/D) formando una secuencia temporal de valores $\{\hat{y}\}$. Seguidamente, la variable secundaria es estimada en el bloque de Reconstrucción Directa, utilizando la función de reconstrucción $R_D(.)$ y los valores de $\{\hat{y}\}$, considerando las variables de interferencia. La función de reconstrucción directa $R_D(.)$ debe ser idealmente la función inversa de $f(.)$.

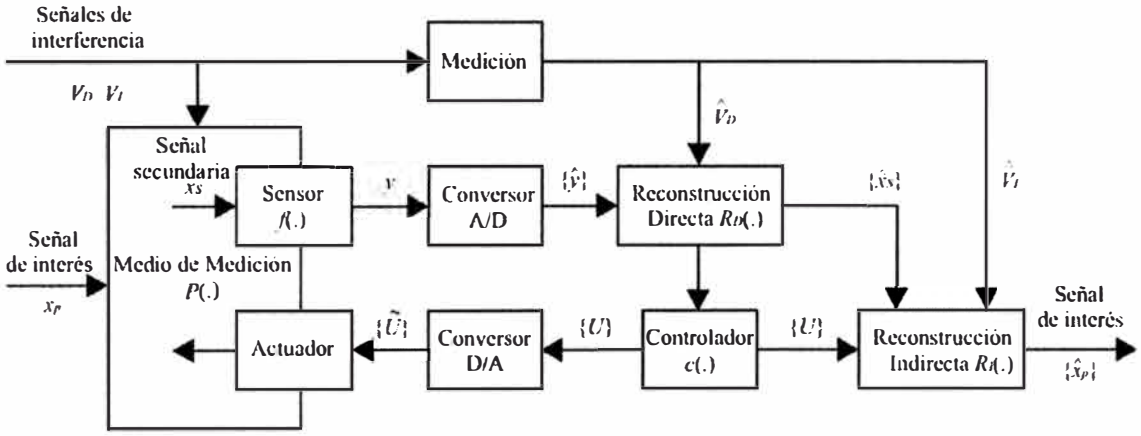


Fig. 2.4 Diagrama detallado del sistema de medición con realimentación.

El sistema de medición puede sufrir interferencia de señales, tal como temperatura del ambiente, que es generalmente el tipo de interferencia mas frecuente en sistemas de medición. En este sistema las señales son clasificadas de acuerdo con el tipo de interferencia:

- V_I para las señales que interfieren en el medio de medición
- V_D para las que interfieren en el sensor de medición de la señal secundaria, cuando este es diferente del medio de medición.

Las señales V_I y V_D deben ser medidas y compensadas en los bloques de reconstrucción descritos por $R_I(.)$ y $R_D(.)$, respectivamente.

El control de la señal secundaria se realiza a través de los bloques: Controlador, Conversor digital – analógico (D/A) y Actuador. La señal $\{U\}$ es generada por el bloque Controlador a partir

de valores de la señal secundaria, utilizando un algoritmo de control especifico. Esa señal puede ser aplicada al medio de medición en la forma analógica, utilizándose un conversor digital – analógico D/A. De esta manera, la secuencia de valores en la salida del bloque controlador es representado por $\{\tilde{U}\}$. El bloque Actuador tiene la función de aplicar esa señal al medio de medición. El bloque de Reconstrucción Indirecta $R_I(.)$ tiene la función de estimar la señal principal $\{\hat{x}_p\}$ a partir de los valores de la señal de control $\{U\}$, la variable secundaria $\{\hat{x}_s\}$, compensando cualquier interferencia de señales externas $\hat{V}_I$.

2.4. Descripción de los Componentes del Sistemas de Medición

A continuación son descritos los componentes principales del sistema de medición presentado en la Figura 2.4, tales como: características de los sensores, acondicionamiento de señales, conversores AD/DA, etc.

2.4.1. Sensores

Sensores convierten variables físicas para variables de señales. Sensores son frecuentemente transductores, siendo que los transductores son dispositivos que convierten una energía de entrada en otra forma de energía. Los sensores pueden ser categorizados en dos clases, dependiendo de cómo interactúan con el medio de medición que ellos miden, pueden ser: Sensores Pasivos y Sensores Activos.

- *Sensores Pasivos*, no adicionan energía como parte del proceso de medida, pero pueden remover energía en su operación. Un ejemplo de sensores pasivos es una Termocupla, el cual convierte una temperatura física en una señal de voltaje. En este caso, el gradiente de temperatura en el ambiente genera un voltaje termoeléctrico que lleva a una variable de señal.
- *Sensores Activos*, adicionan energía para el medio de medida como parte del proceso de medición. Un ejemplo de un sensor activo es un radar o sistema sonar, donde la distancia de algún objeto es medido por un radar o sonar a partir de una onda que es reflejada sobre un objeto.

2.4.2. Sensores Analógicos y Digitales

Sensores analógicos proveen una señal que es continua tanto en magnitud y en el tiempo o espacio, siendo la definición de la palabra analógica como "continua". Si el sensor provee una señal de salida continua que es directamente proporcional a la señal de entrada, entonces es analógico. Muchas variables físicas, tales como corriente, temperatura, desplazamiento, aceleración, velocidad, presión, intensidad de luz, y esfuerzo, tienden a ser de naturaleza continua y son medidas por sensores analógicos y representados por una señal analógica.

Sensores Digitales proveen una señal digital que representada una medida. Sensores digitales son básicamente dispositivos binarios ("*on*" o "*off*"). Esencialmente, señales digitales

existen solamente para valores discretos en el tiempo (o espacio), y con periodo discreto, la señal puede representar solamente un numero discreto de valores de la magnitud. Una variación común es la representación de la señal discretamente muestreada, el cual representa la salida del sensor en valores discretos de su magnitud y tiempo.

2.4.3. Acondicionamiento de Señales

Señales provenientes de sensores por lo general no tienen características útiles para ser utilizados por un indicador (*display*), unidad de almacenamiento, transmisión, o para realizar un procesamiento futuro. Por ejemplo, estas señales pueden tener un bajo nivel de amplitud, potencia, o ancho de banda requerido, o pueden contener interferencias que enmascaran a información deseada.

Acondicionamiento de señales, incluyen: sistema de acoplamiento de señales proveniente de sensores, unidades de amplificación, unidades de filtrado analógico, etc. Una vez realizado el acondicionamiento de señales es derivado a un receptor que usualmente es un conversor Analógico – Digital (A/D) que permite transformar la señal de una región de tiempo continuo a tiempo discreto.

2.4.4. Conversores Analógico – Digital (A/D)

El proceso de convertir una señal en tiempo continuo (analógica) a una secuencia digital que puede ser procesada por un sistema digital, requiere que cuantifiquemos los valores muestreados a un número finito de niveles y representemos cada nivel por un número de bits. El equipo electrónico que realiza esta conversión de una señal analógica a una secuencia digital, se denomina un conversor Analógico – Digital (A/D), (ADC).

En la Figura 2.5 se presenta un diagrama de bloques de los elementos básicos de un conversor A/D, tales como: Muestreo y mantenimiento, Cuantificación y Codificación, Conversión A/D.

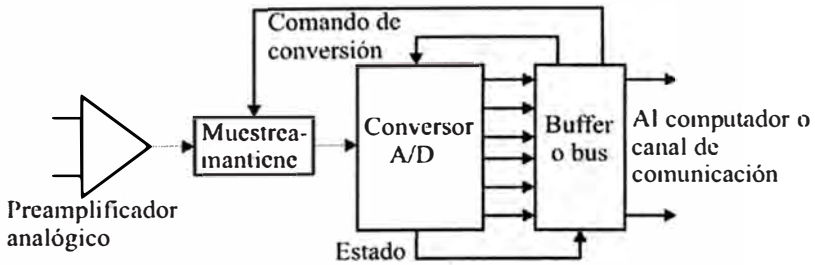


Fig. 2.5. Diagrama de bloques de elementos básicos de un conversor A/D.

- *Muestreo y mantenimiento*, de una señal analógica se realiza mediante un circuito de muestreo y mantenimiento (S/H – *Sample and Hold*), seguidamente la señal muestreada se cuantifica y se convierte a la forma digital. Normalmente el S/H se integra en el conversor A/D. El S/H es un circuito analógico controlado digitalmente que sigue la señal de entrada analógica durante el modo de muestreo y seguidamente el modo de mantenimiento la

mantiene fija al valor instantáneo de la señal en el momento en que el sistema se conmuta del modo de muestreo al modo de mantenimiento.

- *Cuantificación y codificación*, es una tarea básica de un conversor A/D que convierte un rango continuo de amplitudes de entrada en un conjunto discreto de palabras de código digital. Esta conversión implica el proceso de cuantificación y codificación. Cuantificación es el proceso no lineal que traslada una amplitud dada $x(n) \equiv x(nT)$ en el tiempo $t = nT$ en una amplitud x_k, tomada de un conjunto finito de valores. El proceso de codificación en un conversor A/D asigna un único número binario a cada nivel de cuantificación. Si tenemos L niveles, necesitamos por lo menos L números binarios diferentes. Con una longitud de $b + 1$ bits se puede representar 2^{b+1} números binarios distintos. Consecuentemente, se debe tener $2^{b+1} \geq L$, o equivalentemente a $b+1 \geq \log_2 L$. Entonces, el tamaño del escalón o la resolución del conversor A/D viene dado por:

$$\Delta = \frac{R}{2^{b+1}} \tag{2.1}$$

Donde R es el rango del cuantificador.

Los conversores prácticos A/D se diferencian de los conversores ideales en varios aspectos. En la práctica es normal encontrar varias degradaciones. Algunas de estas degradaciones de funcionamiento son debidas a un error de *offset* (la primera transición pueden no ocurrir exactamente en +1/2LSB), error del factor de escala o ganancia (la diferencia entre los valores a los cuales ocurren la primera y ultima transición no son iguales a FS – 2LSB), y errores de linealidad (las diferencias entre valores de transición no son todas iguales o cambian uniformemente). Si el error de linealidad diferencial es suficientemente grande, es posible que se pierdan una o más palabras código. Los datos sobre el funcionamiento de conversores A/D del mercado se especifican en la hoja de especificaciones de los fabricantes.

2.4.5. Conversores Digital – Analógico (D/A)

Un conversor Digital – Analógico (D/A), (DAC) toma una secuencia digital y produce a su salida un voltaje o corriente proporcional al tamaño de la palabra digital aplicada a su entrada. El convertidos digital a analógico (DAC) permite que la señal codificada y cuantizada sea convertida en una señal analógica. En su forma más básica, consiste de un amplificador operacional sumador, como se muestra en la Figura 2.6.

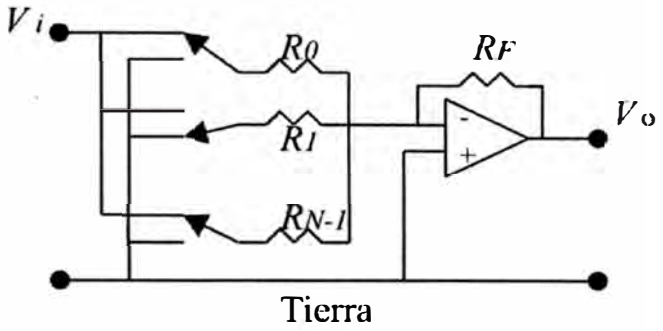


Fig. 2.6. Sistema de conversión digital a analógico.

Los interruptores se encargan de establecer conexiones a tierra o a la entrada. El voltaje de salida V_o es la suma ponderada exclusivamente de las señales en los interruptores que están conectados a la entrada. Si se escoge los valores de los resistores como:

$$R_0 = R \ , \ R_1 = \frac{R}{2} \ , \ R_2 = \frac{R}{2^2} \ \ldots \ R_{N-1} = \frac{R}{2^{N-1}} \tag{2.2}$$

Entonces la salida esta dada por:

$$V_o = -V_i \frac{R_F}{R}\left(b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \ldots + b_1 2^1 + b_0 2^0\right) \tag{2.3}$$

Donde los coeficientes b_k corresponden a la posición de los interruptores, su valor es 1 (si el interruptor conecta la entrada) o 0 (si el interruptor conecta la tierra). Los circuitos reales se basan en codificadores que utilizan solo unos cuantos valores de resistores (como R y $2R$) para superar el problema de seleccionar una gama amplia de valores (en particular para convertidores que tienen gran número de bits).

CAPITULO III
BLOQUE DE CONTROL Y RECONTRUCCIÓN DE VALORES DE MEDICIÓN

3.1. Introducción

La reconstrucción de valores de medición sobre medidas de señales básicas está sujeto a la distorsión sistemática y errores aleatorios, siendo un problema frecuentemente encontrado en aplicaciones de instrumentación. Para dar solución a este tipo de problemas es considerado un bloque de reconstrucción de señales en la estructura del sistema de medición, con la finalidad de obtener una medida de la señal principal con mayor precisión y con estándares de calidad definidos por el proceso en cuestión.

3.2. Bloque para la Medición y Control de Medidas de Interferencia

En la sección 2.3 fueron definidos los principales bloques de un sistema de medición con realimentación, como: sensor de medición, conversor A/D Bloque de Recónstrucción Directa, Bloque de Reconstrucción Indirecta, Controlador, Conversor D/A y Actuador, como se presenta en la Figura 3.1.

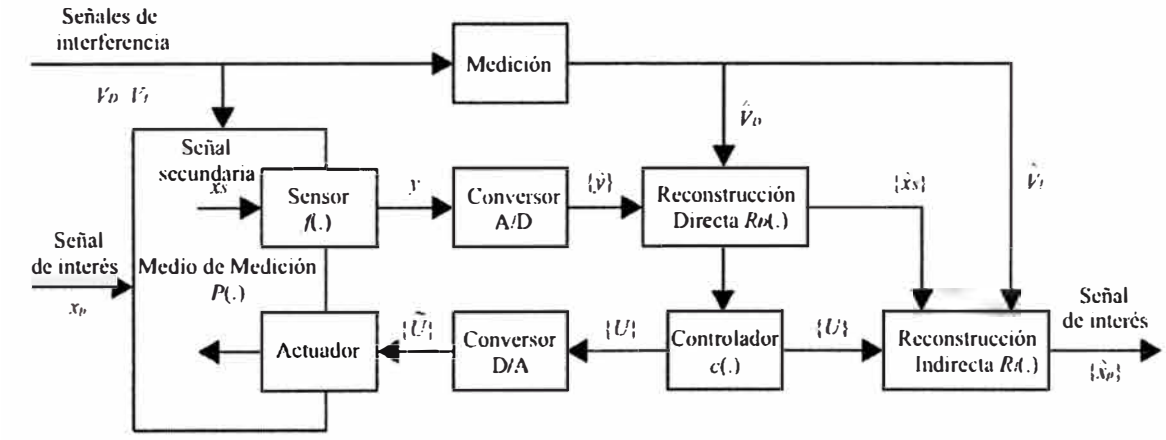


Fig. 3.1. Diagrama de bloques de un sistema de medición con realimentación.

Seguidamente se puede definir un procedimiento general para estimar la variable principal x_p. La dinámica del medio de medición es definida por la función $P(.)$ que relaciona la variable secundaria x_s, con las demás variables relacionadas en el proceso y que puede ser generalmente descrita por medio de ecuaciones diferenciales.

$$x_s(t) = P(xs, x_p(t), U(t), V_D) \qquad (3.1)$$

Donde a función $P(.)$ puede utilizar términos diferenciales de la variable $x_s(t)$.

El sensor para la medición directa de la señal secundaria presenta una respuesta dinámica para la generación de la señal eléctrica, generalmente superior a la del dominio de medición, es

decir, las constantes de tiempo de los sensores son generalmente bien inferiores a las del domino de medición. Consecuentemente, se puede despreciar la dinámica del sensor y su señal eléctrica de salida puede ser obtenido directamente de la señal secundaria a través de una función escalar, dada por:

$$y(t) = f(x_s(t), V_D) \tag{3.2}$$

La señal secundaria debe ser estimada utilizando una función de reconstrucción que debe ser idealmente la función inversa de la función de conversión del sensor utilizado, utilizando los valores cuantizados de la señal eléctrica de salida del sensor y los valores estimados de la señal de interferencia.

$$\hat{x}_s = R_D\left(\tilde{y}, \hat{V}_D\right) \cong \arg_{xs}\left[y = f\left(x_S, V_D\right)\right] \tag{3.3}$$

El bloque de control debe generar una señal de control U a partir de los valores de la señal secundaria x_s y de una señal de referencia de control x_{Sr}. Ese bloque es descrito por una función $c(.)$, que puede utilizar varios valores de la variable secundaria en el tiempo, de acuerdo con el algoritmo de control. La señal de control puede ser definido por:

$$U = c\left(\{\hat{x}_s\}, x_{Sr}\right) \tag{3.4}$$

Donde x_{Sr} es el valor de referencia de control de la variable x_s.

3.3. Diseño del Controlador

El diseño del controlador debe ser realizado de forma que este modifique la dinámica del proceso de medición, disminuyendo las constantes de tiempo asociadas para la estimación final de la variable principal. Este controlador debe realizar el control de la variable secundaria con relación a una determinada referencia, llevando en cuenta principalmente la influencia de la variable principal.

Para el sistema presentado en la Figura 3.1, se considera que las constantes de tiempo relacionadas a la medición de la variable secundaria, así como los atrasos proporcionados en los bloques digitales, son bien menores que las constantes de tiempo dominantes relacionadas con el medio de medición. El diseño del controlador depende básicamente de como las diferentes variables interactúan en el medio de medición, definido por $P(.)$. Algunas consideraciones pueden ser hechas para el diseño del controlador, tales como:

- El error de sistema controlado, en relación a una determinada referencia de la variable secundaria, no es necesariamente igual a cero. Una vez que es necesaria la medición de esta variable para la realización del control, sus valores instantáneos pueden ser utilizados directamente para recuperación de la variable principal. Así también, si el controlador fuerza el error de régimen igual a cero, se puede simplificar la ecuación de reconstrucción de la variable de interés, mas tornando más complicada la ecuación del controlador. De esta

manera, se debe verificar cual de las dos estrategias es mas ventajosa en términos de complejidad y precisión en la estimación.

- El controlador debe mejorar la dinámica de la señal de control U y de la señal secundaria x_s en relación a la señal principal x_p. Consecuentemente, se mejora la dinámica de estimación de la señal principal, una vez que para esto se utilizan los valores de la señal de control y de medición de la señal secundaria.

A continuación se presenta un ejemplo de un diseño de un controlador, suponiendo un proceso dinámico de medición definido en el dominio s (transformada de *Laplace*), dado por:

$$x_s(s) = A(s)x_p(s) + B(s)U(s) \tag{3.5}$$

Se diseña un controlador definido por $c(s)$, como se presenta en la Figura 3.2. Así tenemos que la señal secundaria puede ser expresada por:

$$x_s(s) = \frac{C(s)B(s)}{1+C(s)B(s)}x_{sr}(s) + \frac{A(s)}{1+C(s)B(s)}x_p(s) \tag{3.6}$$

Donde x_{Sr} es la señal de referencia de control para la variable secundaria x_s. La señal de control U puede ser expresada por:

$$U(s) = \frac{C(s)}{1+C(s)B(s)}x_{sr}(s) - \frac{C(s)A(s)}{1+C(s)B(s)}x_p(s) \tag{3.7}$$

El controlador debe ser diseñado de tal manera que mejore la dinámica del sistema, es decir, disminuya las constantes de tiempo asociadas, al segundo termino de las ecuaciones (3.6) y (3.7)

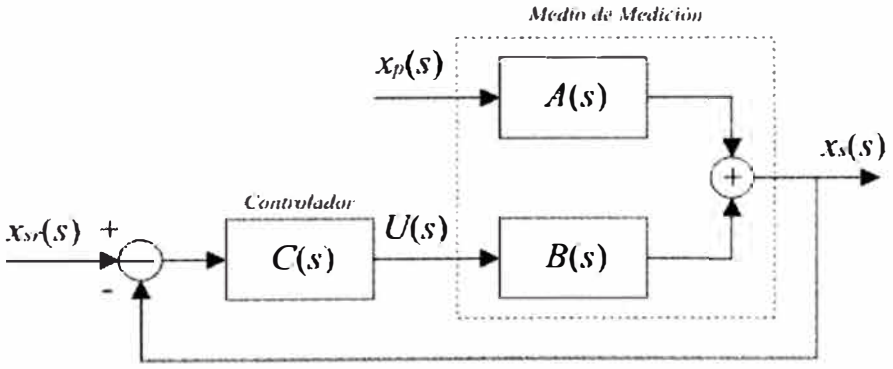


Fig. 3.2. Esquema de control.

3.4. Implementación del Bloque de Reconstrucción Digital de la Señal, Utilizando Aproximación de Funciones

La señal principal puede ser estimada idealmente utilizando la inversa de la función característica del medio de medición $P(.)$, a partir de los valores medidos de la señal secundaria x_s, de las señales de interferencia V y de los valores de la variable de control U. La estimación de esta señal, que presentara una nueva dinámica definida por el controlador aplicado al medio de medición, puede ser descrita por:

$$\hat{x}_p = R_I\left(\{\hat{x}_S\}, U, \hat{V}_I\right) \equiv \arg_{x_p}\left[x_S = P\left(x_s, x_p, \tilde{U}, V_I\right)\right] \tag{3.8}$$

Esa función puede ser representada utilizando ecuaciones de diferencia finita que representa una de las maneras de caracterizar el comportamiento de un sistema discreto lineal invariante en el tiempo (LTI). Una ventaja de utilizar este tipo de representación es que los términos derivativos son descritos a través de ecuaciones en diferencias finitas. Una vez que el sistema realimentado presenta constantes de tiempo menores que del sistema sin realimentación, se puede hacer una aproximación de esta función de reconstrucción para utilizarse apenas el valor de la variable secundaria en el instante de muestreo *k* definido por:

$$\hat{x}_p(k) = \tilde{R}_I\left(\hat{x}_S(k), U(k), \hat{V}_I\right) \tag{3.9}$$

Esa aproximación es valida con el sistema en régimen estacionario, es decir:

$$x_s(k) = x_s(k\text{-}1) = \ldots = x_s(k\text{-}n\text{+}1) \tag{3.10}$$

Para una función de reconstrucción inicial R_I, que utiliza los valores de la variable secundaria en n instantes de muestreo. Esa función presenta resultados menos sensibles al ruido (mejor acondicionadas) considerando un mismo intervalo de muestreo. Fuera del régimen, la estimación de la variable principal, a través de la ecuación (3.9), sigue la nueva dinámica del sistema realimentado que es más rápida que el sistema sin realimentación. De esta manera, esa aproximación debe ser echa caso la perdida de velocidad en la estimación sea aceptable y la mejora del acondicionamiento de los valores de estimación sea mas importante.

3.5. Ejemplo de Diseño de un Controlador

Partiendo del ejemplo de diseño del controlador presentado anteriormente, se define la función de transferencia del medio de medición dado por: $A(s) = B(s) = 0.1/(s+0.1)$. La señal secundaria x_s es medida directamente utilizando un sensor, y es muestreada por un asegurador de orden cero con un periodo de muestreo de $T = 0.2$ s. En este caso, el medio de medición posee constantes de tiempos iguales a 10 s. La señal principal x_p es estimada indirectamente a partir de los valores de U y x_s utilizando dos modos de medición: con y sin realimentación. Para ese sistema, la señal secundaria puede ser descrita a través de ecuaciones diferenciales, en función de x_p y U, por:

A partir de la ecuación (3.5), tenemos que:

$$x_s(s) = \frac{0.1}{s+0.1}x_p(s) + \frac{0.1}{s+0.1}U(s) \tag{3.11}$$

$$s.x_s(s) + 0.1x_s(s) = 0.1x_p(s) + 0.1U(s) \tag{3.12}$$

Transformando al dominio del tiempo:

$$\frac{dx_s}{dt} + 0.1x_s = 0.1x_p + 0.1U \tag{3.13}$$

La señal de interés puede ser estimada, para un instante de muestreo *k*, utilizando la función de reconstrucción, dado por:

$$\hat{x}_p(k-1) = R_I\left(\hat{x}_s(k), \hat{x}_s(k-1), U(k-1)\right) = \frac{\hat{x}_s(k) - \hat{x}_s(k-1)}{0.1T} + \hat{x}_s(k-1) - U(k-1) \qquad 3.14)$$

Donde $\hat{x}_s$ es el valor medido de la señal secundaria.

Se puede obtener una función de reconstrucción simplificada a partir de la función de reconstrucción R_I, despreciando el término de diferencia finita en (3.14), y reescribiendo para utilizar apenas los valores en el instante de muestreo k:

$$\hat{x}_p(k) = \tilde{R}_I\left(\hat{x}_s(k), U(k)\right) = \hat{x}_s(k) - U(k) \qquad (3.15)$$

El sistema de medición en el modo sin realimentación es presentado en la Figura 3.3, y simulado en el intervalo de 0 a 100 s, definiendo la variable U en un valor constante igual a 1 y la señal principal inicialmente igual a 0. En el instante de tiempo igual a 30 s, es modificada la señal principal de 0 para 1, del tipo escalón unitario.

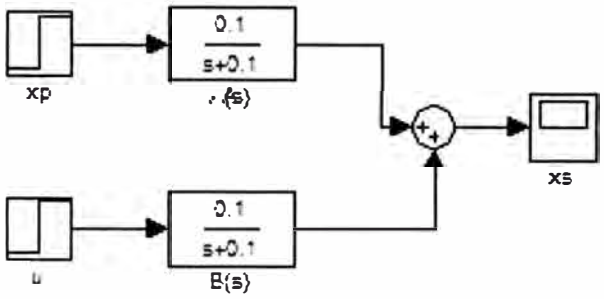


Fig. 3.3. Simulación del sistema sin realimentación

A continuación se presentan los resultados de simulación del sistema de medición sin realimentación: en la Figura 3.4 se presenta el gráfico de la señal secundaria x_s, en el tiempo; y en la Figura 3.5 se presenta la señal principal estimada en el tiempo, utilizando las funciones de reconstrucción dados por las ecuaciones (3.14) y (3.15). En este caso, la estimación utilizando (3.14) presenta un tiempo de respuesta de 10 s.

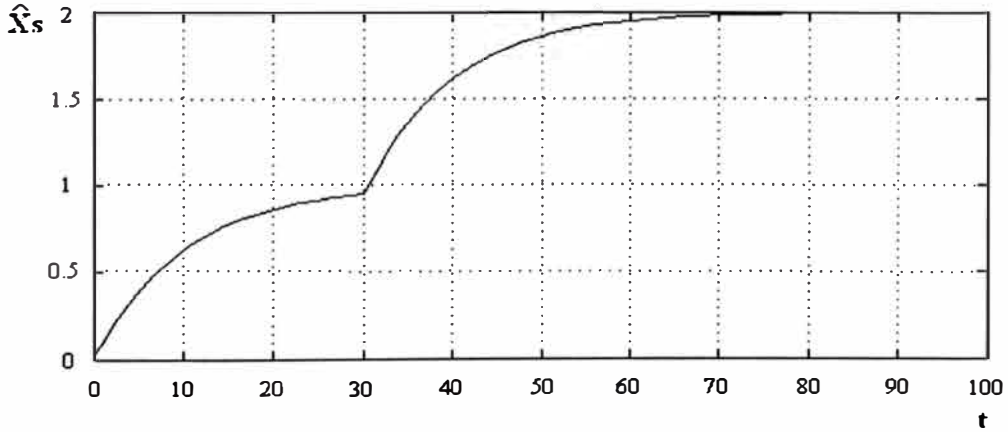


Fig. 3.4. Señal secundaria en el tiempo x_s

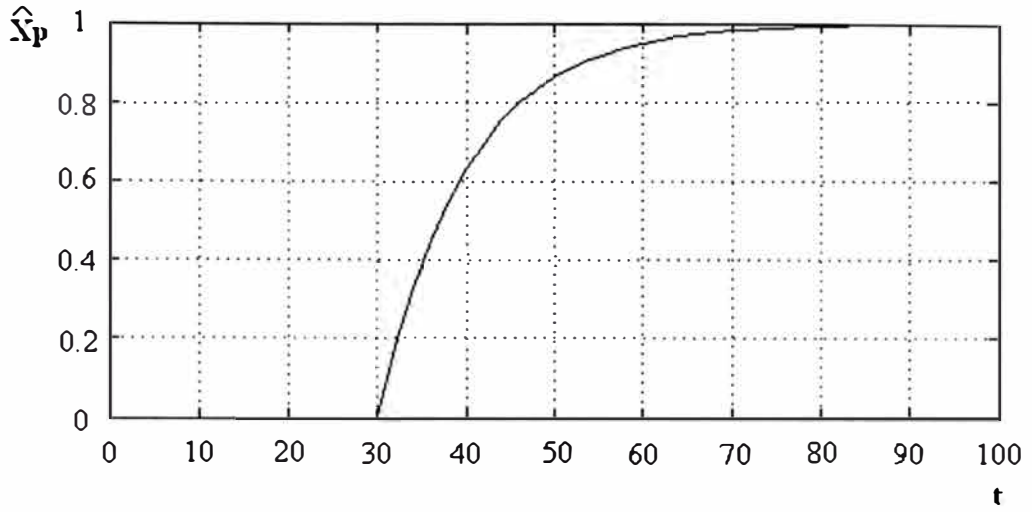


Fig. 3.5. Señal principal en el tiempo x_p

El sistema de medición en el modo con realimentación se presenta en la Figura 3.6, y simulado en el intervalo de 0 a 100 s, se utiliza un controlador proporcional con ganancia de 10, que modifica, las constantes de tiempo de estimación de 10 s para 1 s. La variable de control es U y la señal de referencia x_{Sr} utilizada fue de 1. El valor de la señal de interés es de 0 y después de 30 s aumenta para 1, del tipo escalón unitario.

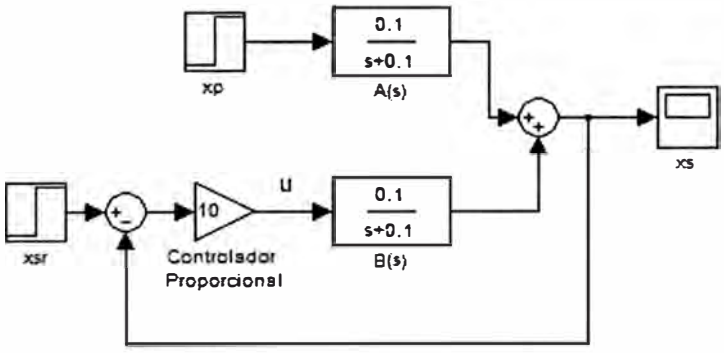


Fig. 3.6. Simulación del Sistema con Realimentación

A continuación se presentan los resultados de simulación del sistema de medición: en la Figura 3.7 se presenta la señal secundaria en el tiempo x_s y en Figura 3.8 se presenta la señal principal en el tiempo x_p, estimada utilizando a función de reconstrucción dada por la ecuación (3.15). En este caso, la estimación utilizando (3.15) presenta un tiempo de respuesta de 1 s utilizando la función de reconstrucción R_I.

