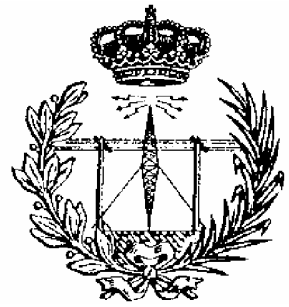
ESCUELA SUPERIOR DE INGENIEROS
UNIVERSIDAD DE SEVILLA
Ingeniería Superior de Telecomunicación

Proyecto Fin de Carrera



Diseño y desarrollo de una herramienta de simulación de convertidores Sigma-Delta cascada tiempo discreto en MATLAB

Autor del Proyecto
Roberto C. Quílez Clemente

Tutor del Proyecto
Luis Miguel Rivas Asensio

Departamento de Ingeniería Electrónica, Grupo de Tecnología Electrónica. Universidad de Sevilla

Sevilla, Diciembre 2003

Alguien me pide una explicación de la teoría de Einstein. Con mucho entusiasmo, le hablo de tensores y geodésicas tetradimensionales.

—No he entendido una sola palabra— me dice, estupefacto.

Reflexiono unos instantes y luego, con menos entusiasmo, le doy una explicación menos técnica, conservando algunas geodésicas, pero haciendo intervenir aviadores y disparos de revólver.

—Ya entiendo casi todo— me dice mi amigo, con bastante alegría. Pero hay algo que todavía no entiendo: esas geodésicas, esas coordenadas...

Deprimido, me sumo en una larga concentración mental y termino por abandonar para siempre las geodésicas y las coordenadas; con verdadera ferocidad, me dedico exclusivamente a aviadores que fuman mientras viajan con la velocidad de la luz, jefes de estación que disparan un revólver con la mano derecha y verifican tiempos con un cronómetro que tienen en la mano izquierda, trenes y campanas.

—¡Ahora sí, ahora entiendo la relatividad!— exclama mi amigo con alegría.

—Sí —le respondo amargamente—, pero ahora no es más la relatividad.

"Uno y el Universo", Ernesto Sábato

A mis padres, mis hermanos y mi novia.

A mi tutor, Luis Miguel Rivas, por su ayuda y dedicación.

A todos con los que he compartido algo durante estos años.

Índice

DISEÑO Y DESARROLLO DE UNA HERRAMIENTA DE SIMULACIÓN DE CONVERTIDORES SIGMA-DELTA CASCADA TIEMPO DISCRETO EN MATLAB

Capítulo 1
Introducción

1.1 Por qué un PFC sobre un CA/D

El carácter eminentemente analógico de nuestro entorno origina la necesidad de utilizar una interfaz para adecuar las distintas señales de información provenientes del mismo a los sistemas digitales.

La mayoría de las fuentes de información prácticas, señales de voz, audio, video, biológicas, etc., son de naturaleza analógica, esto hace necesaria una conversión de formato analógico a digital si se pretende realizar sobre la señal de información algún tipo de procesamiento digital, de filtrado o análisis espectral. El procesado digital de señales proporciona una técnica alternativa al tratamiento de una señal analógica convencional, pues permite aplicar complejos algoritmos matemáticos mediante software o implementación con circuitos lógicos, y, además, aprovecha las ventajas propias de los sistemas digitales frente a los analógicos como veremos posteriormente.

La operación de conversión analógico a digital está sujeta a un compromiso entre la velocidad y la resolución en los CA/D. Esto es, una arquitectura válida para una velocidad alta de operación tendrá una resolución "baja" o no tan buena como su velocidad y viceversa para un mismo consumo de potencia.

La elección de una arquitectura concreta de un CA/D dependerá en la mayoría de las ocasiones de la posibilidad de implementar dicho CA/D utilizando la misma tecnología que la circutería digital de procesamiento posterior. Este hecho haría que sistemas completos se pudieran integrar sobre un mismo substrato semiconductor, y esto se traduce en una reducción de tamaño, consumo de potencia, coste, etc. Por tanto, en la elección de una arquitectura de CA/D y de los bloques de circuito que los

componen, un factor importante a tener en cuenta es su compatibilidad con tecnología estándar VLSI CMOS.

Además del circuito tecnológico y la técnica de circuito empleada, los CA/D se clasifican atendiendo a la razón de sobremuestreo en dos grandes categorías:

- Convertidores de Nyquist, cuya razón de sobremuestreo es igual o ligeramente superior a uno.
- Covertidores de sobremuestreo, cuya razón de sobremuestreo es mucho mayor que uno.

Ante la desventaja que supone utilizar una frecuencia de muestreo superior a la de Nyquist como hacen los CA/D de sobremuestreo (ya que el circuito de muestreo debe funcionar más rápido, y esto se traduce en un mayor consumo de potencia del circuito), no sería conveniente utilizarlos si no fuera por ventajas que presentan, entre otras:

- Relajación de las especificaciones de la circutería analógica a costa de un uso extensivo de la circutería digital.
- El filtro analógico previo al CA/D utilizado para prevenir el solapamiento del espectro de la señal tras muestrearla podría ser tan simple como un circuito pasivo de primer orden, al tener unas especificaciones más relajadas en frecuencia que el caso de CA/D de Nyquist.

Estas ventajas se comprenderán mejor más adelante. En este Proyecto nos vamos a centrar en el estudio de los CA/Ds de sobremuestreo, en concreto en las arquitecturas basadas en modulación Sigma-Delta ($\Sigma\Delta$) y particularmente en la configuración en cascada.

De las arquitecturas existentes para los CA/D, las basadas en modulación Sigma-Delta ($\Sigma\Delta$) presentan un gran atractivo, aparte de las ventajas mencionadas anteriormente, resultan de especial interés en los llamados *sistemas-sobre-chip* debido, entre otras, a las siguientes razones:

- Amplio rango de aplicaciones: los CA/D que podemos desarrollar con esta arquitectura son muy versátiles funcionalmente, pudiendo diseñar CA/D que van desde aplicaciones de instrumentación hasta de telecomunicaciones.
- Robustez y alta tolerancia a imperfecciones de la circutería: Ya que sustituyen precisión de los componentes por procesamiento de señal, los CA/Ds Sigma-Delta ($\Sigma\Delta$) resultan muy apropiados para ser integrados en tecnología CMOS estándar, las cuales están optimizadas para circuitos digitales rápidos, pero resultan poco apropiados para circuitos analógicos precisos

Diferentes técnicas de circuitos son utilizadas en la implementación de los moduladores ΣΔ , como son la de Capacidades Conmutadas (SC del inglés "Switched Capacitor"), Corrientes Conmutadas (SI del inglés "SwItched current") o incluso las más reciente de Tiempo Continuo (CT del inglés "Continuous Time").

Como hemos señalado, una arquitectura especialmente interesante de modulación ΣΔ para la obtención de moduladores de alto orden incondicionalmente estables es la arquitectura en cascada, que se basa en la conexión en cascada de moduladores de bajo orden (orden 1 ó 2 típicamente) donde podemos asegurar la estabilidad. La lógica digital será la encargada de eliminar el ruido de cuantización resultante de cada etapa. Considerando la implementación con un cuantizador de un solo bit, las principales ventajas de esta arquitectura son:

- Se obtienen *SNR* altos para factores de sobremuestreo (*OSR*) de bajo valor.
- Estabilidad garantizada (pues se basan en arquitecturas de lazo simple de bajo orden, y por tanto, estables).
- Utilización del máximo rango de la señal de entrada.

No obstante hay que considerar que esta arquitectura conlleva una alta sensibilidad a ciertas no linealidades circuitales.

Una última característica a destacar de esta arquitectura es que, si bien se complica la parte digital del CA/D al incluir una lógica de cancelación, esto favorece la integración en tecnología VLSI (Very Large Scale Integration) estándar y se beneficia de este modo del escalado tecnológico.

1.2 Objetivos del proyecto y panorama general

Como hemos comentado en el apartado anterior el interés por los convertidores ΣΔ se ha incrementado en los últimos años debido a dos razones:

- A diferencia de otros convertidores que necesitan bloques básicos muy precisos para obtener una resolución alta, los convertidores ΣΔ muestran baja sensibilidad a las imperfecciones de sus bloques básicos.
- El número de aplicaciones con interés industrial ha crecido.

Dado que los moduladores ΣΔ se encuentran en la interfaz analógica-digital, carecen de metodologías semiautomáticas para su diseño, como ocurre con los circuitos puramente digitales. Por lo tanto, se necesitan unas herramientas que ayuden a simular su diseño.

Este proyecto fin de carrera pretende satisfacer la necesidad de simuladores de comportamiento para convertidores $\Sigma\Delta$, mediante una interfaz amigable, versátil y sencillo e intuitivo manejo. No se pretende, no obstante, desarrollar una herramienta definitiva sino establecer una base sobre la que poder construir nuevas ampliaciones y mejoras.

En particular, se fija como objetivo desarrollar un simulador de moduladores $\Sigma\Delta$ en configuración cascada tiempo discreto bajo entorno MATLAB, programa para cálculo técnico y científico, con lenguaje de programación propio.

Este programa, para ciertas operaciones es muy rápido, cuando puede ejecutar sus funciones en código nativo con los tamaños más adecuados para aprovechar sus capacidades de vectorización.. En otras aplicaciones resulta bastante más lento que el código equivalente desarrollado en C/C++ o Fortran. Sin embargo esta herramienta ha sido seleccionada debido a que MATLAB es una magnífica herramienta de alto nivel para desarrollar aplicaciones técnicas, fácil de utilizar y que aumenta la productividad de los programadores a otros entornos de desarrollo. Asimismo, resulta interesante el hecho de que MATLAB dispone de un código básico y de varias librerías especializadas (toolboxes).

MATLAB permite desarrollar de una manera simple interfaces de usuario (conjunto de pantallas con botones, menús, ventanas, etc.), que permiten utilizar de manera muy simple programas realizados en el entorno de Windows.

1.3 Organización del proyecto

El proyecto ha sido organizado por capítulos del siguiente modo:

- *Capítulo 1 Introducción*

  Organización y objetivos del proyecto fin de carrera.

- *Capítulo 2 Conversión Analógico/Digital*

  Conceptos básicos acerca de la conversión Analógico a Digital.

- *Capítulo 3 Modulación Sigma-Delta*

  Fundamentos de los convertidores de sobremuestreo y los moduladores sigma-delta. Configuración en cascada de los mismos.

- *Capítulo 4 Descripción general. Manual de usuario*

  Descripción general del programa de simulación de moduladores sigma-delta tiempo discreto en cascada. Atenderemos aspectos como son la instalación, ejecución e interfaz de usuario y su correspondiente manejo. Obviaremos en todo momento la implementación real que subyace a las mismas (código MATLAB, funciones de programación, variables…) ya que ésta será tratada en detalle en capítulos posteriores

- *Capítulo 5 Funciones, estructura y mapa del código.*

  Aspectos relativos a la programación en MATLAB del proyecto: funciones y estructura del código, algoritmos implementados, sus variables de entrada y salida, la interconexión con otras funciones y sus características y propiedades más interesantes (versatibilidad, flexibilidad, carácter vectorial o escalar…).

- *Capítulo 6 Evolución del proyecto. Test y pruebas*

  En este capítulo describiremos la evolución seguida durante progreso del proyecto. Veremos las pruebas y tests desarrollados a lo largo del mismo, los problemas presentados y soluciones aportadas.

  Comprobaremos también el modo en que ha sido organizado el desarrollo del proyecto, sus fases o etapas.

- *Capítulo 7 Conclusión y futuras líneas de desarrollo*

  Concluimos citando las principales características de la herramienta desarrollada.

  Indicamos posibles ampliaciones o líneas de desarrollo.

- *Bibliografía*

- *Anexo 1: Listado de archivos*

  Listado de los archivos que comprende el proyecto

- *Anexo 2: Código MATLAB*

  Listado completo del código de programación MATLAB.

Capítulo 2
Conversión Analógico/Digital

2.1 Introducción

Un Convertidor Analógico-Digital (CA/D) es un circuito electrónico que transforma una señal continua en el tiempo y en amplitud (señal analógica) en otra señal discreta en el tiempo y cuya amplitud está cuantificada y codificada, generalmente mediante un código binario (señal digital).

El CA/D es uno de los bloques esenciales que integran un sistema electrónico que utiliza procesamiento digital de señal.

Varias son las ventajas del procesamiento en el dominio digital de señales frente al mismo en el dominio analógico. Entre ellas podemos destacar:

- Robustez ante los ruidos e interferencias: como las señales digitales se representan con un número discreto valores eléctricos, resulta muy difícil para una perturbación contaminar una señal digital tanto como para llevarla a otro valor discreto. Esto no ocurre con una señal analógica, donde una pequeña perturbación puede variar el valor de dicha señal y alterar su posterior lectura y/o procesado.
- Diseño y desarrollo de circutería simple: como en último término tenemos secuencias de valores discretos, el diseño es *customizable,* de ahí el que cada vez más el diseño de circuitos digitales tenga más un carácter de *software* (o de programación) que de *hardware* (con elementos físicos). Esto se debe a que los circuitos digitales son más modulares que los analógicos, de manera que se

puede diseñar un bloque digital y su funcionalidad no se altera sustancialmente cuando ese bloque se conecta a otros. Este no es el caso de la circutería analógica.

En la Figura 2.1 se ilustra un esquema general de un sistema de procesamiento digital de señales:

Figura 2-1 Sistema de procesado digital de señales

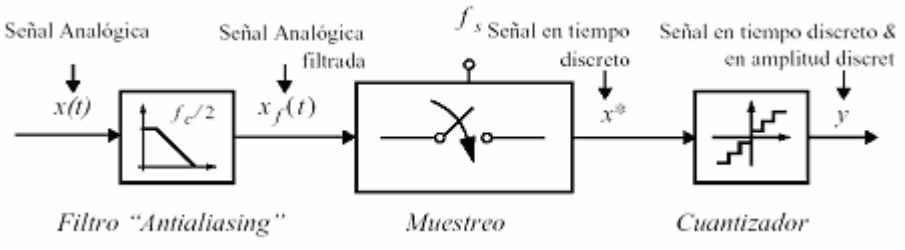
La señal de salida del procesador digital puede necesitar nuevamente una conversión de formato, en este caso de digital a analógico, antes de ser entregada al receptor de la información (Figura 2.1); o, por el contrario, puede no requerir de ninguna conversión, preservando la señal el carácter digital.

2.2 Conceptos básicos

La conversión de una señal analógica a una señal digital incluye tres procesos fundamentales, que son:

- Prefiltrado antisolapamiento (o "antialiasing").
- Muestreo.
- Cuantización.

Los bloques asociados a estos procesos se muestran en la figura 2-2:



tidor A/D

Figura 2-2 Diagrama general de un convertidor A/D

Junto a estos tres procesos fundamentales veremos otros dos, que son:

- Codificación.
- Conversión de la tasa de muestreo.

2.2.1 Filtro analógico antisolapamiento

La señal analógica realmente nunca estará, estrictamente hablando, limitada en banda, por lo que filtramos la señal para que la entrada de nuestro muestreador sea una señal limitada en banda y no se produzca el fenómeno de solapamiento de bandas frecuenciales (o "aliasing"), que conllevaría una pérdida de información de la señal, y por lo tanto podría llegar a inutilizar el sistema. En concreto este filtro eliminará las componentes frecuenciales de la señal de entrada por encima de la frecuencia de Nyquist.

2.2.2 Muestreo

El muestreo es el procedimiento mediante el cual una señal analógica $x_a(t)$, continua en el tiempo, se transforma en una señal en tiempo discreto $x(n)$, formada por las "muestras" tomadas de la señal original en distintos instantes de tiempo discreto. El método de muestreo más utilizado es el muestreo *uniforme* (Figura 2.3), que toma las muestras en intervalos de tiempo equiespaciados (T_s):

$$x(n) = x_a(nT_s), \Leftarrow -\infty < n < +\infty$$

Donde T_s es el periodo de muestreo y $F_s=1/T_s$ es la frecuencia de muestreo.

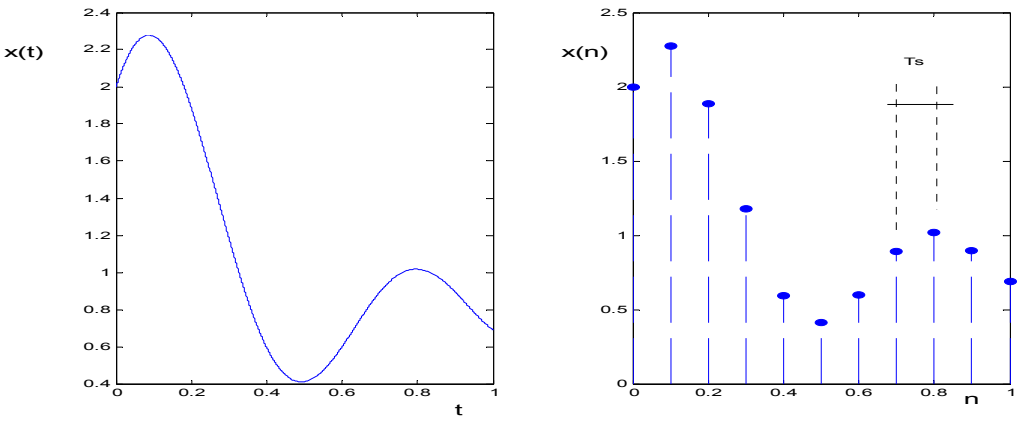


Figura 2-3 Muestreo uniforme de una señal analógica.

2.2.2.1 Teorema del muestreo

Teorema del muestreo:

"Si una señal continua, ***S(t)****, tiene una banda de frecuencia tal que* f_m *se la mayor frecuencia comprendida dentro de dicha banda, dicha señal podrá reconstruirse si distorsión a partir de muestras de señal tomadas a una frecuencia* f_s*, siendo* $2.f_m < f_s$*".*

La señal $x_a(t)$ se puede reconstruir totalmente a partir de sus muestras si la tasa de muestreo o frecuencia F_s cumple que $F_s > 2B$, siendo B la componente espectral máxima de la señal $x_a(t)$.

$F_N = 2B$ se denomina tasa o frecuencia de *Nyquist.*

Teorema de Muestreo $\Rightarrow F_s > F_N$

Si la frecuencia de muestreo F_s es inferior a la frecuencia de *Nyquist* F_N se produce un solapamiento del espectro de la señal (*aliasing*), lo que imposibilita una reconstrucción perfecta de la señal.

2.2.3 Cuantización

La cuantificación o cuantización convierte una señal en tiempo discreto, con la amplitud definida en un intervalo continuo, en una señal en tiempo discreto definida únicamente para un conjunto de valores discretos de amplitud. La señal discreta se limita a un número finito de amplitudes posibles, resultando una diferencia entre la señal cuantizada y la señal sin cuantizar denominada *error de cuantización*.

$$e_q(n) = x_q(n) - x(n) \ , \ x_q(n) = Q[x(n)]$$

Las amplitudes permitidas para la señal discreta se denominan niveles de cuantización. La resolución del cuantizador es la distancia entre dos niveles de cuantificación sucesivos (Δ), cuantos más niveles se usen (L) mayor precisión y menor error de cuantificación.