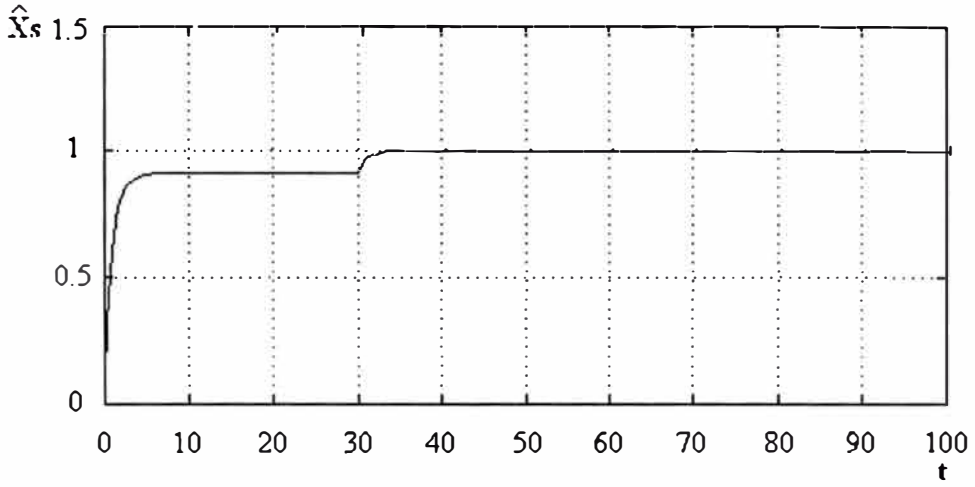


Fig. 3.7. Señal secundaria en el tiempo x_s

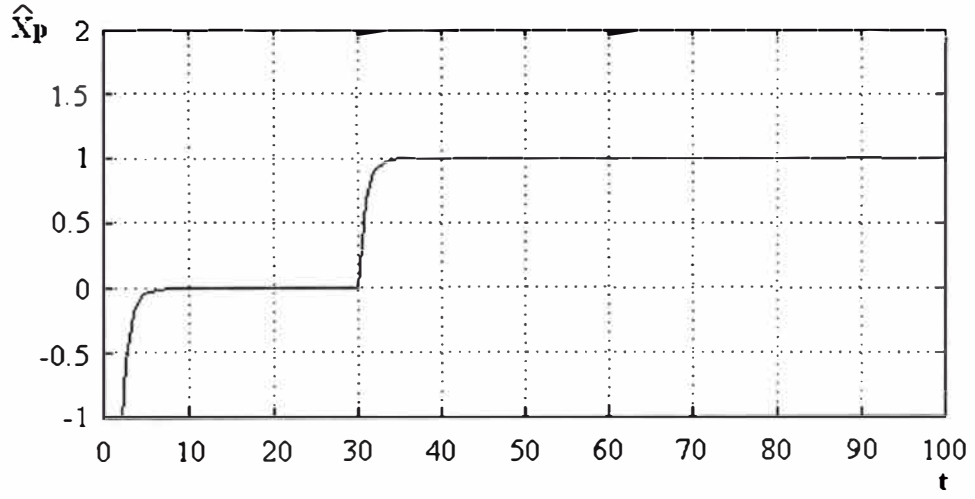


Fig. 3.8. Señal principal en el tiempo x_p

CAPITULO IV
TABLA DE EQUIVALENCIA (LUT – "*Look-Up Table*") PARA RECONSTRUCCION DE VALORES DE MEDICION UTILIZANDO SENSORES NO LINEALES

4.1. Introducción

En el proceso de medición de las variables físicas, las señales de salida de gran parte de los sensores son relacionados de forma no lineal con la señal principal a ser medida y también pueden sufrir interferencias de otras variables no deseadas. En sistemas de medición digital, esas señales deben ser acondicionadas de forma analógica y convertidas para la forma digital por un conversor Analógico – Digital A/D. Seguidamente, deben ser reconstruido de forma a compensar la no linealidad de la función de transferencia del sensor y la influencia de cualquier otra variable de interferencia.

La función de reconstrucción de la variable de medida debe ser la inversa o inversa aproximada de la función de transferencia del sensor. Entre tanto, la implementación de estas funciones en sistemas embebidos, que utilizan por ejemplo microcontroladores o ASICs (*Circuitos Integrados de Aplicación Especifica*), puede ser compleja, teniendo en vista la utilización de aritmética en punto fijo y la velocidad de cálculo relativamente baja. La utilización de tablas de equivalencia (LUT – *Look-Up Table*) con interpolación lineal es una opción interesante para este tipo de problema, comparada con otros tipos de interpolación, tales como *splines* y polinomial, que exigen mayor poder computacional.

La utilización de tablas de equivalencia (LUT) para aproximación de funciones no lineales es bien referenciada en la bibliografía, entretanto, se aborda generalmente el problema de minimización del error de aproximación sin llevar en consideración el número de células necesarias para la construcción de la tabla de equivalencia, es decir, el tamaño de memoria (en bits) necesario para ser implementado en un procesador. Aproximación lineal por partes puede ser utilizada para encontrar los coeficientes de una función lineal por partes, dado los puntos de quiebra de esta función, minimizando el error cuadrático de aproximación. Los valores de los coeficientes y de los puntos de quiebra pueden ser utilizados para encontrar los valores y el número de células de la tabla de equivalencia (LUT). En la aproximación lineal por partes, también se puede optimizar los valores de los puntos de quiebra de la función lineal por partes usando algún método de optimización, con la finalidad de encontrar un número óptimo de puntos de quiebra (o número de células de la tabla) de forma a obtener una resolución (precisión) de reconstrucción especificada.

4.2. Reconstrucción de Señales Utilizando Tablas de Equivalencia (LUT)

Diversos tipos de sensores de medición utilizan sensores que presentan características no lineales entre la señal eléctrica producida y la señal de medición. Los valores de esas señales deben ser reconstruidos a partir de los valores digitales adquiridos utilizando la función inversa (o inversa aproximada) de la función de transferencia del sensor. Entretanto, la función de reconstrucción puede ser aproximada por funciones lineales por partes utilizando una tabla de equivalencia (LUT) con interpolación lineal.

En la Figura 4.1 se presenta la señal de salida del sensor en función de la señal de medición, definido por $y = f(x)$, $f: S \rightarrow \Re$, donde $S \subset \Re$ corresponde al intervalo de medición de la señal principal. Se supone que el circuito de acondicionamiento ajusta la señal de salida del sensor con resolución N bits, la señal digital de salida es definido por: $\tilde{y} = y + \varepsilon_{AD}$, donde ε_{AD} es el error de cuantización definido por: $-q/2 \leq \varepsilon_{AD} \leq q/2$, con q igual a 1 LSB para un conversor Analógico – Digital (A/D) ideal, la variable y pertenece a un conjunto limitado.

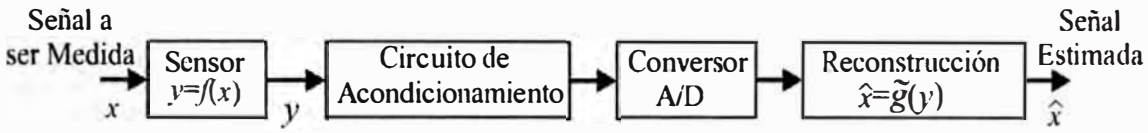


Fig. 4.1. Sistema de medición utilizando una LUT.

La función de reconstrucción ideal debe ser la inversa de la función del sensor, definida por: $g(y) = \arg[y = f(x)]$. Se define entonces, $\tilde{g}(.) \approx g(.)$ como una función lineal por partes que aproxima esa función de reconstrucción, los valores de medición pueden ser estimador por: $\hat{x} = \tilde{g}(y)$. Los Puntos de quiebra de la función aproximada (P_{yi}, P_{xi}), $i = 1, \ldots, m$ deben ser almacenados en una tabla de equivalencia – LUT (m puntos de quiebra) y la función puede ser generada para cada valor de medición utilizando interpolación lineal de los puntos almacenados en la tabla, inmediatamente inferior y superior al valor de medición. Se define también $\hat{\varepsilon}_x(\tilde{y})$ como el error máximo aceptable de medición para cada valor cuantizado de la variable y, que debe ser una asociación del error de cuantización y del error de reconstrucción.

Así, el problema relacionado con el diseño de una tabla de equivalencia (LUT) para tal finalidad consiste en definir.

- El número de puntos de quiebra y sus valores;
- El tamaño en número de bits, de cada posición de memoria de la tabla de equivalencia, de forma que el error de reconstrucción sea siempre: $\varepsilon_x(\tilde{y}) \leq \hat{\varepsilon}_x(\tilde{y})$

4.3. Calculo del Tamaño de la LUT

El procedimiento para el cálculo de la tabla de equivalencia (LUT) consiste en:

- Calcular el valor del error inicial de medición en función de la variable cuantizada
- Definir el error máximo aceptable de medición
- Definir las márgenes aceptables (superior e inferior) de los valores estimados de la señal x
- Aplicar un algoritmo de búsqueda para encontrar los valores y número de puntos de quiebra de la tabla de equivalencia (LUT), de forma que los valores estimados de la señal x están siempre dentro de las márgenes.

Este procedimiento es detallado a continuación: El error de medición inicial, definido por ε_{xo} (usando la función de reconstrucción ideal) en función de los valores de la variable $\tilde{y}$, es inherente al sistema de medición en cuestión y corresponde al mejor caso de reconstrucción. No se puede obtener un aumento de resolución utilizando la reconstrucción con precisión mucho mayor de que el necesario en el intervalo de medición de la señal x. El error de medición inicial puede ser considerado como el peor caso del error de medición debido al efecto de cuantización y a la característica no lineal del sensor. Ese error puede ser calculado substituyéndose los valores de la variable digitalizada y en la función de reconstrucción ideal y considerando el error de cuantización máximo:

$$\varepsilon_{xo}(\tilde{y}) = \max\left\{\left|g(\tilde{y}) - g(\tilde{y}+\varepsilon_y)\right|, \left|g(\tilde{y}) - g(\tilde{y}-\varepsilon_y)\right|\right\} \qquad (4.1)$$

Donde ε_y es el peor caso del error de cuantización, igual a $q/2$.

El error máximo aceptable de medición debe ser siempre mayor o igual al error de medición inicial $\varepsilon_{xo}(\tilde{y}) \le \hat{\varepsilon}_x(\tilde{y})$, pudiendo ser atribuido arbitrariamente o en función del error inicial de medición como:

$$\hat{\varepsilon}_x(\tilde{y}) = w(\tilde{y}).\varepsilon_{xo}(\tilde{y}),\ w(\tilde{y}) \ge 1,\ \forall \tilde{y} \in S \qquad (4.2)$$

Donde w es una función definida siguiendo las necesidades del proyecto y que puede ser considerada como una función de perdida de precisión (resolución) en la reconstrucción. Para valores de w = 1, no hay perdida de resolución en la reconstrucción.

Para el error máximo de medición, se define los límites superior e inferior aceptables de la señal x, respectivamente como: $x_s(\tilde{y}) = g(\tilde{y}) + \hat{\varepsilon}_x(\tilde{y})$ y $x_i(\tilde{y}) = g(\tilde{y}) - \hat{\varepsilon}_x(\tilde{y})$. Se puede entonces, implementar un algoritmo de búsqueda de los puntos de quiebra, tal que las rectas definidas por dos puntos de quiebra consecutivos se encuentren siempre dentro de las dos curvas.

4.4. Formulación del Problema de Aproximación de Funciones No Lineales

Se considera un sistema digital embebido en que se necesita generar valores de una función no lineal, utilizando aproximación de funciones lineales por partes. Para esto, se utiliza un sistema embebido de bajo costo, en que los cálculos sean realizados en punto fijo (*fixed - point*), con resolución limitada y con limitaciones de espacio de memoria debido a las restricciones de la arquitectura del proyecto.

La función no lineal es definida por $y = f(x)$, $x \in [x_{min}, x_{max}]$ e $y \in [y_{min}, y_{max}]$, en que cada elemento del conjunto de valores de entrada x es cuantizado con una resolución equivalente a N bits e los elementos del conjunto de valores de y son generados con una exactitud equivalente a una resolución de N_Y bits. Se asume que los valores de x e y son positivos y sus valores mínimos son ceros, $x_{min} = 0$ e $y_{min} = 0$, que pueden ser obtenidos adicionándose constantes a la función f. Estas constantes pueden ser substraídas después de la generación de los valores de y.

La función no lineal es aproximada por una función lineal por partes considerando que los cálculos pueden ser realizados con una precisión equivalente a N_T bits de resolución, con $N_T \geq N_Y$. Los valores de los puntos de quiebra para reproducir la función aproximada son almacenados en una tabla de equivalencia (LUT – *Look-up Table*) almacenados en un sistema embebido, como es presentado en la Figura 4.2

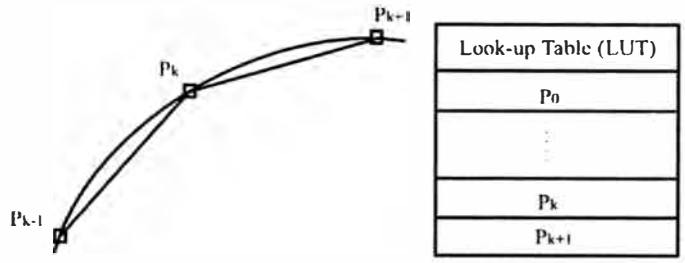


Fig. 4.2. Aproximación lineal por partes de una función no lineal.

4.5. Cuantización, representación de las variables y representación de las variables en escalas diferentes en un sistema digital embebido

En un sistema digital embebido con capacidad de cálculo en punto fijo, y resolución limitada, las variables numéricas pueden ser representadas asumiendo valores enteros, así también la cuantización de los números dependen de la resolución del conversor analógico – digital. Las variables numéricas también pueden ser representadas en otras escalas de forma a obtenerse el máximo de resolución de representación, considerando el error máximo debido a la cuantización.

En esta sección es adoptada una notación para representar las variables en escala real, variables en otra escala y para las aproximaciones:

- Variables en minúscula representan valores en la escala real, p.e: x, y
- Variables en minúscula representan valores en otras escalas, p.e: X, Y
- Variables con sobrescrito "~" representan aproximaciones

4.5.1. Cuantización de las variables

El proceso de cuantización utiliza una función no lineal en que una variable de valores continuos es representada por un conjunto de valores discretos. Cada elemento del segundo conjunto representa un intervalo de variación del primero. En la cuantización uniforme, para una cuantización de n elementos, el intervalo total de variación de la variable continua es dividido en n intervalos iguales y cada intervalo es representado por un único número. Para una resolución de N

bits, se tiene $n = 2^N$ intervalos de cuantización. La cuantización de la variable y en N bits es definida por:

$$\tilde{y} = Q_N(y) \tag{4.3}$$

El intervalo de cuantización puede ser calculado por:

$$\Delta = \frac{y_{max} - y_{min}}{2^N} \tag{4.4}$$

Cuando se usa el valor de salida en el medio del intervalo de los valores de entrada, el error de cuantización tiene los menores valores absolutos máximos. Cada intervalo del intervalo total de variación de la variable continua es representado por un valor posicionado en el medio del intervalo. De esa forma, los valores de cuantización de y, pueden ser calculados por:

$$\tilde{y} = \Delta i + \frac{\Delta}{2};\ i = 0,...,\left(2^N - 1\right) \tag{4.5}$$

La cuantización en la mitad del intervalo en N bits de y puede ser calculada por:

$$\tilde{y} = Q_N(y) = \tilde{y}_i \mid \Delta i \le y < \Delta(i+1) \tag{4.6}$$

4.5.2. Representación de Variables

La variable $x \in [x_{min}, x_{max}]$ puede ser considerada con representación entera una vez que esta es cuantizada con una resolución equivalente a N bits, asumiendo valores enteros X = 0, 1, 2, ...,2^N-1, correspondiente a los números binarios 00...0 hasta 11...1. La transformación de la variable de valores enteros para un valor real de x puede ser realizado a través de:

$$x = \frac{X + 0,5}{2^N}\left(x_{max} - x_{min}\right) \tag{4.7}$$

Se suma 0.5 porque se considera que los valores de X enteros son convertidos en el medio del intervalo.

4.5.3. Representación de Variables en otras Escalas

Los valores de $y \in [y_{min}, y_{max}]$, también pueden ser representados en otras escalas, entretanto, se considera la resolución N_T bits. A pesar de que la resolución deseada de generación de valores de y sea N_Y. Se considera N_T bits porque los cálculos en el sistema embebido son realizados en esa resolución. Además, la resolución de N_Y es considerada para la evaluación final de y debido a las perdidas en la aproximación y en la propagación de la incerteza asociada en la generación de x. El valor verdadero de y es calculado por:

$$y = f(x) \tag{4.8}$$

Considerando el valor máximo de y para todo el rango del intervalo, la representación de y en otra escala (N_T bits), puede ser calculada por:

$$Y = \frac{2^{N_T}}{y_{max} - y_{min}} y - 0,5 \tag{4.9}$$

La cuantización puede ser realizada directamente para los valores de Y con la ventaja de generar números enteros. La representación del valor verdadero por un número entero en otra escala de resolución N_T puede ser obtenida por:

$$\tilde{Y} = \text{round}(Y) = \text{floor}\left(\frac{2^{N_T}}{(y_{max} - y_{min})} y\right) \tag{4.10}$$

Los valores cuantizados de y en escala real pueden ser calculados a partir de (4.10) por:

$$\tilde{y} = \frac{\tilde{Y} + 0,5}{2^{N_T}}(y_{max} - y_{min}) \tag{4.11}$$

Los valores cuantizados de $\tilde{Y}$ e $\tilde{y}$ son los valores que mejor representan Y e y en la escala de resolución de N_T bits. Se considera saturación para el valor máximo $2^{N_T} - 1$ cuando ocurre la saturación.

4.6. Análisis de la Propagación de la Incerteza y Error de Aproximación en un Sistema Digital Embebido

Para evaluar el efecto del error de cuantización de las variables en un sistema digital embebido, se realiza un análisis de la propagación de los errores de cuantización cuando la función no lineal aproximada es implementada en un sistema digital con capacidad de cálculo en punto fijo.

Cada elemento del conjunto de valores de $y \in [y_{min}, y_{max}]$ es generado a una resolución equivalente a N_Y bits desconsiderándose la forma como los valores de y son generados, se tiene que el valor máximo de la incerteza asociada $\tilde{\varepsilon}_y$, para todos los valores de y deben ser:

$$\tilde{\varepsilon}_y = \frac{y_{max} - y_{min}}{2^{N_1+1}} \tag{4.12}$$

En el sistema digital embebido, los valores de la variable independiente x poseen una incerteza asociada a su generación. Consecuentemente, se define $\tilde{x} = \{\tilde{x}_1, ..., \tilde{x}_n\}$ como los n posibles valores de x y $\varepsilon_x = \{\varepsilon_{x_1}, ..., \varepsilon_{x_n}\}$ como las incertezas asociadas, tal que los valores aproximados son calculados por: $\tilde{x} = x + \varepsilon_x$.

La incerteza asociada a la variable independiente x, ε_x, puede ser originada de diversas formas: debido a la cuantización en un conversor Analógico – Digital, redondeo o truncamiento de cálculo, precisión de un codificador o del propio reloj del procesador (*clock*), etc. En el caso de un conversor analógico digital ideal, los valores de los límites de las incertezas son:

$$\varepsilon_{x\,max} = \frac{q}{2} \text{ e } \varepsilon_{x\,min} = -\frac{q}{2} \tag{4.13}$$

Donde q es el paso de cuantización.

Consecuentemente, los valores de la incerteza asociada son limitados por $\varepsilon_{x_i} \in [\varepsilon_{x_i\,min}, \varepsilon_{x_i\,max}]$, para $i = 1, ..., 2^N$, siendo normalmente constantes para todo el conjunto de variación de la variable x.

$$\varepsilon_{x,\max} = \frac{x_{\max} - x_{\min}}{2^{N+1}} \text{ e } \varepsilon_{x,\min} = -\frac{x_{\max} - x_{\min}}{2^{N+1}} \tag{4.14}$$

Una solución para el problema de aproximar la función no lineal f en un sistema digital embebido por funciones lineales por partes es definida como $s = \tilde{f}(\tilde{x})$, en que s es una aproximación de $y = f(x)$. La función aproximada s presenta una incerteza asociada causada por la propagación de la incerteza asociada a la variable libre ε_x, que es propagada a través de la función aproximada s. Consecuentemente, se debe garantir que la incerteza asociada en la generación de los valores de la función aproximada s en relación a la función no lineal y este dentro de la especificación de exactitud del proyecto, o sea:

$$|\varepsilon_s| \leq \hat{\varepsilon}_y \tag{4.15}$$

Donde ε_s es la incerteza (el error) de generación de la función aproximada s, definida como:

$$\varepsilon_s = y - s \tag{4.16}$$

4.6.1 Análisis del Error de Aproximación Considerando la Variable Independiente *X* como exacta

A continuación, se presenta un análisis del error de aproximación, siendo considerado el caso de estudio del análisis del error de aproximación. Se determinan los límites del error de aproximación considerando la variable independiente x como exacta. En este sentido el error de aproximación para todo el conjunto de valores de x es dado por:

$$\varepsilon_{A_i} = \tilde{f}(x_i) - f(x_i) \tag{4.17}$$

El error de aproximación es definido por ε_A, que es limitado por los valores máximos ε_{Amax} y mínimos ε_{Amin}, que son constantes para el conjunto de elementos de x:

$$\varepsilon_{Amin} \leq \varepsilon_A \leq \varepsilon_{Amax} \tag{4.18}$$

$$\varepsilon_{A\max} = \frac{(y_{\max} - y_{\min})}{2^{N_1+1}} \text{ e } \varepsilon_{A\min} = -\frac{(y_{\max} - y_{\min})}{2^{N_1+1}} \tag{4.19}$$

Los valores de la función aproximada s, cualquiera que sea, lineal, cuadrática, etc, deben ser calculados de forma cuantizada en la resolución disponible para el procesador del sistema embebido, con N_1 bits de resolución.

$$s = \tilde{f}(x) \tag{4.20}$$

El cálculo de los valores cuantizados de la función aproximada S, en la escala de resolución de N_1 bits, puede ser encontrado utilizando las ecuaciones (4.9) e (4.10):

$$S = \frac{2^{N_1}}{(y_{\max} - y_{\min})} y - 0.5 \tag{4.21}$$

$$\tilde{S} = \text{round}(S) \tag{4.22}$$

Donde *round* es una función de redondeo.

Debido a la cuantización de la función aproximada en otra escala de resolución, N_T bits, es necesario representar los valores del error de aproximación ε_A, determinado por la ecuación (4.19), en la escala de N_T bits de resolución, siendo determinado por:

$$E_{A\max} = \frac{2^{N_T}}{y_{\max} - y_{\min}} \varepsilon_{A\max} \text{ e } E_{A\min} = \frac{2^{N_T}}{y_{\max} - y_{\min}} \varepsilon_{A\min} \qquad (4.23)$$

Como los cálculos en el procesamiento de las señales del sistema embebido es realizado con una resolución equivalente de N_T bits, los valores de los puntos de quiebra de la función no lineal aproximada por funciones lineales por partes, deben ser calculados y almacenados en la tabla de equivalencia en la escala N_T bits de resolución.

4.7.Normalización de la Función Aproximada

Una normalización de la función aproximada puede ser hecha para facilitar la interpretación de la aproximación con relación a los límites de errores de aproximación. Este proceso de normalización substrae de todas las variables los valores de la función cuantizada y_q a N_T bits de resolución. Entonces se puede usar la siguiente transformación para normalizar la función desconsiderando su forma:

$$S = s - y_q \; ; \; Y = y - y_q \; ; \; E_{A\min} = \varepsilon_{A\min} + Y \; ; \; E_{A\max} = \varepsilon_{A\max} + Y \; ; \; P_y = p_y - y_q \text{ e } P_x = p_x \qquad (4.24)$$

Con está transformación, se trasladan las variables alrededor de cero, de esta manera estas variables asumen valores enteros 0, ±1, ±2,..., e $\varepsilon_A = Y - S$.

4.8. Aproximación de Funciones No Lineales Utilizando Funciones Lineales por Partes

Una tabla de equivalencia (LUT – *Look-up Table*) es un tipo de bloque lógico que contiene células de almacenamiento que son utilizadas para implementación de pequeñas funciones. Cada célula es capaz de almacenar un único valor, así tablas de equivalencia (LUT) de varios tamaños pueden ser creados e implementados en un microcontrolador o procesador.

Estas funciones no lineales pueden ser generadas a partir de tablas de equivalencia, utilizando interpolación lineal por partes, siendo una de las técnicas mas utilizadas, que comparadas con otros tipos de interpolación, tales como *splines* y polinomial, exigen menor poder computacional.

La función aproximada es construida por partes a partir de m puntos de quiebre (puntos comunes de dos segmentos lineales diferentes), $p_x = \{p_{x1},\ldots,p_{xm}\}$, $p_x \subseteq x$ y $p_y = \{p_{y1},\ldots,p_{ym}\}$, $p_y \subseteq y$, $m \leq n$, que son normalizadas en una tabla de equivalencia.

Los puntos de quiebra (p_x, p_y), son almacenados con una resolución equivalente de N_T bits. La función de aproximación $s = \tilde{f}(x)$ es compuesta por m funciones lineales $\tilde{f}_j$ definidas en los intervalos $[p_{xj}, p_{xj+1}]$, con $j = 1,\ldots,m$, $p_{x1} = x_1$ y $p_{xm} = x_n$ es definida en general como:

$$s = \tilde{f}(x) = a_j + b_j\left(x - p_{xj}\right) \qquad (4.25)$$

Donde los valores de los coeficientes a_j y b_j, son calculados a través:

$$a_j = p_{yj} \text{ e } b_j = \frac{p_{yj+1} - p_{yj}}{p_{xj+1} - p_{xj}} \tag{4.26}$$

Para:

$$p_{xj} \leq x < p_{xj+1} \text{ e } \tilde{f}_{m+1}(x_n) = \tilde{f}_m(x_n) \tag{4.27}$$

4.9. Dimensión del Tamaño de la Tabla de Equivalencia (LUT – *Look-Up Table*)

En un sistema digital embebido, la función aproximada es generada a partir de los puntos de quiebra almacenados en la tabla de equivalencia (LUT). Asumiendo que no existe perdida de resolución en el cálculo de los valores de y, la resolución mínima de almacenamiento es de N_T bits, la tabla de equivalencia debe ser tal que el error de cuantización proporcionado (debido a la resolución de la tabla) sea igual al mínimo valor del error de aproximación ε_A, o sea:

$$N_T \geq \log_2\left(\frac{y_{max} - y_{min}}{2\min\left(\{\varepsilon_{A\max}, -\varepsilon_{A\min}\}\right)}\right) \tag{4.28}$$

Donde $\{\varepsilon_{Amax}, -\varepsilon_{Amin}\}$ es la concatenación de los dos vectores.

El procedimiento para determinar el tamaño de la tabla de equivalencia (LUT) debe seguir las siguientes etapas:

- Definir la resolución mínima de generación de los valores de la variable independiente, N_Y;
- Calcular el valor de la componente debida a la propagación de la incerteza ε_x, utilizando (4.14);
- Calcular los valores limites del error de aproximación ε_A utilizando (4.19), caso contrario, se debe modificar la resolución deseada de generación de y o disminuir la incerteza asociada a x;
- Definir la resolución de almacenamiento de la tabla de equivalencia, N_T, utilizando (4.28);
- Utilizar un algoritmo específico para busca los puntos de quiebra de la función aproximada.

El algoritmo para determinar el tamaño de la tabla de equivalencia debe buscar los puntos de quiebra de tal forma que el error de aproximación para cada valor de x se encuentre siempre contenido dentro de los límites aceptables, es decir: $\varepsilon_{A_i} \in [\varepsilon_{A_i \min}, \varepsilon_{A_i \max}]$.

4.10. Caso de Estudio: Aproximación de la Función Seno

Se considera el estudio de caso de la generación de los valores de la función seno en el primer cuadrante, para el análisis de la propagación de la incerteza y del error de aproximación. Para la función seno $y = sen(x)$, por ser simétrica, es necesario aproximar apenas un cuarto de la función $x \in [0, \pi/2]$ e $y \in [0, 1]$, los demás valores pueden ser encontrados a partir de la generación de estos valores.

Se considera que los valores del ángulo x son cuantizados en cuatro bits ($N = 4$) y la incerteza asociada es de ½ LSB que es constante para todo el conjunto de valores de x:

$$\varepsilon_{x\max} = \frac{x_{\max} - x_{\min}}{2^{N+1}} = \frac{\pi/2}{2^5} = 0,049 \qquad (4.29)$$

En la Tabla N° 4.1 se presentan los valores binarios de x y el ángulo equivalente calculado utilizando la primera transformación de la ecuación (4.7)

Tabla N° 4.1. Valores de x y ángulo equivalente con dos espacios decimales

X	0	1	2	3	4	5	6	7
X	0,049	0,147	0,245	0,344	0,442	0,540	0,638	0,736
X	8	9	10	11	12	13	14	15
X	0,834	0,932	1,031	1,129	1,227	1,325	1,423	1,522

En el análisis de la propagación de la incerteza y errores de aproximación en la generación de la función "senoidal" es analizado considerando la variable libre como exacta ($N = N_Y$).

4.10.1. Análisis del Error de Aproximación de la Función Seno Considerando la Variable Independiente x como exacta ($N = N_Y$)

Para evaluar el efecto de cuantización y la propagación de errores en la generación de valores de funciones no lineales en sistemas embebidos, se considera la generación de los valores aproximados de la función seno $y = \text{sen}(x)$, en el primer cuadrante, considerando la variable independiente x como exacta, la exactitud de generación es equivalente a la resolución de x, o sea, $N_Y = N = 4$ bits. Para un cambio de escala de la función aproximada con $N_T = 5$ bits, los limites del error de aproximación son equivalentes a un error de cuantización de 4 bits.

$$\varepsilon_{A\max} = -\varepsilon_{A\min} = \frac{y_{\max} - y_{\min}}{2^{N_T+1}} = \frac{1}{2^5} = 0,03125 \qquad (4.30)$$

En este caso los límites del error de aproximación escalonados son calculados por (4.23)

$$E_{A\max} = 1 \text{ e } E_{A\min} = -1 \qquad (4.31)$$

Se considera los puntos de quiebra $p_x = [0, 6, 12, 15]$ y $p_y = [1, 19, 30, 31]$, en la Figura 4.3 se presenta los valores de los puntos de quiebra, la función aproximada $\tilde{f}$ y los limites de la función aproximada, en función de los valores de X.

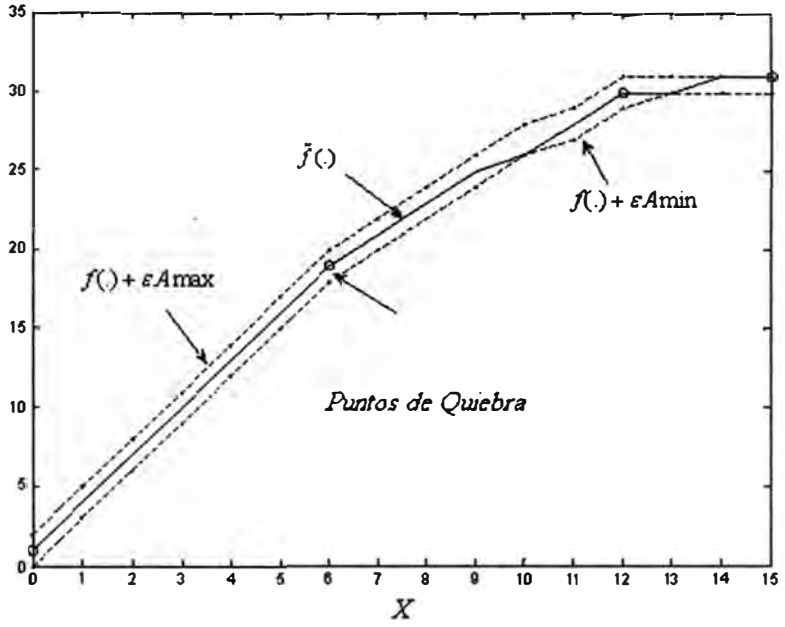


Fig. 4.3. Valores aproximados de y y limites de aproximación.

En la Figura 4.4 se presenta la representación del problema de aproximar la función utilizando la normalización definida por la ecuación (4.24).

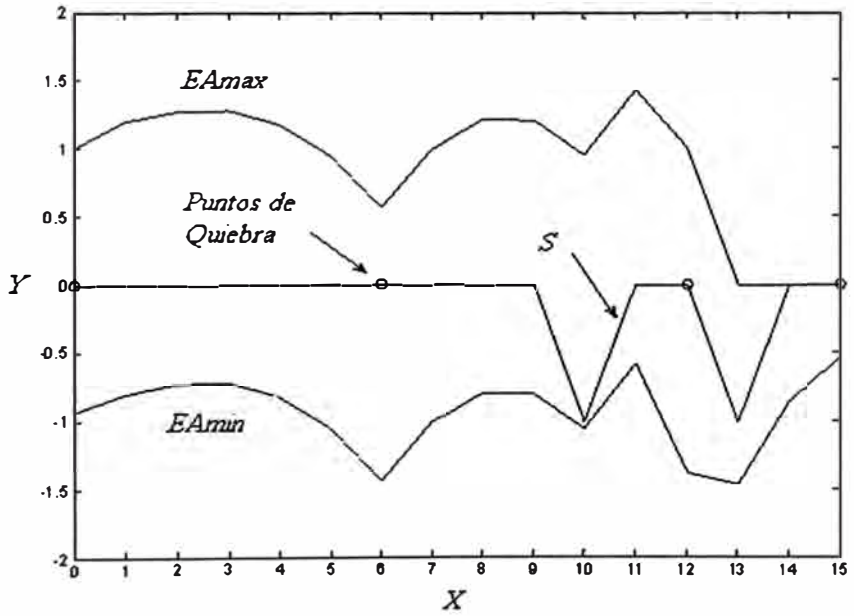


Fig. 4.4. Error de aproximación S y límites de error de aproximación E_{Amin} y E_{Amax} normalizados.

CAPITULO V
DETERMINACION DE APROXIMACION DE FUNCIONES NO LINEALES PARA SISTEMAS EMBEBIDOS UTILIZANDO COMPUTACION EVOLUTIVA

5.1. Introducción

Los sistemas basados en computación evolutiva son particularmente aplicados en problemas complejos de optimización: problemas con diversos parámetros o características que necesitan ser combinadas en búsqueda de la mejor solución, problemas con muchas restricciones o condiciones que no pueden ser representadas matemáticamente y problemas con grandes espacios de búsqueda.

Por ejemplo, los algoritmos genéticos han sido aplicados en diversos problemas de optimización, tales como: optimización de funciones matemáticas, optimización combinatoria, optimización de planeamiento de tareas, problema de optimización de rutas de vehículos, optimización de proyectos de circuitos electrónicos, etc.

Diversos proyectos de sistemas en el mundo real son usualmente abordados como problemas de optimización. Entretanto, solamente algunas combinaciones de las variables relacionadas en el proyecto resultan realmente útiles cuando son utilizadas para dar solución al problema. Consecuentemente, diversos algoritmos de optimización pueden ser implementados para encontrar la mejor combinación de las variables, que resulten en la realización efectiva del proyecto.

Algoritmos de optimización pueden ser clasificados en dos categorías: deterministicos y estocásticos. Los métodos deterministicos son basados en técnicas de búsqueda local, métodos de bifurcación y limitados (*branch and bound*), aproximaciones basadas en descomposición, etc.

Métodos estocásticos incluyen técnicas de búsqueda aleatoria, recocido simulado (*simulated annealing*), métodos de agrupamiento (*clustering*), modelos estocásticos y axiomas. Técnicas evolutivas, también son consideradas estocásticas, cuando son utilizadas en la solución de problemas de optimización.

En la actualidad sistemas basados en computación evolutiva son muy utilizados como herramientas de optimización, como por ejemplo: en problemas de ingeniería, ciencias naturales, biología y ciencia de computación. La idea básica es aplicar el proceso de evolución natural como un paradigma de solución de problemas, utilizando para su implementación un computador.

La utilización de sistemas basados en métodos de aprendizaje para la solución de problemas complejos de optimización necesita de algún tipo de operador de búsqueda iterativa. Este proceso de búsqueda debe explorar el espacio que describa todas las configuraciones posibles de solución y las técnicas evolutivas es una alternativa viable para realizar este procedimiento.

Los algoritmos evolutivos presentan una estructura que corresponde a una secuencia de pasos hasta la solución, siendo esta misma estructura aplicada a una amplia gama de problemas, de una manera robusta y flexible. Consecuentemente, los algoritmos evolutivos son aplicados a problemas complejos, para los cuales otras técnicas conocidas son ineficaces o no pueden ser aplicados.

5.2. Principio Básico de Computación Evolutiva

Computación evolutiva puede ser clasificada como una estrategia de solución concebida para resolver problemas genéricos de optimización y operar en espacios no lineales y no estacionarios, es decir, cuando el problema esta sujeto a pequeñas variaciones en sus especificaciones. Algoritmos evolutivos son métodos de búsqueda estocástica que imitan la evolución biológica natural. Los algoritmos evolutivos son compuestos por una secuencia de pasos hasta la solución, siendo estos pasos los mismos para una amplia gama de problemas. Consecuentemente, computación evolutiva es entendida como un conjunto de técnicas y procedimientos genéricos y adaptables, a ser aplicados en la solución de problemas complejos, para los cuales soluciones clásicas o dedicadas no existen, no son adaptables o presentan fallas cuando son empleadas. Entretanto, las estrategias basadas en computación evolutiva no pueden ser escogidas como la primera abordaje en la búsqueda de la solución de un problema.

En la Figura 5.1se presenta la estructura de un sistema basado en computación evolutiva, básicamente, son modelos computacionales que reciben como entradas:

Una población de individuos (generación inicial), que corresponde a las soluciones candidatas junto a problemas específicos.

Una función que mide la adaptación relativa de cada individuo frente a los demás (función de adaptabilidad, objetivo o *fitness*).

Algoritmos evolutivos operan sobre poblaciones de soluciones potenciales aplicando el principio de sobre vivencia para producir mejores aproximaciones para una solución. En cada generación, un nuevo conjunto de soluciones es creado al participar en un proceso de selección individual de acuerdo a sus niveles de adaptación (*fitness*) y a través de operadores que generan individuos que serán mejores en su ambiente que los individuos de donde fueron inicialmente creados, de la misma manera que una adaptación natural.

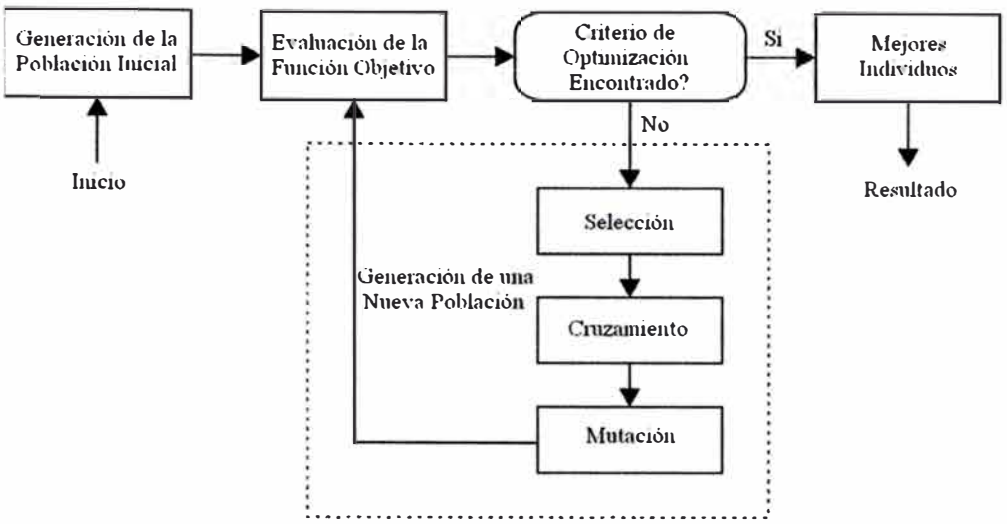


Fig. 5 1. Estructura de un programa evolutivo.

Cada individuo de la población se encuentra descrito a través de una lista ordenada (cromosomas), descrita a partir de un alfabeto finito. Cada atributo de la lista es equivalente a un gene. El tamaño de la lista esta directamente asociado al número mínimo de elementos necesarios para describir cada individuo de la población (solución candidata).

Las principales abordajes de algoritmos evolutivos propuestas en la bibliografía, que fueron desarrolladas independientemente son: *algoritmos genéticos*, *programación evolutiva* y *estrategias evolutivas*, siendo que las principales diferencias entre ellos esta en los operadores genéticos empleados en sus implementaciones.

- Los *algoritmos genéticos*, introducidos por Holland, fueron originalmente propuestos con el objetivo de formalizar modelos de procesos adaptativos, explicar rigurosamente procesos de adaptación en sistemas naturales y desarrollar sistemas artificiales (simulados por computador).
- *Programación evolutiva*, introducido por Fogel, fue originalmente propuesta como una técnica para crear inteligencia artificial a través de la evolución de maquinas de estado finito (FSM – *Finite state machine*). Una FSM es una maquina abstracta que transforma una secuencia de símbolos de entrada en una secuencia de símbolos de salida. Esta transformación depende de un conjunto finito de estados y un conjunto finito de reglas de transición de estados, siendo que el desempeño de una FSM con respecto al ambiente es medido con base en la capacidad de predicción de la maquina, esto es, comparar cada símbolo de salida con el siguiente símbolo de entrada.
- *Estrategias evolutivas*, desarrolladas por Rechenberg y Schwefel, fueron inicialmente propuestas con el objeto de solucionar problemas de optimización de parámetros, tanto discretos como continuos. Las estrategias evolutivas emplean en su implementación operadores de cruzamiento y mutación.

A pesar que los abordajes anteriormente citados han sido desarrollados de forma independiente, sus algoritmos poseen una estructura común. Será usado el término de *algoritmo evolutivo* como una denominación común a todos estos algoritmos. La estructura de un algoritmo evolutivo puede ser dada en la forma.

```
Programa evolutivo
Inicio
   t ← 0
   inicializar P( t )
   evaluación P( t )
   mientras (no termina la condición de parada) hacer
   inicio
        t ( t + 1
        seleccionar P( t ) de P( t - 1)
        evaluación P( t )
   fin
fin
```

La programación evolutiva es un algoritmo probabilístico, el cual mantiene una población de individuos $P(t) = \{x_1', ..., x_n'\}$ en la iteración (generación) t. Cada individuo representa un candidato a la solución del problema en cuestión y, en cualquier programa evolutivo, asume la forma de una estructura de datos S. Durante la iteración t cada solución x_i' es evaluada y produce alguna medida de adaptación, o *fitness*. Entonces, una nueva población es formada en la iteración $t + 1$ por la selección de los individuos mas adaptados. Algunos individuos de la población sufrirán un proceso de alteración por medio de operadores genéticos para formar nuevas soluciones. Existen transformaciones m_i (mutación) que crean nuevos individuos a través de pequeñas modificaciones de atributos en un individuo (m_i: $S \rightarrow S$), y transformaciones de orden superior c_i (cruzamiento), que crean nuevos individuos a través de la combinación de los dos o mas individuos (c_i: $S \times ... \times S \rightarrow S$). Después de un numero de generaciones, la condición de parada debe ser atendida, la cual generalmente indica la existencia en la población, de un individuo que representa una solución aceptable para el problema, o cuando el número máximo de generaciones fue alcanzado.

Abordajes de computación evolutiva tales como *algoritmos genéticos*, *programación evolutiva* y *estrategias evolutivas* difieren en diversos aspectos, dentro los cuales se destacan: estructuras de datos utilizadas para codificar un individuo, operadores genéticos empleados, métodos para crear la población inicial y métodos para seleccionar individuos para la generación siguiente. Entretanto, ellas comparten el mismo principio común: una población de individuos sufre

algunas transformaciones y durante la evolución los individuos compiten por la sobre vivencia. A seguir es presentada una descripción mas detallada de los algoritmos genéticos.

5.3. Algoritmos Genéticos

Los algoritmos genéticos utilizan una terminología originada de la teoría de la evolución natural y de la genética para simular sistemas genéticos. Los algoritmos genéticos procesan poblaciones, siendo que un individuo de la población es representado por un cromosoma, el cual contiene la codificación (*genotipo*) de una posible solución del problema (*fenotipo*). Los cromosomas son usualmente implementados en la forma de estructuras, listas de atributos o vectores, donde cada atributo es conocido como gene.

El proceso de evolución ejecutado por un algoritmo genético corresponde a un procedimiento de búsqueda en un espacio de soluciones potenciales para el problema. Esta búsqueda requiere un equilibrio entre dos objetivos:

- El aprovechamiento de las mejores soluciones;
- La exploración del espacio de búsqueda.

Los algoritmos genéticos pertenecen a la clase de algoritmos probabilísticos, siendo así, no son algoritmos puramente aleatorios, pues combinan elementos de dirección y búsqueda estocástica. Debido a esto, los algoritmos genéticos son más robustos, existiendo una dirección en el proceso de búsqueda. Otra propiedad importante de los algoritmos genéticos es que ellos mantienen una población de soluciones potenciales, al contrario de otros métodos que procesan un solo punto del espacio de búsqueda.

Como fue mencionado anteriormente, algoritmos genéticos realizan un proceso de búsqueda multi – direccional, manteniendo una población de soluciones potenciales intercambiando información entre si. La población es procesada mediante una evolución simulada cuyo objetivo es: a cada generación reproduce soluciones relativamente "buenas", mientras tanto son eliminadas soluciones relativamente "malas". Para distinguir entre las diferentes soluciones se utiliza una función de evaluación o de adaptabilidad (*fitness*) que simula el papel de la presión ejercida por el ambiente sobre los individuos.

La estructura de un algoritmo genético simple es la misma de un programa evolutivo como fue presentado en la Figura 5.1. Durante la iteración t, un algoritmo genético mantiene una población de soluciones potenciales (individuos, cromosomas, lista de atributos o vectores) $P(t) = \{x_1^t, ..., x_n^t\}$. Cada solución x_i^t es evaluada y produce una medida de su adaptación. Luego, una nueva población en la iteración $t + 1$ es formada privilegiando la participación de los individuos más adaptados. Algunos miembros de la nueva población pasan por alteraciones, por medio de operadores de cruzamiento y mutación, para formar nuevas soluciones potenciales. Este proceso se repite hasta que un número pre-determinado de iteraciones sea alcanzado, o hasta que un nivel de adaptación esperado sea alcanzado.

Consecuentemente, para un problema particular, un algoritmo genético debe tener los siguientes componentes:

- Una representación genética para soluciones candidatas o potenciales (proceso de codificación);
- Una manera de crear una población inicial de soluciones candidatas o potenciales;
- Una función de evaluación que hace el papel de la presión ambiental, clasificando las soluciones en términos de su adaptación al ambiente (es decir, su capacidad de resolver el problema);
- Operadores genéticos que alteran las composiciones de las generaciones;
- Valores para los diversos parámetros usados por el algoritmo genético (tamaño de la población, probabilidades de cruzamiento y mutación, etc)

5.3.1. Representación de Individuos (Codificación)

Cada individuo de una población representa un candidato potencial a la solución del problema en cuestión. En el algoritmo genético clásico propuesto por Holland (1975) las soluciones candidatas o potenciales son codificadas en estructuras binarias de tamaño fijo.

La representación binaria de los individuos en estructuras binarias del alfabeto {0, 1} es de la forma:

$$f:\{0,1\}^{l}\rightarrow\Re \qquad (5.1)$$

La representación binaria de los algoritmos genéticos utiliza funciones de codificación y decodificación h: $M\rightarrow\{0,1\}^{l}$ y h^{l}: $\{0,1\}^{l}\rightarrow M$ que facilitan el mapeo de soluciones $Z \in M$ para estructuras binarias $h(Z)\in\{0,1\}^{l}$ y vise versa.

En contraste con la representación de los individuos en algoritmos genéticos, estrategias evolutivas y programación evolutiva están directamente basadas en representaciones con vectores de valores reales, siendo generalmente aplicados para resolver problemas de optimización con parámetros continuos de valores reales, de la forma:

$$f : M\subseteq \Re^{n}\rightarrow\Re \qquad (5.2)$$

La representación o codificación de los individuos es una de las etapas más importantes cuando son definidos algoritmos genéticos. La definición inadecuada de la codificación puede llevar a problemas de convergencia prematura del algoritmo genético. La estructura de un cromosoma debe representar una solución como un todo, y debe ser la mas simple posible.

En problemas de optimización con restricciones, la codificación adoptada puede generar individuos modificados por los operadores de cruzamiento y mutación que sean inválidos. En estos casos, cuidados especiales deben ser tomados en la definición de la codificación y en la aplicación de los operadores.

5.3.2. Definición de la Población Inicial

El proceso de definición de la población inicial para algoritmos genéticos es simple, se crea una población de cromosomas inicializados aleatoriamente, siendo que cada cromosoma es compuesto por una estructura binaria de tamaño finito. En problemas con restricciones, se debe tomar cuidado para no generar individuos inválidos en la etapa de inicialización.

5.3.3. Selección

Una vez escogido el tipo de codificación del cromosoma, el paso siguiente consiste en la definición del método de selección a ser implementado, el cual envuelve la elección de los individuos en la población que crearan descendientes y cuantos descendientes serán creados. El objetivo de la selección consiste en privilegiar los individuos más adaptados al ambiente, consiguiendo que sus descendientes tengan un desempeño mucho mejor.

El proceso de selección debe ser tal que produzca un balance adecuado entre la presión selectiva y la variación introducida por los operadores genéticos. Por ejemplo, métodos de selección con elevada presión selectiva, tienden a generar súper individuos, es decir, individuos con adaptación superior a los demás, reduciendo la diversidad genética de la población. La presencia de un súper individuo puede generar una convergencia prematura del algoritmo genético. Entretanto, métodos de selección con baja presión selectiva tienden a producir progresos muy lentos en el proceso evolutivo. A seguir son descritos algunos de los métodos de selección mas comúnmente encontrados en la bibliografía.

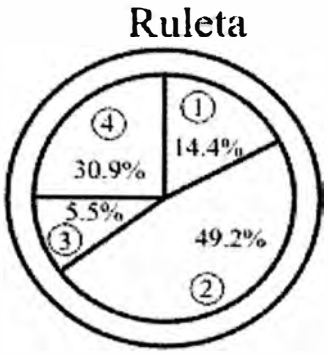
Selección Proporcional a la Función de Adaptación (Fitness)

El método de selección proporcional a la adaptabilidad también conocido como ruleta, atribuye a cada individuo de una población una probabilidad de pasar a la próxima generación proporcional a su adaptación o *fitness*. Consecuentemente, individuos con mayor valor de *fitness* tendrán mayor probabilidad de pasar a la próxima generación. Entretanto, este método de selección puede hacer que individuos con buena adaptación sean perdidos, es decir, no pasen para la próxima generación, esto debido a que este método de selección es típicamente implementado como un operador probabilistico.

Para una población de n individuos $P = \{h_1, h_2, \ldots, h_n\}$, en la cual el i - ésimo individuo h_i tiene asociado a el una medida de adaptación positiva y diferente de cero ($F(h_i) \in \Re_0^+$), la probabilidad $Prob_i$ para que este individuo sea seleccionado es dada por:

$$Prob_i = \frac{F(h_i)}{\sum_{j=1}^{n} F(h_j)} \qquad (5.3)$$

En la selección por ruleta, el *fitness* puede ser representado por segmentos consecutivos en una ruleta imaginaria y la probabilidad de elección de un segmento depende directamente de su tamaño. La Figura 5.2 presenta un ejemplo de cómo seria la ruleta para los individuos representados en la iteración t.



N°	String	*fitness*	%Total
$F(h_1^t)$	01101	169	14.4
$F(h_2^t)$	11000	576	49.2
$F(h_3^t)$	01000	64	5.5
$F(h_4^t)$	10011	361	30.9
Total		1100	100

Fig. 5.2.Selección por ruleta en algoritmos genéticos.

Selección basada en la clasificación (Rank)

La selección proporcional considerando la adaptación de los individuos puede ser problemática si los individuos de la población presentan desempeños muy próximos entre si. Además de eso, si el tamaño de la población es pequeño, la perdida de diversidad genética puede llevar a una convergencia prematura, pues la búsqueda queda reducida a pocos puntos, causando una disminución en el poder de exploración del algoritmo genético. Una opción para evitar el surgimiento de súper individuos (relativos a los demás existentes en la población actual) y la ocurrencia de convergencia prematura es reducir las diferencias entre estos, a través de un mecanismo de selección basado en *rank*. Esta estrategia utiliza las posiciones de los individuos cuando son ordenados de acuerdo con el *fitness* para determinar la probabilidad de selección. Pueden ser usados mapeos lineales o no lineales para determinar la probabilidad de selección.

Para una población de n individuos $P = \{h_1, h_2, \ldots, h_n\}$, son agrupados en orden creciente de adaptabilidad o *fitness*, tal que $F(h_i) < F(h_j)$, para todo $1 \leq i \leq j \leq n$. Se define SP como la presión de selección del ambiente sobre los individuos. En este caso, el valor de la función de adaptación puede ser calculado por:

Ranking Lineal: El ranking lineal permite valores de presión de selección SP entre [1, 2].

$$F(i) = 2 - SP + \frac{2.(SP-1).(i-1)}{n-1} \tag{5.4}$$

Ranking No Lineal: El ranking no lineal permite valores de presión de selección entre [1, n-2]

$$F(i) = \frac{n.v^{n-1}}{\sum_{j}^{n} v^{j-1}}; i = 1,\ldots,n \tag{5.5}$$

Donde v es calculado como la raíz del polinomio:

$$(SP-n)v^{n-1} + SP.v^{n-2} + \ldots + SP = 0 \tag{5.6}$$

En la Figura 5.3 es comparada la selección por clasificación lineal y no lineal, tales mecanismos de selección mejoran el comportamiento del algoritmo genético, con un control mejor de la presión de selección.

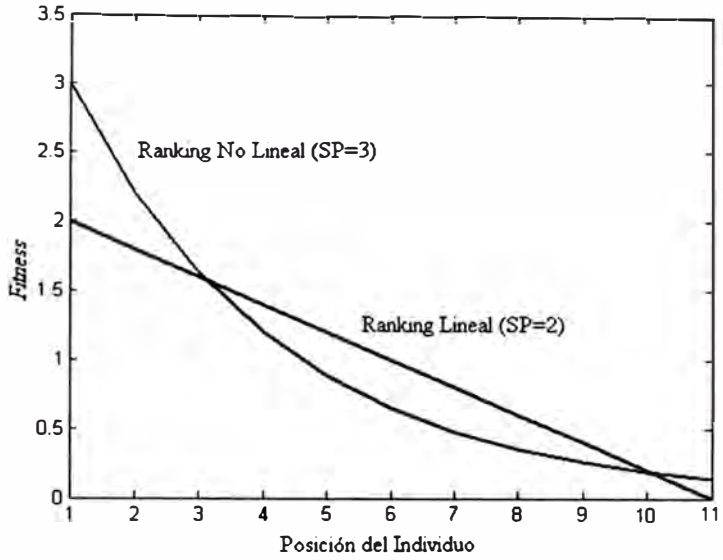


Fig. 5.3. Asignación del *fitness* para el ranking lineal y no lineal.

La selección por clasificación tiene la desventaja de exigir la ordenación de toda la población, lo que puede representar un costo computacional en algunas aplicaciones específicas.

5.3.4. Operadores Genéticos

Después de la elección de la codificación y método de selección a utilizar, el paso siguiente es el proceso de reproducción del algoritmo genético. El proceso de reproducción es responsable por la generación de las nuevas soluciones, producto de la aplicación de operadores genéticos junto a los individuos seleccionados. La variación genética en la población es un punto principal en la evolución, garantizando la exploración de nuevas soluciones. Consecuentemente, los operadores genéticos representan la fuente de la diversidad y variabilidad.

La elección de los operadores genéticos depende fuertemente de la codificación adoptada para la representación genética. Los operadores genéticos más comúnmente utilizados son el *cruzamiento* (también conocido como recombinación) y la *mutación*. El operador de cruzamiento combina los individuos intercambiando sus informaciones genéticas y también adiciona algunas innovaciones. En el proceso de cruzamiento hay una tendencia mayor de generar cromosomas con combinaciones de características existentes en la población, resultando en una convergencia más efectiva del proceso evolutivo. Entretanto, el operador de mutación introduce nuevas combinaciones genéticas hasta entonces inéditas en la codificación. En la mutación, informaciones que fueron pérdidas durante el proceso de evolución pueden ser rescatadas, o informaciones totalmente nuevas pueden ser introducidas.

A continuación se presentan los principales aspectos relacionados a los operadores cruzamiento y mutación.

Operador de Cruzamiento

El operador de cruzamiento es responsable por el intercambio de información genética entre los individuos de una población, produciendo nuevas soluciones candidatas o potenciales con algunas características existentes de los padres. El proceso de cruzamiento depende de la elección de los pares (o grupos) de individuos. La probabilidad de ocurrencia de recombinación entre los individuos de una población es denominada taza de cruzamiento. Esta probabilidad de cruzamiento generalmente varía entre 0.5 y 1. Entretanto, una alta probabilidad de cruzamiento hace con que individuos con una mayor adaptabilidad, sean eliminados antes que el proceso de selección pueda producir un perfeccionamiento. Por otro lado, una baja probabilidad de cruzamiento puede convergir lentamente debido a la baja taza de exploración de las características genéticas.

Cuando es utilizada codificación binaria en la implementación de los algoritmos genéticos, el operador de cruzamiento más simple es el cruzamiento de un punto. Para la aplicación de este operador, son seleccionados dos individuos (padres) y a partir de sus cromosomas (individuos, soluciones potenciales) son generados dos nuevos individuos (hijos). Para generar los hijos, se selecciona un mismo punto de corte aleatoriamente en los cromosomas de los padres, y los segmentos de cromosoma creados a partir del punto de corte son intercambiados. Este tipo de operador impone la quiebra de la secuencia genética definiendo cual es la proporción de información de los padres que cada descendiente recibirá.

Una extensión del operador de cruzamiento de un punto es el cruzamiento de dos puntos en el cual dos puntos de corte son escogidos y el material genético es intercambiado entre ellos. Otra variación es el operador cruzamiento de múltiplos puntos, el cual escoge de forma aleatoria más de una posición de intercambio de información.

Así también existen operadores de cruzamiento especialmente desarrollados para uso con codificación en punto flotante. Un ejemplo es el llamado *cruzamiento aritmético*. Este operador esta definido como una combinación lineal de dos vectores (cromosomas): sean x_1 y x_2 dos individuos seleccionados para cruzamiento, entonces los dos hijos resultantes serán $x_1^{'} = a.x_1 + (1-a).x_2$ y $x_2^{'} = (1-a).x_1 + a.x_2$, siendo a una variable aleatoria definida en el intervalo $a \in [0, 1]$. Este operador es particularmente apropiado para problemas de optimización numérica con restricciones, donde la región factible es convexa. Esto porque, si x_1 y x_2 pertenecen a la región factible, combinaciones convexas de x_1 y x_2 serán también factibles. Así, se garantiza que el operador de cruzamiento no genera individuos inválidos.

Entretanto, no hay ningún operador de cruzamiento que claramente presente un desempeño superior a los demás. Esto porque, cada operador de cruzamiento es particularmente eficiente para una determinada clase de problemas y extremadamente ineficiente para otras.

Operador de Mutación

El operador de mutación modifica aleatoriamente un o más genes de un cromosoma. La probabilidad de ocurrencia de mutación en un gene es denominada taza de mutación. Usualmente, son atribuidos valores pequeños para la taza de mutación. La principal contribución del operador es crear una variabilidad extra en la población, mas sin destruir el progreso obtenido con la búsqueda.

Considerando una codificación binaria, el operador de mutación simplemente intercambia el valor de un gene en un cromosoma. La aplicación del operador de mutación depende de la elección del punto de mutación o posición escogida aleatoriamente, siendo que todos los puntos tienen la misma probabilidad de ser escogidos. Así, si un gene es seleccionado para mutación y tiene el valor 1, su valor pasara a ser 0 después de la mutación, y vise versa.

En el caso de problemas con codificación en punto flotante, los operadores de mutación más populares son: *mutación uniforme*, *mutación no uniforme* y *mutación gaussiana*.

- El operador de *mutación uniforme*, selecciona aleatoriamente un componente $k(\{1, 2, \ldots, n\}$ del cromosoma $\mathbf{x} = [x_1 \ldots x_k \quad x_n]$ y genera un individuo $\mathbf{x}' = [x_1 \ldots x'_k \ldots x_n]$, siendo x'_k un numero aleatorio (con distribución de probabilidad uniforme) muestreado en el intervalo $x_k \in [x_{k\min}, x_{k\max}]$.
- El operador de *mutación no uniforme*, fue especialmente desarrollado para problemas de optimización con restricciones y codificación en punto flotante, destinada a realizar la sintonía fina a los individuos de la población.
- En el caso del operador de mutación gaussiana, todos los componentes de un cromosoma $\mathbf{x} = [x_1, \ldots., x_n]$ son modificados en la forma:

$$\mathbf{x}' = \mathbf{x} + N(0,\sigma) \qquad (5.7)$$

Donde $N(0, \sigma)$ es un vector de variables aleatorias gaussianas independientes, con media cero y varianza σ.

5.4. Algoritmo Evolutivo para Determinación de Aproximación de Funciones

Diversas metodologías pueden ser implementadas para determinar la aproximación de funciones relacionales (pares ordenados de la forma entrada – salida), por ejemplo aproximación utilizando: análisis de regresión, redes neurales, lógica nebulosa (*Fuzzy Logic*), computación evolutiva, etc.

En la determinación de aproximación de funciones se requiere un método automatizado que realice la búsqueda de las soluciones aproximadas, considerándose los siguientes aspectos de las funciones: no diferenciables, complejas, discontinuas y con espacio de búsqueda con múltiplos modos, es decir, varias soluciones mínimo o máximo locales. En este panorama se puede implementar un algoritmo genético para encontrar soluciones, con un proceso de evolución robusto de una representación de las soluciones.

Para la implementación de aproximación de funciones no lineales en sistemas embebidos de bajo costo con capacidad de cálculo en punto fijo, deben considerarse aspectos como:

- Incertezas asociadas a las variables libres
- Saturación en la representación de números
- Efectos de cuantización debido a la resolución de almacenamiento de los puntos de quiebra.

Funciones aproximadas pueden ser implementadas utilizándose aproximación por funciones lineales por partes, determinándose el tamaño de la tabla de equivalencia (LUT), que será utilizada. Consecuentemente, la implementación del algoritmo genético debe alcanzar los siguientes objetivos:

- Calcular el número y las posiciones de los puntos de quiebra necesarios para reproducir los valores aproximados de la función no lineal a partir de los puntos de quiebra.
- Determinar el tamaño de la tabla de equivalencia (LUT) en bits llevando en consideración aspectos como restricciones y límites para aproximación de funciones en sistemas embebidos.

5.5. Formulación para Determinar Aproximaciones de Funciones a través de Algoritmos Genéticos

Para la determinación de funciones no lineales por partes utilizando algoritmos genéticos, se debe realizar una formulación del problema que envuelve la definición de los cromosomas (individuos potenciales), método de selección, operadores de cruzamiento y mutación.

En la formulación y en la implementación del algoritmo genético utilizado, algunas extensiones fueron incluidas, con el objetivo de realizar el proceso de búsqueda de soluciones más eficiente. Esas extensiones envuelven la utilización de aproximación de funciones utilizando funciones lineales por partes y una normalización del problema para un mejor entendimiento de las soluciones en los límites de errores de aproximación.

5.5.1. Aproximación de Funciones No Lineales por Funciones Lineales por Partes

Una función no lineal definida por $y = f(x)$, puede ser aproximada a través de funciones lineales por partes. A función aproximada $s = \tilde{f}(x)$ es construida por partes a partir de valores almacenados en una tabla de equivalencia (LUT), que contiene m puntos de quiebra p_{xi} y p_{yi}, con $i = 1, \ldots, m$. Los puntos de quiebra p_{xi} son cuantizados con una resolución equivalente a N bits y los puntos de quiebra p_{yi} son determinados con una exactitud equivalente a una resolución de N_T bits de forma que los valores generados de la función tenga una resolución equivalente de N_Y bits (considerando las incertezas de propagación y de aproximación). La función no lineal es aproximada considerando una exactitud equivalente del sistema embebido de N_T bits de resolución, con $N_T \geq N_Y$, es decir, los puntos de quiebra son almacenados en la tabla de equivalencia con una resolución equivalente de N_T bits.