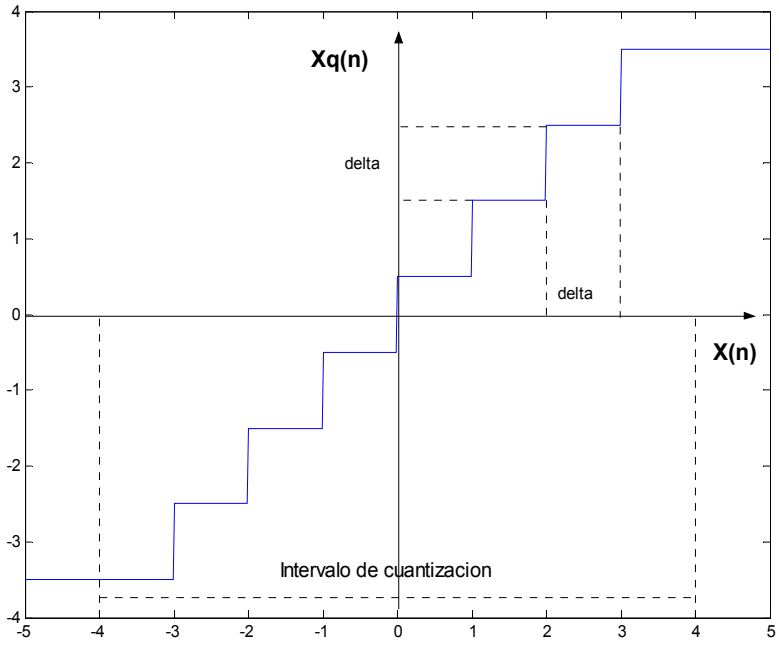


Figura 2-4 Cuantizador de 8 niveles.

2.2.4 Codificación

La codificación consiste en asignar a cada valor discreto de la señal cuantizada $x_q(n)$ una secuencia de *b* bits. El número de niveles de cuantización está fijado por los bits de precisión del codificador, y viceversa. Si disponemos de b bits en el codificador se pueden representar un total de 2^b números binarios, luego el número de niveles de cuantización máximo se define como $L = 2^b$.

Alternativamente, se puede decir que el número mínimo de bits del codificador debe cumplir la relación: $b \geq \log_2 L$

2.2.5 Conversión de la tasa de muestreo

El proceso de cambiar la frecuencia de muestreo de una señal digital a otra frecuencia dada se denomina conversión de la tasa de muestreo. Este proceso ofrece varias posibilidades para el factor de conversión:

- Diezmado o Submuestreo (1/D): consiste en reducir la tasa de muestreo por un factor entero D. Produce un ensanchamiento del espectro de la señal digital de entrada respecto de la nueva frecuencia.
- Interpolación o Sobremuestreo (I): incrementa la tasa de muestreo por un factor entero I. Produce una compresión del espectro de la señal digital de entrada respecto de la nueva frecuencia.

- Factor racional I / D: es una combinación de los dos casos anteriores.

2.3 Convertidor A/D

En este capítulo hemos tratado aspectos fundamentales de la conversión analógica/digital sin considerar en ningún momento su arquitectura interna o diseño de circuito. Los convertidores han sido tratados como *cajas negras* tales que sólo se han discutido sus relaciones entre entrada y salida.

Definiremos a continuación ciertas figuras de mérito de los CA/D que nos posibilitarán concretar las mejoras obtenidas como resultado del sobremuestreo y conformación del ruido que trataremos en el siguiente capítulo.

2.3.1 Figuras de Mérito

En este sub-apartado vamos a definir figuras de mérito típicas de los CA/Ds, de manera que en posteriores secciones podamos argumentar cómo mejorar el compromiso entre velocidad y resolución en el caso de los CA/D de sobremuestreo.

2.3.1.1 Relación señal ruido

Se define la relación señal-ruido (*SNR* del inglés "Signal to Noise Ratio") como el cociente entre la potencia de salida a la frecuencia de la entrada (supuesta ésta sinusoidal) y la potencia en banda del ruido total. Si tenemos en cuenta tan solo el ruido de cuantización y que la amplitud de la entrada sinusoidal al CA/D es A, la *SNR* es:

$$SNR\big|_{dB} = 10\log\left(\frac{A^2/2}{P_e}\right)$$

donde *log* significa logaritmo en base diez. Vemos que a priori la *SNR* aumenta de forma monótona con el nivel de entrada. Pero esto sólo ocurre hasta un cierto valor, donde un exceso en el nivel de la señal de entrada del cuantizador provocará un aumento desmesurado en el ruido de cuantización (una vez la amplitud de la entrada sobrepase el fondo de escala del cuantizador), y por tanto una bajada abrupta de *SNR* a la salida del cuantizador, y si extendemos el estudio a los moduladores sigma-delta, a la salida del modulador.

2.3.1.2 Rango dinámico

Se define el rango dinámico (*DR* del inglés "Dinamic Range") como el cociente entre la potencia a la salida a la frecuencia de una sinusoide con una amplitud igual al rango completo del cuantizador y la potencia a la salida para una sinusoide que tenga la misma frecuencia y amplitud tal que no sea distinguible del fondo de ruido; esto es, con una *SNR*=0dB. Usualmente se expresa en dB.

Idealmente el rango dinámico de entrada viene limitado por la escala completa de entrada del cuantizador. Por otro lado la potencia de salida para una sinusoide de amplitud tal que su *SNR*=0dB vale P_e. Por tanto,

$$DR\big|_{dB} = 10\log\left[\frac{\left(x_{FS}/2\right)^2}{2P_e}\right]$$

2.3.1.3 Resolución equivalente

Se define resolución equivalente de un CA/D (y se representa por *b*) como:

$$b(bits) = \frac{DR(dB) - 1.76}{6.024}$$

Es importante indicar que *b* puede no coincidir con el número de bits del cuantizador usado en el CA/D.

Capítulo 3
Modulación Sigma-Delta

3.1 Introducción

Recientemente, los convertidores A/D y D/A de sobremuestreo están tomando popularidad para aplicaciones de alta resolución y velocidad media-baja tales como de audio digital de alta calidad. Entre las razones más importantes para este auge se incluyen las siguientes:

- Los convertidores de sobremuestreo permiten relajar las especificaciones de la circutería analógica en contrapartida de una circutería digital más compleja. Este intercambio es deseable cara a tecnologías modernas submicra con fuentes de 3.3V, donde la compleja circutería digital de alta-velocidad es fácilmente realizable en menor área mientras que la realización de circuitos analógicos de alta resolución es complicada debido a fuentes de alimentación de bajo voltaje y pobre impedancia de salida de transistor (causado por efectos de canal-corto). Con los convertidores de sobremuestreo, los componentes analógicos han reducido sus requerimientos cara a tolerancias pareadas y ganancias de los amplificadores.
- Los convertidores de sobremuestreo simplifican los requerimientos de los filtros analógicos anti-solapamiento (*anti-aliasing*) para convertidores A/D y filtros de alisado (*smoothing*) para D/A. Por ejemplo, para un CA/D suele ser requerido un solo con filtro anti-solapamiento de primer orden, que puede ser realizado en el mismo chip o, en el peor caso, en un chip externo de muy bajo coste. Además, un muestreador y mantenedor (*sample and hold*) no suele ser necesario a la entrada de un convertido A/D de sobremuestreo.

En este capítulo comenzaremos discutiendo los aspectos básicos del sobremuestreo. Comprobaremos después que podremos obtener una resolución extra de bits fruto de muestrear mucho más rápido de la frecuencia de Nyquist. Además, veremos asimismo, que esta resolución extra puede ser conseguida con tasas de sobremuestreo inferiores empleando *"noise-shaping"* que traduciremos como *moldeado* o c*onformación* (emplearemos ambos términos indistintamente) del espectro del ruido de cuantización a través del uso de la realimentación. El uso de la conformación del ruido de cuantización a señales sobremuestreadas lo referiremos como modulación sigma-delta. Discutiremos finalmente los moduladores $\Sigma\Delta$ de primer y segundo orden y su configuración en cascada.

3.2 Sobremuestreo sin conformación de ruido

En esta sección trataremos las ventajas de muestrear a tasas superiores a la de Nyquist. Comprobaremos que se puede obtener un rango dinámico extra ensanchando el espectro de la potencia del ruido de cuantización sobre un rango de frecuencias mayor. Este incremento, no obstante, será sólo de 3dB por cada duplicado de la tasa de muestreo. Para obtener un rango dinámico mucho mayor conforme se incrementa la tasa de muestreo, se puede usar el moldeado del ruido a través de realimentación como discutiremos en la próxima sección (Ver apartado 3.3).

3.2.1 Modelo del ruido de cuantización

Comenzaremos modelando un cuantizador como la adición de un ruido de cuantización *e(n)* como muestra la siguiente figura 3-1.

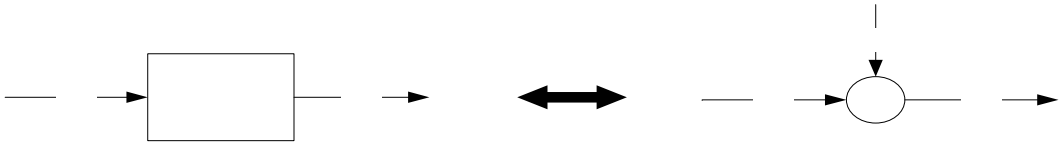


Figura 3-1 Cuantizador y su modelo lineal

La señal de salida *y(n)*, es igual al valor cuantizado más próximo de *x(n)*. El error de cuantización es la diferencia entre los valores de entrada y salida. Este modelo es exacto si se reconoce que el error de cuantización no es una señal independiente, sino que puede estar fuertemente relacionada con la señal de entrada, *x(n)*. Este modelo lineal es aproximado cuando las suposiciones son hechas sobre las propiedades estadísticas de *e(n)*, tales como que *e(n)* es un ruido blanco (*white-noise*). Sin embargo, incluso con la aproximación, este modelo permite una sencilla comprensión de los moduladores $\Sigma\Delta$ y, con algunas excepciones, es razonablemente precisa.

3.2.2 Suposición de ruido blanco

Si *x(n)* es muy activa, *e(n)* puede ser aproximado como una variable aleatoria uniformemente distribuida entre ±Δ/2, donde Δ equivale a la diferencia entre dos niveles de cuantización adyacentes. De este modo la potencia del ruido de cuantización vale $\Delta^2/12$ y es independiente de la frecuencia de muestreo, f_s. También, la densidad espectral de *e(n), $S_e(f)$,* es blanca (constante en frecuencia) y toda su potencia está concentrada en $\pm f_s/2$.

Asumiendo ruido de cuantización blanco, la densidad espectral del ruido de cuantización, $S_e(f)$, será como la de la figura 3-2

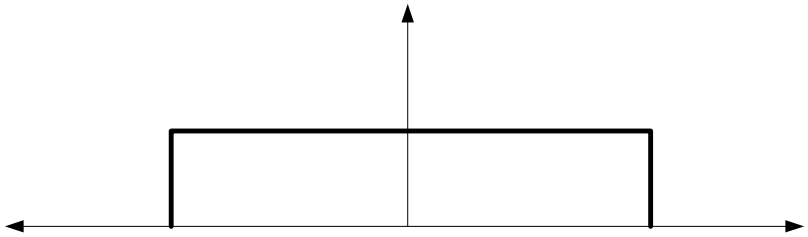


Figura 3-2 Densidad espectral del ruido de cuantización asumida

La altura de la densidad espectral se puede calcular como consecuencia de que la potencia total del ruido es $\Delta^2/12$ y ésta equivale al área bajo $S_e(f)$, contenida entre $\pm f_s/2$, o matemáticamente,

$$\int_{-f_s/2}^{f_s/2} S_e^2(f)df = \int_{-f_s/2}^{fs/2} k_x^2 df = k_x^2 f_s = \frac{\Delta^2}{12}$$

Solucionando la relación anterior

$$k_x = \left(\frac{\Delta}{\sqrt{12}}\right)\sqrt{\frac{1}{f_s}}$$

3.2.3 Ventajas del sobremuestreo

El sobremuestreo se da cuando las señales de interés están limitadas en banda a f_0 mientras que la tasa de muestreo es f_s, donde $f_s > 2f_0$ (siendo $2f_0$ la *frecuencia de Nyquist*). Definimos la tasa de sobremuestreo, *OSR* (*oversampling ratio,* también aparece en la literatura identificada por *M*), como:

$$OSR \equiv \frac{f_s}{2f_0}$$

Después de la cuantización, una vez que las señales de interés están todas por debajo de f_0, $y_1(n)$ es filtrado por $H(f)$ para generar la señal $y_2(f)$, como muestra la figura 3-3. Este filtro elimina el ruido de cuantización mayor que f_0.

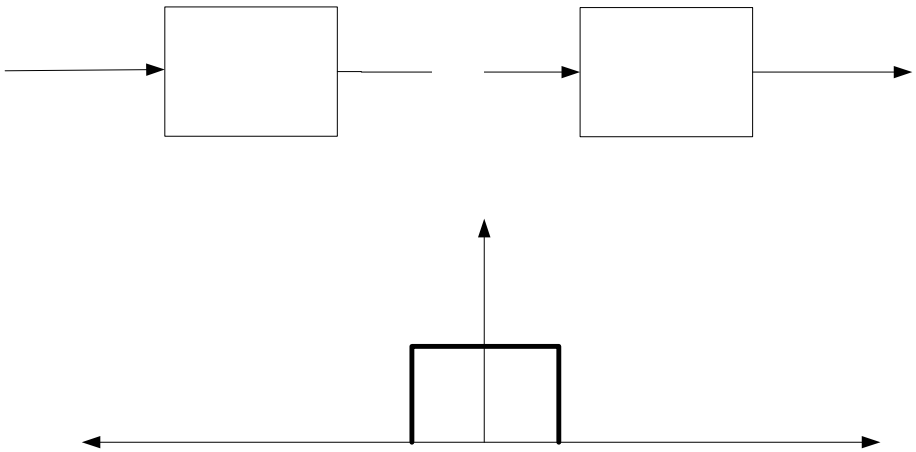


Figura 3-3 (a) Posible sistema de sobremuestreo sin conformación de ruido. (b) Respuesta frecuencial del filtro para eliminar gran parte del ruido de cuantización.

Asumiendo que la señal de entrada es una onda sinusoidal, su máximo valor de pico es $A=2^N(\Delta/2)$. Para este máximo de la onda sinusoidal, la potencia de la señal, P_s, es:

$$P_s = \left(\frac{A}{\sqrt{2}}\right)^2 = \left(\frac{\Delta 2^N}{2\sqrt{2}}\right)^2 = \frac{\Delta^2 2^{2N}}{8}$$

La potencia de la señal dentro de $y_2(n)$ permanece siendo la misma que anteriormente ya que asumimos que el contenido frecuencial de la señal está por debajo de f_0. Sin embargo, la potencia del ruido de cuantización es reducida a

$$P_e = \int_{-f_s/2}^{f_s/2} S_e^2(f)\left|H(f)\right|^2 df = \int_{-f_o}^{f_0} k_x^2 df = \frac{2f_0}{f_s}\frac{\Delta^2}{12} = \frac{\Delta^2}{12}\left(\frac{1}{OSR}\right)$$

De esta forma, duplicando *OSR* (es decir, muestreando a dos veces la tasa) se decrementa la potencia del ruido de cuantización en un medio o, equivalentemente, 3dB (ó 0.5 bits).

También podemos calcular el máximo *SNR* (en dB) al ser la relación entre la potencia máxima de la señal sinusoidal y la del ruido de cuantización en la señal $y_2(n)$.

$$SNR_{\max} = 10\log\left(\frac{P_s}{P_e}\right) = 10\log\left(\frac{3}{2}2^{2N}\right) + 10\log(OSR)$$

lo que equivale también a

$$SNR_{\max} = 6.02N + 1.76 + 10\log(OSR)$$

El primer término es el *SNR* debido al cuantizador de *N*-bits mientras el término en *OSR* es la mejora de *SNR* obtenida por el sobremuestreo. Aquí comprobamos que el sobremuestreo directo proporciona una mejora del *SNR* de 3dB/octava o, equivalentemente, 0.5 bits/octava. La razón para esta mejora del *SNR* a través del sobremuestreo es que cuando las muestras cuantizadas son promediadas, la parte correspondiente a la señal crece linealmente, mientras que la del ruido lo hace como la raíz cuadrada de la suma de cuadrados.

3.2.4 La ventaja de CA/D de 1-bit

Mientras que el sobremuestreo incrementa la relación señal-ruido, no lo hace con la *linealidad*. Algún tipo de *auto-calibración* debe ser empleada para obtener la linealidad requerida. La ventaja de un D/A de 1-bit es que es inherentemente lineal. Esta linealidad es el resultado de un convertidor D/A de 1-bit teniendo solo dos valores de salida, y como dos puntos definen una línea recta, no se requerirá calibración. Esta linealidad inherente es uno de los principales motivos por los que es empleado el sobremuestreo con convertidores de 1-bit. . De hecho, muchos convertidores de audio, en el presente, emplean convertidores de 1-bit para realizar convertidores lineales de 16 a 18 bits (con moldeado de ruido). Por último, mencionar que hay otras ventajas en el empleo de técnicas de sobremuestreo, tales como reducir los requerimientos para los filtros analógicos de anti-solapamiento y alisado.

3.3 Sobremuestreo con conformación de ruido

En esta sección discutiremos beneficios obtenidos por la conformación del ruido de cuantización a través de la realimentación. Aquí, comprobaremos un incremento mucho más significativo del rango dinámico cuando la señal de entrada es sobremuestreada en comparación con el sobremuestreo sin moldeado de ruido. Ilustraremos, de este modo los principios básicos de los convertidores $\Sigma\Delta$.

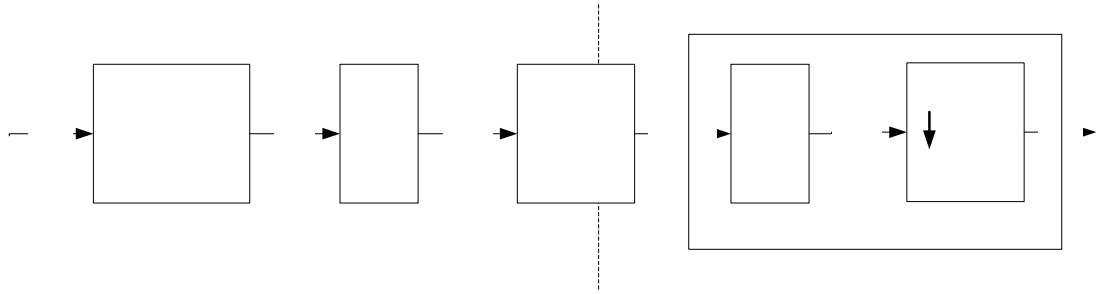


Figura 3-4 Diagrama de bloques de CA/D de sobremuestreo

La arquitectura de un sistema de modulación $\Sigma\Delta$ se muestra en la figura 3-4. La primera etapa consiste en un filtro *anti-aliasing* en tiempo continuo para limitar en

banda la señal de entrada a frecuencias inferiores a la mitad de la frecuencia de sobremuestreo, f_s. Cuando la tasa de sobremuestreo es grande, el filtro *anti-aliasing* puede ser bastante simple, tal como un *filtro RC paso-bajo*. A continuación del filtro *anti-aliasing*, la señal en tiempo continuo $x_c(t)$, es muestreada por un *sample&hold* (muestrea y mantiene). Esta señal es después procesada por el modulador ΣΔ, que convierte la señal analógica en una señal digital con conformación de ruido y baja resolución. El tercer bloque en el sistema es un *diezmador*. Éste convierte la señal digital sobremuestreada de baja resolución en una digital de alta resolución a una tasa de muestreo inferior que suele ser dos veces la frecuencia del ancho de banda deseado de la señal de entrada. El filtro de diezmado puede ser conceptualmente pensado como un filtro paso-baja formado por varias etapas. Es importante reseñar que en múltiples realizaciones, donde el modulador ΣΔ es implementado mediante capacidades conmutadas, el bloque independiente correspondiente al *Sample&Hold* no es necesario, ya que la señal en tiempo continuo es inherentemente muestreada por los interruptores y la entrada de las capacidades de los SC (switched-capacitors) ΣΔ.