5.5.2. Normalización de las Soluciones

En la sección 4.6 fue definido el error de aproximación como $\varepsilon_A = s - y$, limitado por $[\varepsilon_{Amin}, \varepsilon_{Amax}]$. El problema de aproximación consiste en determinar valores de la función aproximada s tal que el error de aproximación ε_A se encuentre dentro de estos límites de aproximación. Para facilitar la interpretación de la aproximación con relación a los límites del error de aproximación de ε_A, una normalización de las soluciones fue establecida en la sección 4.7. Este proceso de normalización substrae de todas las variables los valores de la función cuantizada y_q a N_T bits de resolución. Con esta transformación, se desplazan las variables alrededor de cero, de esta manera el error de aproximación normalizado asume valores enteros limitados por un error normalizado E_{Amin} y E_{Amax}.

Una ventaja de utilizar esta normalización en el problema de aproximación de funciones es que se realiza la búsqueda de las soluciones de los puntos de quiebra a través de límites de errores de aproximación sin importar la forma de la función no lineal en cuestión.

5.6. Procedimiento de Solución

En la metodología propuesta se desarrolla un algoritmo evolutivo cuyo objetivo es de buscar una solución óptima del problema de aproximación de funciones. El algoritmo evolutivo desarrollado fue un algoritmo genético clásico y su desempeño fue evaluado a través de un estudio de caso. Los resultados presentados con el uso de este algoritmo fueron satisfactorios para casos con resoluciones de N y N_Y hasta 12 bits.

5.6.1. Algoritmo Jerárquico

El procedimiento propuesto posee dos niveles jerárquicos, como representado en la Figura 5.4. En el primer nivel se define el tamaño de la tabla de equivalencia (LUT) ($m \times N_T$), con m puntos de quiebra y una resolución equivalente de N_T bits. En el segundo nivel es implementado un algoritmo evolutivo, que consiste de un algoritmo genético clásico, que busca una solución factible para la condición especificada en el primer nivel.

Si la solución es encontrada, está es transferida al primer nivel, que procederá a reducir el tamaño de la tabla de equivalencia. Con esta nueva condición, el segundo nivel es nuevamente activado. Se una solución factible no es encontrada, se considera como resultado la ultima solución encontrada.

En la Figura 5.4 son presentadas las especificaciones de entrada del problema, tales como: el conjunto de valores cuantizados de la variable independiente x, el conjunto de valores cuantizados de la variable dependiente y y los errores de aproximación normalizados E_{Amin} y E_{Amax}, definido en la sección 4.7 por la ecuación 4.24.

Una vez que las especificaciones de entrada $\{x, y_q, E_{Amin}, E_{Amax}\}$ son transferidas para el algoritmo genético, las soluciones obtenidas para las especificaciones definidas son las posiciones de los puntos de quiebra (p_x, p_y) y el tamaño de la tabla de equivalencia ($m \times N_T$).

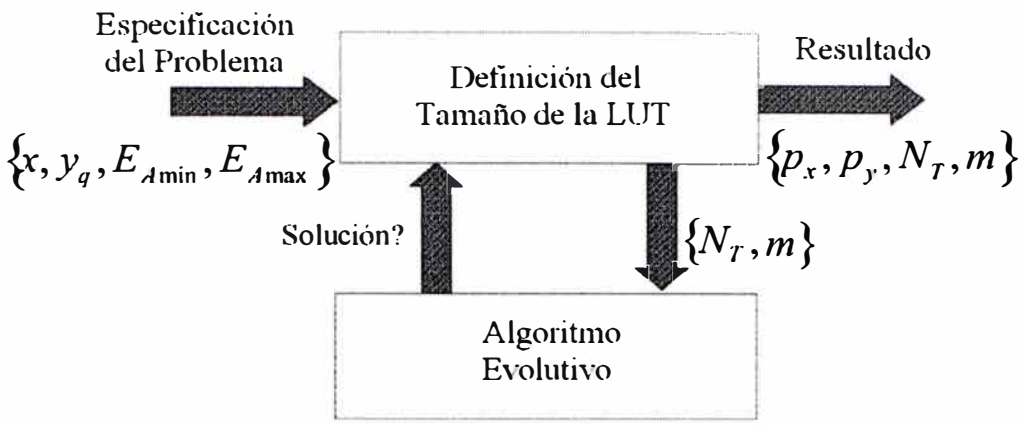


Fig. 5.4. Diagrama del Procedimiento Jerárquico de búsqueda.

El algoritmo de más alto nivel busca las soluciones de aproximación a partir de un par de resoluciones (N y N_Y) y número de puntos de quiebra inicial (m). Encontrando la solución para el tamaño de la LUT en bits ($N_T \times m$), seguidamente, el algoritmo disminuye un es esos parámetros y realiza una nueva búsqueda para encontrar una solución. Ese procedimiento es representado en la Figura 5.5.

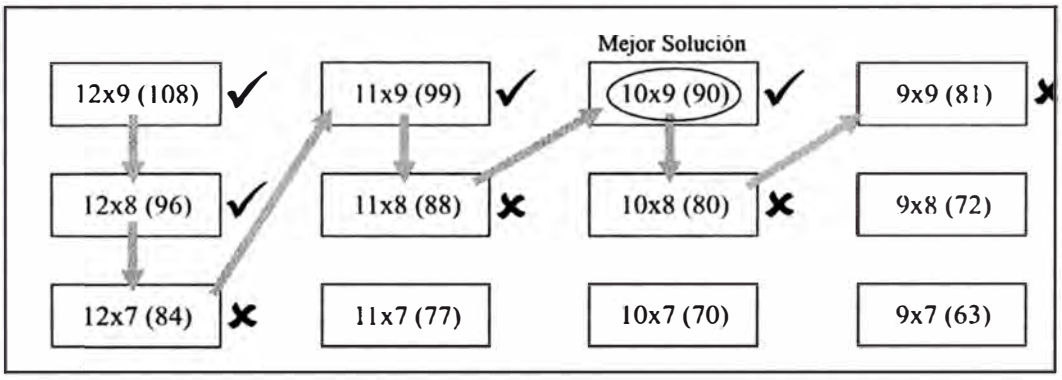


Fig. 5.5. Definición del tamaño de la LUT con solución para $N_T = 10$ y $m = 9$.

5.6.2. Algoritmo Genético Clásico

Para resolver el segundo nivel, se desarrollo inicialmente un algoritmo genético clásico, cuyos objetivos son determinar el mínimo tamaño de la tabla de equivalencia ($m \times N_T$), y determinar las posiciones de los puntos de quiebra con las especificaciones determinadas en el primer nivel de jerarquía.

El algoritmo genético realiza el siguiente procedimiento:

- El algoritmo se inicia con la creación de una población inicial aleatoria de individuos, representándose los individuos con una codificación de números enteros proyectados en una región viable;
- Se evalúa la función de adaptabilidad de cada individuo de acuerdo con un criterio predefinido. Después de la evaluación de la función de adaptación (*fitness*), los individuos

son seleccionados utilizando una puntuación basada en clasificación no lineal y selección a través de ruleta, así, los individuos que tienen mayor probabilidad pasaran su información genética para la siguientes generación;

- Se aplica la operación de recombinación a dos individuos seleccionados de acuerdo con una clasificación no lineal. La operación de recombinación intercambia el material genético entre los individuos produciendo soluciones candidatas o potenciales con características de los padres;
- Se aplica la operación de mutación a los individuos de la población que simulara una mudanza aleatoria de algunos bits en el cromosoma de los hijos. El operador de mutación provee un mecanismo para exploración de nuevas regiones del espacio de solución evitando una convergencia prematura para una solución mínima local;
- Se ordenan los individuos de la población actual, en base al valor de la función de adaptabilidad (*fitness*), siendo el objetivo principal mantener los mejores individuos para la siguiente generación.

Este proceso evolutivo es repetido para un número especificado de generaciones o hasta que algún criterio de parada sea alcanzado. Para el problema abordado en este trabajo, el algoritmo propuesto es descrito resumidamente a seguir:

```
Programa evolutivo
Inicio
   t ← 0
   inicializar P( t )
   evaluación P( t )
   mientras (no termina la condición de parada) hacer
   inicio
        t ( t + 1
        seleccionar P( t ) de P( t - 1)
        evaluación P( t )
   fin
fin
```

A continuación se desarrolla aspectos que envuelven la implementación del algoritmo genético clásico, como por ejemplo: representación de los individuos, determinación de la función de adaptabilidad, método de selección y operadores de reproducción genética.

Inicialización de la Población de Individuos

En la codificación se define la manera como los genes son almacenados en el cromosoma los cuales representan parámetros que serán optimizados por el algoritmo genético. En el algoritmo implementado se utiliza una representación de los cromosomas con codificación entera, en que cada cromosoma tiene un tamaño de m puntos de quiebra (p_{xi}, p_{yi}), $i = 1, \ldots, m$, representados por valores enteros con tamaños de palabras de N y N_T bits respectivamente, como representado en la Figura 5.6. El cromosoma es inicializado aleatoriamente, consecuentemente, proyectado dentro de una región viable de variación de los puntos de quiebra: $p_{xi} \in [0, 2^N\text{-}1]$ y $p_{yi} \in [0, 2^{N_T}\text{-}1]$, $i = 1,\ldots, m$.

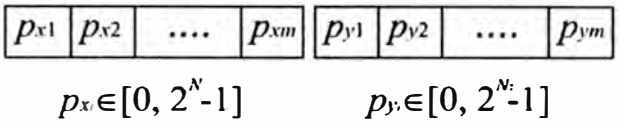


Fig. 5.6. Representación del Cromosoma.

El tamaño de la población inicial es definido por *Pop*, consecuentemente la población inicial se encuentra constituida por *Pop* cormos somos, en que cada cromosoma representa una solución candidata para el problema. En la Figura 5.7 se presenta la estructura de la población de individuos a ser optimizada por el algoritmo genético.

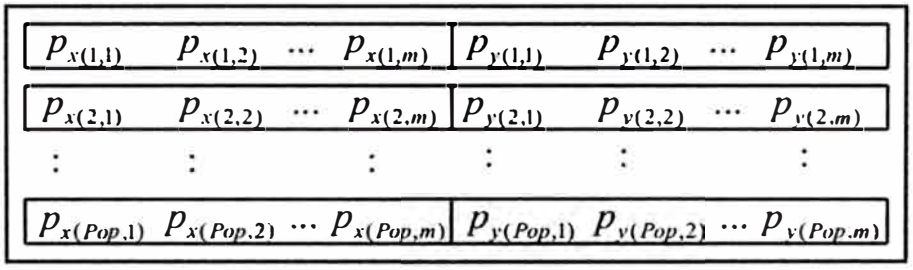


Fig. 5.7. Estructura de la población total de individuos.

Evaluación de la Función de Adaptabilidad (Fitness) de las Soluciones Candidatas

El error de aproximación de cada individuo de la población es dado por $\varepsilon_i = s_i - y_q$, donde s_i es la función de aproximación del individuo i, y_q es el valor cuantizado de la función y. El algoritmo genético tiene que minimizar el error de aproximación considerando sus límites. Para cada individuo, la función y el error de aproximación son calculados utilizando funciones lineales por partes. Como los límites del error de aproximación no son constantes para el intervalo de variación de x (debido a la propia cuantización y a la propagación de la incerteza en la variable independiente) el error de aproximación normalizado es determinado por:

$$\varepsilon_{AN}(x) = \left\{ \frac{\varepsilon_A(x)}{E_{A\max}(x)} \middle| \varepsilon_A(x) > 0 \;;\; \frac{\varepsilon_A(x)}{E_{A\min}(x)} \middle| \varepsilon_A(x) < 0 \right\} \qquad (5.8)$$

Así, cada individuo de la población tiene asociado una función de adaptabilidad, que refleje su superioridad, comparada con otras en la población. Se define la función de adaptabilidad como *fit*, para cada individuo *i*, determinada como una combinación de tres parámetros:

$$fit(i) = f_1(i) + \sqrt{f_2^2(i) + f_3^2(i)} / w_f \qquad (5.9)$$

Donde *wf* es un factor de ponderación, que fue ajustado para 40, y f_1, f_2 y f_3, son valores máximo, medio y varianza, respectivamente, dados por:

$$f_1(i) = \max\left(\varepsilon_{AN}^2\right) \; ; \; f_2(i) = mean\left(\varepsilon_{AN}^2\right) \; ; \; f_3(i) = std\left(\varepsilon_{AN}^2\right) \qquad (5.10)$$

La función de adaptabilidad, definida en la ecuación (5.9), fuerza al algoritmo a minimizar el máximo del error de aproximación normalizado considerando la minimización también de los valores medio y varianza de este error. El factor de ponderación tienen el objetivo de enmascaramiento del primer parámetro pro los otros dos.

Seguidamente es analizado el comportamiento de la función de adaptabilidad propuesta en la ecuación (5.9), para esto, se hace uso del caso de estudio presentado en la sección 4.10, en la cual fue analizado el error de aproximación de la función seno considerado la variable independiente *x* como exacta, es decir, $N = N_Y$. Fue considerado que la variable libre *x* es cuantizada con una resolución equivalente de $N = 4$ bits y los valores de la función aproximada son generados con una resolución de $N_T = 6$ bits.

Inicialmente se considera que los puntos de quiebra son determinados sobre la curva de la función cuantizada. En la Figura 5.8 se presenta el comportamiento de la función de adaptabilidad definido en la ecuación (5.9), en que los puntos p_x varían en el intervalo: $0 < p_{x1} < p_{x2} < 15$. En este caso se determina el valor mínimo de la función de adaptación $fit_{min} = 0.434$, con las soluciones de las posiciones de los puntos de quiebra en $p_x = [0, 7, 1215]$ y $p_y = [3, 43, 60, 63]$.

En la Figura 5.9 se presenta la función aproximada con valor mínimo de la función de adaptación. La función es aproximada por funciones lineales por partes a partir de las posiciones de los puntos de quiebra almacenados en una tabla de equivalencia de un sistema embebido. En la Figura 5.10 se presenta la normalización de la función aproximada, se observa que el error normalizado *S*, se encuentra dentro de los límites del error normalizado E_{Amin} y E_{Amax}. Se observa que el mapeamento de los valores de los puntos $p_{x1} = 7$ y $p_{x2} = 12$ se encuentran sobre el eje cero, pues los puntos p_{y1} y p_{y2} se encuentran determinados sobre la función cuantizada y_q.

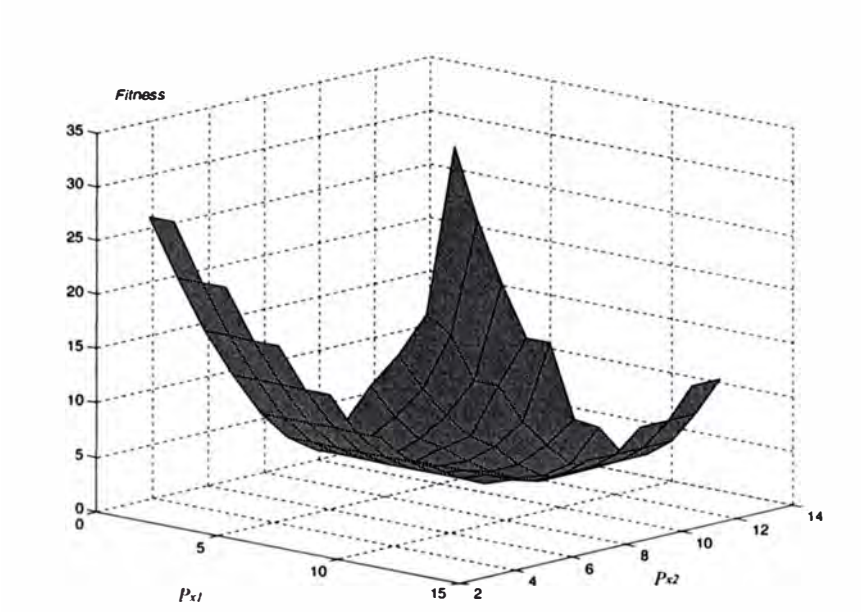


Fig. 5.8. Comportamiento de la función de adaptación con puntos de quiebra sobre la función cuantizada, con parámetros $N = N_Y = 4$, $N_T = 6$ y $m = 4$.

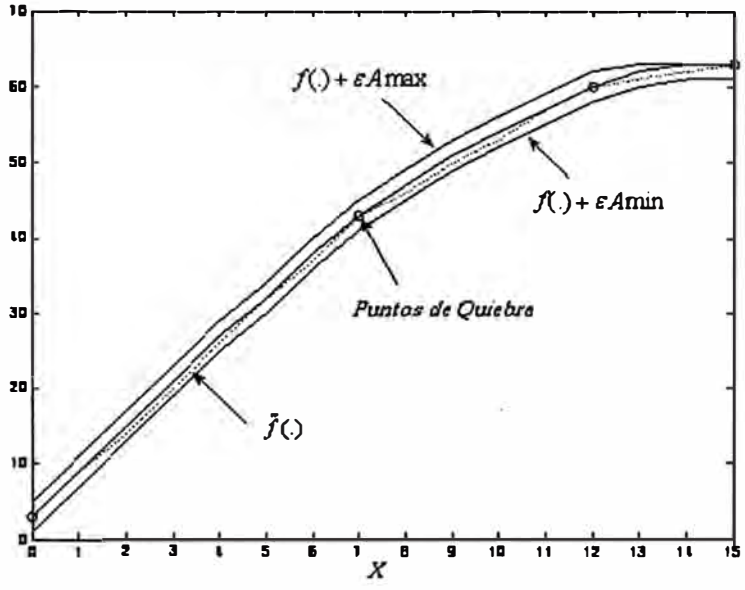


Fig. 5.9. Función "seno" en el primer cuadrante aproximado por funciones lineales por partes.

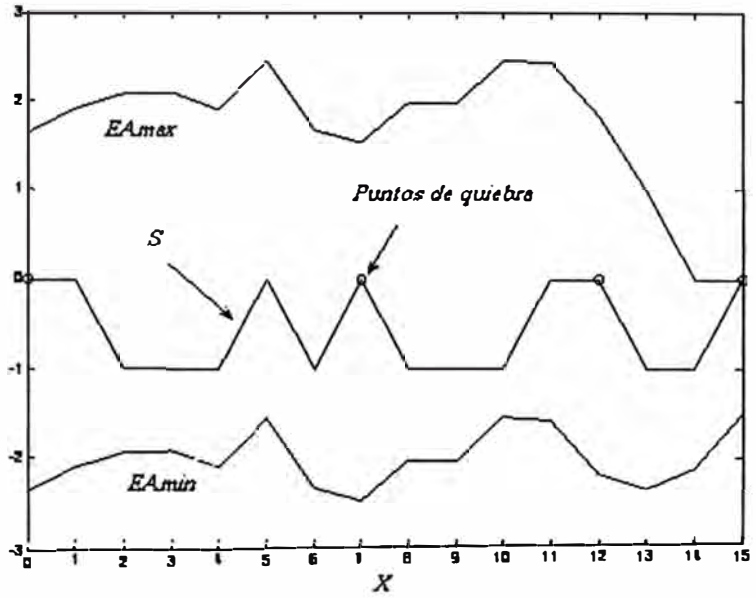


Fig. 5.10. Error de aproximación S y límites de error de aproximación $E_{A\min}$ y $E_{A\max}$ normalizados.

Método de Selección

El método de selección envuelve la elección de los individuos en la población que generaran descendentes, así como el número de descendente a ser creados. A partir de la población principal n_{ind} individuos son seleccionados para generar nuevos cromossomos, utilizando una puntuación basada en atributos no lineales y selección a través de la ruleta.

$$Prob(i) = \frac{n_{ind}.v^{n_{ind}-1}}{\sum_{j}^{n_{ind}} v^{j-1}} \quad ; \; i = 1,\ldots,n_{ind} \tag{5.11}$$

Donde v es la raíz real y positiva del polinomio.

$$(SP - n_{ind}).v^{n_{ind}-1} + SP.v^{n_{ind}-2} + \ldots + SP = 0 \tag{5.12}$$

Donde SP es el parámetro que controla a presión de selección.

Se considera un ejemplo con $n_{ind} = 40$, el individuo con mejor *fit* es localizado en la posición 1 y el individuo con peor *fit* es localizado en la posición 40, la probabilidad *Prob* de cada individuo en la población es calculado por la ecuación (5.11), donde el valor de la probabilidad depende de la posición de cada individuo en la población, como presentado en la Figura 5.11.

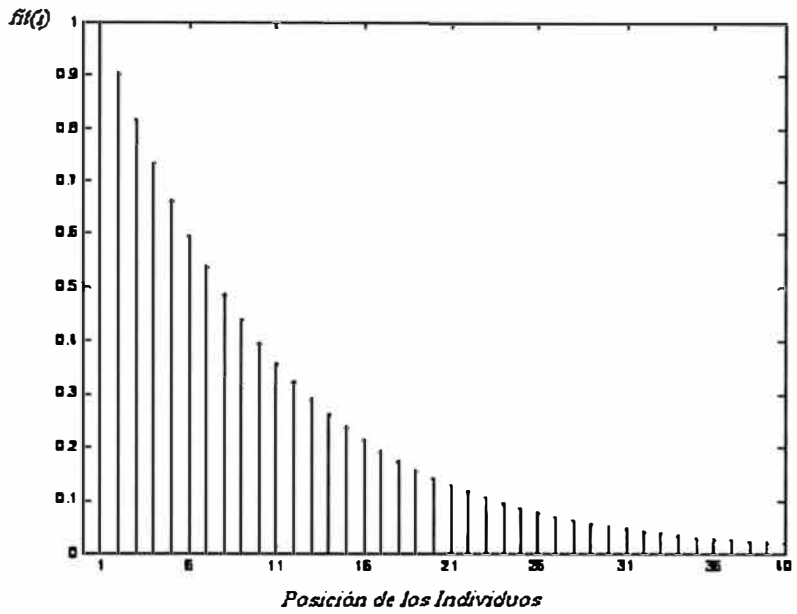


Fig. 5.11. Probabilidad de selección basada en la clasificación no lineal

Cruzamiento de los Individuos

El siguiente paso, después del proceso de selección, es aplicar la operación de cruzamiento en los individuos de la población. El cruzamiento intercambia parte de la información genética entre dos individuos, produciendo nuevas soluciones potenciales con algunas características de los padres.

En este algoritmo genético se aplica el operador de cruzamiento binario simple (de un punto) en que los cromosomas genitores son quebrados en una determinada posición (escogida aleatoriamente), siendo los segmentos resultantes permutados entre si, formando así dos descendientes.

En este algoritmo genético los cromosomas fueron definidos utilizándose una codificación entera, así para la aplicación del operador de cruzamiento binario simple, se requiere hacer el mapeo de la representación del cromosoma de entero para binario. Consecuentemente, en la representación binaria el cromosoma tiene un tamaño en bits de $m\times(N+N_T)$. La Tabla N° 5.1 contiene los parámetros y el tamaño en bits del cromosoma. Se observa que cuando las especificaciones del problema son mayores el tamaño del cromosoma en bits es muy grande. Esto es un inconveniente debido al esfuerzo computacional necesario para realizar la operación de cruzamiento.

Tabla N° 5.1. Tamaño del Cromosoma en bits

N	N_T	m	Tamaño en bits
4	7	4	44

8	10	9	162
9	11	13	260
10	12	17	374
11	14	22	550
12	15	31	837

Mutación de los Individuos

Después de la operación de cruzamiento los individuos de la población actual sufren un proceso de mutación, con el objetivo de introducir individuos con nueva codificación en la población, teniendo como consecuencia una mayor exploración del espacio de búsqueda.

En este sentido se hace un mapeo de individuos de representación entera para representación binaria. Seguidamente se aplica el operador de mutación binario intercambiándose los valores binarios de 1 para cero y vise versa. En la Figura 5.12 se presenta un ejemplo de una mutación binaria para una variable entera. Para todos los cromosomas, se define con que probabilidad (P_m) cada gene presente en el cromosoma será modificado.

Antes de la Mutación	114	1 1 1 0 0 1 0
Despues de la Mutación	98	1 1 0 0 0 1 0

Punto de Mutación

Fig. 5.12. Representación de la Operación de Mutación

Selección de la Nueva Población

Una vez concluida la selección probabilística basada en la posición a través del método de la ruleta y el proceso de reproducción, los mejores individuos son seleccionados para componer una nueva población. Este proceso de selección es del tipo $(\mu + \lambda)$-ES (ES – Estrategia Evolutiva), siendo que μ individuos producen λ hijos. La nueva población temporaria de $(\mu + \lambda)$ individuos es reducida por un proceso de selección deterministica, en que los μ mejores individuos pasaran definitivamente para la próxima generación.

CAPITULO VI
APLICACIONES

En esta sección, se presentan dos aplicaciones prácticas de las consideraciones y definiciones presentadas en las secciones anteriores. Primeramente, la aproximación de la función no lineal "seno" aproximada en el primer cuadrante utilizando funciones lineales por partes. Como segunda aplicación, la reconstrucción de medida de un sistema de medición de temperatura utilizando un Termistor. En ambas aplicaciones es utilizado el algoritmo genético desarrollado en la sección V, para determinar en ambos casos, el mínimo numero de puntos de quiebra (m) y la resolución necesaria para que sean almacenados en la tabla de equivalencia (N_T) en un sistema digital embebido. Para realizar las implementaciones fue considerado el procesador digital de señales DSP56811, disponible en los laboratorios del centro de investigación de la facultad de ingeniería eléctrica y electrónica.

6.1. Generación de Ondas Senoidales Utilizando una Tabla de Equivalencia Optimizada (LUT – "Look-Up Table")

Diversas aplicaciones de electrónica necesitan sistemas que generen señales o valores utilizando funciones no lineales. En sistemas digitales de posicionamiento, generalmente se necesita convertir la información de ángulo en posición, utilizando funciones trigonometricas. En instrumentación, diversas aplicaciones necesitan generan funciones no lineales para reconstrucción de los valores de medición, que pueden ser utilizadas para compensación de no linealidades del sensor.

En esta sección, para verificación de los algoritmos desarrollados, se considera el estudio de caso de aproximar una función seno en el primer cuadrante utilizando aproximación lineal por partes. Para dar solución a este problema, se utiliza el algoritmo genético descrito en la sección V, para determinara los puntos de quiebra y la memoria mínima necesaria de almacenamiento para resolver el problema de aproximación lineal de funciones por partes considerando las especificaciones de errores y propagación de incertezas para aproximación de funciones en sistemas embebido de bajo costo y resolución limitada, como presentado en la sección IV. Los valores digitales de la función seno son generados a partir de los puntos de quiebra solución obtenidos a partir del algoritmo desarrollado, que son almacenados en una tabla de equivalencia (LUT) del sistema embebido.

El procedimiento propuesto posee dos niveles de jerarquía. En el primer nivel se define el tamaño de la tabla de equivalencia ($m \times N_T$), con m puntos de quiebra y una resolución equivalente a

N_T bits. En el segundo nivel se implementa un algoritmo genético clásico que buscara una solución factible para la condición especificada en el primer nivel, como presentado en la Figura 6.1. Si la solución es encontrada, esta es derivada al primer nivel, que procederá a reducir el tamaño de la tabla de equivalencia. Con esta nueva condición, el segundo nivel es nuevamente activado. Si una solución factible no es encontrada, se considera como resultado la última solución encontrada.

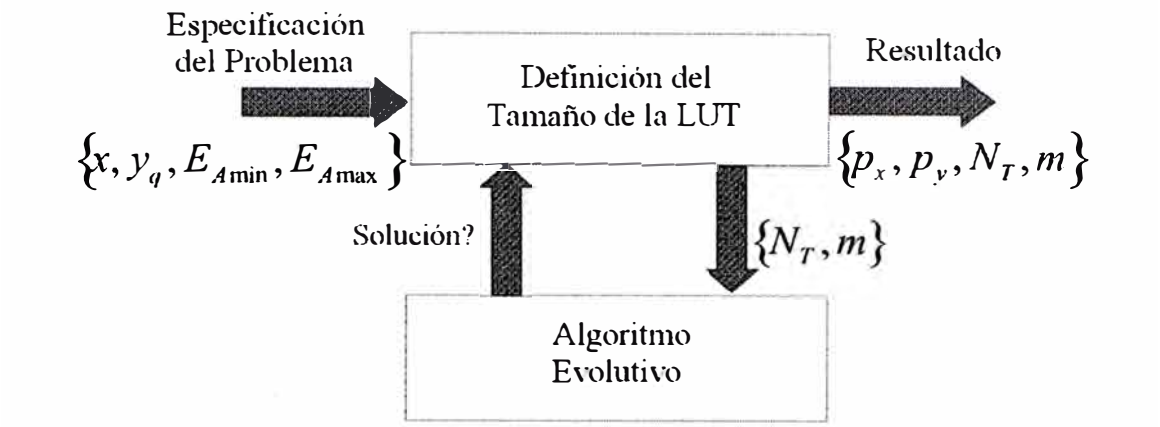


Fig. 6.1. Algoritmo jerárquico evolutivo compuesto por un algoritmo genético clásico

A continuación se presentan los resultados computacionales obtenidos para el estudio de caso de aproximación de la función seno en el primer cuadrante. Se considera la función seno definida como $y = \text{sen}(x)$, con limites $x \in [0, \pi/2]$ y $y \in [0, 1]$. Se analiza el caso en que no existe incerteza asociada a la variable libre, es decir $N_Y = N$.

Los resultados presentados con el uso de este algoritmo fueron satisfactorios para casos de resoluciones de N y N_Y hasta 12 bits. En la Tabla N°6.1 se resume los parámetros y resultados utilizados en las simulaciones y el tamaño mínimo de la tabla de equivalencia (LUT) en bits, obtenido para cada aproximación especificada. En cada caso se utiliza una probabilidad de cruzamiento $P_c = 0.9$ y una probabilidad de mutación $P_m = 0.04$. Para cada especificación de parámetros se determina un número de individuos de la población. Las simulaciones fueron realizadas utilizando el programa MATLAB para un computador Pentium IV, 3 GHz y memoria RAM de 512MB.

Tabla N°6.1. Parámetros y resultados de la aproximación utilizando el algoritmo genético

Parámetros de entrada			Resultados			
N	N_Y	Población	N_T	Puntos de quiebra: m	Tamaño de la LUT (bits)	Tiempo computacional por época (s)
8	8	40	10	9	90	0,188
9	9	40	11	13	143	0,280
10	10	50	12	17	204	0,485
11	11	60	14	22	308	1,93
12	12	80	15	31	465	6,15

A continuación, se presentan los resultados de la aproximación de la función seno con parámetros $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$. En la Figura 6.2 se presenta una solución para la representación de la aproximación de la función seno, siendo $S = y - y_q$ el error de la función de aproximación normalizado, E_{Amin} y E_{Amax} los limites de errores de aproximación normalizados. El valor de *fitness* obtenido fue de 0.9627. Se observa que la solución S no sobrepasa los límites de los errores de aproximación, condición establecida por el algoritmo para determinar el conjunto de puntos de quiebra como soluciones.

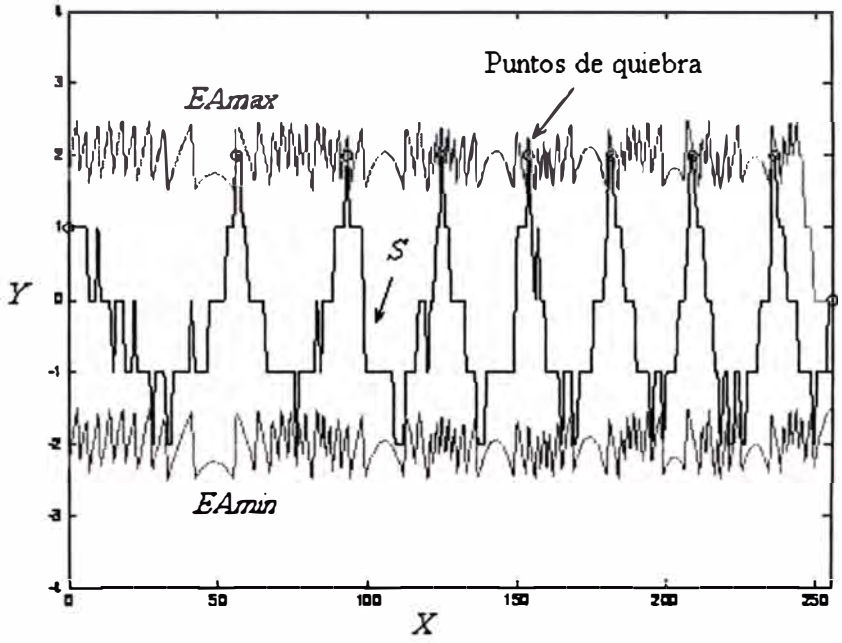


Fig. 6.2. Aproximación para la función seno con $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$.