3.3.1 Modulador sigma-delta

En la figura 3-5 se muestra un modulador ΣΔ general con moldeado de ruido y su modelo lineal.

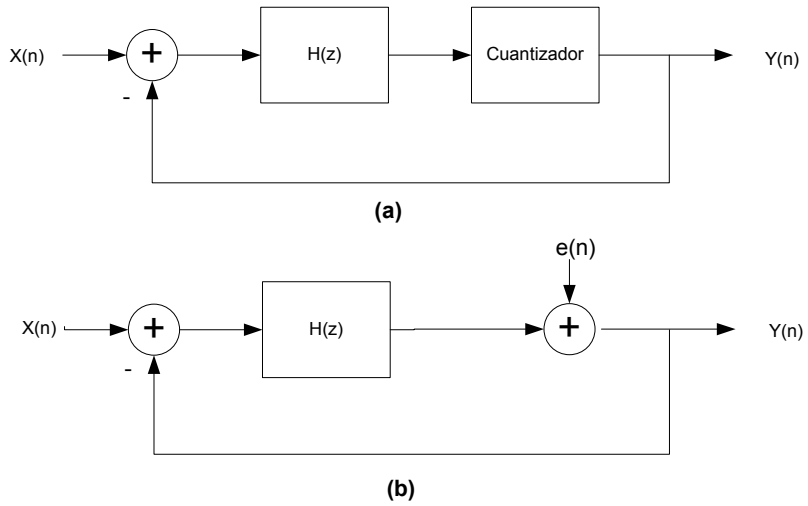


Figura 3-5 Un modulador y su modelo lineal: (a) un modulador sigma-delta general (estructura interpolativa); (b) modelo lineal del modulador mostrando la adición de ruido de cuantización.

Esta configuración es conocida como *estructura interpolativa* y es análoga a un amplificador realizado empleando un amplificador operacional y realimentación. En esta analogía, la realimentación reduce el efecto del ruido de la etapa de salida del amplificador operacional en la señal de salida del amplificador en el bucle cerrado a bajas frecuencias cuando la ganancia del amplificador es elevada. A altas frecuencias, cuando la ganancia del amplificador es baja, el ruido no es reducido. Observar que el cuantizador es mostrado aquí para el caso general en el que se pueden dar varios niveles

de salida. Mientras la mayoría de los convertidores de sobremuestreo actuales hacen uso de cuantizadores de 1-bit (sólo dos niveles de salida) debido a razones ya discutidas (apartado 3.2.4), realmente no hay motivo por el que restringirnos a tales implementaciones. De hecho, los convertidores multibit, están ganando popularidad.

Considerando el modelo mostrado en la figura 3-5-b con dos entradas independientes (lo que es una aproximación, ver 3.2.1 y 3.2.2), podemos tomar una función de transferencia de la señal, $S_{TF}(z)$, y una función de transferencia del ruido, $N_{TF}(z)$.

$$S_{TF}(z) \equiv \frac{Y(z)}{X(z)} = \frac{H(z)}{1+H(z)}$$

$$N_{TF}(z) \equiv \frac{Y(z)}{E(z)} = \frac{1}{1+H(z)}$$

Observar que los ceros de la función de transferencia del ruido, $N_{TF}(z)$, serán los mismos que los polos de $H(z)$. En otras palabras, cuando $H(z)$ sea infinito, vemos que $N_{TF}(z)$ es cero. Podemos escribir también la señal de salida como la combinación de la señal de entrada y la de ruido, siendo cada una filtrada por la correspondiente función de transferencia. Tenemos, en el dominio frecuencial:

$$Y(z) = S_{TF}(z)X(z) + N_{TF}(z)E(z)$$

Para conformar el ruido de cuantización de una forma eficaz, escogeremos $H(z)$ tal que su magnitud sea grande de 0 a f_0 (sobre la banda de frecuencia de interés). Con dicha elección, la función de transferencia de la señal, $S_{TF}(z)$, aproximará la unidad en la banda de interés de modo similar a un amplificador operacional en configuración de realimentación con ganancia unidad. Además, la función de transferencia del ruido, $N_{TF}(z)$, se aproximará a cero en la misma banda se frecuencias. Por lo tanto, el ruido de cuantización es reducido en la banda frecuencial de interés mientras que la señal no se ve afectada. El ruido de alta frecuencia no es reducido por la realimentación ya que hay una pequeña ganancia de bucle a altas frecuencias. Sin embargo, un filtrado posterior puede eliminar el ruido de cuantización fuera de banda con mínimos efectos sobre la señal deseada.

Antes de escoger funciones específicas para $H(z)$, reseñar que el nivel máximo de la señal de entrada en banda, $x(n)$, debe permanecer dentro de los niveles máximos de la señal de realimentación, $y(n)$; en caso contrario, la gran ganancia de $H(z)$ provocará que la señal de entrada al cuantizador sature. De hecho, muchos moduladores requieren que la señal de entrada sea significativamente más pequeña que los niveles extremos de salida del cuantizador para mantener el modulador estable, Sin embargo, el nivel máximo de la señal de entrada, $x(n)$, para frecuencias donde la ganancia de $H(z)$ es pequeña, no provocará necesariamente que la señal de entrada al cuantizador sature.

3.3.2 Modulador Sigma-Delta de primer orden

Para realizar un moldeado de ruido de primer orden, la función de transferencia del ruido, $N_{TF}(z)$, debería tener un cero en continua (z=1) por lo que el ruido de

cuantización es filtrado paso-alta. Dado que los ceros de $N_{TF}(z)$ son iguales a los polos de *H(z)*, podemos obtener un moldeado de ruido de primer orden tomando *H(z)* como un integrador en tiempo discreto (polo en z=1).

$$H(z) = \frac{1}{z-1}$$

El diagrama de bloques se muestra en la figura 3-6:

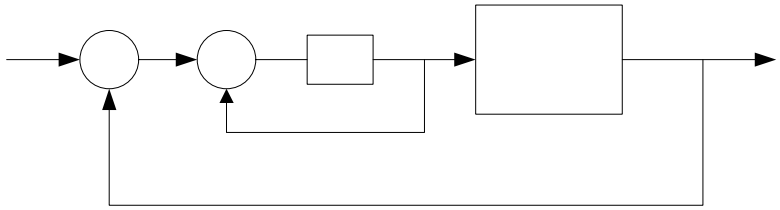


Figura 3-6 Modulador sigma-delta de primer orden

3.3.2.1 Estudio en dominio temporal

Desde el punto de vista del dominio temporal, si la realimentación opera correctamente y el sistema es estable, entonces la señal de entrada al cuantizador está acotada ($\neq\infty$). Como el integrador tiene ganancia infinita en *dc,* el promedio del valor a la entrada del integrador en tiempo discreto debe ser exactamente cero (media de *u(n)-y(n)* igual a cero). Este resultado implica que el valor medio (valor en *dc*) de *u(n)* debe ser igual al valor medio (valor en *dc*) de *y(n)*.

3.3.2.2 Función de transferencia

El primer paso en el diseño de los convertidores es la obtención de las ecuaciones teóricas de los mismos. Para ello se aplica la *transformada Z*, en la que los retrasos son fácilmente tratables. En cuanto a los cuantizadores, se considera el modelo ya visto en el que a la señal de entrada se le añade el ruido de cuantización. Un detalle importante a señalar en este punto es el tratamiento de los cuantizadores de 1-bit. Para fijar la pendiente de estos cuantizadores (indefinida en principio) se considera un bloque de ganancia delante de ellos. El valor de esta ganancia se toma como el inverso de la ganancia del bucle externo de la etapa completa. Consideraremos aquí las ganancias de los amplificadores para cada entrada.

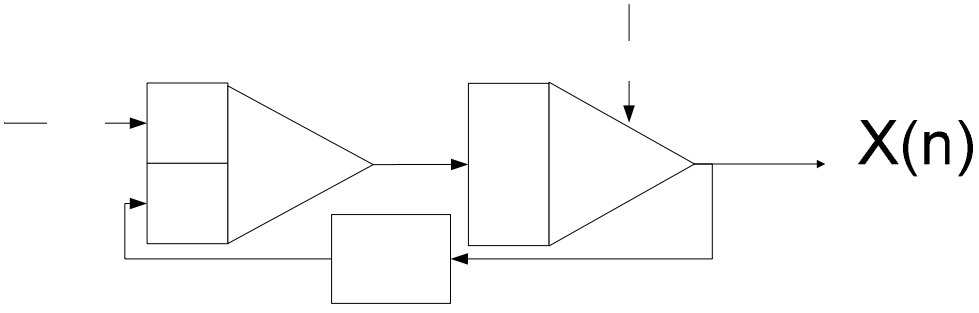


Figura 3-7 Modulador sigma-delta de primer orden

Desde el punto de vista del dominio frecuencial, la función de transferencia de la señal, $S_{TF}(z)$, viene dada por

$$S_{TF}(z) = \frac{Y(z)}{X(z)} = \frac{\frac{g_1}{g_1'}H(z)}{1+H(z)} = \frac{\frac{g_1}{g_1'}\,1/(z-1)}{1+1/(z-1)} = \frac{g_1}{g_1'}z^{-1}$$

de donde hemos obtenido la condición de realización

$$\frac{g_1}{g_1'} = 1 \Rightarrow g_1' = g_1$$

tal que resulte

$$S_{TF}(z) = \frac{Y(z)}{X(z)} = z^{-1}$$

y la función de transferencia del ruido, $N_{TF}(z)$

$$N_{TF}(z) = \frac{Y(z)}{E(z)} = \frac{1}{1+1/(z-1)} = (1-z^{-1})$$

Comprobamos que la función de transferencia de la señal es un simple retraso, mientras que la función de transferencia del ruido es un diferenciador en tiempo discreto (filtro paso-alto).

3.3.2.3 Relación señal ruido (SNR)

Para calcular la magnitud de la función de transferencia del ruido, $|N_{TF}(f)|$, sustituimos $z = e^{jwT} = e^{j2\pi f/f_s}$

$$N_{TF} = 1 - e^{-j2\pi f/f_s} = \frac{e^{j\pi f/f_s} - e^{-j\pi f/f_s}}{2j} \times 2j \times e^{-j\pi f/f_s} = \sin\left(\frac{\pi f}{f_s}\right) \times 2j \times e^{-j\pi f/f_s}$$

Tomando la magnitud en ambos lados de la igualdad, tenemos una función paso-alta

$$|N_{TF}(f)| = 2\sin\left(\frac{\pi f}{f_s}\right)$$

La potencia del ruido de cuantización sobre la banda de frecuencias de *0* a f_0 viene dada por

$$P_e = \int_{-f_0}^{f_0} S_e^2(f)|N_{TF}(f)|^2\,df = \int_{-f_0}^{f_0}\left(\frac{\Delta^2}{12}\right)\frac{1}{f_s}\left[2\sin\left(\frac{\pi f}{f_s}\right)\right]^2 df$$

y, haciendo la aproximación $f_0 << f_s$ ($OSR >> 1$), podemos aproximar $sin((\pi f)/f_s)$ a $(\pi f)/f_s$, resultando

$$P_e \cong \left(\frac{\Delta^2}{12}\right)\left(\frac{\pi^2}{3}\right)\left(\frac{2f_0}{f_s}\right)^3 = \frac{\Delta^2\pi^2}{36}\left(\frac{1}{OSR}\right)^3$$

Asumiendo que el máximo de la potencia de la señal es el obtenido anteriormente, el SNR máximo para este caso resulta

$$SNR_{\max} = 10\log\left(\frac{P_s}{P_e}\right) = 10\log\left(\frac{3}{2}2^{2N}\right) + 10\log\left[\frac{3}{\pi^2}(OSR)^3\right]$$

o, equivalentemente

$$SNR_{\max} = 6.02N + 1.76 - 5.17 + 30\log(OSR)$$

Observamos que duplicando el *OSR* se produce un incremento de *SNR* para el modulador de primer orden de 9dB o, equivalentemente, una ganancia de 1.5 bits/octava. Podemos comparar este resultado con el de 0.5 bits/octava obtenido para el sobremuestreo sin conformación de ruido (Ver 3.2.3).

3.3.3 Modulador Sigma-Delta de segundo orden

3.3.3.1 Función de transferencia

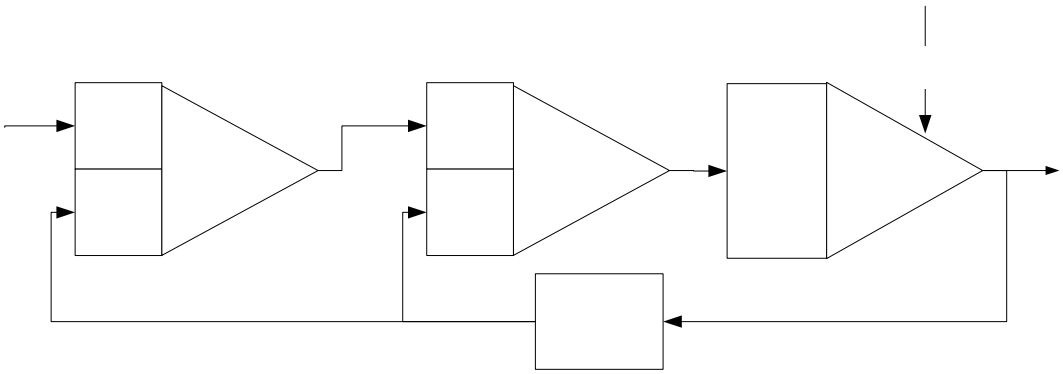


Figura 3-8 Modulador Sigma-delta de segundo orden

Tal y como describimos para el modulador de primer orden, también aquí, para compensar problemas de la pendiente del cuantizador de un bit, al modelo del cuantizador le añadimos una ganancia que se corresponde con la del bucle de realimentación más externo; $1/(g_1'g_2)$, en este caso.

La salida, $Y(z)$, será función de la entrada, $X(z)$, y el ruido de cuantización, $E(z)$,

$$Y(z) = N_{TF}(z)E(z) + S_{TF}(z)X(z)$$

Desde el punto de vista del dominio frecuencial, la función de transferencia de la señal, $S_{TF}(z)$, viene dada por

$$S_{TF}(z)=\frac{Y(z)}{X(z)}=\frac{\frac{g_1}{g_1'}H(z)^2}{1+\frac{g_2'}{g_1g_2}H(z)+H(z)^2}=\frac{\frac{g_1}{g_1'}z^{-2}}{1+\left(\frac{g_2'}{g_1g_2}-2\right)z^{-1}+\left(\frac{g_2'}{g_1g_2}-2\right)z^{-2}}$$

y la función de transferencia del ruido, $N_{TF}(z)$

$$N_{TF}(z)=\frac{Y(z)}{E(z)}=\frac{1}{1+\frac{g_2'}{g_1g_2}H(z)+H(z)^2}=\frac{(1-z^{-1})^2}{1+\left(\frac{g_2'}{g_1g_2}-2\right)z^{-1}+\left(\frac{g_2'}{g_1g_2}-2\right)z^{-2}}$$

En este caso resultan las siguientes condiciones de realización

$$\frac{g_2'}{g_1g_2}-2=0\Rightarrow g_2'=2g_1g_2$$

$$\frac{g_1}{g_1'}=1\Rightarrow g_1'=g_1$$

Necesarias para eliminar términos del denominador tal que las correspondientes funciones de transferencia queden

$$S_{TF}(z)=\frac{Y(z)}{X(z)}=z^{-2}$$

$$N_{TF}(z)=\frac{Y(z)}{E(z)}=(1-z^{-1})^2$$

Así la salida $Y(z)$ será

$$Y(z)=(1-z^{-1})^2E(z)+z^{-2}X(z)$$

3.3.3.2 Relación señal ruido (SNR)

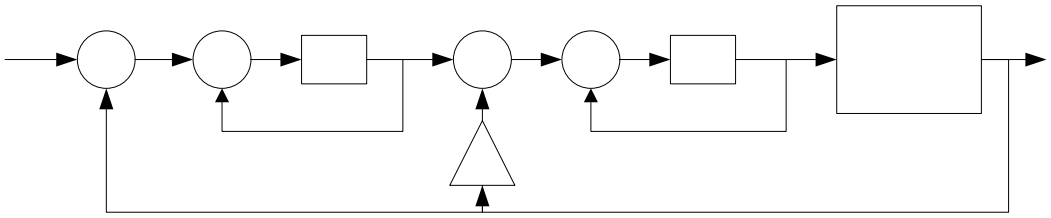


Figura 3-9 Modulador sigma-delta de segundo orden

El modulador mostrado en la figura 3-9 realiza un moldeado de ruido de segundo orden (la función de transferencia del ruido, $N_{TF}(z)$, es una función paso-alta de segundo orden). Nótese que hemos impuesto $g_2'=2g_1g_2=2$ como consecuencia de lo deducido en el apartado anterior.

Para este modulador la función de transferencia de la señal viene dada por

$$S_{TF}(f) = z^{-2}$$

y la función de transferencia del ruido

$$N_{TF}(f) = (1 - z^{-1})^2$$

cuya magnitud resulta

$$\left|N_{TF}(f)\right| = \left[2\sin\left(\frac{\pi f}{f_s}\right)\right]^2$$

resultando una potencia del ruido de cuantización en la banda de interés tal que

$$P_e \cong \frac{\Delta^2 \pi^4}{60}\left(\frac{1}{OSR}\right)^5$$

De nuevo, asumiendo que el máximo de potencia de la señal es el obtenido con anterioridad, el SNR máximo para este caso será

$$SNR_{\max} = 10\log\left(\frac{P_s}{P_e}\right) = 10\log\left(\frac{3}{2}2^{2N}\right) + 10\log\left[\frac{5}{\pi^4}(OSR)^5\right]$$

o, equivalentemente

$$SNR_{\max} = 6.02N + 1.76 - 12.9 + 50\log(OSR)$$

Podemos comprobar que doblando el OSR mejora el SNR para un modulador de segundo orden en 15dB o, una ganancia de 2.5 bits/octava.

3.3.4 Modulador Sigma-Delta de orden L

En el caso general, cuando hay L integradores, es decir, para moduladores de orden L, en caso de que el sistema sea estable, puede demostrarse que la potencia del ruido de cuantización en la banda de la señal viene dada aproximadamente por

$$P_e = \frac{\Delta^2}{12}\frac{\pi^{2L}}{2L+1}\left(\frac{2f_0}{f_s}\right)^{2L+1}$$

y la función de transferencia del ruido vendrá dada por

$$N_{TF}(z) = (1 - z^{-1})^L$$

Así pues, al aumentar el orden, L, del sistema, se incrementa la atenuación del ruido en las frecuencias más bajas (banda de la señal), lo que contribuye de forma positiva en la relación señal a ruido del modulador, y, por el contrario, en las frecuencias altas el ruido es amplificado.