En la Tabla N° 6.2 se resume los puntos de quiebra (p_x, p_y) solución y el error de la función aproximada S encontrada por el algoritmo jerárquico evolutivo utilizando el algoritmo genético clásico.

Tabla N° 6.2. Puntos de quiebra para $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$

Puntos de quiebra (m)	1	2	3	4	5	6	7	8	9
p_x	0	56	93	125	154	182	209	236	255
p_y	4	349	557	714	833	923	984	1018	1023
S	1	2	2	2	2	2	2	2	0

En la Figura 6.3 se presenta la implementación de los puntos de quiebra en una tabla de equivalencia (LUT) con parámetros $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$. A partir de los valores almacenados en la LUT los otros valores de la función seno pueden ser obtenidos por aproximación lineal por partes, procedimiento descrito en la sección IV.

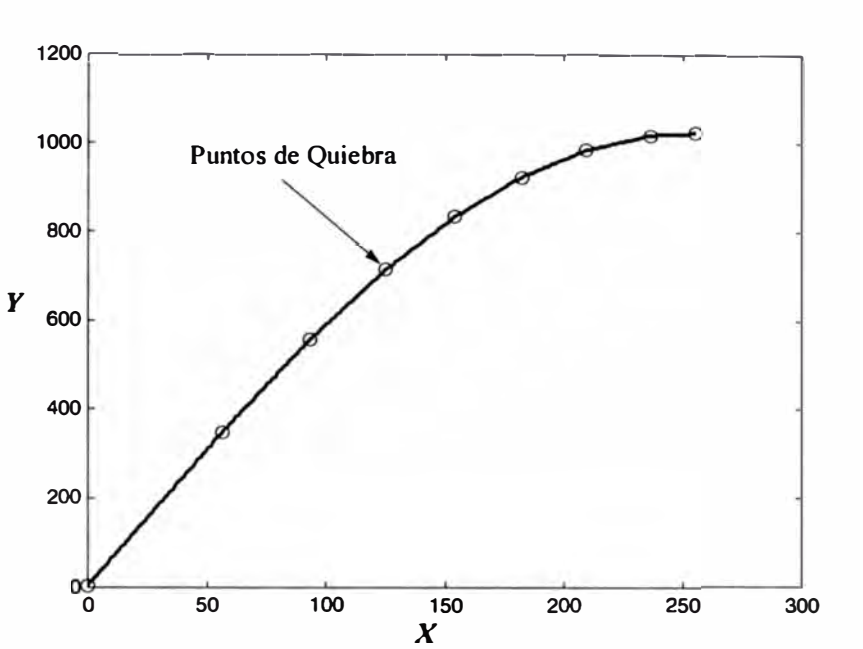


Fig. 6.3. Puntos de quiebra implementados en una tabla de equivalencia (LUT) con parámetros $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$.

Una vez determinada la aproximación lineal por partes de la función seno en el primer cuadrante, es posible generar la forma de onda de una senoidal completa, a partir de los puntos de quiebra almacenados en la memoria del sistema embebido. Para esto se define la variable $X \in [0, 255]$, que es el valor de entrada a la LUT para generar los valores en el primer cuadrante. Así también, se define la variable $K \in [0, 1023]$, que son los valores que permitirán generar la onda senoidal completa utilizando solamente la onda senoidal en el primer cuadrante. A partir del reracionamiento de las variables X y K es posible generar la onda senoidal completa, como se presenta en la Figura 6.4:

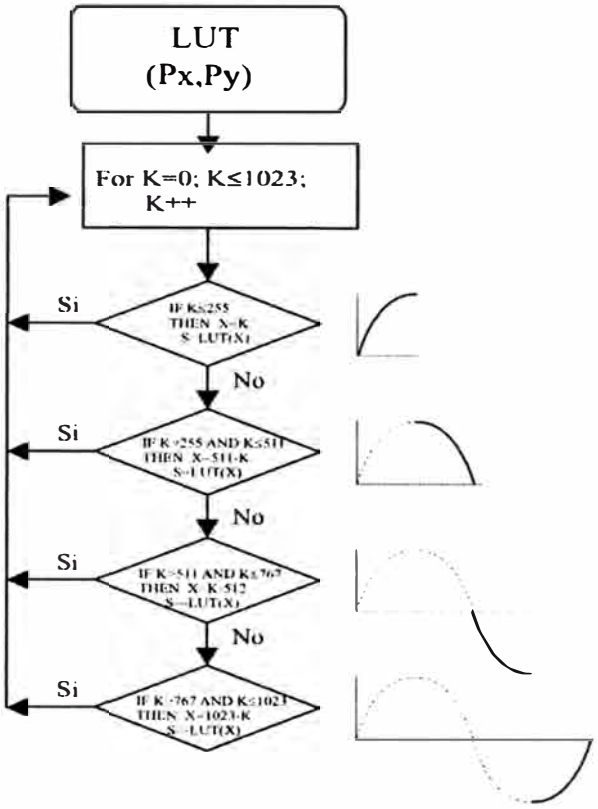


Fig. 6.4. Generación de la onda senoidal completa.

En la Figura 6.5 se presenta la manera de acceso de las variables X y K a la memoria interna del sistema embebido para generar la onda senoidal completa.

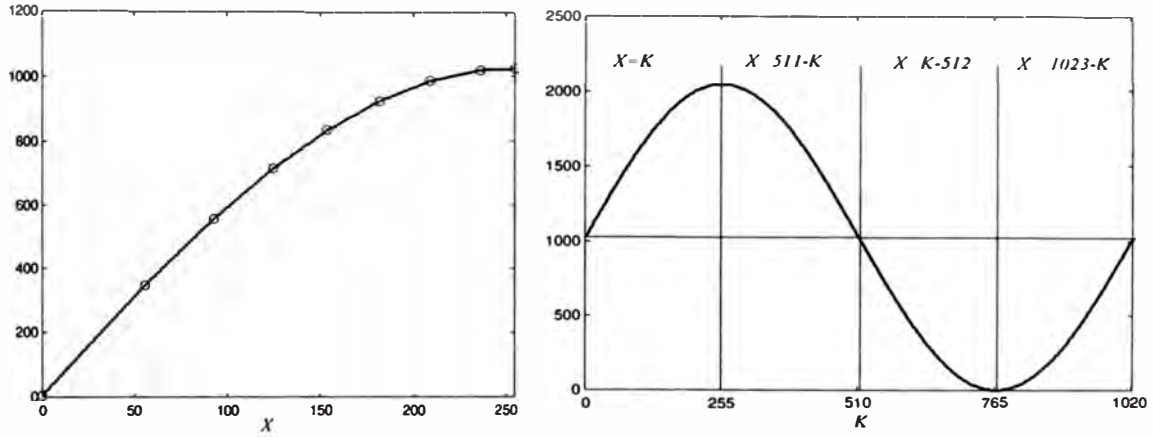


Fig. 6.5. Generación de la onda seno completa a partir de un cuarto de onda

Se debe destacar que en la generación de la onda senoidal completa fueron utilizadas las soluciones de los puntos de quiebra encontradas por el algoritmo evolutivo propuesto, para la aproximación de la función seno el primer cuadrante. Consecuentemente, se pueden generar funciones senoidales a partir de los puntos de quiebra de un cuarto de onda seno almacenados en la memoria del sistema embebido, con la ventaja de utilizar menos espacio de memoria, que a comparación con soluciones convencionales de generadores de ondas digitales, almacenan todos los

valores de la función seno, con un uso considerable del espacio de la memoria disponible en los procesadores de sistemas embebidos.

La frecuencia de la forma de onda digital generada es definido por:

$$f = \frac{1}{nT}\ (Hz) \tag{6.1}$$

Donde:

n : Numero de componentes generados por la tabla de equivalencia

T : Intervalo de tiempo entre las componentes sucesivas de la LUT (en segundos).

Para el caso de estudio la tabla de equivalencia genera n = 256 puntos, que a partir de ellos se genera la onda completa. Para una onda senoidal de frecuencia f = 60 Hz es necesario definir un tiempo de acceso a la memoria de la LUT de T = 65.104μseg.

6.1.1. Análisis del Error de Cuantificación

A partir de las soluciones de los puntos de quiebra encontradas por el algoritmo evolutivo se realiza un análisis de los errores de cuantificación, determinando los efectos de cuantificación debido al funcionamiento del conversor analógico – digital (A/D).

El proceso de codificación del conversor A/D asigna un número binario a cada nivel de cuantificación. Así para L niveles, se necesitan por lo menos L números binarios diferentes. Con una longitud de $b + 1$ bits se puede representar 2^{b+1} números binarios distintos. Consecuentemente, se debe tener $2^{b+1} \geq L$ o, equivalentemente $b+1 \geq \log_2 L$. Entonces, el tamaño del escalón o la resolución del conversor A/D es definido por:

$$\Delta = \frac{R}{2^{b+1}} \tag{6.2}$$

Donde R es el rango del cuantificador.

El modelo matemático para el error de cuantificación $e_q(n)$ se muestra en la Figura 6.6. Para llevar a cabo el análisis hacemos las siguientes suposiciones sobre las propiedades estadísticas.

1. El error $e_q(n)$ se distribuye uniformemente sobre el rango $-\Delta/2 < e_q(n) < \Delta/2$
2. La secuencia del error $\{e_q(n)\}$ es una secuencia estacionaria de ruido blanco
3. La secuencia de error $\{e_q(n)\}$ está incorrelada con la secuencia $x(n)$.
4. La secuencia $x(n)$ tiene media cero y es estacionaria.

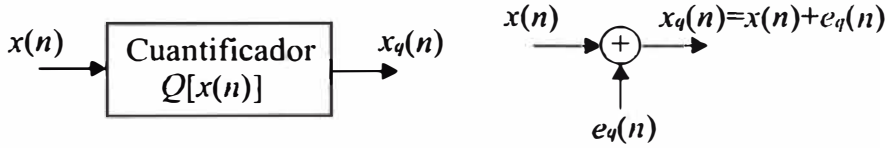


Fig. 6.6. Modelo matemático del ruido de cuantificación

Bajo estas consideraciones, el efecto del ruido aditivo $e_q(n)$ en la señal deseada, se puede cuantificar evaluando la relación (potencia) señal a ruido de cuantificación (SQNR), que se puede expresar en escala logarítmica (en decibelios dB) como:

$$SQNR = 10\log\left(\frac{P_x}{P_n}\right) \tag{6.3}$$

Donde $P_x = \sigma_x^2 = E\left[x^2(n)\right]$ es la potencia de la señal y $P_n = \sigma_e^2 = E\left[e_q^2(n)\right]$ es la potencia del ruido de cuantificación.

Si el error de cuantificación (e_q) se distribuye uniformemente en el rango (-Δ/2; Δ/2) como se muestra en la Figura 6.7, el valor medio del error es cero y la varianza (potencia del ruido de cuantificación) es definido por:

$$P_n = \sigma_e^2 = \int_{\Delta/2}^{\Delta/2} e^2 p(e)de = \frac{1}{\Delta}\int_{\Delta/2}^{\Delta/2} e^2 de = \frac{\Delta^2}{12} \tag{6.4}$$

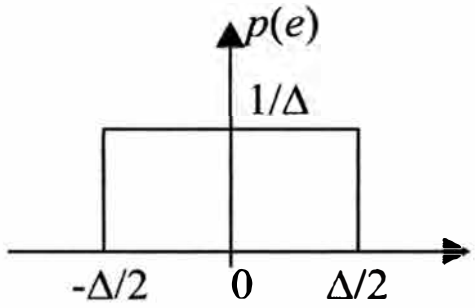


Fig. 6.7. Función de densidad de probabilidad para el error de cuantificación

Combinando (6.2) con (6.4) y sustituyendo en (6.3), la expresión para la relación señal a ruido (SQNR) se convierte en:

$$SQNR = 10\log\frac{P_x}{P_n} = 20\log\frac{\sigma_x}{\sigma_n} \tag{6.5}$$

$$SQNR = 6.02b + 16.81 - 20\log\frac{R}{\sigma_x}\ (dB) \tag{6.6}$$

El último término de (6.6) depende del rango R del cuantificador del conversor A/D y de las propiedades estadísticas de la señal de entrada (varianza σ_x). Asumiendo que $x(n)$ tiene distribución Gausiana y el rango del cuantificador se extiende desde $-3\sigma_x$ hasta $3\sigma_x$ (es decir $R = 6\sigma_x$), la ecuación (6.6) se convierte en:

$$SQNR = 6.02b + 1.25\ (dB) \tag{6.7}$$

En esta última ecuación (6.7) se observa que a cada *bit* adicional en el cuantificador, se incrementa la relación señal a ruido de cuantificación en 6 dB aproximadamente. En la Tabla N°6.3 se resume los valores de *SQNR* para diferentes valores de resolución (en bits) del cuantificador.

Tabla N°6.3. Relación señal a ruido para diferentes resoluciones

N (bits)	*SQNR* (dB)
8	49.41
9	55.43
10	61.45
11	67.47
12	73.49

Como resultado, el número efectivo de bits de alguna manera es menor que el número de bits en el conversor A/D, debido a la presencia del ruido generada en la cuantificación de la función aproximada, es decir para un conversor de 10 bits puede tener solo 8 bits efectivos de precisión.

6.1.2. Implementación de la Función Seno Aproximada utilizando un Procesador Digital de Señales DSP56811

Para validar los aspectos teóricos y prácticos presentados de este trabajo, es realizada la implementación de la tabla de equivalencia (LUT) en un procesador digital de señales (DSP) de la familia DSP56800 de Motorola, procesador de punto fijo, utilizado como una unidad de procesamiento de propósito general (CPU), diseñado para un control eficiente de operaciones para el procesamiento digital de señales (ver **ANEXO A**). Primeramente, es implementada la LUT para la generación de la función seno en el primer cuadrante a partir de los puntos de quiebra soluciones encontradas por el algoritmo jerárquico evolutivo. Seguidamente, a partir de la generación de los valores de la función seno en el primer cuadrante, es generada la onda seno completa (un periodo), utilizando las propiedades e identidades trigonometricas.

- ***Implementación de la Función Seno en el Primer Cuadrante***

La aproximación de la función seno en el primer cuadrante es implementado utilizando los puntos de quiebra (p_x, py) solución encontrados para las especificaciones $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$.

A continuación se presenta el código fuente para esta especificación, escrito en C – ANSI utilizando características específicas para su implementación en el procesador digital de señales DSP56811 (A este código fuente es asignado el nombre de "**LUTsenoC.c**"). En la Tabla N°6.4 son definidas las variables utilizadas en el código implementado.

Tabla N°6.4. Variables utilizadas en el código fuente implementada.

Variable	Representación
input	Valor de entrada a la LUT
s	Valor de la función aproximada $s = f(\text{input}) = coef\_a + coef\_b(\text{input}-p_x)$
m	Numero de puntos de quiebra
p_x, p_y	Puntos de quiebra
coef_a, coef_b	Coeficientes de la función lineal por partes

```
/*************************************************************/
/*LUTsenoC.c*/
/*************************************************************/
#include <stdio.h>
#include <dsp56811.h>
int main()
{
        int input=236,m=9,s,i;
        /*****************************************************/
        /*PUNTOS DE QUIEBRA m=9*/
        /*****************************************************/
        int px[9]={0,56,93,125,154,182,209,236,255};
        int py[9]={4,349,557,714,833,923,984,1018,1023};
        /*****************************************************/
        /*COEFICIENTES*/
        /*****************************************************/
        int coef_a[8]={4,349,557,714,833,923,984,1018};
        int coef_b[8]={6,6,5,4,3,2,1,0};
        /*****************************************************/
        /*Funcion por partes s=f(x)=a+b(x-px)*/
        /*****************************************************/
        for (i=1;i<m;i++)
        {
                if(input>=px[i-1] & input<px[i]){
                s=coef_a[i-1]+coef_b[i-1]*(input-px[i-1]);
                }
```

```
        }
        if(input>=px[m-1]){
                s=py[m-1];
        }
        While(1);
}
/*************************************************************/
```

Seguidamente, el código fuente es compilado y enlazado utilizando el programa enlazador, propio de la familia de procesadores DSP56800, "m568c":

C:>\m568c –g –mr LUTsenoC.c

El proceso de compilación genera el siguiente archivo: "**LUTsenoC.CLD**", que es la versión ensamblada usada por el procesador (hardware). Una vez realizados los pasos anteriores, es ejecutado el DEBUG-EVM para depurar el programa, cargando el archivo LUTsenoC.CLD en el *debuger* y transfiriendo el programa a la tarjeta DSP56800EVM. En la Figura 6.8 se presenta este procedimiento de compilación y depuración del código fuente implementado.

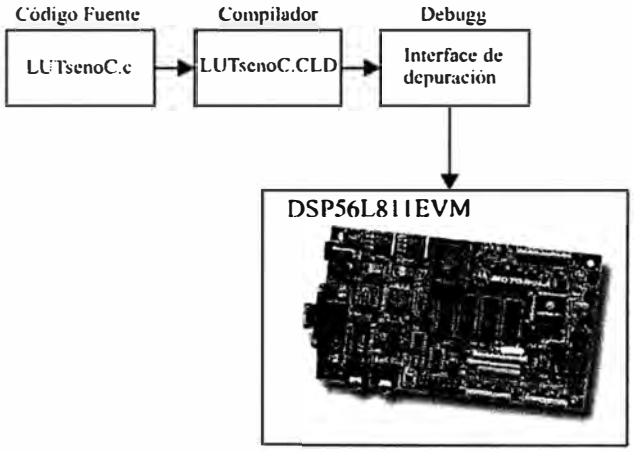


Fig. 6.8. Proceso de compilación y depuración del código fuente implementado.

Como ejemplo de aplicación, se asigna a la variable de entrada el valor de input = 236, y a través del algoritmo implementado, este debe entregar como resultado en la variable de salida el valor de s = 1018, como presentado en la Figura 6.9.

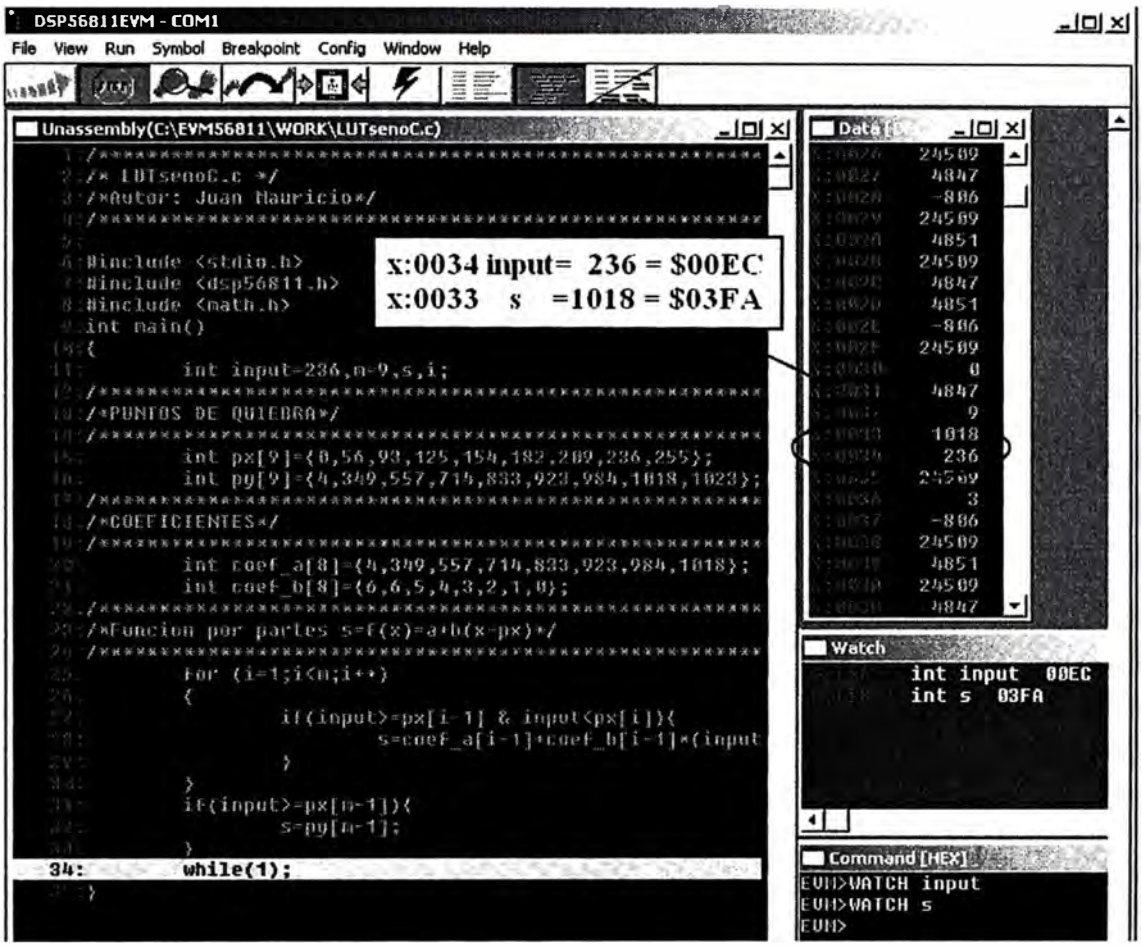


Fig. 6.9. Archivo LUTseno.CLD transferido al DEBUG - DSP56811EVM.

En la Figura 6.9, se observa cuatro secciones utilizadas para depurar el código fuente:

- *Memoria de programa* (*Unassembly*), en la cual se encuentran las instrucciones de código, que realizaran la implementación del algoritmo en cuestión. Como se observa el codigo fuente implementado es cargado y ejecutado en el DEBUG.
- *Memoria de datos* (*Data*), en la cual se encuentran los datos utilizados por la memoria de programa. Aquí se observa que los registros de datos con direcciones x:0034 y x:0033 contiene los datos de entrada (input) y salida (s), respectivamente.
- *Command,* sección utilizada para ejecutar instrucciones en línea de comando, para acceder o modificar los valores de registros de datos.
- *Watch,* sección utilizada para visualizar los valores de los registros de datos. Como se observa en la Figura 6.9, los valores representados se encuentran en base hexadecimal.

- ***Implementación de la Función Seno Completa***

A partir de la implementación de la aproximación de la función seno en el primer cuadrante, el código fuente es ampliado con el objetivo de generar una onda senoidal completa. Adicionalmente, es utilizado el CODEC de audio del procesador digital DSP56811, con la finalidad de transferir las ondas senoidales generadas en el procesador para una computadora (PCs) a través del puerto de entrada de audio del computador, como representado en la Figura 6.10. Una vez

transferidas las formas de ondas para un computador estas son almacenadas y analizadas utilizando el programa MATLAB para ondas senoidales con diferentes frecuencias.

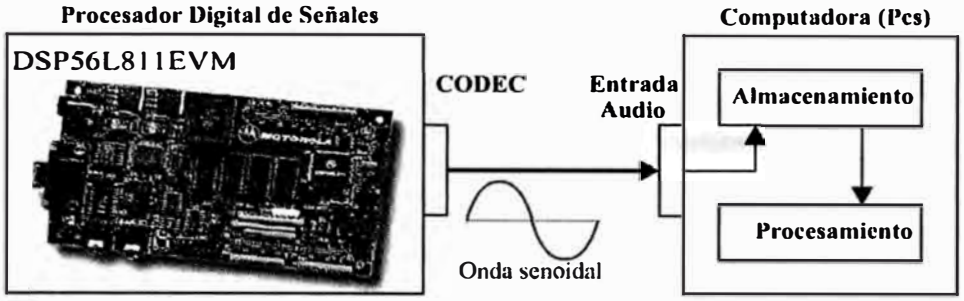


Fig. 6.10. Análisis de las ondas senoidales generadas en el DSP56811

A continuación se presenta el código fuente para generar una onda seno completa con $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 9$, escrito en C – ANSI utilizando características específicas para su implementación en el procesador digital de señales DSP56811 (A este código fuente es asignado el nombre de "**LUTsenoC.c**").

```
/* LUTsenoC.c */
#include <stdio.h>
#include <dsp56811.h>
#include <math.h>
unsigned int LUT(unsigned int input);
void main(void){
    unsigned int s,k,x;
    while(1)
    {
        for(k=0;k<1024;k++){
            /*Primer Cuadrante*/
            if (k<=255){
                x = k;
                s = 1024 + LUT(x);}
            /*Segundo Cuadrante*/
            if (k>255 & k<=511){
                x = 511 - k;
                s = 1024 + LUT(x);}
            /*Tercer Cuadrante*/
            if (k>511 & k<=767){
                x = k - 512 ;
```

```
                    s = 1024 - LUT(x);}
                /*Cuarto Cuadrante*/
                if (k>767 & k<=1023){
                    x = 1023 - k;
                    s = 1024 - LUT(x);}
            }
        }
}
unsigned int LUT(unsigned int input){
        unsigned int px[9]={0,56,93,125,154,182,209,236,255};
        unsigned int py[9]={4,349,557,714,833,923,984,1018,1023};
        unsigned int m=9,i,salida;
        /*Coeficientes*/
        unsigned int coef_a[8]={4,349,557,714,833,923,984,1018};
        unsigned int coef_b[8]={6,6,5,4,3,2,1,0};
        /*Funcion por partes s=f(x)=a+b(x-px)*/
        for (i=1;i<m;i++){
            if(input>=px[i-1] & input<px[i]){
                salida=coef_a[i-1]+coef_b[i-1]*(input-px[i-1]);}
        }
        if(input>=px[m-1]){
            salida=py[m-1];
        }
        return salida;
}
```

El almacenamiento de la onda seno es realizado por la "*Grabadora de Sonidos*", programa accesorio del sistema operativo en uso. Las características de este gravador es que la frecuencia de muestreo es de f_s = 44.1 KHz, con codificación PCM a 16 bits, estéreo.

Generación de una Onda Seno con frecuencia 1Hz

Para analizar la forma de onda senoidal, se realizo la grabación de la señal por tiempo de 1.75seg, considerando la frecuencia de muestreo de 44.10KHz, se tiene un total de 1.75*44100 = 77175 puntos almacenados durante la grabación de la señal.

Fig. 6.11. Grabación de la señal por un tiempo de 1.75seg.

A continuación se presenta el código en MATLAB para la lectura del archivo de sonido, y el procedimiento para graficar la señal en el dominio del tiempo, a partir de los valores almacenados durante la grabación de la señal.

```
[y,fs,nbits]=wavread('c:\salida\sonido');
b = [y(:,1)];
figure
plot(b,'k'),title('Seno')
t=1:length(b);
t=t/44100;                %Normalizacion (seg)
figure
plot(t,round(b8*1000)/1000,'k'),title('Normalizado - Seno')
```

En la Figura 6.12 se presenta la forma de onda senoidal generada, para una frecuencia de 1Hz. Se debe observar que ha sido considerada la frecuencia de muestreo de 44.100KHz, utilizada por el grabador de sonido.

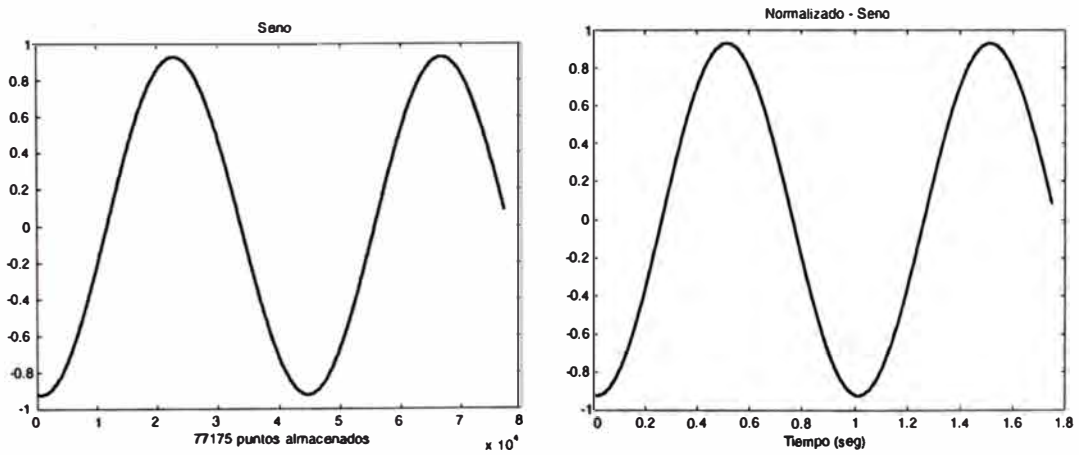


Fig. 6.12. Generación de la forma de onda seno con una frecuencia de 1Hz.

En la Figura 6.13 se presenta el error de aproximación *S* para la forma de onda seno para un periodo completo. Se observa que el error de aproximación *S* se encuentra dentro de los errores mínimo y máximo, generados debido a la cuantificación,

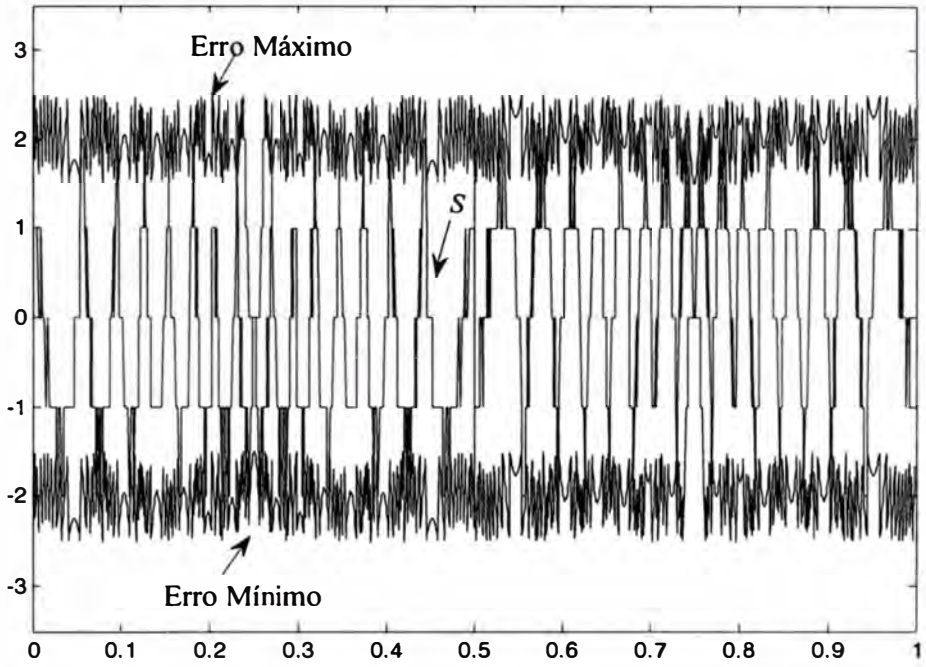


Fig. 6.13. Error de aproximación para un periodo de la forma de onda seno completa.

Una manera de cuantificar los errores producidos debido a los efectos de la cuantificación del conversor analógico digital, es determinar los valores de la media y varianza de las señales de errores. La *media* o esperanza es una medida de donde se centra la distribución. La *varianza* mide la dispersión de la distribución en torno a la media. Cuanto menor es la dispersión, tanto más pequeña es la varianza. Esta es también una medida de la potencia de ***ca*** (corriente alterna) de una señal. El valor de la varianza se conoce como la desviación estándar y proporciona una medida de la incertidumbre en una medición física. En la Tabla N° 6.5 se resumen los valores de la media y varianza de los errores de aproximación S, error máximo de aproximación y el error mínimo de aproximación, respectivamente. A partir de este resultado, se comprueba que el error presenta una varianza aproximadamente de 2, lo que concuerda con el valor observado en la Figura 6.13.