Además, cada vez que se duplica el factor de sobremuestreo, la potencia de ruido se divide por el factor

$$\frac{P_e\big|_L^1}{P_e\big|_L^2} = \frac{\frac{\Delta^2}{12}\frac{\pi^{2L}}{2L+1}\left(\frac{2f_0}{f_s}\right)^{2L+1}}{\frac{\Delta^2}{12}\frac{\pi^{2L}}{2L+1}\left(\frac{2f_0}{2f_s}\right)^{2L+1}} = 2^{2L+1}$$

donde

$P_e\big|_L^1$: Potencia del ruido para *OSR*

$P_e\big|_L^2$: Potencia del ruido para (2·*OSR*)

Este decremento en la potencia de ruido también provoca una mejora de la relación señal a ruido del sistema, en concreto:

$$SNR\big|_2(dB) = SNR\big|_1(dB) + \Delta SNR(dB)$$

$$\Delta SNR(dB) = 10\log(2^{2L+1}) = 3(2L+1)dB$$

donde

$SNR\big|_1$: *SNR* para *OSR*

$SNR\big|_2$: *SNR* para (2·*OSR*)

Considerando la relación existente entre el número de bits del cuantizador, *N*, y la *SNR* proporcionada por el sistema (*SNR*(dB)~6·*N*), el incremento de 3·(2L + 1) dB en la relación señal a ruido del modulador , al duplicar la tasa de muestreo, equivale a obtener una precisión extra de (L + 0.5) bits.

No obstante, el hecho de que la *SNR* del modulador ΣΔ aumente con el orden del sistema, no implica que sea preferible la utilización ΣΔ de un de mayor orden en una determinada aplicación, puesto que existen dificultades para la implementación de sistemas de alto orden (L = 3) estables.

En los sistemas ΣΔ de alto orden es necesario usar limitadores de amplitud para evitar problemas de inestabilidad. Los moduladores de alto orden sólo pueden ser condicionalmente estables, por lo que presentan una elevada sensibilidad a los parámetros del circuito.

Por este motivo nos centraremos en moduladores de primer y segundo orden (cuya estabilidad podremos garantizar) pero distribuidos en configuración cascada como veremos a continuación.

3.3.5 Modulador Sigma-Delta en cascada

Esta arquitectura también es conocida en la literatura como *multietapa* o *MASH*. En la figura podemos apreciar su estructura que atendiendo a la nomenclatura habitual de esta arquitectura denominaríamos *"modulador ΣΔ en cascada L_1-L_2...-L_N"* y que se basa en la conexión en cascada de moduladores de bajo orden (orden 1 ó 2 típicamente) donde podemos asegurar la estabilidad. La lógica digital será la encargada de eliminar el

ruido de cuantización resultante de cada etapa. El resultado final es que se obtiene la señal de entrada al modulador junto con el ruido de cuantización de la última etapa, el cual es conformado por una función que tiene igual orden al número de integradores de las cascada, por lo que será más eficiente esta conformación que si solo tuviésemos una etapa.

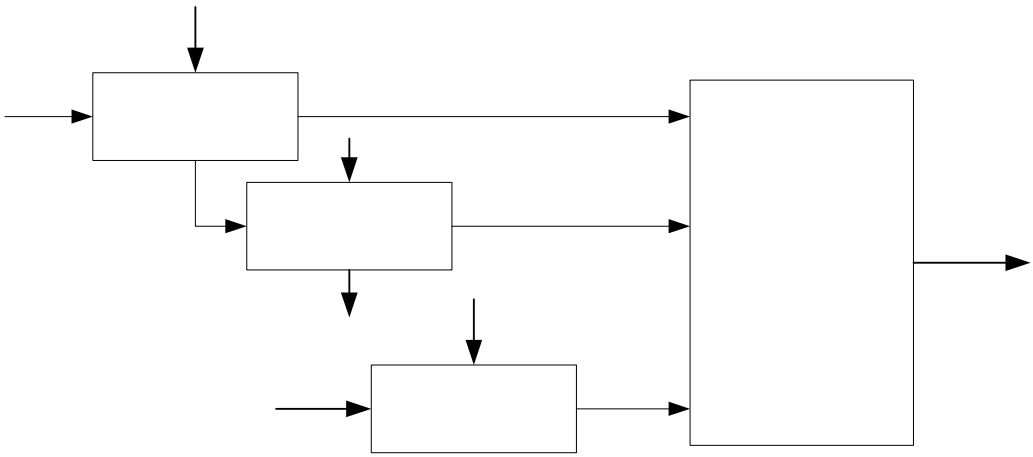


Figura 3-10 Diagrama de bloques de modulador Sigma-Delta en cascada L_1-L_2-…-L_N

Considerando la implementación con un cuantizador de un solo bit, las principales ventajas/desventajas de esta arquitectura son:

- Ventajas:
  - Se obtienen *SNR* altos para factores de sobremuestreo (*OSR*) de bajo valor.
  - Estabilidad garantizada (pues se basan en arquitecturas de lazo simple de bajo orden, y por tanto, estables).
  - Utilización del máximo rango de la señal de entrada.
- Desventajas:
  - Alta sensibilidad a ciertas no linealidades circuitales.

Una ultima característica a destacar de esta arquitectura es que, si bien se complica la parte digital del CA/D al incluir una lógica de cancelación, esto favorece la integración en tecnología VLSI (Very Large Scale Integration) estándar y se beneficia de este modo del escalado tecnológico.

3.3.5.1 Modulador Sigma-Delta en cascada 2-2

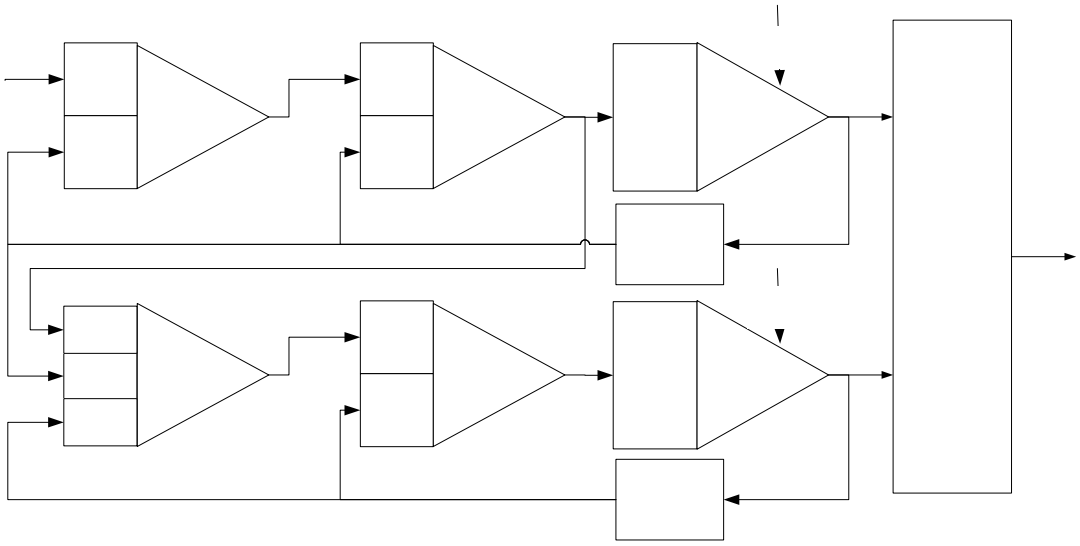


Figura 3-11 Esquema de un modulador sigma-delta cascada 2-2 con cuantizadores de 1-bit

Del modulador de segundo orden sabemos

$$Y_1(z) = (1 - z^{-1})^2 E_1(z) + z^{-2} X(z)$$

para lo cual vimos que debe cumplirse

$$\frac{g_2^{'}}{g_1 g_2} - 2 = 0 \Rightarrow g_2^{'} = 2 g_1 g_2$$

$$\frac{g_1}{g_1^{'}} = 1 \Rightarrow g_1' = g_1$$

De acuerdo al esquema de la figura, y sin más que operar y añadiendo una nueva condición de realización durante el desarrollo, $g_4^{'} = 2 g_3^{"} g_4$, para eliminar términos en "z" del denominador, se obtiene

$$Y_2(z) = \left(\frac{g_3 g_2 g_1 - g_3^{'}}{g_3^{"}} \right) z^{-4} X(z) + \left[\left(\frac{g_3 g_2 g_1 - g_3^{'}}{g_3^{"}} \right) z^{-2} (1 - z^{-1})^2 - \left(\frac{g_3 g_2 g_1}{g_3^{"}} \right) z^{-2} \right] E_1(z) + (1 - z^{-1}) E_2(z)$$

Ahora, mediante el bloque digital de cancelación, y tomando como entrada las salidas del modulador $Y_1(z)$, $Y_2(z)$ hemos de eliminar el ruido de cuantización correspondiente a la primera etapa de la cascada del modulador, $E_1(z)$.

Para ello tomamos la siguiente configuración para el bloque digital de cancelación

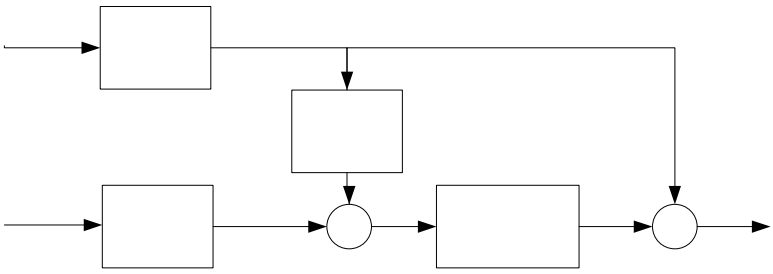


Figura 3-12 Bloque digital de cancelación para modulador sigma-delta cascada 2-2

Donde tenemos que

$$d_0 = \frac{g_3^{'}}{g_1 g_2 g_3} - 1$$

$$d_1 = \frac{g_3^{''}}{g_1 g_2 g_3}$$

y la salida Y(z) resulta

$$Y(z) = z^{-4} X(z) + \frac{g_3^{''}}{g_1 g_2 g_3}(1 - z^{-1})^4 E_2$$

Observamos que la señal de entrada no tiene ninguna constante multiplicativa y que hemos conseguido anular el ruido de cuantización del primer cuantizador, E_1.

Por último, es importante reseñar que hemos considerado en este caso que todos los cuantizadores de la cascada son de 1-bit. Es habitual, no obstante, tomar el último de los cuantizadores de más de un bit para afinar la digitalización del ruido de la etapa anterior. En este caso no contaríamos con el bloque multiplicativo $1/(g_3'' g_4)$ a la entrada del segundo cuantizador. Las condiciones de realización resultan, de este modo más restrictivas

$$g_1 = g_1^{'}$$
$$g_2^{'} = 2 g_1 g_2$$
$$g_4^{'} = 2$$
$$g_4 = \frac{1}{g_3^{''}}$$

siendo la salida $Y_2(z)$

$$Y_2(z) = \left(g_4 g_3 g_2 g_1 - g_4 g_3^{'}\right) z^{-4} X(z) + \left[\left(g_4 g_3 g_2 g_1 - g_4 g_3^{'}\right) z^{-2} (1 - z^{-1})^2 - \left(g_4 g_3 g_2 g_1\right) z^{-2}\right] E_1(z) + (1 - z^{-1}) E_2(z)$$

Por lo que respecta a la bloque digital de cancelación, éste sería de la misma estructura que el implementado anteriormente y sólo habríamos de ajustar los coeficiente d_0 y d_1 para eliminar el ruido de cuantización de la primera etapa.

3.3.5.2 Modulador Sigma-Delta en cascada 2-1-1

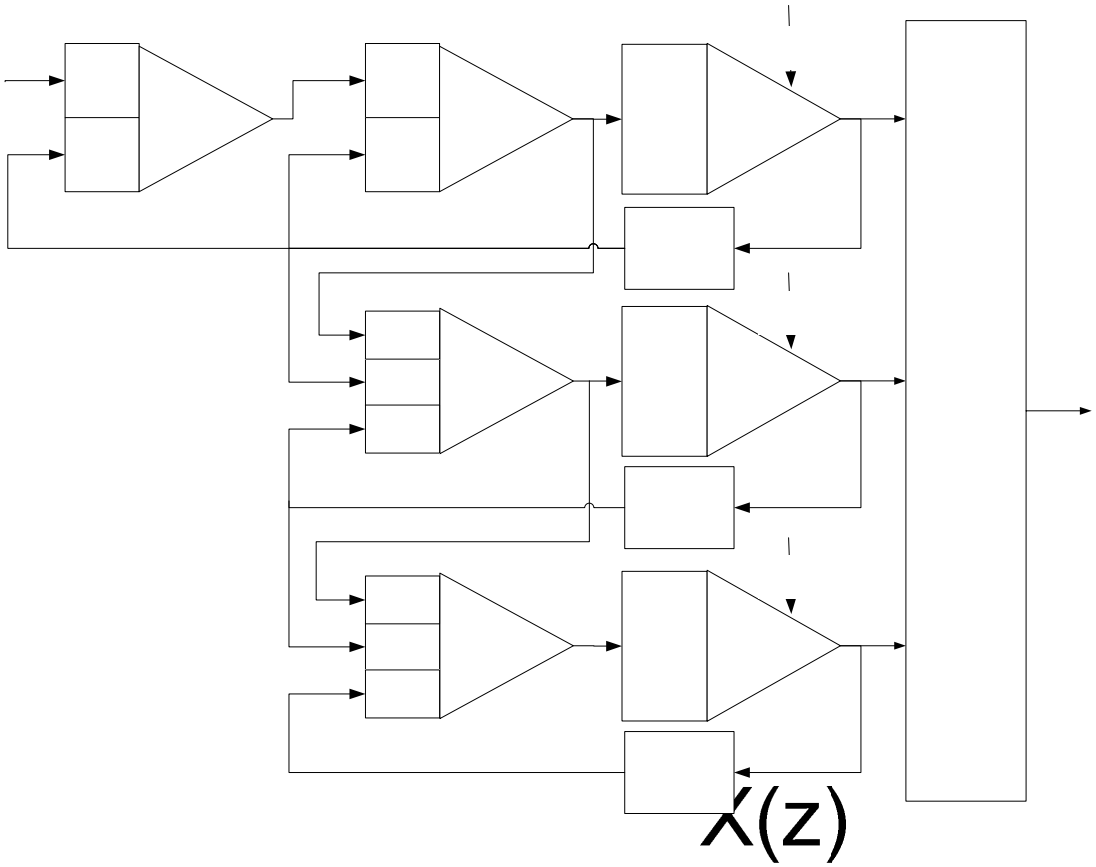


Figura 3-13 Modulador sigma-delta 2-1-1 con último cuantizador genérico

En la figura 3-13 se muestra el esquema de un modulador sigma-delta en cascada 2-2-2 con el último cuantizador genérico.

Mediante un desarrollo análogo al de los moduladores vistos anteriormente (orden 1, 2 y 2-2, ver apartados 3.3.2, 3.3.3 y 3.3.5.1 respectivamente) llegamos a las siguientes expresiones para las salidas:

$$Y_1(z) = (1-z^{-1})^2 E_1(z) + z^{-2} X(z)$$

$$Y_2(z) = (1-z^{-1})\ E_2(z) + \left[\frac{g_3 g_2 g_1 - g_3^{'}}{g_3^{''}} z^{-1}(1-z^{-1})^2 - \frac{g_3 g_2 g_1}{g_3^{''}} z^{-1} \right] E_1(z) + \frac{g_3 g_2 g_1 - g_3^{'}}{g_3^{''}} z^{-3} X(z)$$

$$Y_3(z) = (1-z^{-1})\ E_3(z) - g_4 g_3^{''} z^{-1} E_2(z)$$

donde para evitar la aparición de elementos en el denominador, se han hecho las siguientes suposiciones:

$$g_1 = g_1^{'}$$
$$g_2^{'} = 2g_1g_2$$
$$g_4^{''} = 1$$
$$g_4^{'} = g_4g_3^{''}$$

Obsérvese que la salida $Y_3(z)$ no depende ni de la entrada $X(z)$ ni del ruido de cuantización de la primera etapa $E_1(z)$.

Vemos ahora cual será la estructura del bloque de cancelación de ruido que tendrá como entrada Y_1, Y_2, Y_3:

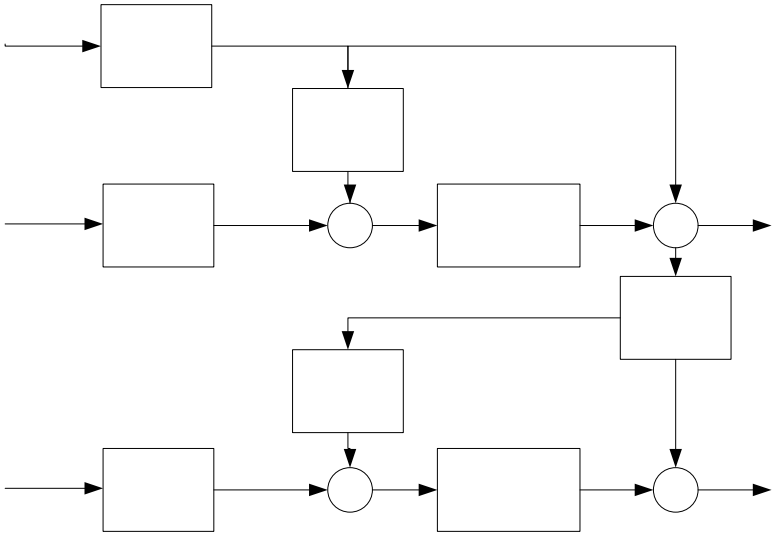


Figura 3-14 Bloque de cancelación de ruido para modulador sigma-delta 2-1-1

Donde los coeficientes son:

$$d_0 = \frac{g_3^{'}}{g_1g_2g_3} - 1$$

$$d_1 = \frac{g_3^{''}}{g_1g_2g_3}$$

$$d_2 = 0$$

$$d_3 = \frac{g_4^{''}}{g_1g_2g_3g_4}$$

y la salida total del modulador $Y_{211}(z)$ resulta

$$Y_{211}(z) = z^{-4}X(z) + \frac{g_4^{''}}{g_1g_2g_3g_4}(1-z^{-1})^4E_3$$

Nótese que este ejemplo contempla y también el modulador 2-1 ya que podemos tomar la salida $Y_{21}(z)$ de la figura 3-14 que resulta:

$Y_1(z)$

$$Y_{21}(z) = z^{-3}X(z) + \frac{g_3^{''}}{g_1 g_2 g_3}(1-z^{-1})^3 E_2$$

3.3.5.3 Modulador Sigma-Delta en cascada orden L

De forma análoga a la desarrollada en los casos anteriores, una vez que se tiene la salida de cada una de las etapas, se calculan los coeficientes y los bloques que hay que colocar en la parte digital para cancelar los errores de cuantización de las etapas intermedias. La salida total debe tener la forma:

$$Y(z) = z^{-L}X(z) + K(1-z^{-1})^{-L}E_N(z)$$

Donde *L* es el orden total del convertidor y *N* el número de etapas. En el caso de la arquitectura 2-2 , *L* vale 4 y *N* vale 2, mientras que para la 2-1-1, *L* vale 4 y *N* vale 3. La constante *K* aparece tras la cancelación del ruido y, tal y como se desprende de los ejemplos anteriores, está relacionada con la inversa de los coeficientes de cada etapa. Si su producto es un valor pequeño, su inverso será muy grande. Esto hace que el ruido se vea magnificado por una constante mayor que la unidad. Por tanto hay una pérdida en la *SNR* con respecto al caso ideal.

3.3.5.4 Selección de coeficientes

Aunque el proyecto en cuestión no contempla el estudio en sí de los moduladores sigma-delta sino el proporcionar una herramienta de simulación para ello, comentaremos aquí algunos aspectos relativos a dicho estudio dado que es interesante tenerlos en mente cara a que las herramientas proporcionadas resulten lo más útiles posible.

Para la elección de los valores de los coeficientes a la entrada de los integradores, se han de tener en cuenta las siguientes condiciones:

- Deben cumplirse las condiciones impuestas por las ecuaciones.
- Deben adaptar la señal a la entrada de los integradores para que a la salida de los mismos no sobrepase el rango impuesto por la tensión de alimentación.
- Los valores han de ser lo mayor posible para que la inversa del producto de los coeficientes directos sea lo menor posible.
- En lo posible, elegirlos de modo que los coeficientes digitales (función de lo valores analógicos) queden potencia de dos. Esto simplifica enormemente las operaciones digitales ya que no harían falta multiplicadores, sino simplemente registros de desplazamiento.
- Los valores han de ser sintetizables en la tecnología de realización escogida.

Dichas condiciones entran en conflicto unas con otras, ya que el adaptar la señal de entrada exige valores pequeños de los coeficientes (especialmente en las etapas de segundo orden), pero por otro lado, queremos los valores tan grandes como sea posible. Además las relaciones entre los coeficientes impuestos por las ecuaciones nos impiden cambiarlos de forma independiente.

3.3.5.5 Sensibilidad de parámetros

Vemos otro aspecto importante relativo al estudio de los moduladores sigma-delta.

La sensibilidad de cada parámetro viene determinada por su influencia en la cancelación de ruido de cuantización de las distintas etapas. Lógicamente, el ruido más importante en el sistema es el de cuantización de la primera etapa. Por tanto, las variables que estén relacionadas de una u otra forma con este ruido serán las más sensibles.

En general, el ruido de cuantización puede aparecer en la salida por varios motivos:

- Errores en la cancelación de ruido en la parte digital: Los parámetros digitales se determinan a partir de los valores de los parámetros analógicos. Una variación en cualquiera de ellos hace que la cancelación no sea perfecta.
- Comportamiento indeseado de las etapas: Una variación en los parámetros que aparecen en las ecuaciones de comportamiento de las etapas hace que dichas condiciones no se cumplan. Aparecen denominadores en las funciones de transferencia de señal y ruido que no se espera en la parte digital (cuyos bloques están calculados sobre el comportamiento correcto de todas las etapas). Por tanto, la cancelación tampoco será exacta y parte del ruido de cuantización pasará a la salida.
- Paso de la señal de entrada a la tercera etapa: En el ejemplo correspondiente al modulador 2-1-1 (apartado 3.3.5.2) los coeficientes están escogidos de forma que la señal de entrada no entra en la tercera etapa. Esto simplifica enormemente la parte digital de cancelación de ruido

3.3.6 No linealidades

Comentamos a continuación algunas de las no linealidades presentes en los moduladores sigma-delta.

3.3.6.1 Variaciones en los coeficientes de los integradores

Los coeficientes de ganancia de los integradores modelan diversos aspectos relativos a la implementación circuital del modulador por lo que no tendremos un control exacto sobre el valor de los mismos. Esta no linealidad corresponde a la variación de dichos coeficientes respecto a los valores deseados por desapareamiento de elementos que dependerán de la tecnología escogida.

3.3.6.2 Pérdidas en los integradores

La ganancia real en continua de los amplificadores no es infinita y el modulador se verá afectado por este hecho.

Para el caso ideal, ganancia infinita, teníamos:

$$H(z) = \frac{z^{-1}}{1 - z^{-1}}$$

Que se corresponde con la siguiente estructura:

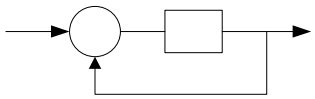


Figura 3-15 H(z), modelo ideal

Sin embargo, en la práctica la ganancia del amplificador limita la ganancia en continua del integrador. Esto produce pérdidas en los integradores. La función de transferencia del integrador, teniendo en cuenta este efecto, puede modelarse como:

$$H(z) = \frac{z^{-1}}{1 - (1 - \varepsilon)z^{-1}}$$

que se corresponde con el siguiente diagrama de bloques:

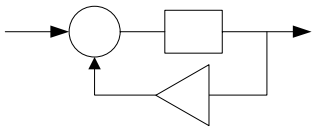


Figura 3-16 H(z), modelo con pérdidas

Observamos como se desplaza el polo de $z = -1$.

3.3.6.3 Jitter

Una posible fuente de aumento de potencia del ruido de cuantización es la resultante de que se produzca un muestreo no uniforme de la señal de entrada debido al efecto jitter sobre el reloj.

Este efecto se da cuando las muestras de la señal de entrada no son tomadas a intervalos equiespaciados de tiempo ($T_s \neq cte$, ver apartado 2.2.2).

Si no muestreamos a intervalos regulares de tiempo, la relación señal ruido a la salida se verá afectada.

Capítulo 4
Descripción general. Manual de usuario

4.1 Introducción

En este capítulo realizaremos una descripción general del programa de simulación de moduladores sigma-delta en cascada. Atenderemos aspectos como son la instalación, ejecución e interfaz de usuario y su correspondiente manejo. Obviaremos en todo momento la implementación real que subyace a las mismas (código MATLAB, funciones de programación, variables…) ya que ésta será tratada en detalle en capítulos posteriores.

La interfaz de usuario se ha orientado de modo que aspectos complejos de programación y simulación sean transparentes al mismo tal que procesos de distinta implementación de código se le muestran al usuario de modo uniforme y similar. Asimismo, se ha hecho especial hincapié en acentuar la sencillez de manejo del simulador. Para ello se han agrupado los controles con funciones relacionadas de forma intuitiva y ordenada.

El esquema conceptual general del simulador es

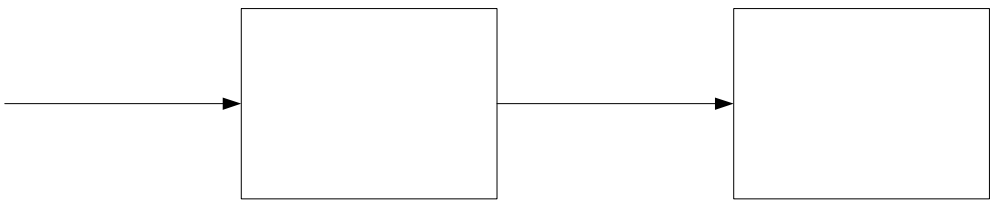


Figura 4-1 Esquema conceptual de simulación

Deberemos, pues, configurar la que será nuestra señal de entrada, aquella que será objeto de la modulación sigma-delta, establecer sus características y obtener asimismo información relativa a la misma.

Por otro lado, habremos también de fijar la arquitectura, parámetros y particularidades del modulador en cuestión.

Y finalmente, para la entrada y modulador establecidos con anterioridad, obtener una determinada salida sobre la que podremos realizar el procesamiento digital de señal que estimemos oportuno. Incluiremos aquí el estudio de dicha salida y la caracterización del proceso completo mediante gráficas y figuras de mérito.

De este modo, parece lógico organizar la interfaz de usuario en tres bloques diferenciados (que se corresponderán con tres *ventanas* de MATLAB) de acuerdo a los señalados en la figura 4-2.

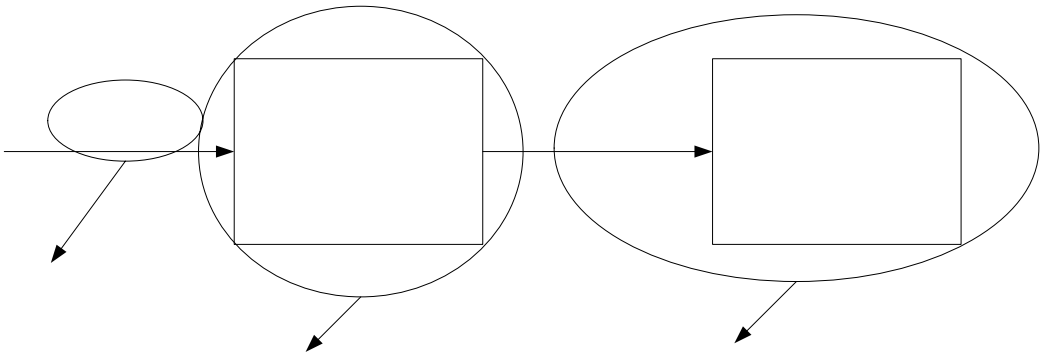


Figura 4-2 Esquema conceptual y bloques de interfaz

Como se desprende de la figura 4-2, la interfaz gráfica del programa estará dividida en los tres bloques siguientes:

- Generar señal de entrada
  - Datos de la entrada
  - Representación gráfica (dominio temporal y frecuencial)
  - Obtención de la frecuencia de muestreo
- Configurar modulador
  - Configurar arquitectura.
  - Fijar coeficientes de ganancia de los integradores
  - Configurar no linealidades
- Salida
  - Simulación

- Representación gráfica
- Figuras de mérito
- Enventanado

Describiremos en los apartados siguientes cada uno de estos bloques en detalle; sus entradas y salidas y su modo de operación.

4.2 Instalación y ejecución

El simulador se desarrolla bajo entorno MATLAB por lo que no requiere una instalación propiamente dicha. El conjunto de funciones necesarias para su ejecución se encuentran recogidas en una carpeta denominada *sd_cascada*.

Es necesario, pues, iniciar el programa MATLAB, versión 6.5 o superior (*release 13*), entorno bajo el cual ha sido desarrollado el proyecto, y fijar como espacio de trabajo la ubicación de la carpeta *sd_cascada*. Posteriormente, sólo resta ejecutar desde la línea de comandos de MATLAB *sd_cascada.* Acto seguido se abren tres ventanas: *entrada*, *simulador* y *salida*, que se corresponden con los bloques conceptuales descritos anteriormente (Ver figura 4-3).

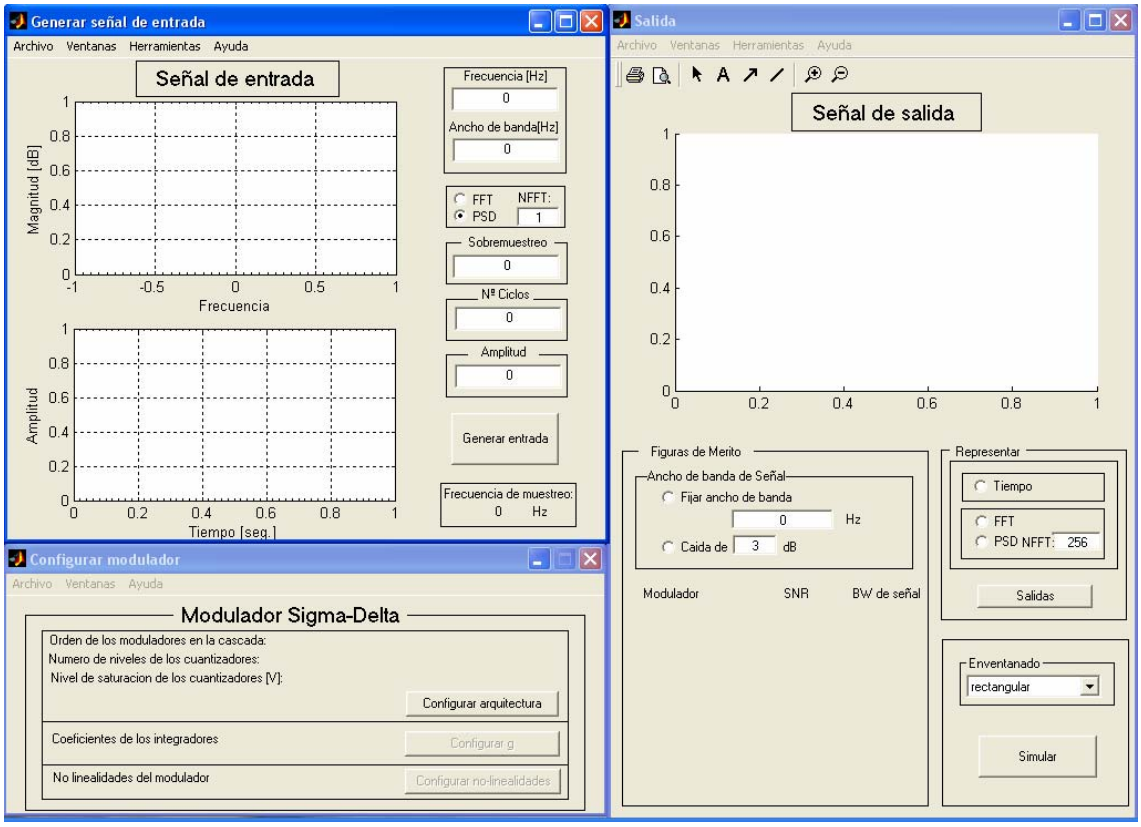


Figura 4-3 Disposición de ventanas al comienzo de la ejecución del programa

4.3 Generar señal de entrada

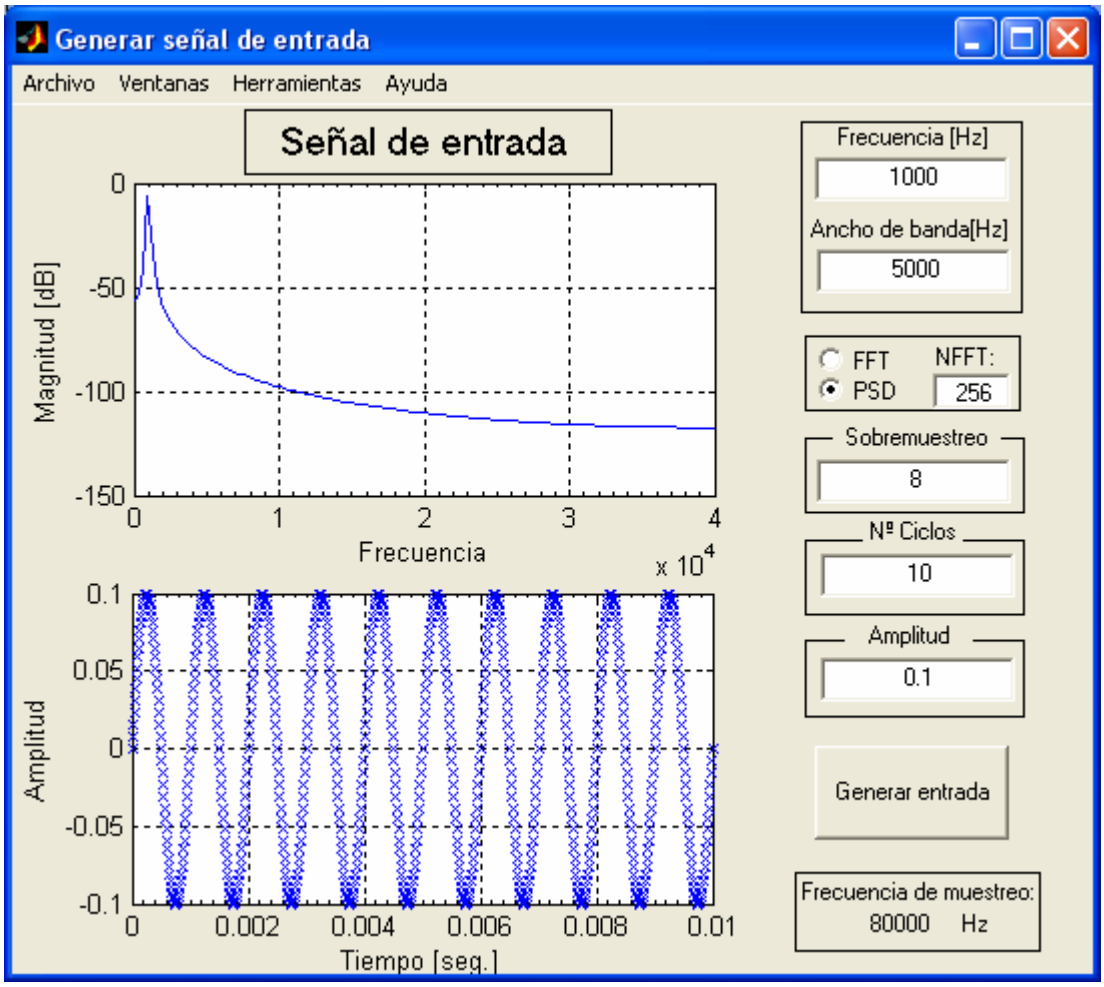


Figura 4-4 Interfaz para generar señal de entrada

En esta ventana (figura 4-4) estableceremos la que será nuestra señal sinusoidal muestreada de entrada al modulador.

Encontramos dos ejes para la representación gráfica de la señál; temporal (abajo) y frecuencial (arriba). A la derecha se sitúan los controles mediante los que fijaremos las características de la entrada: "Frecuencia de la señal", "ancho de banda", "tasa de sobremuestreo", "número de ciclos" del tono y "amplitud" del mismo.

Mediante el botón *"generar entrada"* confirmamos la captura de los datos introducidos y la señal pasará a ser representada. Asimismo, en la esquina inferior derecha se nos informa de la frecuencia de muestreo de la entrada.

Disponemos finalmente de una barra de menú en la parte superior de la ventana. Esta barra es descrita en el apartado 4.6.

4.4 Configurar el modulador

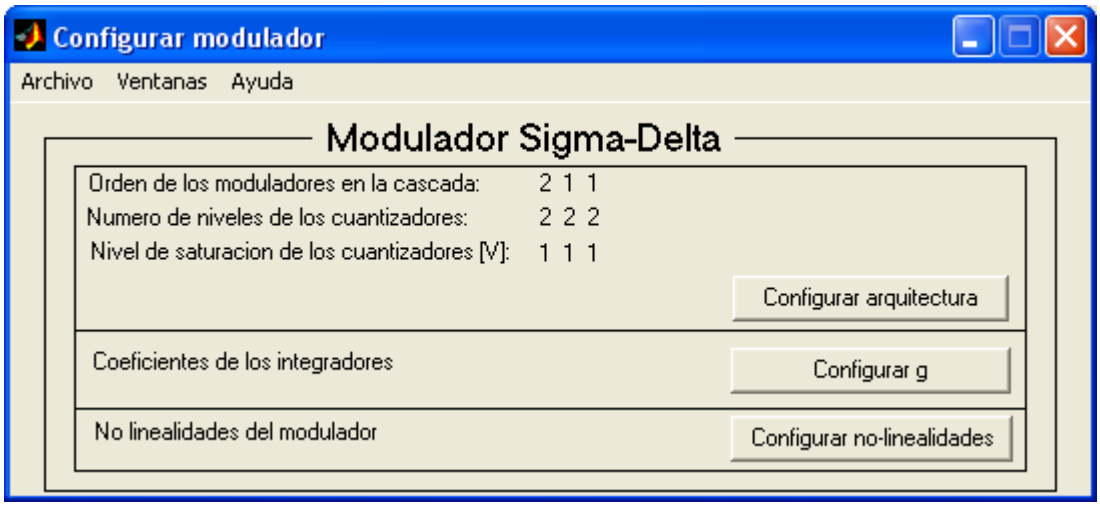


Figura 4-5 Interfaz de configuración del modulador

Esta ventana (figura 4-5) nos permite configurar el modulador sigma-delta en cascada que deseemos, su arquitectura, coeficientes de ganancia de los integradores y no-linealidades.