Tabla N° 6.5. Media y Varianza de los errores de aproximación

	Media	**Varianza**
***S*: Error de aproximación**	0	1.0920
Error máximo de aproximación	-2.9967*10-16	1.9645
Error mínimo de aproximación	1.0538*10-16	2.0323

Generación de una Onda Seno con frecuencia 100Hz

Seguidamente, es generada una onda forma de onda seno a una frecuencia de 100Hz. Para esto se realizo la grabación por un tiempo de 0.5 seg., como presentado en la Figura 6.14. Considerando la frecuencia de muestreo de 44.100KHz, fueron almacenados un total de 0.5*44100 = 22050 puntos de la señal. En la Figura 6.15 se presenta la forma de onda senoidal generada, para una frecuencia de 100Hz.

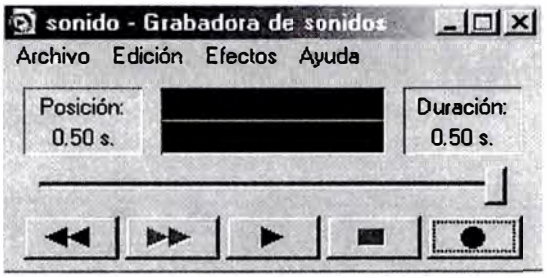


Fig. 6.14. Grabación de la señal por un tiempo de 0.5 seg.

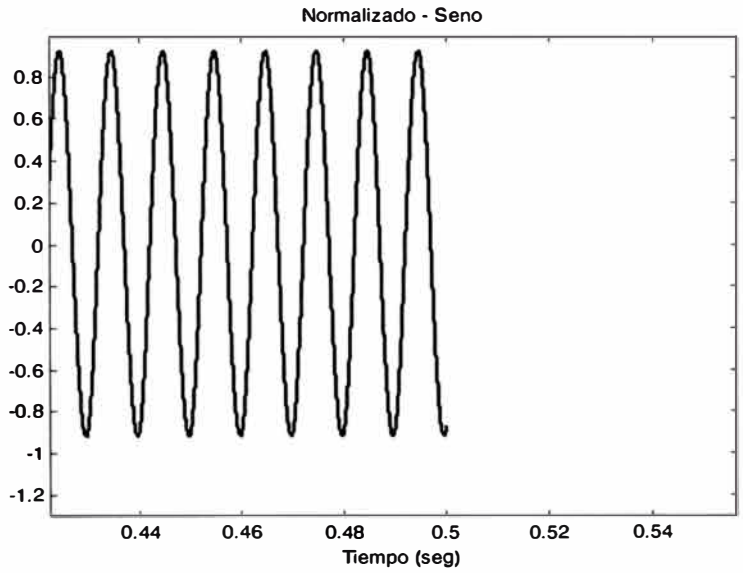


Fig. 6.15. Generación de la forma de onda seno con una frecuencia de 100Hz.

A continuación, se presentan otra solución para la aproximación de la función seno con parámetros configurados en $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 13$. En la Figura 6.16 se presenta una solución para la representación de la aproximación de la función seno, siendo $S = y - y_q$ el error de la función de aproximación normalizado, E_{Amin} y E_{Amax} los limites de errores de aproximación normalizados. El valor de *fitness* obtenido fue de 0.9051. Se observa que la solución S no sobrepasa los límites de los errores de aproximación.

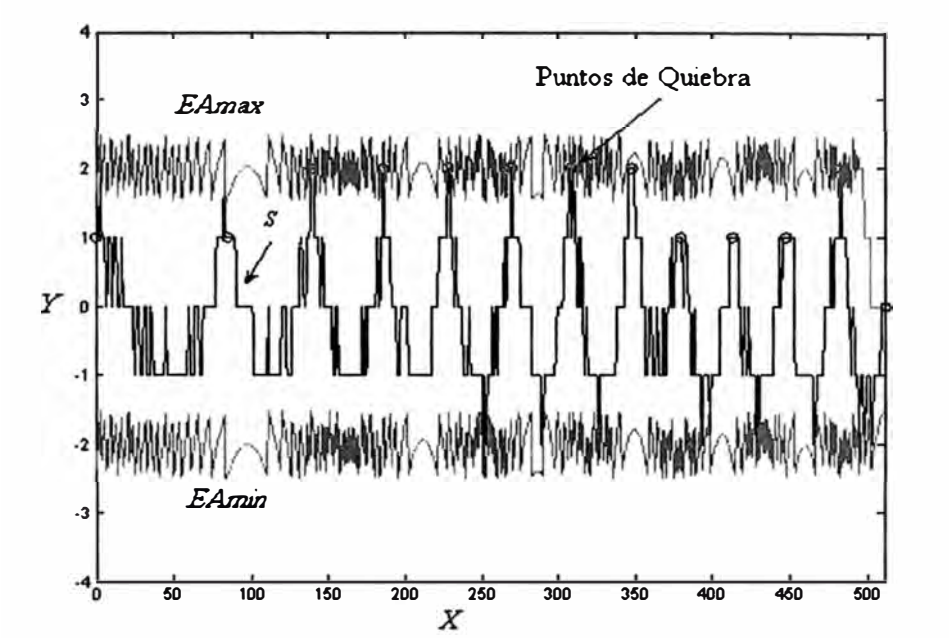


Fig. 6.16. Aproximación para la función seno con $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 13$.

A continuación se resume los puntos de quiebra (p_x, p_y) encontrados por el algoritmo jerárquico evolutivo.

p_x = [0 85 139 186 228 269 308 347 379 413 447 482 511]

p_y = [4 532 851 1110 1322 1508 1663 1794 1882 1956 2009 2041 2047]

En la Figura 6.17 se presenta la implementación de los puntos de quiebra en una tabla de equivalencia (LUT) con parámetros $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 13$. A partir de los valores almacenados en la LUT los otros valores de la función seno pueden ser obtenidos por aproximación lineal por partes, procedimiento descrito en la sección IV.

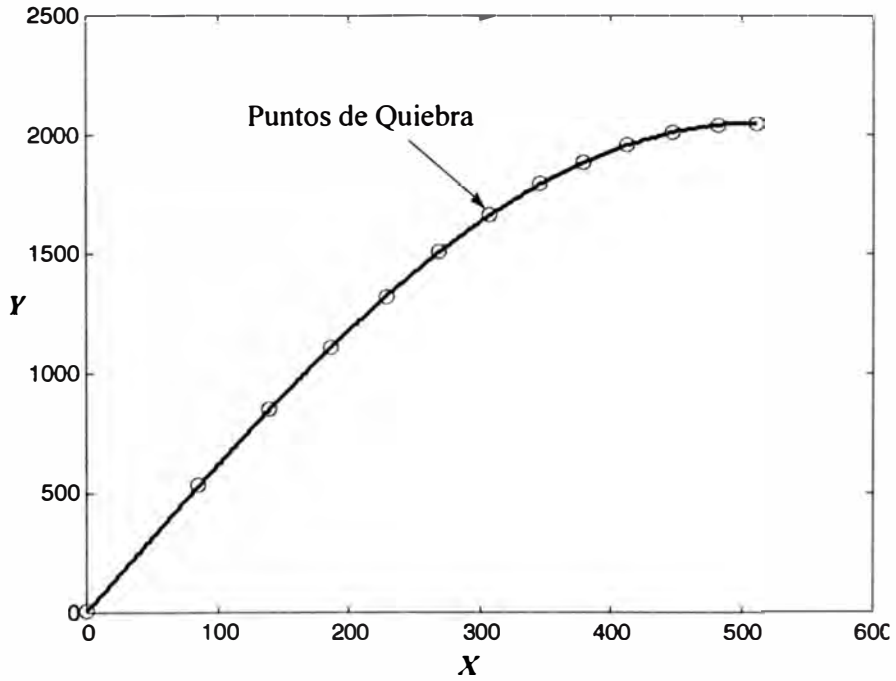


Fig. 6.17. Puntos de quiebra implementados en una tabla de equivalencia (LUT) con parámetros $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 13$.

Para esta solución encontrada, la aproximación de la función seno en el primer cuadrante es implementada en un Procesador Digital de Señales DSP56811. Así la aproximación de la función seno es implementado utilizando los puntos de quiebra (p_x, py) solución encontrados para las especificaciones $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 13$. A seguir se presenta el código fuente para esta especificación, escrito en C – ANSI.

```
/*********************************************************/
/* LUTseno9.c */
/*Autor: Juan Mauricio*/
/*********************************************************/
#include <stdio.h>
#include <dsp56811.h>
int main()
{
        int input=413,m=13,s,i;
        /*********************************************************/
        /*PUNTOS DE QUIEBRA*/
        /*********************************************************/
        int px[13]={0,85,139,186,228,269,308,347,379,413,447,482,511};
int py[13]={4,532,851,1110,1322,1508,1663,1794,1882,1956,2009,2041,2047};
        /*********************************************************/
        /*COEFICIENTES*/
        /*********************************************************/
int coef_a[12]={4,532,851,1110,1322,1508,1663,1794,1882,1956,2009,2041};
        int coef_b[12]={6,6,6,5,5,4,3,3,2,2,1,0};
        /*********************************************************/
        /*Funcion por partes s=f(x)=a+b(x-px)*/
        /*********************************************************/
        for (i=1;i<m;i++)
        {
                if(input>=px[i-1] & input<px[i]){
                        s=coef_a[i-1]+coef_b[i-1]*(input-px[i-1]);
                }
        }
        if(input>=px[m-1]){
                s=py[m-1];
```

```
        }
        while(1);
}
```

Como ejemplo de aplicación, la variable de entrada asume el valor de input = 413, y a través del algoritmo implementado, este debe dar como resultado a la variable de salida el valor de s = 1956. Así el código fuente es compilado e enlazado utilizando el programa enlazador, propio de la familia de procesadores DSP56800, "m568c":

C:>\m568c –g –mr LUTsenoC9.c

El proceso de compilación genera el siguiente archivo: "LUTsenoC9.CLD", que es la versión ensamblada usada por el procesador (hardware). Una vez realizados los pasos anteriores, es ejecutado el DEBUG-EVM para depurar el programa, cargando el archivo LUTseno9.CLD en el *debuger* y transfiriendo el programa a la tarjeta DSP56800EVM, como se presenta en la Figura 6.18.

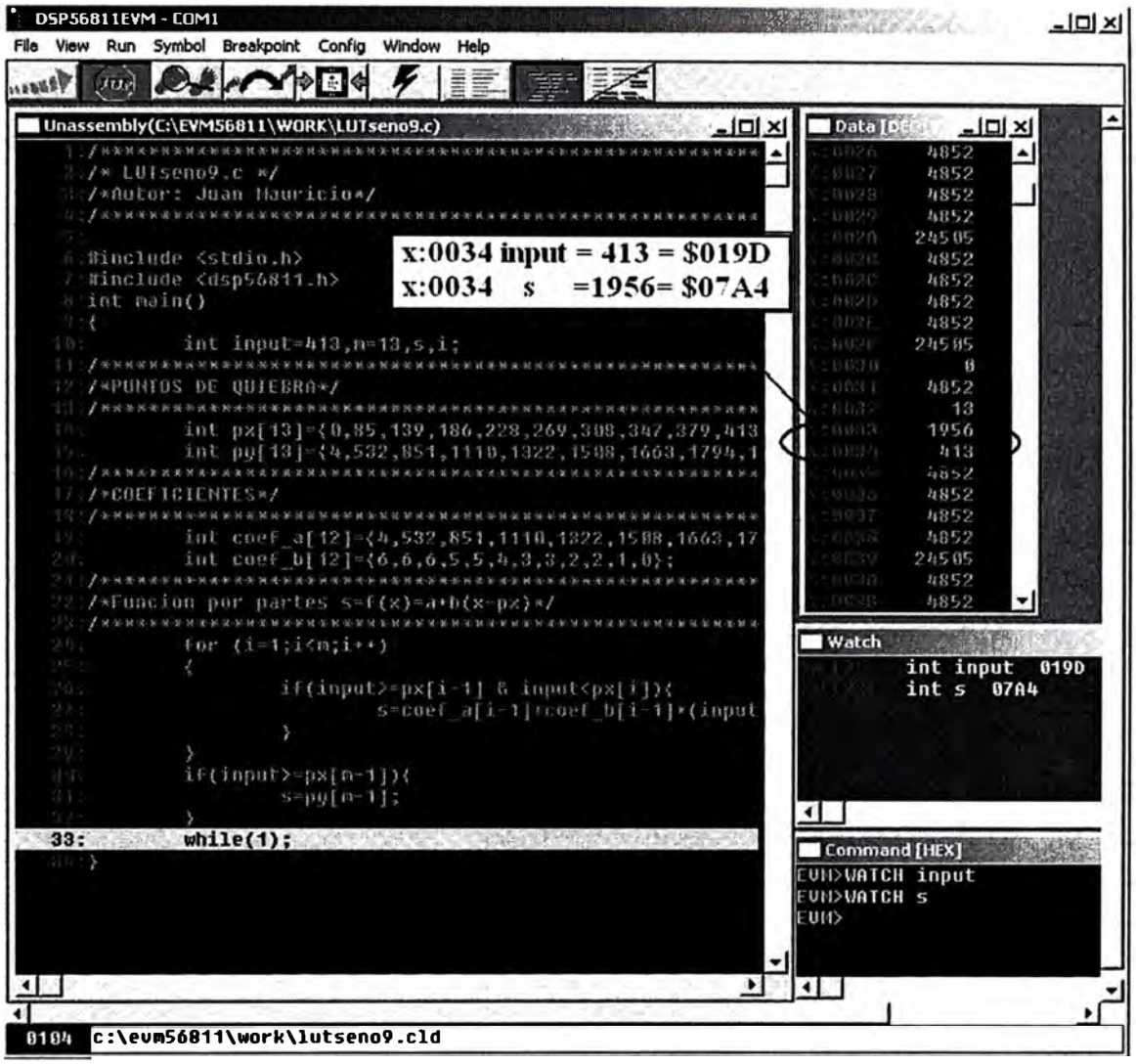


Fig. 6.18. Archivo LUTseno9.CLD transferido al DEBUG - DSP56811EVM.

A continuación se analiza el desempeño de la taza de convergencia del algoritmo genético. Para esto, se determinada los parámetros de N y N_Y, desde 8 bits hasta 12 bits como especificado en la Tabla N° 6.6. Para cada caso fueron realizadas 100 simulaciones del algoritmo genético, determinándose los valores de la media final de épocas y la varianza en que son encontradas las soluciones. En cada caso se utiliza una probabilidad de cruzamiento $P_c = 0.9$ y una probabilidad de mutación $P_m = 0.04$.

A partir de estos resultados, se puede concluir que el algoritmo jerárquico evolutivo encuentra soluciones para el problema de aproximar a función seno en el primer cuadrante, para diferentes especificaciones de entrada, entretanto, el costo computacional aumenta a medida que las resoluciones N, N_Y y N_T aumentan, esto debido a que el espacio de búsqueda de las soluciones también aumenta.

Tabla N° 6.6. Tazas de convergencia del algoritmo genético para cada caso.

Parámetros				Algoritmo Genético Clásico	
N	N_Y	N_T	m	Media de época final	Varianza media
8	8	10	9	62	35
9	9	11	13	40	1,8
10	10	12	17	341	65
11	11	14	22	850	83
12	12	15	31	1522	194

6.2. Reconstrucción de Temperatura de un Sensor Termistor Utilizando un Procesador Digital de Señales DSP56811

Los termistores son resistores ajustables cuyos valores son modificados en función de la temperatura. Los termistores son utilizados en una gran variedad de aplicaciones, incluyendo limitación de corrientes, como sensor de temperatura, protección contra el recalentamiento de equipos tales como motores eléctricos, etc. Con el objetivo de utilizar un termistor como un sensor de temperatura, se considera un sistema de medición que incluye un bloque de reconstrucción digital, en la cual se implementa la función inversa de la función no lineal del termistor.

Para este propósito, como segunda aplicación, se presenta la reconstrucción de temperatura de un termistor. Un sistema de medición de temperatura utilizando un termistor es un sistema que utiliza un sensor con características no lineales entre la señal a ser medida y la señal eléctrica producida por esta. La dependencia de la resistencia del termistor en función de la temperatura puede ser descrito por:

$$R_{th} = R_{in} e^{\beta / T} \tag{6.8}$$

Donde R_{th} y T son la resistencia del termistor (Ω) y temperatura (K – Kelvin), respectivamente. Los parámetros tienen valores iguales a $R_{in} = 3.33m\Omega$ y $\beta = 4000K$.

Se considera un sistema de medición de temperatura para un intervalo de valores de 0 a 50°C. Una de las configuraciones convencionales de medición de temperatura con termistor, consiste en utilizar una fuente de corriente, como se presenta en la Figura 6.19. A partir de esta configuración, el termistor es excitado por una fuente de corriente constante $I = 10\mu A$, y la señal eléctrica representando la señal principal es la tensión y sobre el termistor.

Para esta configuración es considerada la variable x como la temperatura e y como la señal eléctrica medida, realizando un mapeamento no lineal $y = f(x)$, definiéndose la siguiente ecuación:

$$y = f(x) = I.R_{in}.e^{\beta / x} \tag{6.9}$$

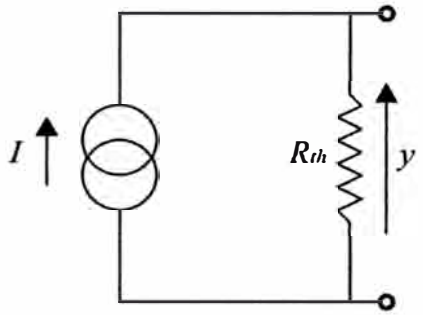


Fig. 6.19. Configuración de medición de temperatura usando un termistor utilizando la excitación de una fuente de corriente constante *I*.

En la Figura 6.20 se presenta la señal eléctrica *y* en función de la temperatura (convertida para °C), señal *x*.

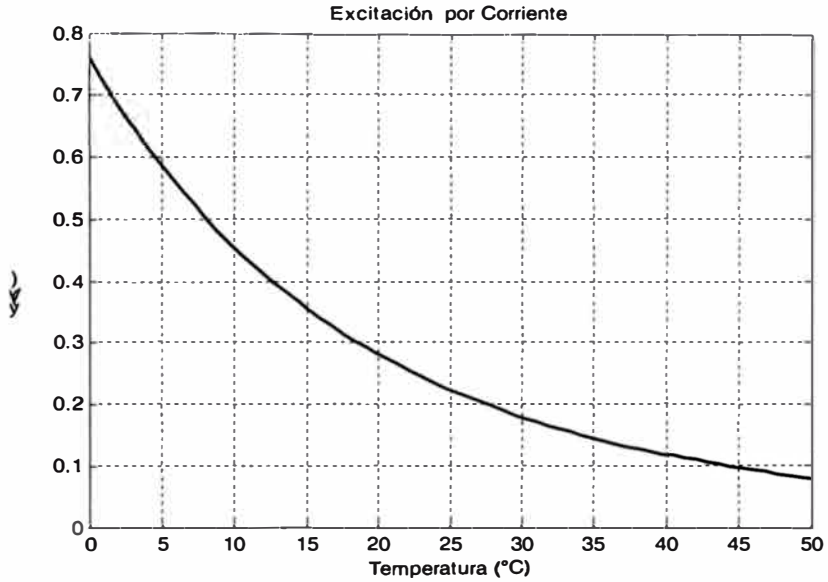


Fig. 6.20. Señal eléctrica *y* en función de la temperatura (convertida para C), señal *x*.

A partir de la ecuación (6.9) se puede obtener la función no lineal para implementar en el bloque de reconstrucción directa $R_D(.)$ de la señal de medición, calculando la función inversa (o inversa aproximada) de la función no lineal del sensor, definido por:

$$\hat{x} = R_D(\tilde{y}) = \frac{\beta}{\ln\left(\frac{\tilde{y}}{R_{in}}\right)} - 273.15 \qquad (6.10)$$

En la Figura 6.21 se presenta el diagrama de bloques del sistema de medición con reconstrucción de la señal medida.

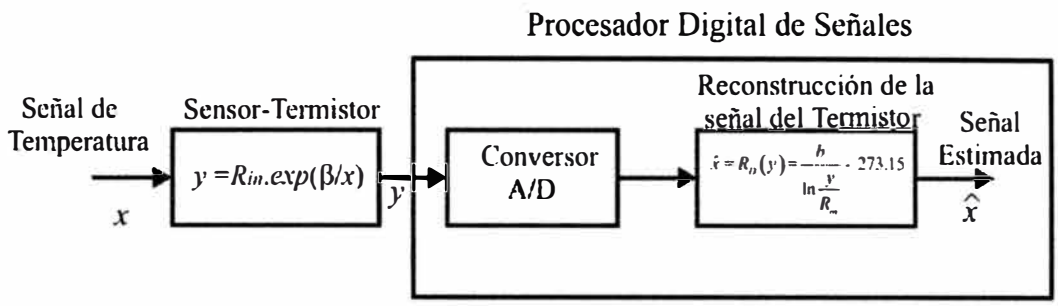


Fig. 6.21. Sistema de medición de temperatura con reconstrucción de la medida

En la Figura 6.22 se presenta la característica no lineal de la función de reconstrucción directa definida por (6.10), que representa variación de temperatura en función de la resistencia.

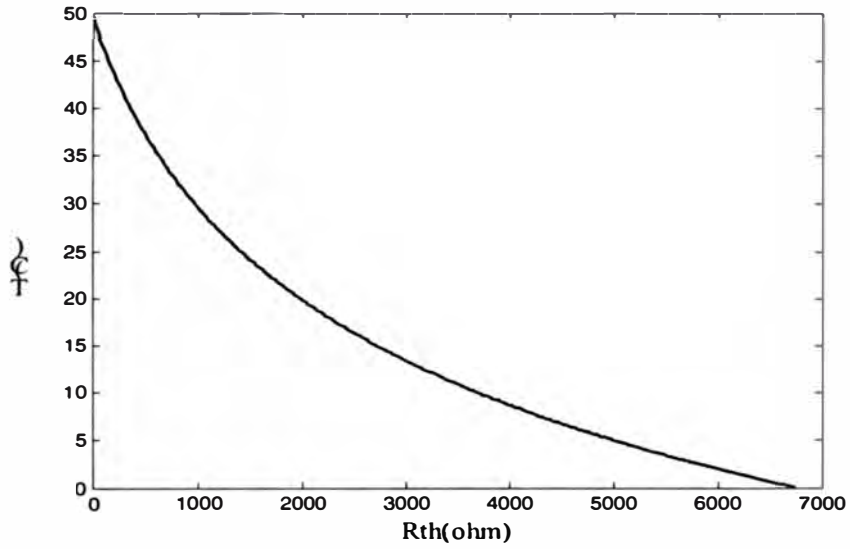


Fig. 6.22. Representación de la señal de reconstrucción (Resistencia vs Temperatura).

A partir de la definición de la función de reconstrucción $R_D(.)$, se utiliza el procedimiento de determinación de funciones no lineales utilizando el algoritmo jerárquico evolutivo. En el primer nivel se define el tamaño de la tabla de equivalencia ($m \times N_T$), con m puntos de quiebra y una resolución equivalente a N_T bits. En el segundo nivel se implementa un algoritmo genético clásico que buscara una solución factible para la condición especificada en el primer nivel, como presentado en la Figura 6.23.

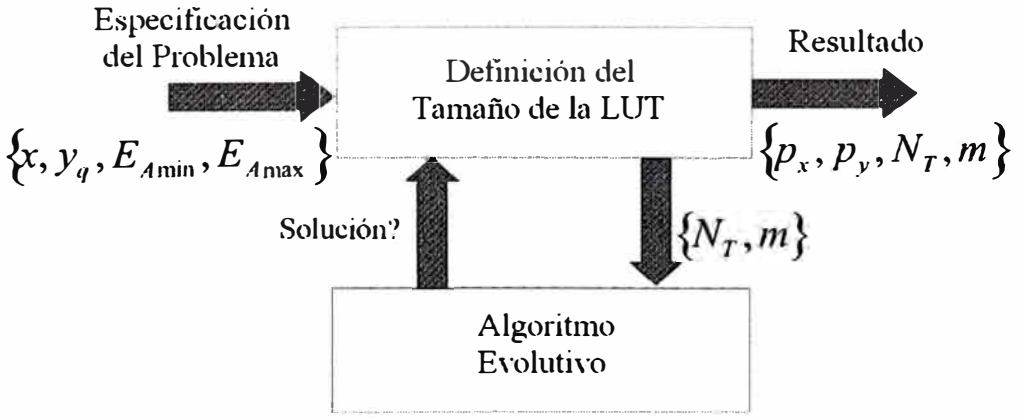


Fig. 6.23. Algoritmo jerárquico evolutivo compuesto por un algoritmo genético clásico

A continuación se presentan los resultados computacionales obtenidos para el estudio de caso de aproximación de la función de reconstrucción directa del termistor. Se considera la función no lineal definida como:

$$y = R_D(x) = \frac{\beta}{\ln\left(\frac{x}{R_{in}}\right)} - 273.15 \qquad (6.11)$$

con limites $x \in [0, 6773]\Omega$ e $y \in [0, 50]$°C. Se analiza el caso en que no existe incerteza asociada a la variable libre, es decir $N_Y = N$.

A continuación, se presentan los resultados de la aproximación de la función no lineal de reconstrucción con parámetros $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 17$. En la Figura 6.24 se presenta una solución para la representación de la aproximación de la función de reconstrucción, siendo $S = y - y_q$ el error de la función de aproximación normalizado, E_{Amin} y E_{Amax} los limites de errores de aproximación normalizados. El valor de *fitness* obtenido fue de 0.9986. Se observa que la solución S no sobrepasa los límites de los errores de aproximación.

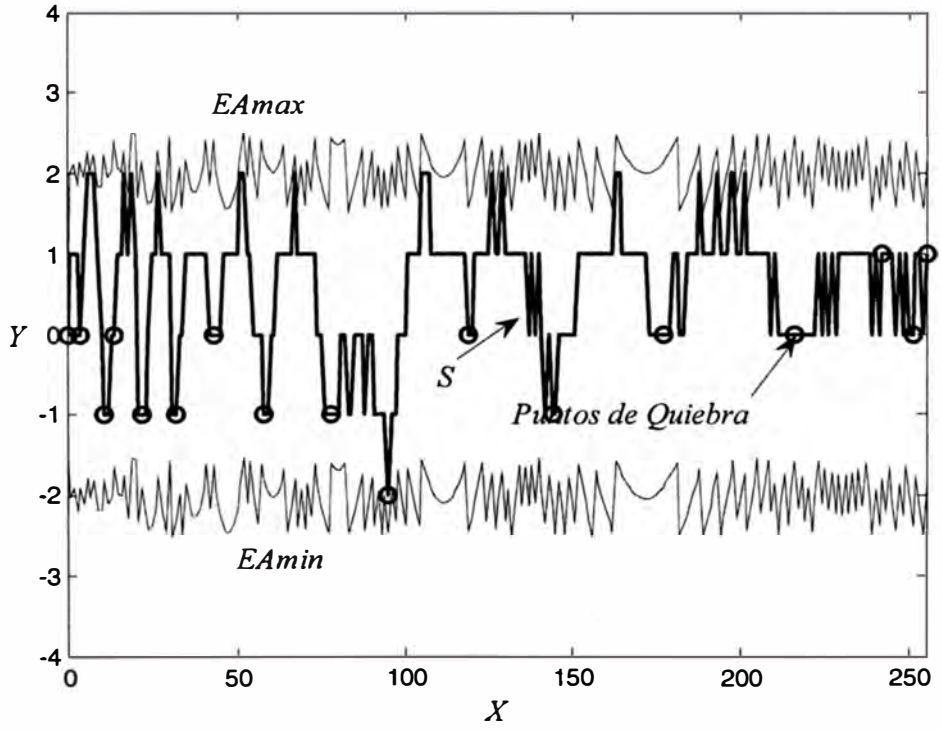


Fig. 6.24. Aproximación para la función de reconstrucción con $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 17$

A continuación se resume los puntos de quiebra (p_x, p_y) encontrados por el algoritmo jerárquico evolutivo.

p_x = [0 4 11 14 22 32 43 58 78 95 119 144 177 216 242 251 255]

p_y = [1023 957 859 824 740 655 578 491 399 335 262 196 126 57 19 6 1]

En la Figura 6.25 se presenta la implementación de los puntos de quiebra en una tabla de equivalencia (LUT) con parámetros $N = 9$, $N_Y = 9$, $N_T = 11$ y $m = 17$. A partir de los valores

almacenados en la LUT los otros valores de la función de reconstrucción pueden ser obtenidos por aproximación lineal por partes, procedimiento descrito en la sección IV.

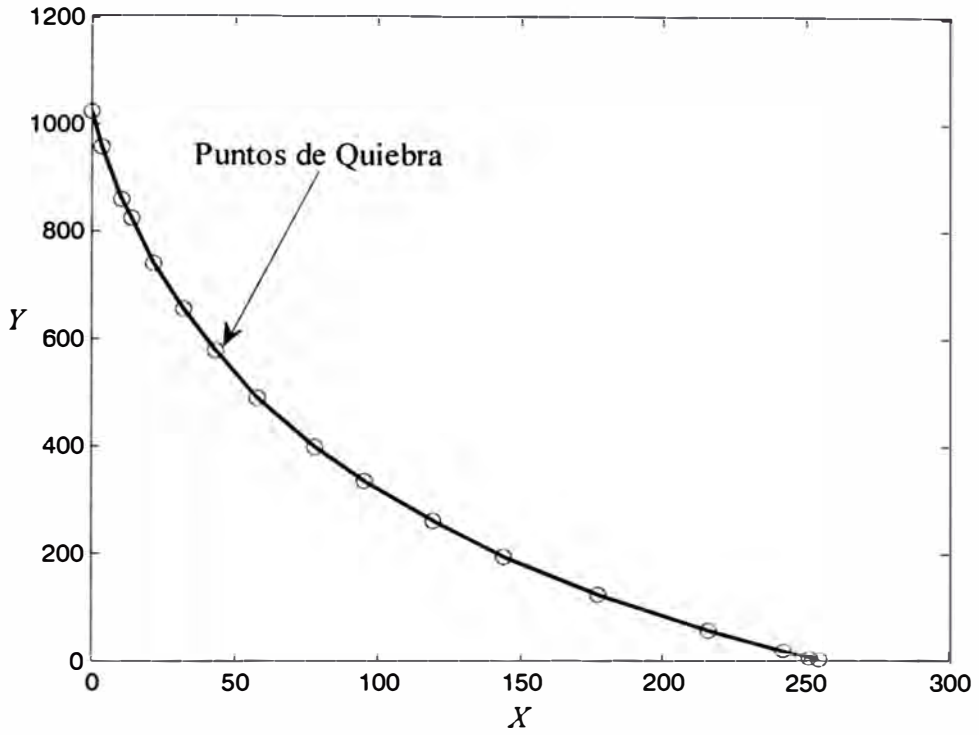


Fig. 6.25. Puntos de quiebra implementados en una tabla de equivalencia (LUT) con parámetros $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 17$.