Disponemos de tres botones que abren nuevas ventanas de configuración:

- *Configurar arquitectura*
- *Configurar 'g'*
- *Configurar no linealidades*

Inicialmente sólo está activo el botón *"configurar arquitectura"*.

Una vez fijada una arquitectura concreta mediante el primero de los configuradores, podremos establecer los coeficientes *g* y las *no linealidades* del modulador accediendo a sus correspondientes configuradores a través de los botones.

En la interfaz se muestran los datos correspondientes a la arquitectura seleccionada:

- Orden de los moduladores en cascada
- Número de escalones de cuantización de los cuantizadores
- Nivel de saturación de los cuantizadores.

4.4.1 Configurar la arquitectura del modulador

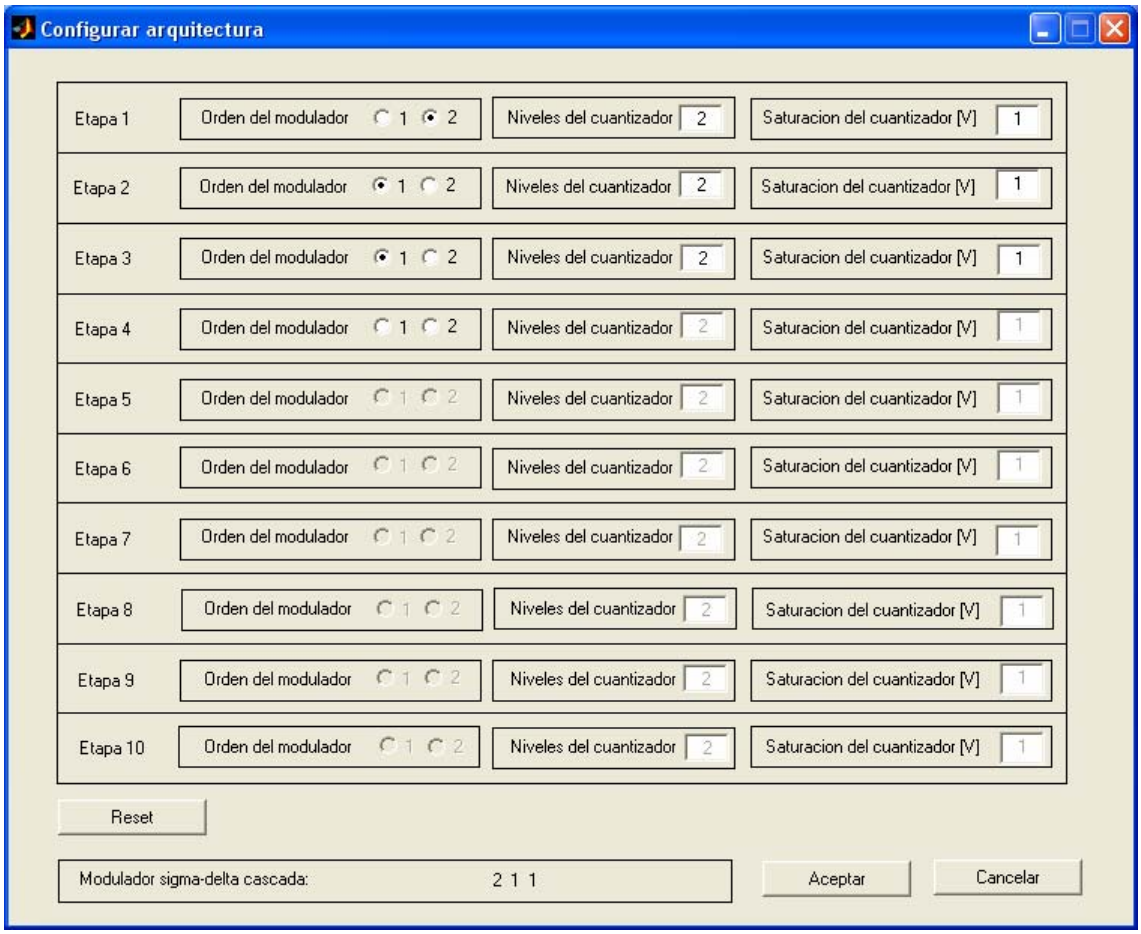


Figura 4-6 Interfaz correspondiente al configurador de arquitectura

Establecemos la arquitectura deseada para el modulador sigma-delta.

Las etapas de la cascada están organizadas por filas, de forma análoga a como se representa la estructura circuital del modulador.

- En una primera columna mediante un selector marcaremos (1 ó 2) según sea el orden del modulador en la etapa de la cascada (primer o segundo orden).
- A continuación, en la siguiente columna, se fija el número de niveles del cuantizador o escalones de cuantización de dicha etapa (2 niveles por defecto, cuantizador de 1-bit).
- Finalmente, establecemos el nivel de saturación del cuantizador en la etapa en cuestión, es decir, la tensión máxima que este es capaz de cuantizar sin cometer un error superior a la mitad del escalón de cuantización.

Una vez fijada una etapa, se hacen activos los controles relativos a la etapa siguiente. En la parte inferior se refleja el modulador seleccionado. También se dispone de un reset para comenzar de nuevo la configuración de la arquitectura.

4.4.2 Configurar los coeficientes "g"

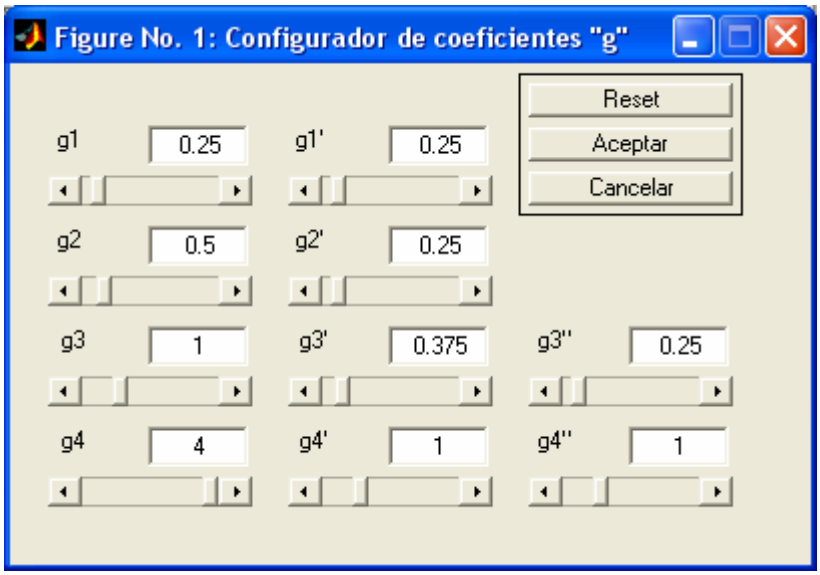


Figura 4-7 Interfaz correspondiente a los coeficientes de ganancia de los integradores

Los coeficientes "g" son característicos de cada arquitectura. Al acceder a esta ventana, fijada una arquitectura previamente, se mostrarán de forma ordenada únicamente los coeficientes relativos a la arquitectura en cuestión.

Podemos introducir los datos mediante deslizadores o cajas de texto indistintamente.

La nomenclatura correspondiente a los coeficientes de ganancia de los integradores es la misma que se sigue en el Capítulo 3.

Si se modifica el número de etapas de modulación o el orden de los moduladores de cualquiera de las etapas, el configurador resetea los coeficientes introducidos previamente ya que estos no tienen por qué ser los mismos que los de la arquitectura anterior a la modificación.

Un cambio exclusivamente en el número de escalones de cuantización o nivel de saturación de los cuantizadores no conlleva un reseteo de los coeficientes de ganancia de los integradores ya que la arquitectura no se ve modificada y los coeficientes pueden seguir siendo los mismos.

4.4.3 Configurar no-linealidades del modulador

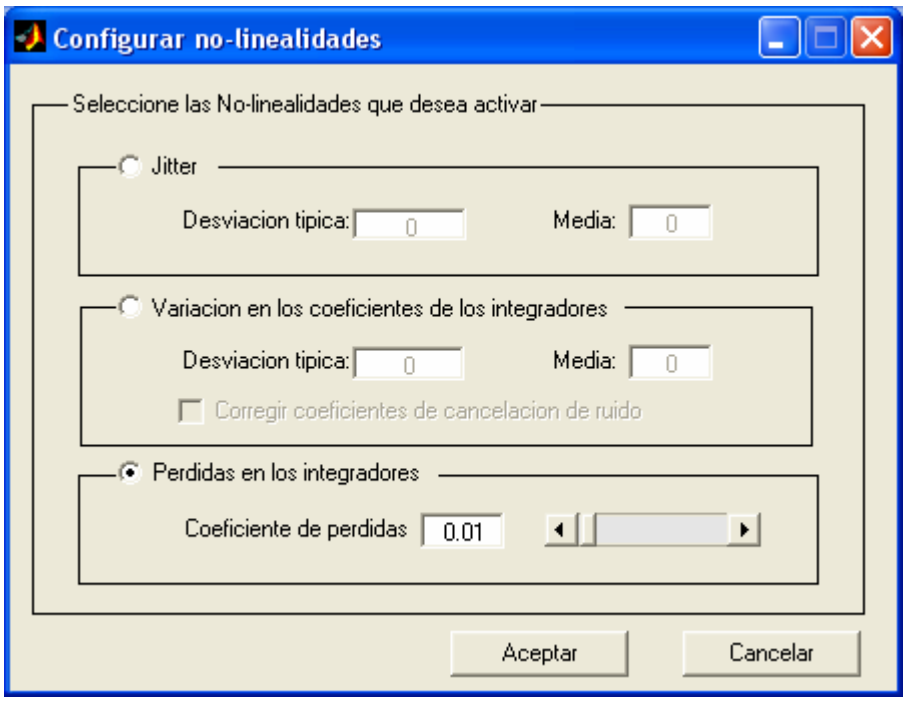


Figura 4-8 Interfaz para configurar no-linealidades

Mediante esta ventana se configuran las no linealidades del modulador. Al seleccionar el marcador correspondiente a una no-linealidad se activan los controles relativos a la misma.

Se dispone de tres no linealidades que pueden ser seleccionadas conjuntamente:

- Jitter (ver 3.3.6.3)
  - Desviación típica
  - Media
- Variación de los coeficientes de los integradores (ver 3.3.6.1)
  - Desviación típica
  - Media
  - Corregir coeficientes de cancelación de ruido:
    - Si se marca esta opción los coeficientes correspondientes a la cancelación de ruido son calculados a partir de las ganancias modificadas de los integradores (considerando la no-linealidad).
    - En caso contrario, los coeficientes no son corregidos y se calculan con las ganancias introducidas a partir del "configurador g", sin incluir las modificaciones resultantes de la no-linealidad.
- Pérdidas en los integradores (ver 3.3.6.2)
  - Coeficiente de pérdidas

4.5 Salida

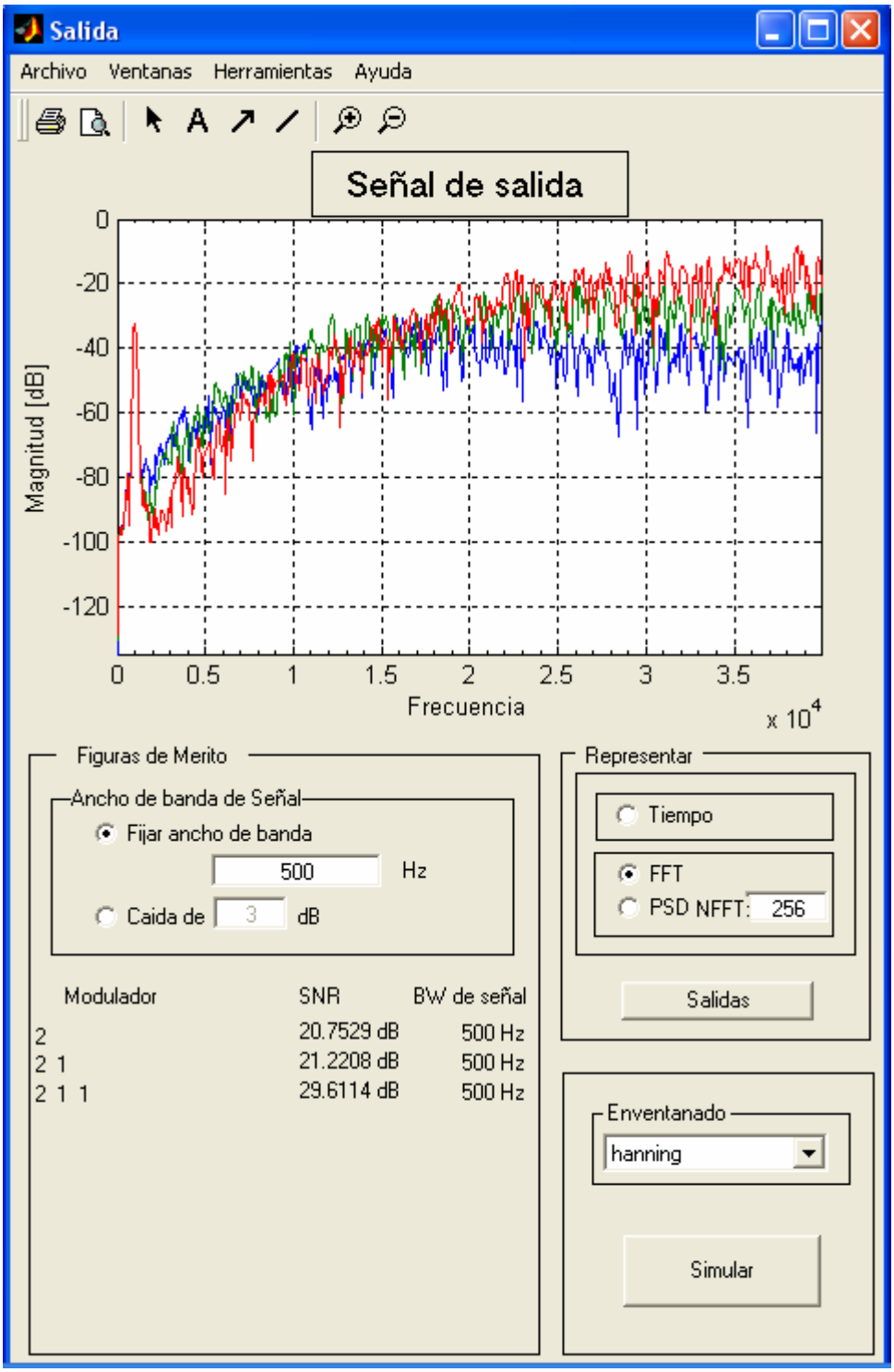


Figura 4-9 Interfaz correspondiente a la salida

En esta ventana, una vez establecidas una entrada y un modulador, se realiza la simulación, el procesado de señal y son mostrados los resultados gráficos y figuras de mérito de la modulación.

Se distinguen cuatro secciones en la ventana de la figura 4-9:

- *Ejes de representación gráfic*a (En la parte superior). Mostrará graficas relativas a los datos de salida. Puede representar:
  - *Dominio temporal*
  - *PSD (dominio frecuencial)*
  - *FFT (dominio frecuencial)*
- *Cuadrante de control de representación gráfica* (En medio a la derecha, "Representa"). Controla aquello que será representado en los ejes:
  - *Dominio temporal*
  - *PSD (dominio frecuencial)*
  - *FFT (dominio frecuencial)*
  - *Salidas:* Selección de las salidas del modulador que deseemos representar.
- *Cuadrante de simulación y procesado de señal* (Abajo a la derecha). Incluye:
  - Botón de simulación: Ejecuta la simulación.
  - Procesado de señal: Enventanado que se desea realizar a la salida.
    - Rectangular
    - Blackman
    - Hanning
    - Hamming
- *Cuadrante de Figuras de Mérito* (abajo a la izquierda). Muestra las figuras de mérito relativas al modulador que nos permitirán estimar la bondad del mismo.
  - Fijar ancho de banda de señal para calcular SNR
    - BW fijo: Se fija un ancho de banda de señal centrado en la frecuencia del tono de entrada.
    - Caída en dB: el ancho de banda de señal queda determinado por las frecuencias en que la señal cae el número introducido de decibelios respecto al valor de señal en la frecuencia de la sinusoide de entrada.
  - Relación de SNR y anchos de banda de señal de las salidas seleccionadas. El cálculo de la SNR se realizará sobre la FFT o PSD según esté seleccionada una u otra.

4.5.1 Seleccionar las salidas a representar

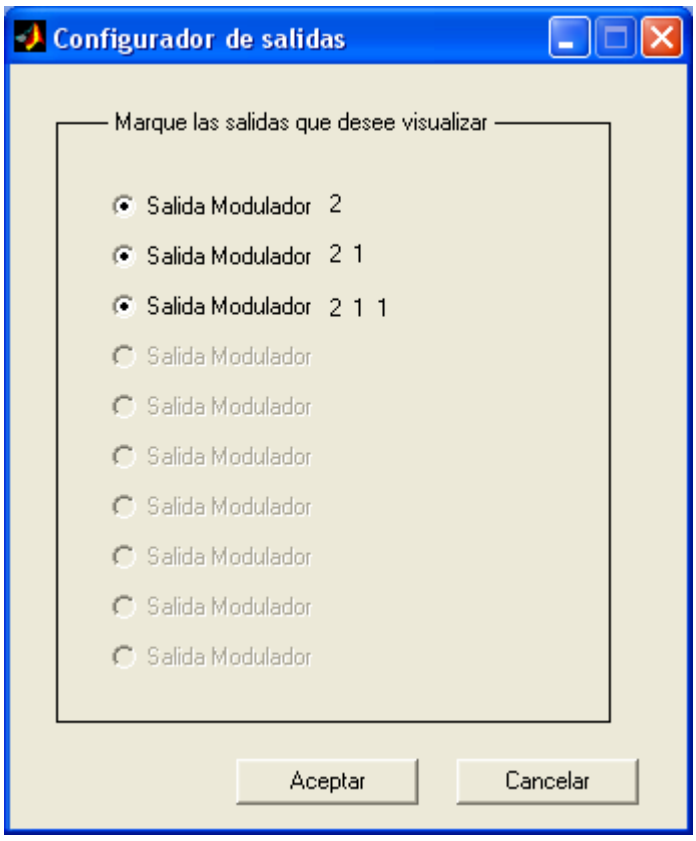


Figura 4-10 Interfaz que permite seleccionar las salidas a representar

Esta ventana (figura 4-10) se activa al pulsar el botón salidas.

Permite seleccionar aquellas salidas en la cascada del modulador que deseemos representar gráficamente.

Esta opción permite, para una misma simulación, comparar moduladores de un determinado orden con otros de menos etapas en la cascada.

4.6 Menú

Figura 4-11 Barra de menú

En las tres ventanas globales anteriores: *generar señal de entrada*, *configurar modulador* y *salida*, en la parte superior de las mismas, encontramos un menú como el de la figura 4-11.

En el menú se nos ofrecen nuevas opciones de forma ordenada. Han sido agrupadas en*: Archivo, Ventanas, Herramientas y Ayuda*.

Dada la similitud entre los menús relativos a las tres ventanas, veremos, a modo de ejemplo, el concerniente a la entrada e iremos indicando las particularidades para los otros dos casos (modulador y salida).

4.6.1 Archivo

En el menú *Archivo* se ofertan las siguientes opciones:

- *Cargar Entrada*: Abre una ventana a través de la que podremos explorar ficheros y carpetas de Windows y seleccionar una entrada entre las guardadas previamente en archivos ".mat". La entrada es representada en la interfaz.

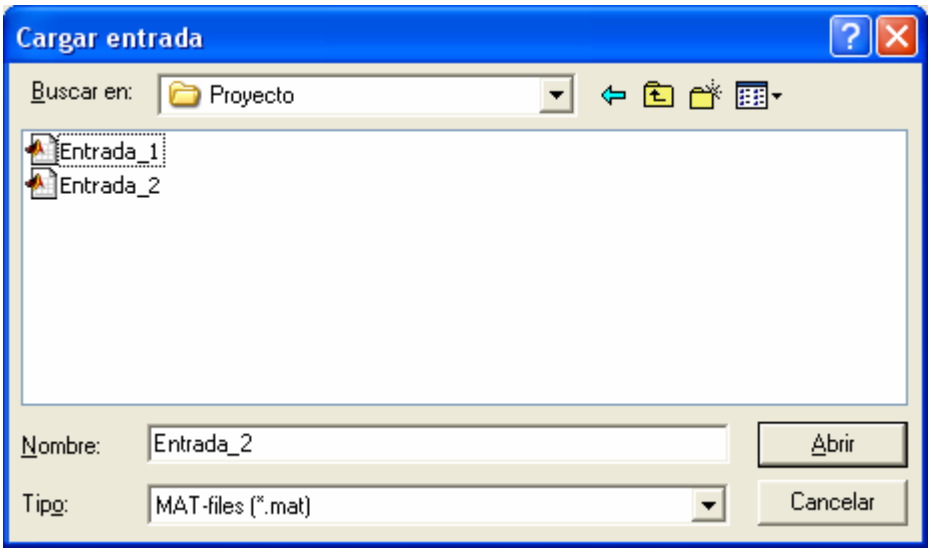


Figura 4-12 Interfaz para cargar entrada

Es importante reseñar aquí que, en el caso de la salida, al cargar una simulación, dado que los resultados son relativos a una entrada y un modulador en particular, se cargarán, la entrada, el modulador y la salida correspondiente a la simulación que fue guardada.

- *Guardar Entrada*: Abre una ventana con la que explorar ficheros y carpetas de Windows, captura el nombre de fichero deseado y guarda la entrada activa en la ubicación deseada en un archivo del tipo "nombre.mat".

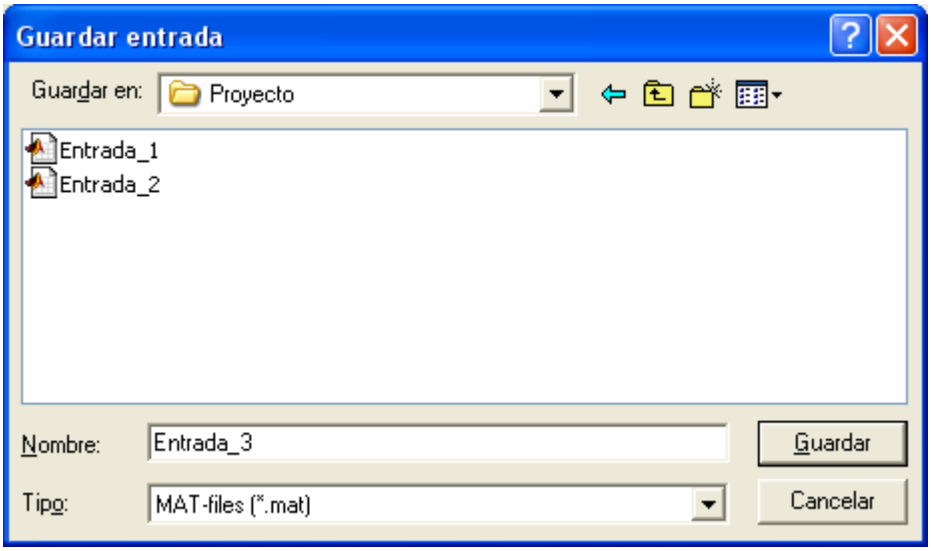


Figura 4-13 Interfaz para guardar entrada

Al igual que en el caso anterior, en la salida, al guardar una simulación, se salvarán no únicamente los resultados de la misma sino también la entrada y el modulador a los que dicha salida corresponde.

- *Previsualizar Impresión*:

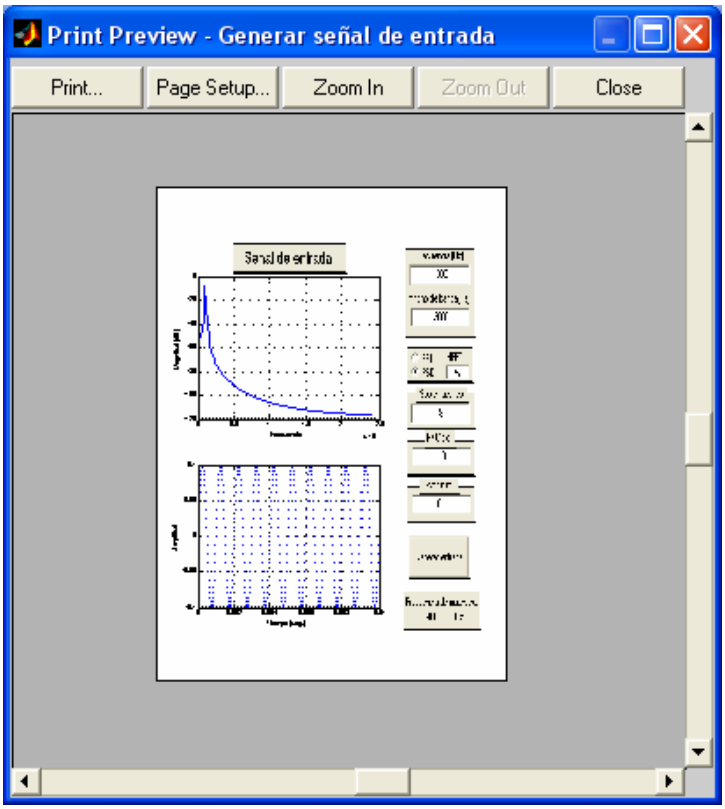


Figura 4-14 Interfaz correspondiente a la previsualización de impresión

Permite previsualizar la impresión de la entrada y configurar aspectos relativos a la misma, como son:

- Tamaño y posición del gráfico el papel.
- Tipo y orientación del papel.
- Líneas, ejes y colores, fondo…
- Imprimir gráficos y controles, o sólo gráficos.
- Control de impresora, número de copias, calidad…

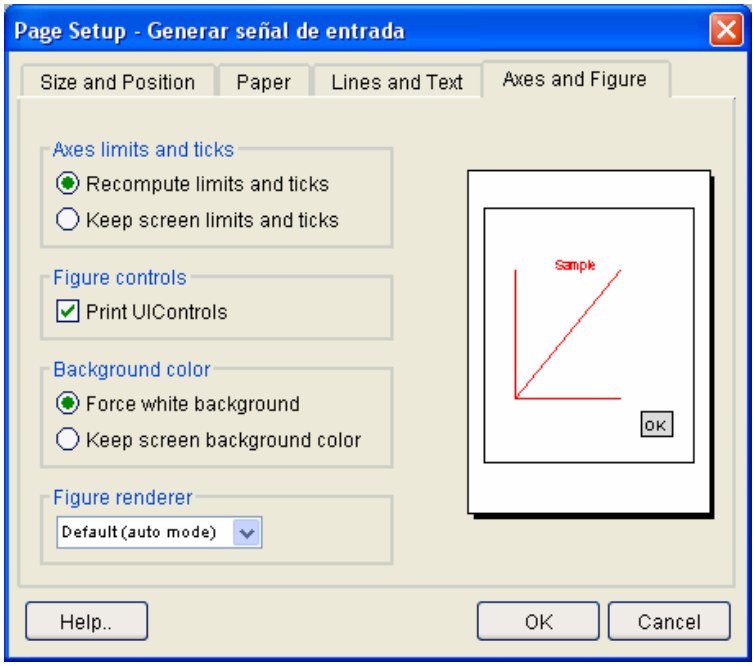


Figura 4-15 Interfaz para fijar propiedades de la página

- *Imprimir*
  - Control de impresora, propiedades
  - Número de copias
  - Intervalo de impresión
  - Calidad…

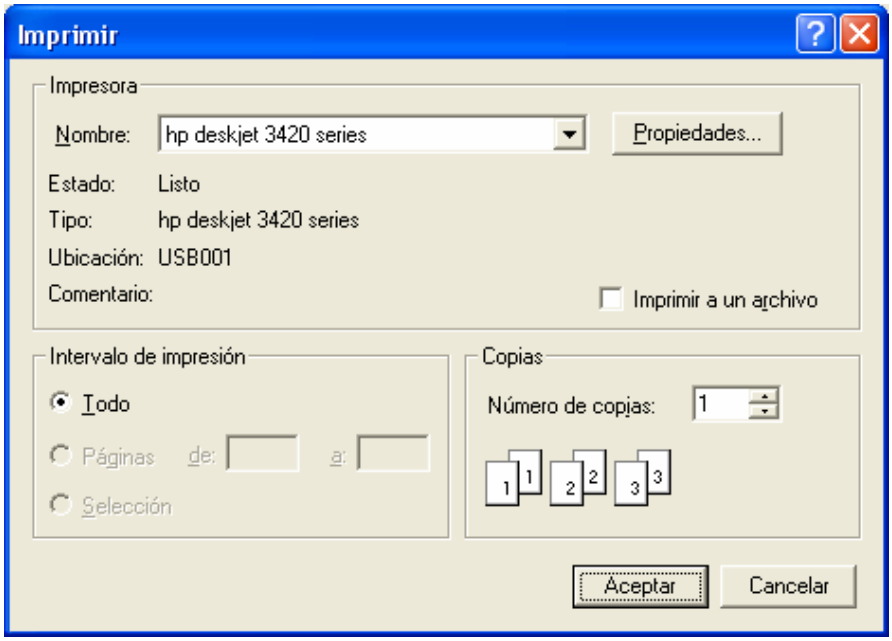


Figura 4-16 Interfaz de control de impresora

- *Salir*: Abre una ventana para confirmar si se desea salir. En caso afirmativo cierra la ventana activa.

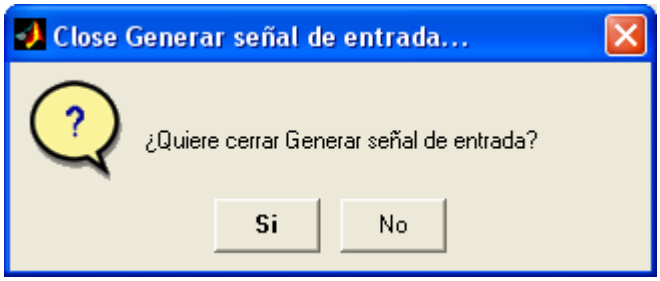


Figura 4-17 Interfaz de confirmación

4.6.2 Ventanas

Selecciona la ventana que se desea escoger como ventana activa. Si ya estaba abierta se trae al frente, en caso en caso contrario, se abre una nueva.

La ventana activa es reflejada con un marcador en el menú desplegable.

4.6.3 Herramientas

Esta utilidad del menú sólo se oferta en las ventanas de entrada y salida (y no en la de modulador) ya que es relativa a aspectos gráficos y éstas son las únicas ventanas con ejes para la representación de datos.

- *Activa Barra de Herramientas*: Activa una barra de herramientas con iconos de acceso directo a las siguientes utilidades:

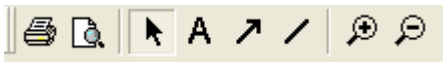


Figura 4-18 Barra de herramientas de la interfaz

  - Imprimir
  - Previsualizar Impresión
  - Editar
  - Insertar Puntero
  - Insertar línea
  - Insertar texto
  - Zoom In
  - Zoom Out
- *Desactiva Barra de Herramientas*: Anula la opción anterior.
- *Zoom On*: Representa la sección seleccionada con el cursor sobre el gráfico.
- *Zoom Off*: Anula la opción anterior

4.6.4 Ayuda

En las tres ventanas principales podemos acceder a través del menú a la documentación del proyecto en versión "HTML".

Al seleccionar "Documentación de Ayuda" se abre el navegador web predeterminado.

El índice proporciona un listado completo de hipervínculos a los distintos apartados del proyecto.

Esta es una forma, cómoda e intuitiva de disponer de toda la información relativa al proyecto durante la ejecución del programa.

Además, en la línea de comandos de MATLAB también podemos escribir "help nombre_de_función" y nos aparecerá una descripción de la función, parámetros de entrada, de salida y funciones relacionadas.

Capítulo 5
Funciones, estructura y mapa del código.

5.1 Introducción

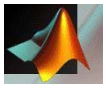
En este capítulo desarrollaremos aspectos relativos a la programación en MATLAB del proyecto como son: las funciones y estructura del código, los algoritmos implementados, sus variables de entrada y salida, la interconexión con otras funciones y sus características y propiedades más interesantes (versatibilidad, flexibilidad, carácter vectorial o escalar…).

Para facilitar la comprensión de la organización del código de programación (funciones, variables y archivos de MATLAB) nos apoyaremos en mapas conceptuales. Partiremos de una visión general para adentrarnos después en cada uno de los módulos y profundizar así en la descripción de los mismos.

La nomenclatura que emplearemos en dichos mapas es la que sigue:

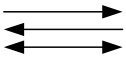
Usuario: Recogida de datos, selecciones o eventos originados por el usuario.

Incluye el código correspondiente a la interfaz de usuario. Archivos ".fig" y ".m" de la GUI.



Archivo de MATLAB "funcion.m". Suele incluir funciones o conjunto de instrucciones y comandos de MATLAB

Archivo de MATLAB "nombre.mat". Archivo de datos a través del cual se comunican distintas ventanas de la interfaz.



Flujo de información o invocación. Una u otra dirección y bidireccional.

Según el color la información será:

- **Negro**: Relativa a funciones '.m'
- **Azul**: Relativa a intervención del usuario.
- **Rojo**: Comunicación entre interfaces y ficheros '.mat'

datos.mat

5.2 Guía de estilo

Las funciones vectoriales en MATLAB son mucho más rápidas que sus contrapartidas escalares. Es por esto que, en la medida de lo posible, se ha intentado vectorizar los algoritmos de cálculo, realizándolos con vectores y matrices, y no con escalares dentro de bucles.

Aunque los vectores y matrices pueden ir creciendo a medida que se necesita, es mucho más rápido reservarles toda la memoria necesaria al comienzo del programa. Utilizamos para ello la función *zeros*, de modo que la memoria reservada es continua.

También se ha empleado la función *profile* para conocer en qué sentencias de cada función se gasta la mayor parte del tiempo de cálculo. De esta forma se han descubierto "cuellos de botella" y se han hecho algoritmos mucho más eficientes.

El programa ha sido desarrollado incrementalmente, comprobando cada función o componente añadido. De esta forma, al construir sobre algo que ya ha sido probado y que funciona, al aparecer algún error, lo más probable es que se deba a lo último que se ha añadido, y de esta manera la búsqueda de errores quedaba más acotada y ha resultado mucho más sencilla. El *debugger* ha sido una herramienta muy útil a la hora de acortar ese tiempo de puesta a punto.

En este mismo sentido, puede decirse que pensar bien las cosas al programar (sobre una hoja de papel en blanco, mejor que sobre la pantalla del PC) ha resultado rentable, porque se ha disminuido más que proporcionalmente el tiempo de depuración y eliminación de errores.

Otro objetivo de la programación ha sido mantener el código lo más sencillo y ordenado posible. Al pensar en cómo hacer el programa o en cómo realizar una determinada tarea se ha pensado siempre primero en la solución más sencilla, y luego se han planteado otras cuestiones como la eficiencia.

Especial incidencia se ha hecho en potenciar la flexibilidad del código. Las funciones desarrollan algoritmos generales y no restringidos únicamente al desarrollo propio y concreto del proyecto. Se implementan algoritmos enormemente potentes cuya limitación viene dada por la memoria disponible y no por el algoritmo descrito en sí mismo. Atendiendo a este motivo se ha evitado también en todo momento el empleo de variables globales.

Finalmente, el código ha sido escrito de manera clara y ordenada, introduciendo comentarios, utilizando líneas en blanco para separar distintas partes del programa, sangrando las líneas para ver claramente el rango de las bifurcaciones y bucles, utilizando nombres de variables que recuerdan el significado de la magnitud física correspondientes, etc.

Por lo que respecta a aspectos concretos de la programación, como se ha comentado, se ha optado por evitar el uso de variables globales de modo que la comunicación entre funciones se realiza a través del paso de parámetros como argumento o, en caso de que las funciones pertenezcan a la misma interfaz de usuario y sean relativas al *"callback"* de algún elemento de control, empleando la estructura *handles*, que es visible desde todas ellas. La comunicación entre las distintas interfaces de usuario se efectúa a través de archivos de datos ".mat" , mediante los comandos *save* y *load*, que guardan matrices y vectores de forma selectiva y en ficheros con un determinado nombre

5.3 Ejecución del programa

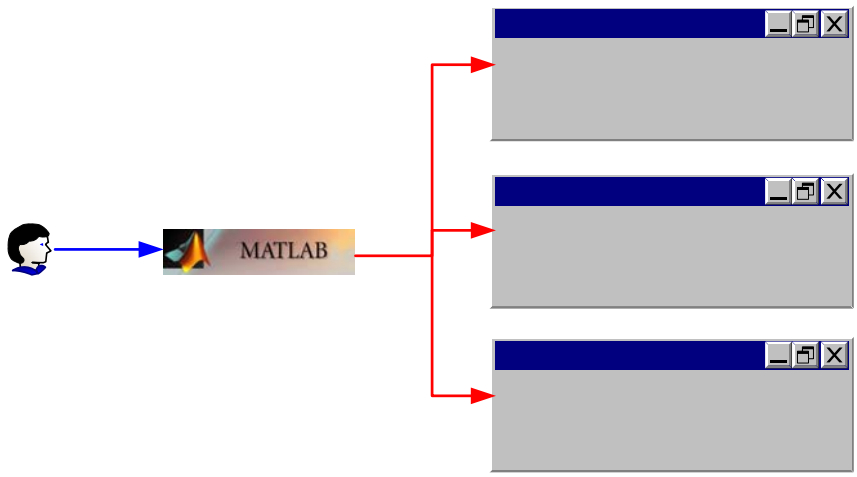


Figura 5-1 Diagrama de ejecución del programa

Para comenzar el programa, el usuario deberá ejecutar desde la línea de comandos de MATLAB la función "sd_cascada". Esta función inicializa ciertas variables y realiza una llamada a las tres funciones correspondientes a las ventanas de interfaz de usuario programadas:

- GUI_entrada: Interfaz de usuario para generar señal de entrada.
- GUI_modulador: Interfaz de usuario para configurar modulador.
- GUI_salida: Interfaz de usuario para generar la salida y su procesamiento digital.