Para esta solución encontrada, la aproximación de la función de reconstrucción no lineal es implementada en un Procesador Digital de Señales DSP56811. Así la aproximación de la función de reconstrucción es implementada utilizando los puntos de quiebra (p_x, py) solución encontrados para las especificaciones $N = 8$, $N_Y = 8$, $N_T = 10$ y $m = 17$. A seguir se presenta el código fuente para esta especificación, escrito en C – ANSI.

```
/*************************************************************/
/* LUTter.c */
/*Autor: Juan Mauricio*/
/*************************************************************/
#include <stdio.h>
#include <dsp56811.h>
int main()
{
        int input=216,m=17,s,i;
        /*************************************************************/
        /*PUNTOS DE QUIEBRA*/
        /*************************************************************/
```

```
        int px[17]={0,4,11,14,22,32,43,58,78,95,119,144,177,216,242,251,255};
    int py[17]={1023,957,859,824,740,655,578,491,399,335,262,196,126,57,19,6,1};
        /*********************************************************/
        /*COEFICIENTES*/
        /*********************************************************/
int coef_a[16]={1023,957,859,824,740,655,578,491,399,335,262,196,126,57,19,6};
        int coef_b[16]={-17,-14,-12,-11,-9,-7,-6,-5,-4,-3,-3,-2,-2,-1,-1,-1};
        /*********************************************************/
        /*Funcion por partes s=f(x)=a+b(x-px)*/
        /*********************************************************/
        for (i=1;i<m;i++)
        {
            if(input>=px[i-1] & input<px[i]){
                s=coef_a[i-1]+coef_b[i-1]*(input-px[i-1]);
            }
        }
        if(input>=px[m-1]){
            s=py[m-1];
        }
        while(1);
    }
```

Como ejemplo de aplicación, la variable de entrada asume el valor de input = 216, y a través del algoritmo implementado, este debe dar como resultado a la variable de salida el valor de s = 57.

Así el código fuente es compilado e enlazado utilizando el programa enlazador, propio de la familia de procesadores DSP56800, "m568c":

C:>\m568c –g –mr LUTter.c

El proceso de compilación genera el siguiente archivo: "LUTter.CLD", que es la versión ensamblada usada por el procesador (hardware). Una vez realizados los pasos anteriores, es ejecutado el DEBUG-EVM para depurar el programa, cargando el archivo LUTter.CLD en el *debuger* y transfiriendo el programa a la tarjeta DSP56800EVM, como se presenta en la Figura 6.26.

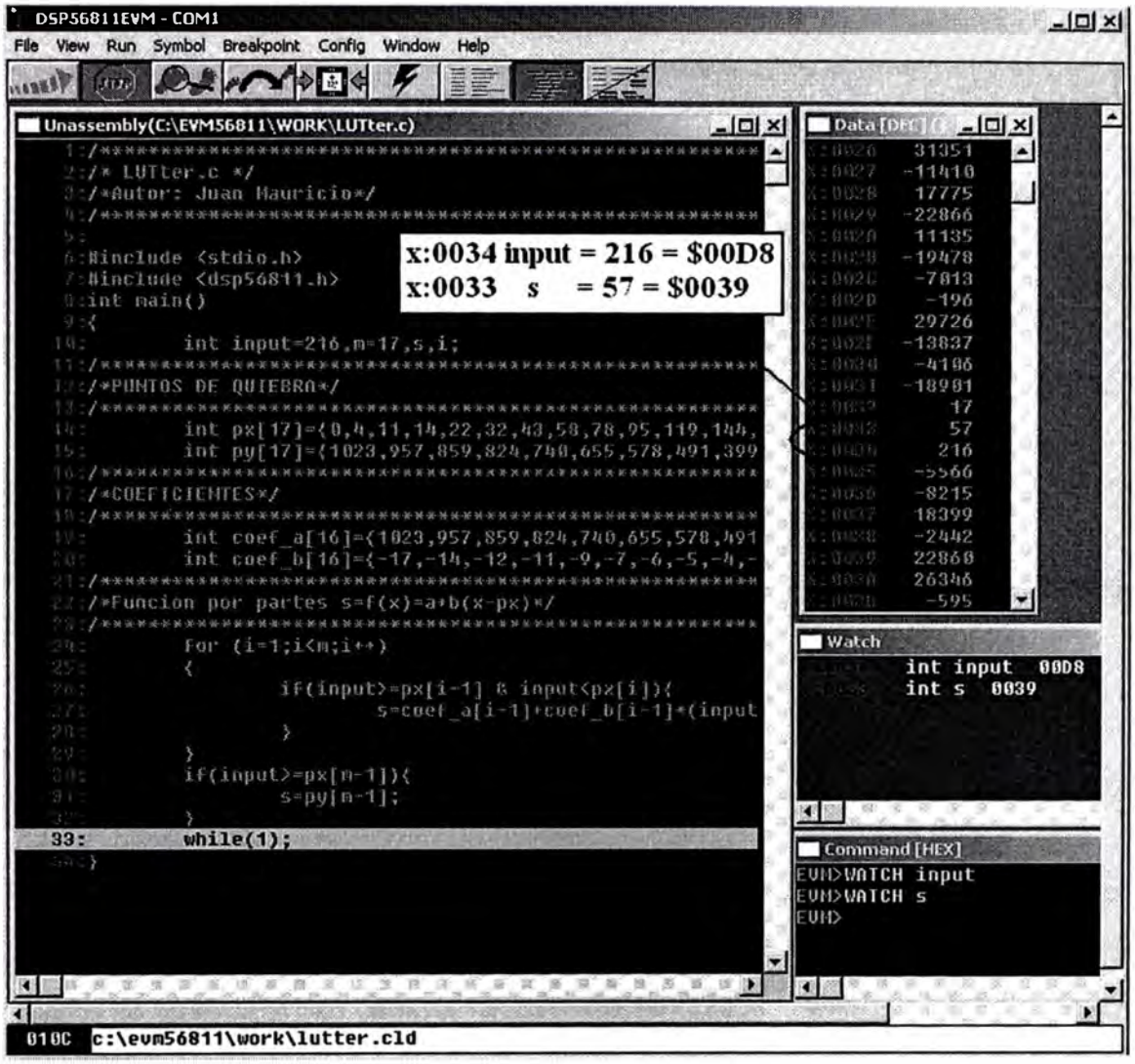


Fig. 6.26. Archivo LUTter.CLD transferido al DEBUG - DSP56811EVM.

CONCLUSIONES Y RECOMENDACIONES

A partir del desarrollo, análisis y de los resultados obtenidos de este trabajo, pueden ser realizadas las siguientes **conclusiones**:

1. Fueron definidos sistemas de medición de la forma más general posible con la finalidad de que sean aplicados a cualquier tipo de forma de medición. Fueron clasificados los sistemas de medición según su forma de medición directa o indirecta y según la actuación en el medio de medición sin realimentación o con realimentación.
2. El bloque de control, elemento del sistema de medición, puede ser considerado en sistema en los cuales se puede actuar sobre el ambiente de medición, es decir, sistemas de medición realimentados, disminuyendo las constantes de tiempo asociadas a la medición y mejorando la calidad de la señal.
3. El bloque de reconstrucción de señales, permite eliminar las no lineales presentadas por sensores con características no lineales, entretanto, la implementación de este bloque de reconstrucción requiere la determinación de la función inversa aproximada de la función del sensor, que puede ser implementada utilizando funciones lineales por partes, almacenando los puntos de quiebra en una tabla de equivalencia (LUT) y obteniendo los valores de la función aproximada a través un una interpolación lineal.
4. Para la implementación de aproximación de funciones no lineales en sistemas embebidos de bajo costo con capacidad de cálculo en punto fijo, fueron considerados aspectos como: incerteza asociadas a las variables libres, saturación con en la representación de números y efectos de cuantización debido a la resolución de almacenamiento de los puntos de quiebra. Para esto, fueron analizados restricciones y limites para aproximación de funciones lineales en sistemas embebidos.
5. La utilización de algoritmos genéticos para la determinación de funciones de aproximación demostró ser una alternativa viable para resolver este tipo de problemas. El objetivo de este algoritmo genético fue la de determinar funciones de aproximación por funciones lineales por partes, encontrando el mínimo numero de puntos de quiebra, sus valores y la resolución de estos valores, para que sean almacenados en la tabla de equivalencia (LUT) de un sistema digital embebido, utilizado el mínimo tamaño de memoria (en bits).
6. Los casos de estudios presentados en esta tesis probaron que el algoritmo genético desarrollado tiene un buen desempeño, encontrando soluciones próximas del óptimo con resoluciones de N e N_Y hasta 12 bits.

7. El algoritmo genético desarrollado puede ser implementado para determinar las funciones de aproximación de sensores no lineales por funciones lineales por partes encontrando el mínimo tamaño de memoria de la tabla de equivalencia (LUT), necesaria para obtener los valores aproximados de la función no lineal.

Así también, son presentadas **recomendaciones** para trabajos futuros a ser realizados a partir de este trabajo:

1. Analizar el caso de aproximación de funciones multivariables, para la implementación de tablas de equivalencia de múltiples variables, en sistemas embebidos de resolución limitada y de bajo costo.
2. Analizar el caso de aproximación de funciones no lineales con resoluciones de generación mayor de 12bits, considerando los casos en la cual no hay convergencia en una solución del algoritmo evolutivo jerárquico.

ANEXO A
PROCESADOR DIGITAL DE SEÑALES DSP56800

La familia de procesadores DSP56800 es un Procesador Digital de Señales (DSP) de 16 bits, utilizado como una unidad de procesamiento de propósito general (CPU), diseñado para un control eficiente de operaciones para el procesamiento digital de señales.

Este procesador tiene un conjunto de instrucciones para diferentes modos de direccionamiento de direcciones, instrucciones de manipulación de bits, las cuales permiten escribir códigos de procesamiento de señales inmediatamente.

Adicionalmente, este procesador presenta una variedad de periféricos adicionados al procesador, tales como: puertos, *timers* de propósito general, puertos de entrada-salida (I/O) de propósito general, direccionamiento de memoria.

1. Características del Procesador Digital de Señales DSP 56800

El procesador digital de señales DSP56800 es un procesador programable de 16 bits CMOS, que incluye en su arquitectura una Unidad Aritmética Lógica (ALU) de 16 bits, una Unidad Generadora de Direcciones (AGU) de 16 bits, un Codificador de Programa, On-Chip Emulation (OnCE), buses asociados para comunicación, y un conjunto de instrucciones para realizar operaciones de manera eficiente y rápida. A seguir, se presentan las características básicas de procesador DSP56800:

- 25 millones de instrucciones por segundo (MIPS) a 40MHz
- Multiplicador- Acumulador (MAC) paralelo de 16×16 bits
- Dos acumuladores de 16 bits (A y B) incluyendo bits de extensión
- Hardware DO e instrucción de repetición REP
- Tres buses de datos de 16 bits (X0, Y0, Y1)
- Tres buses de direcciones de 16 bits
- Un bus de datos dedicados a periféricos
- Bajo consumo de potencia (HCMOS)

En la Figura 1 se presenta el diagrama de bloques del procesador DSP56800.

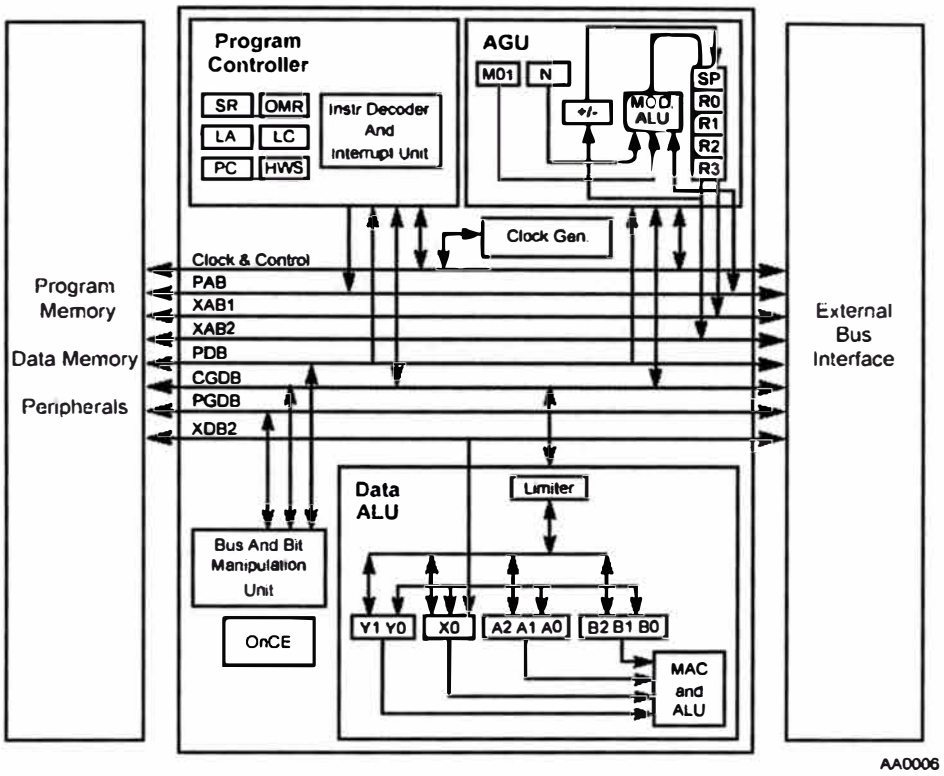


Fig. 1. Diagrama de bloques del DSP56800

2. Bloque de Periféricos

Los siguientes bloques de periféricos están disponibles para los miembros de la familia DSP56800

- Modulo Phase Locked Loop (PLL)
- Cristales de 32 y 38.4KHz
- Cristales con frecuencia de 1MHz
- Factor multiplicador programable
- Modulo Timers de 16 bits
- Tres Timers independientes de 16 bits, cada uno puede ser usado como un reloj en cada pin, un reloj oscilador o la salida de un PLL
- Modulo de interface serial sincrona (SSI)
- Interface Serial (SPI)
- Programación de puertos de propósito general de entrada-salida I/O

3. Unidad Aritmética Lógica (ALU)

La unidad aritmética lógica (ALU) realiza las operaciones aritméticas y lógicas de los datos. El ALU esta compuesto por los siguientes registros:

- Tres registros de entrada de 16 bits (X0, Y0, y Y1)
- Dos registros acumuladores de 32 bits (A y B)
- Dos registros acumuladores de extensión de 4 bits (A2 y B2)

- Un acumulador desplazador (AS)
- Un limitador de datos
- Una unidad MAC multiplicador acumulador

Los buses de multiplicación del ALU realizan operaciones aritméticas complejas (tales como multiplicación y acumulación) en para con dos transferencias de memoria.

El ALU realiza las siguientes instrucciones en un solo siglo de reloj:

- Multiplicación (Con y sin redondeo)
- Multiplicación con inversión de producto (Con y sin redondeo)
- Multiplicación y acumulación (Con y sin redondeo)
- Multiplicación y acumulación con inversión de producto (Con y sin redondeo)
- Adición y sustracción
- Comparación
- Operaciones lógicas(AND, OR, y EOR)
- Complemento a uno
- Complemento a dos(negación)
- Desplazamiento aritmético y lógico
- Rotación
- Redondeo
- Valor absoluto
- Saturación (limitación)

En las Figuras 2 y Figura 3 se presentan el diagrama de bloques del ALU y el modelo de programación del ALU, respectivamente.

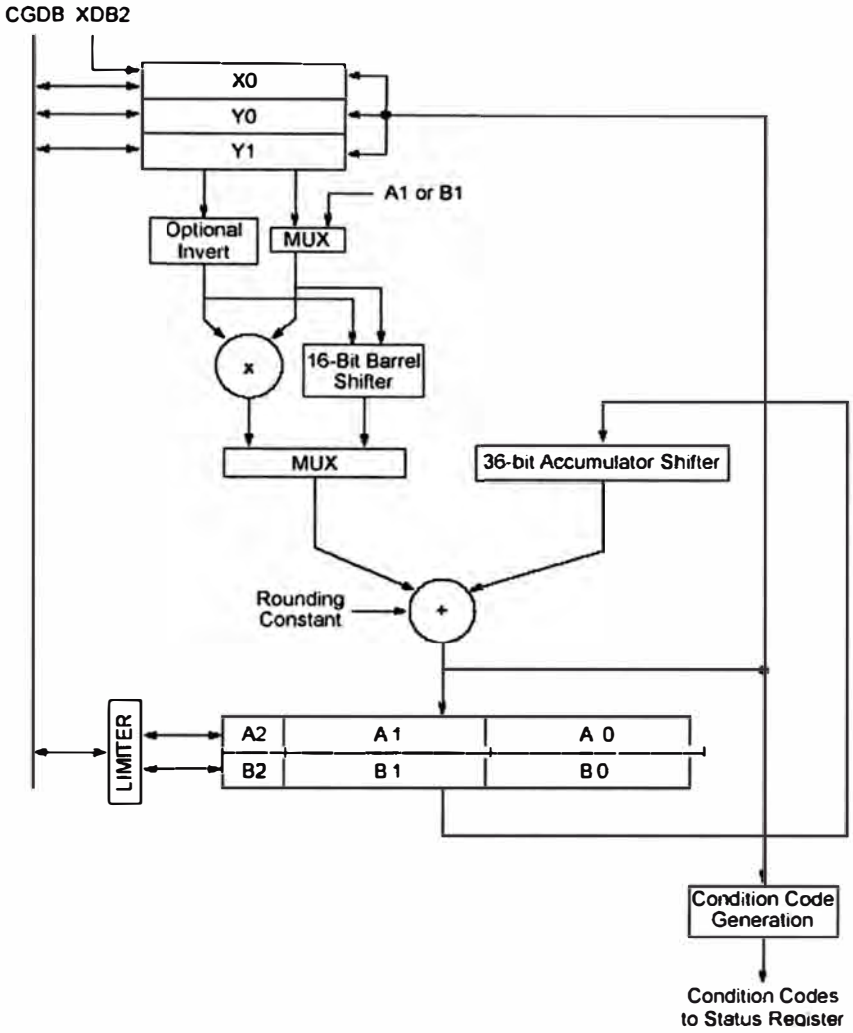


Fig. 2. Diagrama de bloques del ALU

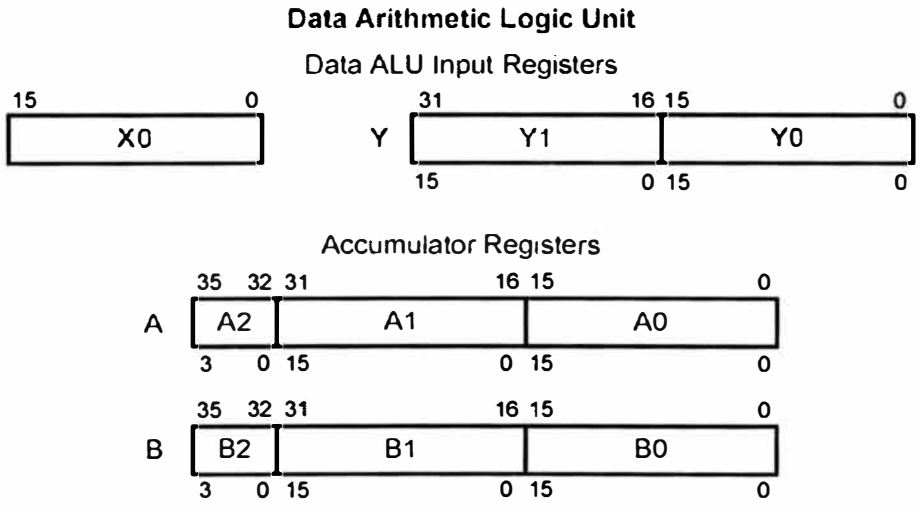


Fig. 3. Modelo de programación del ALU

3.1. Multiplicador Acumulador (MAC) y Unidad Lógica

El multiplicador acumulador (MAC) y la unidad lógica es la principal unidad de procesamiento del DSP. Este bloque realiza la multiplicación, adición, sustracción, operaciones lógicas, y otras operaciones aritméticas a excepción desplazamientos.

Este acepta tres registros de entrada y una salida de 36 bits. Las operaciones aritméticas en el MAC se realizan independientemente y en paralelo con acceso a memoria sobre el CGDB, XDB2 y PDB. El ALU provee registros pipeline para datos de entrada y salida.

3.2. Acumulador Shifer (AS)

El acumulador shifter(AS) es un desplazador paralelo sincronizado con una entrada de 36bits y una salida de 36bits. Las operaciones realizadas por esta unidad son las siguientes:

- No realiza desplazamiento - ADD, SUB, MAC, etc
- 1 bit de desplazamiento a la izquierda - ASL, LSL, ROL
- 1 bit de desplazamiento a la derecha - ASR, LSR, ROR
- Forzar a cero - MPY, IMPY(16)
- La salida del desplazamiento va directamente al MAC como entrada

4. Unidad Generadora de Direcciones (AGU)

La unidad generadora de direcciones (AGU) realiza todos los cálculos de direccionamiento de memoria y almacenamiento de direcciones necesarias al direccionar datos en la memoria. El AGU opera en paralelo con otras unidades del procesador para minimizar el cálculo de generación de direcciones.

El AGU contiene dos ALU's, las cuales generan direcciones de 16 bits por cada ciclo de instrucción; uno para el bus de direcciones externa (XAB1) o bus de direcciones de programa (PAB), y para el bus de direcciones externa (XAB2). El ALU puede direccionar directamente 65536 localizaciones sobre el bus XAB1 o XAB2 y 65536 localizaciones sobre el bus PAB, 131 072 palabras de datos de 16 bits en total, soportando un conjunto completo de modos de direccionamiento. Esta unidad aritmética puede ser del tipo aritmética lineal o aritmética modulo.

El AGU incluye los siguientes registros:

- Cuatro registros de dirección (R0-R3)
- Un registro apuntador de pila (SP)
- Un registro offset (N)
- Un registro modificador (MO1)
- Un modulo de unidad aritmética
- Una unidad incrementadora/decrementadora

Los registros de dirección son de 16 bits, estos registros pueden contener una dirección o datos. Cada registro de dirección puede proveer una dirección para los buses de dirección XAB1 y PAB. Para instrucciones que leen dos valores de la memoria de datos, R3 provee una dirección para el XAB2, y R0 o R1 provee una dirección para el XAB1. El registro modificador y el offset son registros de 16 bits que controlan la actualización de los registros de direcciones. El registro offset puede también ser utilizado para almacenar datos de 16 bits. Los registros del AGU pueden ser leídos o escritos por el CGDB como un operando de 16 bits.

4.1. Arquitectura y Modelo de Programación

Los cuatro registros de direcciones de punteros (R0-R3) y el SP son usados para generación de direcciones en el registro de modo de direccionamiento indirecto, como se observa en la Figura 4. El registro *offset* puede ser usado por las cuatro direcciones de puntero de registros, donde el registro modulo puede ser usado por el R0 o por los registros de punteros R0 y R1, como se presenta en la Figura 5.

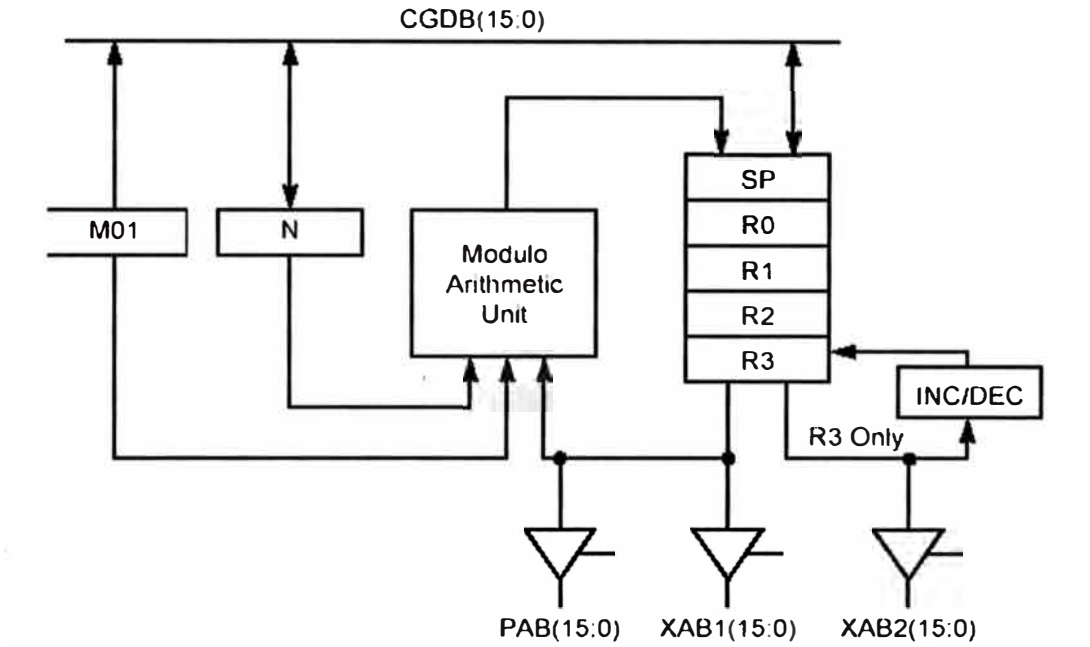


Fig. 4. Diagrama de bloques de la Unidad Generadora de Direcciones

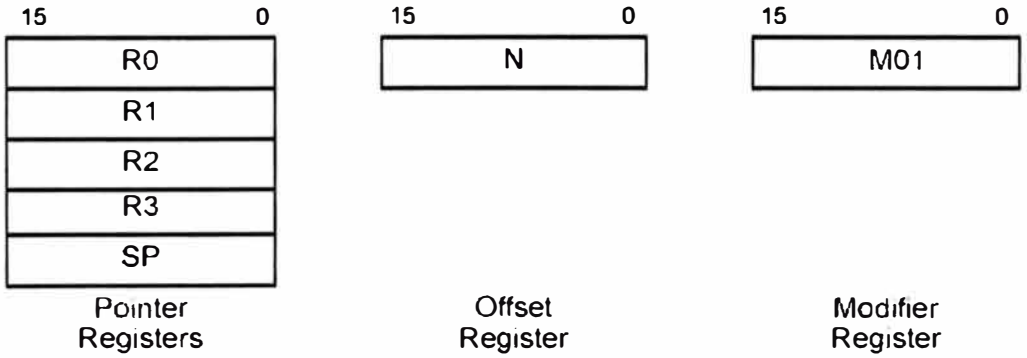


Fig. 5. Modelo de Programación AGU

4.2. Registro de Direcciones (R0 – R3)

El registro de direcciones consiste de cuatro registros de 16 bits R0-R3 (Rn) que usualmente contienen direcciones usadas como punteros en memoria. Cada registro puede ser leído o escrito por el bus CGDB. Cada registro de dirección puede ser usado como entrada para la unidad modulo aritmético para el registro de actualización.

4.3. Stack – Pointer Register (SP)

El registro stack pointer (SP) es un registro único de 16 bits que es usado implícitamente en todas las instrucciones PUSH e instrucciones POP. El SP es usado explícitamente para referencia a la memoria cuando se usa con la dirección de registro de modo indirecto.

4.4. Offset Register (N)

El registro offset (N) usualmente contiene valores de desplazamiento usados para actualizar las direcciones de punteros. Este registro puede ser usado al actualizar o indexar con cualquiera de los registros de direcciones (R0-R3, SP). Este registro offset puede ser leído o escrito por el bus CGDB. El registro offset es usado como entrada a la unidad modulo aritmético.

4.5. Modifier Register (M01)

El registro modificador (M01) especifica si es usada la aritmética lineal o modulo cuando se calcula una nueva dirección y puede ser leído o escrito por el CGDB. Este registro modificador es automáticamente leído cuando el registro de dirección R0 es usado en un cálculo de dirección y puede opcionalmente ser usado también cuando R1 es usado.

Este registro no tiene efecto sobre el cálculo de direcciones hecho con los registros R2, R3 o SP. Este es usado como una entrada a la unidad modulo aritmético. Este registro modificador es predeterminado durante un proceso *reset* a $FFFF(aritmética lineal).

4.6. Unidad Modulo Aritmético

La unida modulo aritmético puede actualizar un registro de dirección o el SP durante un ciclo de instrucción. Este es capaz de realizar aritmética lineal y modulo. El contenido del registro modificador especifica el tipo de aritmética a ser realizado en una actualización del cálculo de un registro.

4.7. Unidad Incrementadota/Decrementadora

La unidad incrementadora/decrementadora es usada para actualizar el cálculo de direcciones durante la lectura de la memoria de datos. Con el registro R3 no es usado el incrementador/decrementador.

4.8. Modos de Direccionamiento

El conjunto de instrucciones para la operación de modos de direccionamiento, optimizado para una alta performance de procesamiento de señales, para un eficiente controlador de código. Todos los cálculos de direcciones son realizados en la unidad generadora de direcciones para minimizar la ejecución en el tiempo.

5. La Tarjeta DSP56811EVM

La Tarjera DSP56811EVM de Motorota es una plataforma de bajo costo diseñada para familiarizar al usuario con el procesador digital de señales DSP56811. Con 16 bits de precisión y con *Codec* de audio y sonido la tarjeta es ideal para la implementación y demostración de

algoritmos de procesamiento de señales, así como para el aprendizaje de la arquitectura y el conjunto de instrucciones del procesador DSP56811.

En la Figura 6 se presenta los principales componentes de la tarjeta DSP56811EVM y en la Figura 7 se presenta el diagrama funcional del DSP56811 incluyendo sus unidades de memoria de datos y de programa, así como sus periféricos disponibles.

6. DEBUG – EVM56800

Existen dos maneras de desarrollar aplicaciones utilizando el modulo DSP56811EVM. Primeramente, se puede escribir instrucciones de código en "*assembler*" específicas para implementar aplicaciones de procesamiento de señales, entretanto, la manera de programación puede convertirse compleja para ciertas aplicaciones. Otra manera de desarrollar aplicaciones, consiste en escribir instrucciones de código en "C-ANSI" utilizando la herramienta de desarrollo DSP C de la familia de procesadores DSP56800 de punto fijo. Esta herramienta consiste de:

- Lenguaje C para implementar aplicaciones en el C Compiler DSP56800, incluyendo variaciones y extensiones para implementaciones de aplicaciones con la tarjeta DSP56800EVM.
- DSP Shell, es un programa que provee una interfase integral entre el C Compiler DSP56800, el Assembler Motorola DSP56800 y el enlazador motorola DSP.
- Rutinas y librerías especificas del DSP56800

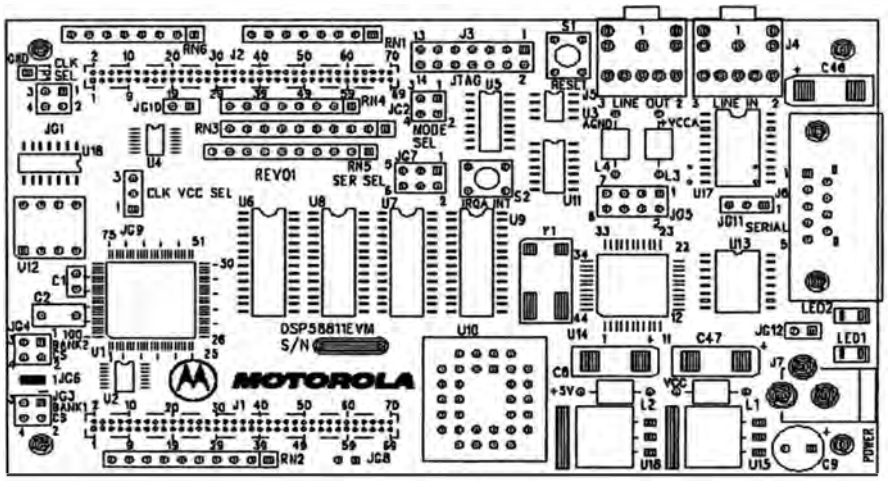


Fig. 6. Principales componentes del *layout* de la tarjeta DSP56811EVM

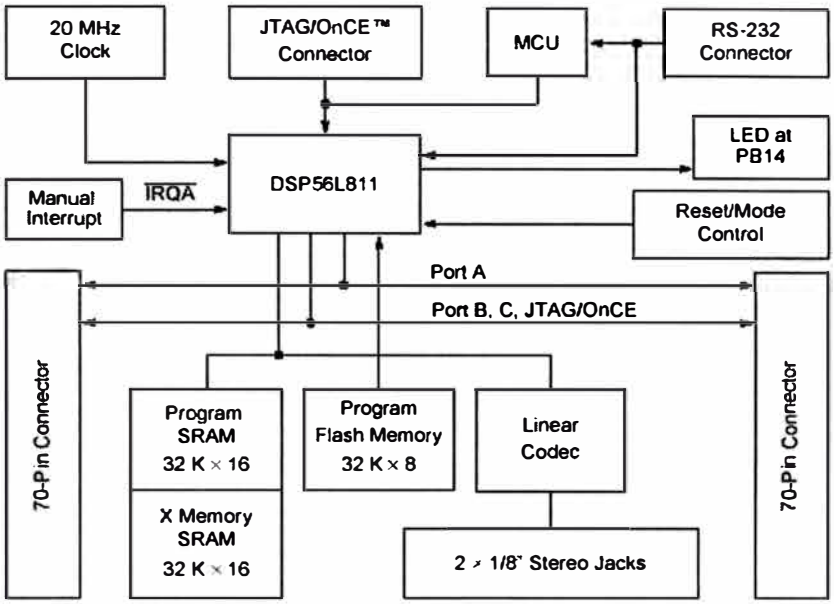


Fig. 7. Diagrama funcional del DSP56811

7. Compilación y Enlace de Programas

Para compilar y enlazar un programa, tomamos como ejemplo un programa escrito en lenguaje de programación C:

```
/*suma.c*/
#include <stdio.h>
#include <dsp56811.h>
int main()
{
    int v1,v2,v3,v4;
    v1 = 1;
    v2 = 2;
    v3 = v1 + v2;
    while(1);
}
```

El ejemplo es guardado en un directorio del computador con nombre "suma.c" y compilado con el programa enlazador m568c:

C:>\m568c –g suma.c

Esta instrucción enlaza el programa "suma.c", produciendo el programa ejecutable con nombre "suma.cld" que será transferido a la tarjeta DSP56800EVM a través de la interfase DEBUG -EVM56800 como presentado en la Figura 8. La opción –g causa que el compilador genere información del *debuger*. En esta figura se observa que las variables utilizadas: v1, v2 y v3 son almacenadas en la memoria de datos del procesador.

Fig. 8. Archivo suma.CLD transferido al DEBUG - DSP56811EVM

ANEXO B
PROGRAMA: DETERMINACION DE FUNCIONES NO LINEALES POR ALGORITMOS GENETICOS

1. Código fuente principal del Algoritmo Genético

Este algoritmo permite determinar las soluciones de los puntos de quiebra (Px, P_y) y la resolución N_T de almacenamiento en la tabla de equivalencia. Los parámetros de entrada son N, NY, *Breakpoint* (Numero de puntos de quiebra), y como salida obtenemos los valores de (P_x, P_y N_T).

Durante la ejecución del código es creado una estructura llamada "*s*" que engloba las variables utilizadas durante el proceso de ejecución del algoritmo, como: P_x (puntos de quiebra en el eje *x*), P_y (Puntos de quiebra en el eje *y*).

```
clear;clc;close all;
%****************************************
%Autor: Juan Mauricio Villanueva
%Ultima modificacion: 19-01-06
%****************************************

      %0. Initial Parameters
N=8;
NY=8;
NT=10;
BreakPoints=9;
Generations=500;
PopSize=80;
Pc=0.9;
Pm=0.04;
n inf=0.0001;
wf=40;
makespan=10;
fitness lim=1;
      %1. Limits of The Approximation Erro
[q]=quantiza(N,NY,NT);
      %2. Initialization (Generacion de la población inicial)
[s]=population(PopSize,BreakPoints,N,NY,NT,q,n_inf,wf);
h=waitbar(0,'En ejecución, por favor, aguarde...');
for loop=1:Generations
```

```
      waitbar(loop/Generations,h);
      %3. Evaluation
      s_pai=s;
      [eval,s]=evaluation(s,N,q,fitness_lim);
      fitness(loop)=s.fit(1);
      %4. Termination test
      if eval.sol==1
            disp('SOLUCION'); break;
      end
      %5. Selection
      s5=s;
      [s]=sel_nolinear_rank(n_inf,wf,N,q,s);
      histograma_x(loop,:)=s.sel_ind_x;
      histograma_y(loop,:)=s.sel_ind_y;
      %6. Crossover
      s6=s;
      [s]=crossover(Pc,N,NY,NT,n_inf,wf,q,s);n_cruze(loop)=s.num_cruze;
      %n_cruze(loop)=0;
      %7. Mutation
      s7=s;
      [s]=mutation(Pm,N,NY,NT,n_inf,wf,q,s);n_muta_x(loop)=s.num_muta_x;n_muta_y
(loop)=s.num_muta_y;
      %n_muta_x(loop)=0;n_muta_y(loop)=0;
      %8. Elitist strategy
      s_hijos=s;
      [s]=elitist_strategy(N,NY,NT,n_inf,wf,q,s_padres,s_hijos,s);
      %9. Local Search
      s9=s;
      [s,taza_LS]=local_search(N,NY,NT,q,s,n_inf,wf,makespan);TLS(loop)=taza_LS;
      fit_min(loop)=min(s.fit); fit_max(loop)=max(s.fit);
      erro_min(loop)=min(s.erro); erro_max(loop)=max(s.erro);
      %10. Measure The Genetic Diversity (gmd)
      [Pc,Pm,gmdl]=gmd(Pc,Pm,s); Prob_Pc(loop)=Pc;
      Prob_Pm(loop)=Pm;gmd_geracion(loop)=gmdl;
      %11. Progresso de las soluciones
      plot(q.x,s.S(1,:),s.Px(1,:),s.S(1,s.Px(1,:)+1),'ko',q.x,q.EAmax_nor,q.x,q.
EAmin_nor),title('Error de aproximacion'), axis([0 (2^N-1) -4 4]),
      grid
end
```

2. Código fuente de la función "quantiza.m"

Función creada para determinar los valores de cuantificación de las variables X, Y y determinar los valores de los errores mínimos y máximos de cuantificación. Presenta como valores de entrada las resoluciones N, N_Y y N_T, y como salida la estructura "q", la cual contiene los valores de las variables X, Y, E_{Amin} y E_{Amax}.

```
function [q]=quantiza(N,NY,NT)
      %Primera Tranformacion
xmin=0;
xmax=pi/2;
X=0:(2^N)-1;%Quantizacion del eje X
x=((X+0.5)/2^N)*xmax;
y=sin(x);
      %Segunda Tranformacion
Yescalonado=((2^NT)/max(y))*y-0.5;
Yquant=round(Yescalonado);
for i=1:2^N
      if Yquant(i)>=2^NT-1
            Yquant(i)=2^NT-1;
      end
end
yquant_real=(Yquant+0.5).*max(y)./2^NT;
      %error sin incerteza de propagacion, constante en el  eje 'x'
eAmax=1/2^(N+1);
eAmin=-1/2^(N+1);
for i=1:2^N
      EAmax(i)=((2^NT)/max(y))*eAmax;
      EAmin(i)=-((2^NT)/max(y))*eAmin;
end
      %*****
lim_sup_quant=Yquant+EAmax;
lim_inf_quant=Yquant-EAmin;
for i=1:2^N
      if lim_sup_quant(i)>=2^NT-1
            lim_sup_quant(i)=2^NT-1;
      end
end
EAmax=lim_sup_quant-Yquant;
EAmin=Yquant-lim_inf_quant;
      %*****
      %Tercera Transformacion
Ynor=Yescalonado-Yquant;
%Limite Minimo Normalizado
```

```
EAmin_nor=-EAmin+Ynor;
%Limite Maximo Normalizado
for i=1:2^N
      if Yescalonado(i)>=2^NT-1;
            Yescalonado(i)=2^NT-1;
      end
end
for i=1:2^N
      if lim_sup_quant(i)>=2^NT-1
            Ynor(i)=0;
      end
end
        %Propagacion sin incerteza
EAmax_nor=EAmax+Ynor;

for i=1:2^N
    if Yquant(i)>=2^NT-1
        EAmax_nor(i)=0;
    end
end
    %*****
quant_x=X;
Y=Yquant;                %Valor entero
quant_y=Y;

q=struct('linf',{lim_inf_quant},'lsup',{lim_sup_quant},'EAmin_nor',{EAmin_nor},'
EAmax_nor',{EAmax_nor},'x',{quant_x},'y',{quant_y});
```

3. Código fuente de la función "population.m"

Este código permite determinar la población inicial, creada aleatoriamente, cuyos valores cambiaran durante la evolución de las soluciones potenciales, hasta converger en una solución. Esta función admite como valores de entrada las variables N, N_Y, N_T y los valores de cuantificación, teniendo como salida la estructura "s" la cual contiene los valores de los puntos de quiebra (P_x, P_y).

```
function [s]=population(PopSize,BreakPoints,N,NY,NT,q,n_inf,wf)

MIN=ceil(q.EAmin_nor);
MAX=floor(q.EAmax_nor);
mx=zeros(PopSize,BreakPoints);

%Puntos x
M=randint(PopSize,BreakPoints-2,[1,2^N-2]);
```

```
x=[zeros(PopSize,1) M (2^N-1)*ones(PopSize,1)];
mx=sort(x')';   %ordena en forma ascendente

%Puntos y
%my=floor(q.EAmax_nor(mx+1));
%puntos_y=q.y(mx+1)+my;
my=randint(PopSize,BreakPoints,[-(NT-NY) (NT-NY)]);
pontos_y=q.y(mx+1)+my;


f=func_nor(mx,pontos_y,N,PopSize);

for i=1:PopSize
    S(i,:)=round(f(i,:)-q.y);
    %ERROR de cada individuo
    ERRO(i)=sum(abs(S(i,:)));
    %Calculo del Fitness
    fit(i)=fitness(q,S(i,:),n_inf,wf);
end

individuo=1:PopSize;
num_cruze=0;
num_ncruze=0;
num_muta_x=0;
num_muta_y=0;

sel_ind_x=0;
sel_ale_x=0;

sel_ind_y=0;
sel_ale_y=0;
Ybits=0;
%Estrutura de la poblacion X Y
s=struct('Px',{mx},'Py',{pontos_y},'erro',{ERRO},'fit',{fit},'S',{S},'ind',{indi
viduo},'Y',{my},'Ybits',{Ybits},'num_cruze',{num_cruze},'num_ncruze',{num_ncruze
},'num_muta_x',{num_muta_x},'num_muta_y',{num_muta_y},'sel_ind_x',{sel_ind_x},'s
el_ale_x',{sel_ale_x},'sel_ind_y',{sel_ind_y},'sel_ale_y',{sel_ale_y});
```

4. Código fuente de la función "evaluation.m"

Este código permite evaluar la función de adaptabilidad (*fitness*) de los individuos de la población actual. El objetivo de esta evaluación es determinar el grado de adaptabilidad de los individuos en el ambiente. Esta función admite como valores de entrada las estructuras de cuantificación "*q*" y de población "*s*" y entrega como salida los valores de la evaluación.

```
function [eval,s]=evaluation(s,N,q,fitness_lim)

[PopSize BreakPoints]=size(s.Px);
    %Ordenamiento de menor a mayor fitness
[fit_ord indice]=sort(s.fit);
s.fit=fit_ord;
s.erro=s.erro(indice);
s.Px=s.Px(indice,:);
s.Py=s.Py(indice,:);
s.S=s.S(indice,:);
s.Y=s.Y(indice,:);
num_cruze_x=0;
num_ncruze_x=0;
ind_cruze_x=0;
ind_ncruze_x=0;
offspring_x=0;
num_cruze_y=0;
num_ncruze_y=0;
ind_cruze_y=0;
ind_ncruze_y=0;
offspring_y=0;

   %Calculo del fitness normalizado para la poblacion de tamaño PopSize
fitness=s.fit/sum(s.fit);%fitness normalizado[0 - 1]
    %Evaluacion de la solucion
for i=1:PopSize
    a=find(s.fit<=fitness_lim);
    if length(a)==0
        solucion=0; %No Existe Solucion
    else
        solucion=1; %Existe Solucion
    end
end
eval=struct('sol',{solucion},'fitness',{fitness});
```

5. Código fuente de la función "sel_nolinear_rank.m"

Este código permite realizar la selección no lineal de los individuos mejor adaptados al ambiente, con la finalidad que estos individuos reproduzcan individuos con características de mejor adaptación al ambiente (problema). Este algoritmo presenta como salida la estructura "*s*", en la cual modifica la estructura de los individuos con valores de adaptabilidades mejores.

```
%Non-Linear Ranking
%Fitness segun su posicion
function [s]=sel_nolinear_rank(n_inf,wf,N,q,s)
```

```
[PopSize BreakPoints]=size(s.Px);
SP=4; %SELECTIVE PRESSURE
polinomio(1)=SP-PopSize;
polinomio(2:PopSize)=SP*ones(1,PopSize-1);
raiz=roots(polinomio);
v=raiz(1);
suma=0;
for i=1:PopSize
    suma=suma+v^(i-1);
end
for i=1:PopSize
    fitselec(i)=PopSize*(v^(PopSize-i))/suma;
end
prob = fitselec/PopSize;
prob_nor=prob/max(prob);

%Seleccion de individuos X
for i=1:PopSize
    aleatorio_x(i)=rand(1);
    a=find(prob_nor>=aleatorio_x(i));
    SELECCION_IND_X(i)=max(a);
end

%Seleccion de individuos Y
for i=1:PopSize
    aleatorio_y(i)=rand(1);
    a=find(prob_nor>=aleatorio_y(i));
    SELECCION_IND_Y(i)=max(a);
end

%sel=struct('ind',{SELECCION_IND},'ale',{aleatorio});

s.sel_ind_x=SELECCION_IND_X;
s.sel_ale_x=aleatorio_x;

s.sel_ind_y=SELECCION_IND_Y;
s.sel_ale_y=aleatorio_y;

%Calculo del Fitness con los nuevos individuos seleccionados X e Y

s.Px=s.Px(s.sel_ind_x,:);
s.Py=s.Py(s.sel_ind_y,:);
s.Y=s.Y(s.sel_ind_y,:);
```

```
f=func_nor(s.Px,s.Py,N,PopSize);
for i=1:PopSize
    S(i,:)=round(f(i,:)-q.y);
    erro(i)=sum(abs(S(i,:)));
    %calculo del fitness
    fit(i)=fitness(q,S(i,:),n_inf,wf);
end
s.erro=erro;
s.fit=fit;
s.S=S;
```

6. Código fuente de la función "crossover.m"

Este código permite realizar el cruzamiento de dos individuos seleccionados denominados "padres", generando dos nuevos individuos denominados "hijos". El objetivo es generan individuos con características de los padres que se adecuen mejor al ambiente. Esta función admite como entrada la estructura de la población "*s*" y de cuantificación "*q*". La función presenta como salida la estructura de la población "*s*" de individuos modificados por el operador de cruzamiento.

```
%Cruzamiento individual simple de 1 punto i
%cada punto de quiebra y cruzamiento y no todo el cromossomo
function [s]=crossover(Pc,N,NY,NT,n_inf,wf,q,s)

[PopSize BreakPoints]=size(s.Px);
%Probabilidad de cada individuo
Probabilidad=rand(1,PopSize);
k=1;
m=1;
no_cruze=0;
for i=1:PopSize
    if Probabilidad(i)<=Pc
        cruze(k)=i;    %indice de los individuos a ser cruzados
        k=k+1;
    else
        no_cruze(m)=i;
        m=m+1;
    end
end
if mod(length(cruze),2)~=0
    cruze(length(cruze)+1)=randint(1,1,[1,PopSize]);
    no_cruze=no_cruze(1,1:length(no_cruze)-1);
end

num_cruze=length(cruze);
if no_cruze==0
```

```
    num_ncruze=0;
else
    num_ncruze=length(no_cruze);
end
%cruze   ->Indice de los individuos que se van a cruzar
%no-cruze->Indice de los individuos que no se van a cruzar

%****************************************************************
%CRUZAMIENTO DE X
for i=1:length(cruze)/2
    punto_cruze_x=randint(1,BreakPoints,[1 (N-1)]);
    for j=1:BreakPoints
        offspring_x(2*i-1,j)=floor(s.Px(cruze(2*i-
1),j)/2^punto_cruze_x(j)).*2^punto_cruze_x(j) +
mod(s.Px(cruze(2*i),j),2^punto_cruze_x(j));

offspring_x(2*i,j)=floor(s.Px(cruze(2*i),j)/2^punto_cruze_x(j)).*2^punto_cruze_x
(j) + mod(s.Px(cruze(2*i-1),j),2^punto_cruze_x(j));
    end
end

if no_cruze~=0
    for i=length(cruze)+1:PopSize
        offspring_x(i,:)=s.Px(no_cruze(i-length(cruze)),:);
    end
end
%****************************************************************
%CRUZAMIENTO DE Y
%Conversion de binario a decimal
MATRIZ=s.Y;
BITS_SIGNO=decbin_signo(MATRIZ,(NT-NY));
k=0;
for i=1:length(cruze)/2
    punto_cruze_y=randint(1,BreakPoints,[1 ((NT-NY+1)-1)]);
    for j=1:BreakPoints
        offspring_y(2*i-1,1+k:punto_cruze_y(1,j)+k)=BITS_SIGNO(2*i-
1,1+k:punto_cruze_y(1,j)+k);

offspring_y(2*i,1+k:punto_cruze_y(1,j)+k)=BITS_SIGNO(2*i,1+k:punto_cruze_y(1,j)+
k);

        offspring_y(2*i-1,punto_cruze_y(1,j)+1+k:(NT-
NY+1)+k)=BITS_SIGNO(2*i,punto_cruze_y(1,j)+1+k:(NT-NY+1)+k);
```

```
        offspring_y(2*i,punto_cruze_y(1,j)+1+k:(NT-NY+1)+k)=BITS_SIGNO(2*i-
1,punto_cruze_y(1,j)+1+k:(NT-NY+1)+k);

        k=k+(NT-NY+1);
    end
    k=0;
end

if no_cruze~=0
    for i=length(cruze)+1:PopSize
        offspring_y(i,:)=BITS_SIGNO(i,:);
    end

end
s.Ybits=offspring_y;
%*****************************************************************
s.num_cruze=num_cruze;
s.num_ncruze=num_ncruze;
```

7. Código fuente de la función "mutation.m"

Este código permite realizar la mutación de la población actual con el objetivo de modificar algunos individuos de la población. Esta función acepta como valores de entrada la población actual "*s*" y los valores de cuantificación "*q*", y entrega como valores de salida la estructura de población "*s*" modificada por el operador de mutación.

```
%mutacion bit a bit
function [s]=mutation(Pm,N,NY,NT,n_inf,wf,q,s)

[PopSize BreakPoints]=size(s.Px);

%*******************************************
%Mutaçao X
binario_x=decbin(s.Px,N);
taza_mutx=0;
for i=1:PopSize
    for j=(N+1):(N-1)*BreakPoints
        ran=rand(1);
        if ran<=Pm
            taza_mutx=taza_mutx+1;
            if binario_x(i,j)==1
                binario_x(i,j)=0;
            else
                binario_x(i,j)=1;
            end
```

```
        end
    end
end
decimal_x=bindec(N,binario_x);
%*****************************************
%Mutacion Y
taza_muty=0;
for i=1:PopSize
    for j=((NT-NY+1)+1):((NT-NY+1)-1)*BreakPoints
        ran=rand(1);
        if ran<=Pm
            taza_muty=taza_muty+1;
            if s.Ybits(i,j)==1
                s.Ybits(i,j)=0;
            else
                s.Ybits(i,j)=1;
            end
        end
    end
end
decimal_y=bindec_signo((NT-NY),s.Ybits,s);
%*****************************************

s.Px=sort(decimal_x')';
s.Y=decimal_y;
for i=1:PopSize
    f=floor(q.EAmax_nor(s.Px(i,:)+1));
    c=ceil(q.EAmin_nor(s.Px(i,:)+1));
    for j=1:BreakPoints
        if s.Y(i,j)>f(j)
            s.Y(i,j)=f(j);
        end
        if s.Y(i,j)<c(j)
            s.Y(i,j)=c(j);
        end
    end
end

s.Py=q.y(s.Px+1)+s.Y;

s.num_muta_x=taza_mutx;
s.num_muta_y=taza_muty;
%*****************************************
%Calculo del fitness
```

```
f=func_nor(s.Px,s.Py,N,PopSize);
for i=1:PopSize
    S(i,:)=round(f(i,:)-q.y);
    erro(i)=sum(abs(S(i,:)));
    %calculo del fitness
    fit(i)=fitness(q,S(i,:),n_inf,wf);
end
s.erro=erro;
s.fit=fit;
s.S=S;
```

8. Código fuente de la función "elitist_strategy.m"

Esta función permite seleccionar a los individuos de manera deterministica, para esto son agrupadas las poblaciones "anterior" y "actual" y a partir de esta población total son seleccionados los mejores individuos que pasaran a la siguiente generación.

```
%Seleccion elitista
function [s]=elitist_strategy(N,NY,NT,n_inf,wf,q,s_padres,s hijos,s)

[PopSize BreakPoints]=size(s.Px);

%Junto os "descendientes" con los "padres"
Px=[s_hijos.Px;s_padres.Px];
Py=[s_hijos.Py;s_padres.Py];
erro=[s_hijos.erro,s_padres.erro];
fit=[s_hijos.fit,s_padres.fit];
S=[s_hijos.S;s_padres.S];
Y=[s_hijos.Y;s_padres.Y];

competencia=struct('Px',{Px},'Py',{Py},'erro',{erro},'fit',{fit},'S',{S},'Y',{Y}
);

%Elimino los elementos repetidos con el mismo valor de fitness
A=competencia.fit;
for i=1:(length(A)-1)
    if length(A)<i
        break
    end
    B=find(A~=A(i));
    A=[A(i) A(B)];
end
for i=1:length(A)
    ind=find(competencia.fit==A(i));
    indice(i)=ind(1);
```

```
end
%Los elementos no repetidos con sus indice son:
no_rep_fit=competencia.fit(indice);
no_rep_erro=competencia.erro(indice);
no_rep_Px=competencia.Px(indice,:);
no_rep_Py=competencia.Py(indice,:);
no_rep_S=competencia.S(indice,:);
no_rep_Y=competencia.Y(indice,:);
no_rep=struct('Px',{no_rep_Px},'Py',{no_rep_Py},'erro',{no_rep_erro},'fit',{no_r
ep_fit},'S',{no_rep_S},'Y',{no_rep_Y});
%Ordeno de menor a mayor fitness y seleciono un tamaño de PopSize individuos
[fit_ord in]=sort(no_rep.fit);
individuos=in(1:PopSize);
s.fit=fit_ord(1:PopSize);
s.erro=no_rep.erro(individuos);
s.Px=no_rep.Px(individuos,:);
s.Py=no_rep.Py(individuos,:);
s.S=no_rep.S(individuos,:);
s.Y=no_rep.Y(individuos,:);
s.num_cruze=0;
s.num_ncruze=0;
s.num_muta_x=0;
s.num_muta_y=0;
s.sel_ind_x=0;
s.sel_ale_x=0;
s.sel_ind_y=0;
s.sel_ale_y=0;
```

9. Código fuente de la función "local_search.m"

Función creada para generar algunos individuos con mejor adaptación utilizando un algoritmo de búsqueda local. Estas soluciones substituirán los peores individuos de la población actual.

```
function [s,taza_LS,f2]=local_search(N,NY,NT,q,s,n_inf,wf,makespan)

%TLS=Taxa of Search Local
[PopSize BreakPoints]=size(s.Px);
s0=s;
Px1=s.Px;
Py1=s.Py;
Y1=s.Y;
Px2=s.Px;
Py2=s.Py;
```

```
Y2=s.Y;
delta=makespan;
taza_LS=0;
for i=1:round(PopSize/4)
    for j=2:(BreakPoints-1)
        delta_inf=s.Px(i,j)-s.Px(i,j-1)-1;
        delta_sup=s.Px(i,j+1)-s.Px(i,j)-1;
        %Busca hacia la izquierda <-
        if delta_inf>=delta
            x_neighbor_inf=s.Px(i,j)-randint(1,1,[1 delta]);
            Px1(i,j)=x_neighbor_inf;
            my1=floor(q.EAmax_nor(Px1(i,j)+1));
            Py1(i,j)=q.y(Px1(i,j)+1)+my1;
            %calculo fitness con x_neighbor_inf
            [a1,b1]=sort(Px1(i,:));
            Px1(i,:)=a1;
            Py1(i,:)=Py1(i,b1);
            f1=func_nor(Px1(i,:),Py1(i,:),N,1);
            S1=round(f1-q.y);
            erro1=sum(abs(S1));
            fit_inf=fitness(q,S1,n_inf,wf);
            if fit_inf<s.fit(i)
                taza_LS=taza_LS+1;
                s.fit(i)=fit_inf;
                s.Px(i,:)=Px1(i,:);
                s.Py(i,:)=Py1(i,:);
                s.erro(i)=erro1;
                s.S(i,:)=S1;
                s.Y(i,j)=my1;
            end
        end
        %Busca hacia la derecha->
        if delta_sup>=delta
            x_neighbor_sup=s.Px(i,j)+randint(1,1,[1 delta]);
            Px2(i,j)=x_neighbor_sup;
            my2=floor(q.EAmax_nor(Px2(i,j)+1));
            Py2(i,j)=q.y(Px2(i,j)+1)+my2;
            %calculo fitness con x_neighbor_sup
            [a2,b2]=sort(Px2(i,:));
            Px2(i,:)=a2;
            Py2(i,:)=Py2(i,b2);
            f2=func_nor(Px2(i,:),Py2(i,:),N,1);
            S2=round(f2-q.y);
            erro2=sum(abs(S2));
```

```
            fit_sup=fitness(q,S2,n_inf,wf);
            if fit_sup<s.fit(i)
                taza_LS=taza_LS+1;
                s.fit(i)=fit_sup;
                s.Px(i,:)=Px2(i,:);
                s.Py(i,:)=Py2(i,:);
                s.erro(i)=erro2;
                s.S(i,:)=S2;
                s.Y(i,j)=my2;
            end
        end
    end
end
```

BIBLIOGRAFIA

[1] ALAN BURNS, ANDY WELLINGS, "Real-Time Systems and Programming Languages", Addison Wesley, 1996.

[2] VAHID F., GIVARGIS, T., "Embedded System Design: A unified Hardware/Software Approach", Department of computer Science and Engineering University of California, 1999.

[3] CATUNDA, S.Y.C., SAAVEDRA, O.R., "Constraints definition and evaluation of piecewise polynomial approximation functions for embedded systems", 19th IEEE Proceedings of Instrumentation and Measurement Technology Conference 2002, pp: 1103 – 1108, vol2.

[4] MORAWSKI, R.Z., "Unified approach to measurand reconstruction, Instrumentation and Measurement", IEEE Transactions on, Volume: 43, Issue: 2, pp: 226 – 231, 1994.

[5] MORAWSKI, R.Z., "Digital signal processing in measurement Microsystems", Instrumentation & Measurement Magazine, IEEE, Volume: 7, Issue: 2, June 2004, pp: 43 – 50.

[6] RANDY FRANK, "Understanding Smart Sensors", Artech House, Boston – London, 1996.

[7] JOHN G. WEBSTER, "Measurement, Instrumentation and Sensors – HANDBOOK", Chapman & Hall, 1999.

[8] JOHN G. PROAKIS, DIMITRIS G. MANOLAKIS, "Tratamiento Digital de Señales, Principios, Algoritmos y Aplicaciones", Prentice Hall, 3ra Edición, 1998.

[9] The Math Works, "Simulink Fixed Point, For Use With Simulink, User's Guide", Version 5, 2004.

[10] CATUNDA, S.Y.C., NAVINER, J. F., DEEP, G.S., FREIRE, R.C.S., "Optimized look-up table for inter-mesurand compensation", Instrumentation and Measurement Technology Conference, 2001, IMTC 2001, Proceedings of the 18th IEEE, Vol. 1, 21-23 May 2001, pp: 128-132.

[11] CATUNDA, S.Y.C., NAVINER, J.F., DEEP, G.S., FREIRE, R.C.S., "Look-up table otimizada para reconstrução de valores de medição utilizando sensores não lineares". XIII Congresso Brasileiro de Automática, Florianópolis, Brasil, Setembro 2000.

[12] FLAMMINI, A., MARIOLI, D., TARONI, A., "Application of an optimal look-up table to sensor data processing", Instrumentation and Measurement, IEEE Transactions on, Vol. 48, Issue: 4, Aug. 1999, pp: 813-816.

[13] CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., e MORAIS, M.R.A., "Determinação de tabela de equivalência e pontos de quebra de funções de aproximação lineares por partes usando computação evolutiva", 6° SBIA Simpósio Brasileiro de Automação Inteligente, 14-17 Setembro 2003, pp: 662-667.

[14] AHMED, M.A., DEJONG, K.A., "Function Approximator design using Genetic Algorithms, Evolutionary Computation", IEEE International Conference on, 13-16 April 1997, pp: 519 – 524.

[15] CATUNDA, S.Y.C., SAAVEDRA, O.R., "Constraints definition and evaluation of piecewise polynomial approximation functions for embedded systems", 19th IEEE Proceedings of Instrumentation and Measurement Technology Conference 2002, pp. 1103-1108 vol2.

[16] MAURICIO, J.M., CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., "Determinação de aproximação de funções não lineares para sistemas embarcados utilizando algoritmos genéticos híbridos", VII Simpósio Brasileiro de Automação Inteligente, São Luis, Brasil, Setembro 2005.

[17] GOLDBERG, D.E., "Genetic Algorithms in Search, Optimization, and Machine Learning", Reading Mass.: Addison – Wesley Pub. Co., 1989.

[18] MICHALEWICZ, Z., "Genetic Algorithms + DataStructures = Evolution Programs", Third, Revides and Extended Edition, Pub. Springer, 1996.

[19] AHMED, M.A., DEJONG, K.A., "Function Approximator design using Genetic Algorithms", Evolutionary Computation, IEEE International Conference on, 13-16 April 1997, pp: 519 – 524.

[20] The MathWorks, "Simulink Fixed Point, For Use With Simulink, User's Guide" Version 5, 2004.

[21] CATUNDA, S.Y.C., NAVINER, J.F., DEEP, G.S., FREIRE, R.C.S., "Look-up table otimizada para reconstrução de valores de medição utilizando sensores não lineares". XIII Congresso Brasileiro de Automática, Florianópolis, Brasil, Setembro 2000.

[22] CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., e MORAIS, M.R.A., "Determinação de tabela de equivalência e pontos de quebra de funções de aproximação lineares por partes usando computação evolutiva", 6° SBIA Simpósio Brasileiro de Automação Inteligente, 14-17 Setembro 2003, pp: 662-667

[23] MAURICIO, J.M., CATUNDA, S.Y.C., SAAVEDRA, O.R., FONSECA NETO, J.V., "Aproximação de funções: Uma abordagem utilizando computação evolutiva híbrida", 8th Simpósio Brasileiro de Redes Neurais, São Luis, Brasil, Setembro 2004, 3590.

[24] FOGEL, D.B., "An introduction to simulated evolutionary optimization", Neural Networks, IEEE Transactions on, Vol. 5, Issue :1, Jan. 1994, pp : 3-14.

[25] HARTMUNT POHLHEIM, "GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB", http://www.geatbx.com/docu/index.html, acessado em 09/11/2004.

[26] BACK, T., HAMMEL, U., SCHWEFEL, H.P., "Evolutionary computation: comments on the history and current state", Evolutionary computation, IEEE Transactions on, Volume: 1, Issue: 1, April 1997, pp: 3 – 17.

[27] VON ZUBEN, F.J., "Computação Evolutiva: Uma Abordagem Pragmática", Notas de aula da disciplina IA707 DCA/FEEC/Unicamp/Brasil, 2000.

[28] MICHALEWICZ, Z., "Heuristic Methods for Evolutionary Computation Techniques", Journal of Heuristics, 1995, Vol. 1, N° 2, pp: 177-206.

[29] YEARY, M.B., FINK, R.J., BECK, D., GUIDRY, D.W., BURNS, M., "A DSP-based mixed-signal waveform generator", Instrumentation and Measurement, IEEE Transactions on, Volume: 53, Issue: 3, June 2004, pp: 665-671.