5.4 Mapa del proyecto

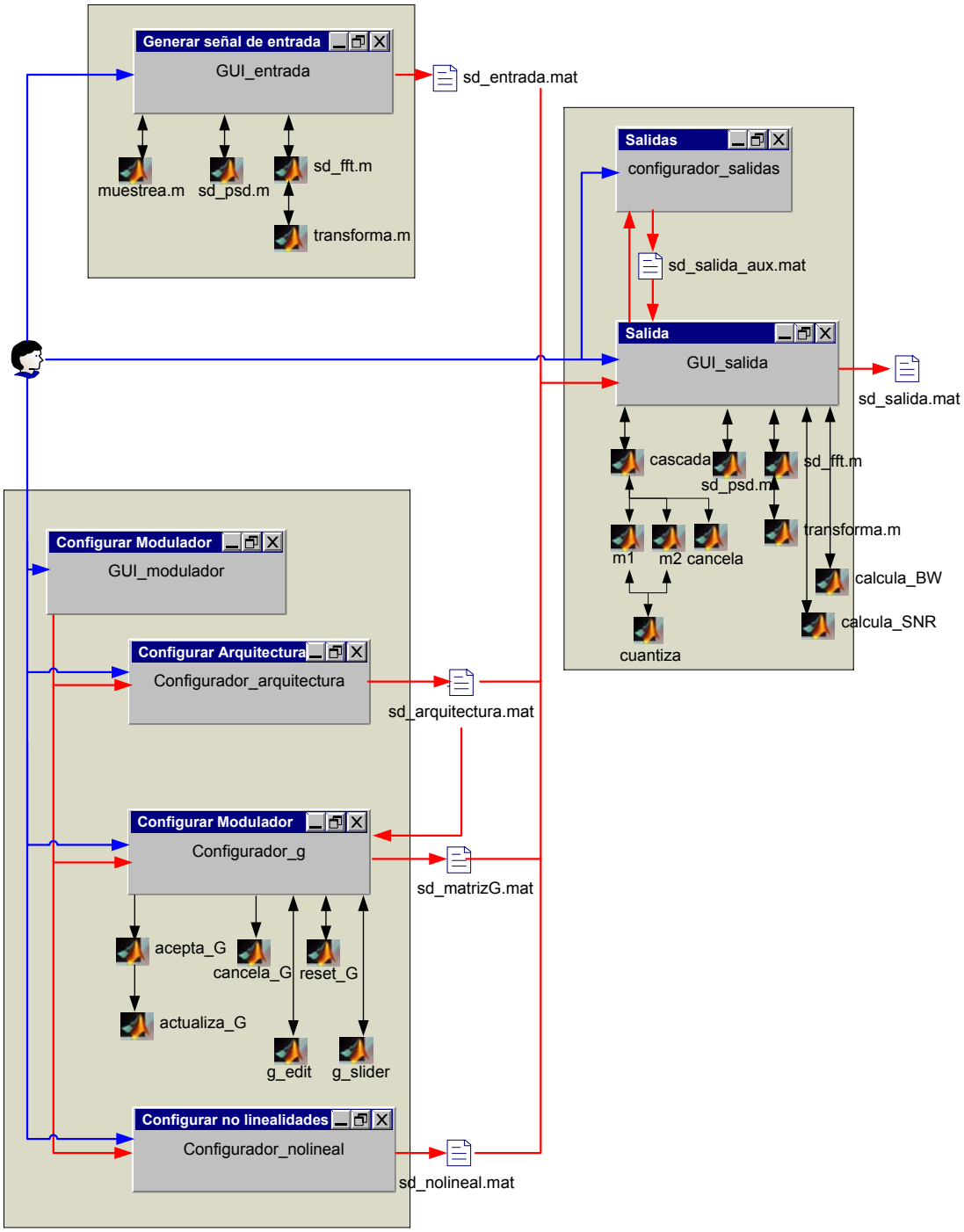


Figura 5-2 Mapa del proyecto

La figura 5-2 se corresponde con el mapa conceptual del proyecto completo. En los apartados siguientes analizaremos cada una de las partes que comprenden la totalidad del programa; su funcionabilidad y relaciones contextuales.

A grandes rasgos apreciamos los tres grandes bloques principales de interfaz de usuario y como, a través de lo mismos, podemos acceder a nuevas interfaces o funciones de MATLAB directamente. Se observa también que la interconexión entre GUIs (interfaces de usuario) se realiza siempre a través de ficheros de datos ".mat"

Se han obviado en la representación gráfica las funciones propias de las interfaces así como los "callback" asociados a los controles y eventos de las mismas, ya que conllevaría tal complejidad en la representación que ésta carecería de sentido al ser su objeto el facilitar la comprensión global del proyecto.

5.5 GUI_entrada

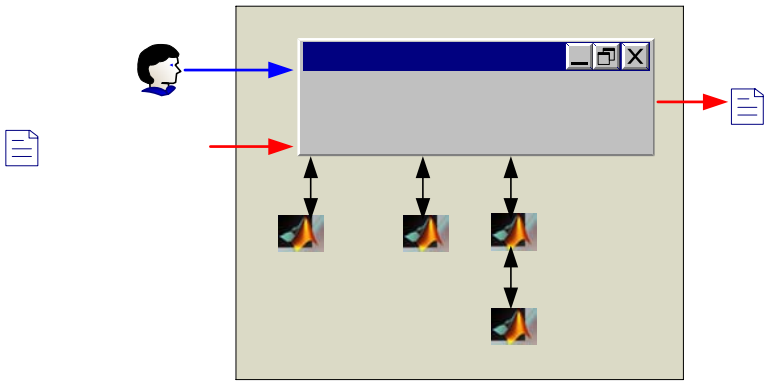


Figura 5-3 Mapa conceptual de la entrada

A través de esta interfaz el usuario introducirá los datos relativos a la entrada sinusoidal del simulador. La señal de entrada propiamente dicha será generada al producirse un evento sobre el botón *"generar entrada"*. Se representará, además, dicha señal tanto en el dominio temporal como en el frecuencial.

La función asociada a este botón es:

entrada_Callback(hObject, eventdata, handles)

Esta función captura los datos de la interfaz y los asigna a sus correspondientes variables:

```
% Tomamos datos de entrada del usuario a traves de la interfaz GUI
Fx = str2double(get(handles.frecuencia,'String')); %frecuencia de la señal sinusoidal de entrada
```

```
M = str2double(get(handles.sobremuestreo,'String')); %tasa de sobremuestreo
n_ciclos = str2double(get(handles.ciclos,'String')); %numero de ciclos de la señal  de entrada
B = str2double(get(handles.ancho_de_banda,'String')); %ancho de banda
A = str2double(get(handles.amplitud,'String')); %amplitud de la señal sinusoidal de entrada
```

Acto seguido, con dichos datos, pasa a muestrear la señal y generar la secuencia de entrada mediante la función "muestrea.m".

```
%Muestreamos la señal
[x Fs kTs]=muestrea(Fx,M,n_ciclos,B,A,m_jitter,(dt_jitter)^2);
```

donde "m_jitter" y "dt_jitter" son la media y desviación típica del efecto jitter, tomados del fichero "sd_nolineal.mat".

A la secuencia obtenida se le realiza la FFT y la PSD mediante las funciones "sd_fft" y "sd_psd" respectivamente.

```
%Calculamos la fft y psd
[Xw_fft,eje_f_fft]=sd_fft(x,Fs);
NFFT= str2double(get(handles.e_nfft,'string'));%tomamos nfft de la GUI
[Xw_psd,eje_f_psd]=sd_psd(x,NFFT,Fs,rectangle);
```

Finalmente representa la secuencia de entrada en el dominio temporal y frecuencial (PSD ó FFT, según se haya seleccionado). En el caso de la FFT, dado que la señal en el dominio temporal es real, su transformada en frecuencia será par. Es por esto que mostramos sólo las componentes espectrales correspondientes a frecuencias positivas. Para ello:

```
%dibujamos en frecuencia la fft
axes(handles.ejes_frecuencia) % seleccionamos los ejes adecuados
%al ser real la señal en el tiempo, en frecuencia su espectro sera par
%dibujamos solo el espectro correspondiente a "frecuencias positivas"
plot(eje_f_fft,20*log10(abs(Xw_fft)));
%fijamos los ejes para mostrar solo las frecuencias positivas
axis([0,eje_f_fft(end),min(20*log10(abs(Xw_fft))), max(20*log10(abs(Xw_fft)))]);
```

5.5.1 Funciones invocadas

Pasamos a describir las funciones que son llamadas durante el desarrollo correspondiente a la función *"callback"* del botón *generar entrada*. Para cada caso veremos:

- Declaración de la función en Matlab.
- Descripción general de la función.
- Parámetros de entrada a la función.
- Parámetros de salida devueltos por la función
- Código, algoritmos, comentarios de especial interés.

Para ver la totalidad del código, remitirse al anexo del proyecto.

5.5.1.1 muestrea

Declaración:

```
function [muestreada,Fs,kTs] =muestrea (Fx,M,n_ciclos,B,A,m,v)
```

Descripción:

Muestrea una señal sinusoidal para los parámetros de entrada dados generando una secuencia.

Parámetros de entrada:

- Fx: Frecuencia del tono.
- M: Tasa de sobremuestreo.
- n_ciclos: Número de periodos del tono.
- B: Ancho de banda.
- A: Amplitud.
- m: media de efecto Jitter
- v: varianza de efecto Jitter

Parámetros de salida:

- muestreada: secuencia resultante.
- Fs: Frecuencia de muestreo.
- kTs: Eje para representación temporal.

Código de interés:

Si no se pasa la media y la varianza no hay efecto jitter, (por defecto m=v=0)

```
Fn=2*B;%Frecuencia de Niquist respecto a B
Fs=M*Fn;%Frecuencia de sobremuestreo
Ts=(1/Fs);%Periodo de sobremuestreo
kTs=[0:Ts:(n_ciclos*(1/Fx))];%vector correspondiente al eje x en representación temporal
kTs=kTs+sqrt(v)*randn(size(kTs))+m;%efecto jitter
muestreada=A*sin(2*pi*Fx*kTs);%generamos la frecuencia para el tono deseado
```

Para modelar el efecto Jitter se ha considerado una distribución normal.

5.5.1.2 sd_psd

Declaración:

```
function [Yw,F]=sd_psd(Y,NFFT,Fs,ventana)
```

Descripción:

Aplica la PSD a las filas de una matriz de entrada. Función matricial.

Parámetros de entrada:

- Y: matriz de vectores por filas.
- NFFT: parámetro de la "psd". Tamaño de las secciones superpuestas.
- Fs: frecuencia de muestreo.
- ventana: enventanado a realizar.

Parámetros de salida:

- Yw: matriz de transformadas PSD por filas.
- F: eje de frecuencias.

Código de interés:

Esta función opera matricialmente por filas, como podemos observar en el siguiente extracto de código:

```
for (i=1:1:n_etapas)
[Yw(:,i),F]=psd(Y(i,:),256,Fs,ventana); %transformamos cada fila de entrada
end
Yw=Yw';% trasponemos para organizar las salidas por filas
F=F';% trasponemos para devolver un vector fila
```

5.5.1.3 sd_fft

Declaración:

```
function [Yw,F]=sd_fft(Y,Fs)
```

Descripción:

Aplica la FFT a las filas de una matriz de entrada. Función matricial.

Invoca la función *transforma.m*

Parámetros de entrada:

- Y: matriz de vectores por filas.
- Fs: frecuencia de muestreo.

Parámetros de salida:

- Yw: matriz de transformadas FFT por filas.
- F: eje de frecuencias.

Código de interés:

Al igual que en el caso de la PSD esta función opera matricialmente por filas:

```
%realizamos la FFT por filas
for (i=1:1:n_etapas)
  Yw(i,:)=transforma(Y(i,:)); %transformamos cada salida y las almacenamos por filas
end
```

5.5.1.4 transforma

Declaración:

```
function transformada=transforma(secuencia,N);
```

Descripción:

Realiza la transformada de Fourier discreta FFT de N puntos de un vector.

- Si sólo se pasa un argumento, toma N como la potencia de 2 mayor o igual que la longitud de la secuencia para agilizar la FFT.
- Si N es menor que la longitud de la secuencia, se trunca la secuencia.
    - "Warning: Se esta introduciendo efecto aliasing".
- Si N es mayor que la longitud de la secuencia; se rellena con ceros.

  - "Warning: Rellenando con ceros. Zero padding".
- Se devuelve la FFT centrada en cero.

Parámetros de entrada:

- secuencia: vectores al que realizar la FFT.
- N: Número de puntos de los que se desea realizar la transformada.

Parámetros de salida:

- transformada: Transformada de Fourier discreta FFT de N puntos de secuencia.

Código de interés:

Realizamos la transformada de Fourier discreta de forma que esté centrada en cero:

```
transformada= (1/length(secuencia))*fftshift(fft(secuencia,N));
% fft de N puntos, centrando en w=0. Escalamos en magnitud para que no sea función de la longitud
```

5.6 GUI_modulador

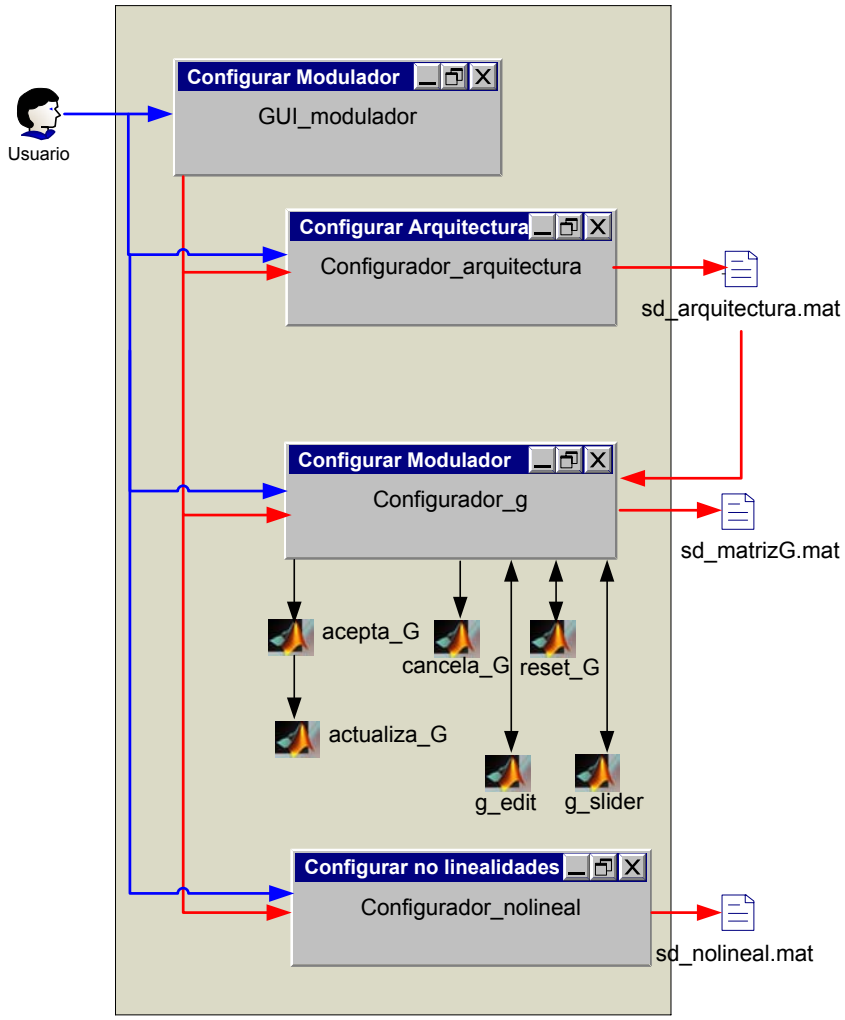


Figura 5-4 Mapa conceptual del modulador

Esta interfaz de usuario permite la configuración del modulador sigma-delta en cascada: número de etapas, orden de las mismas, cuantizadores, parámetros g y no linealidades.

Muestra la arquitectura seleccionada en la ventana:

- Orden de los moduladores de la cascada
- Número de niveles de los cuantizadores
- Nivel de saturación de los cuantizadores

Da acceso a tres interfaces de configuración a través de los "callbacks" de tres botones:

- *configurador_ arquitectura.*
- *configurador_g*

- *configurador_ nolineal*

Vemos, a continuación, en detalle, cada una de estas tres interfaces de usuario las funciones y comandos asociados a las mismas.

5.6.1 configurador_arquitectura

Esta interfaz no invoca funciones especialmente diseñadas para el desarrollo del proyecto. Se vale únicamente de los "callbacks" de los elementos de la interfaz y de la "opening_function" de la misma (función que se ejecuta al abrir la interfaz). Todas estas funciones están incluidas en *configurador_arquitectura.m*.

Para la comunicación entre distintas funciones de la interfaz se crea la clase *orden* (almacenará un vector con el orden de cada etapa de la cascada) dentro de la estructura *handles*, visible por todas las funciones de *configurador_arquitectura.m*.

```
%Creamos clases en la estructura para almacenar datos
handles.orden; %vector con el orden de cada etapa de la cascada
guidata(hObject,handles);
```

El orden se irá actualizando al pulsar sobre uno de los marcadores de orden (1 ó 2, primer o segundo orden) en cada etapa. Para el caso de que seleccionáramos un modulador de segundo orden en la primera etapa de la cascada, se ejecutaría la función *mod1_2_Callback* que haría:

```
handles.orden(1)=2;%orden de la primera etapa es 2
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));%escribimos el orden
set(handles.mod1_1, 'Value', 0);
set(handles.mod1_2, 'Value', 1);%activamos el marcador correspondiente
%habilitamos los controles del siguiente modulador
set(handles.mod2_1,'Enable','on');
set(handles.mod2_2,'Enable','on');
set(handles.cuant1,'Enable','on');
set(handles.sat1,'Enable','on');
```

Es decir, se guarda la información correspondiente al orden de dicha etapa y se habilitan los controles del siguiente modulador por si se quiere añadir una etapa más a la cascada.

Por lo que respecta a los cuantizadores, sus escalones de cuantización y nivel de saturación son capturados al pulsar aceptar.

```
%recuperamos el orden
orden=handles.orden;
%recuperamos numero escalones de cuantizacion y niveles de saturacion de los cuantizadores
for(i=1:1:length(orden))
   guarda_cuant=['cuant(',int2str(i),')=str2double(get(handles.cuant',int2str(i),',''String''));'];
   eval(guarda_cuant);
   guarda_sat=['sat(',num2str(i),')=str2double(get(handles.sat',int2str(i),',''String''));'];
   eval(guarda_sat);
end
```

Al seleccionar una determinada arquitectura y pulsar aceptar, los datos (*orden*, *cuant, sat* y variables de control) son volcados al fichero "sd_arquitectura.mat", archivo que cargarán otras interfaces para comprobar la arquitectura del modulador y el estado de dichas variables de control.

```
%guardamos el orden del modulador, y los niveles y rangos de los cuantizadores
save sd_arquitectura orden cuant sat m_reset;
```

5.6.2 configurador_g

Esta herramienta, a diferencia del resto de interfaces de usuario, ha sido desarrollada sin la ayuda de *"GUIDE"* de *MATLAB*, debido al carácter no estático de la configuración de la misma, dado que los elementos a mostrar y su disposición están íntimamente relacionados con la arquitectura previamente escogida para el modulador.

Los coeficientes de ganancia para cada integrador dependerán del número de moduladores y del orden de cada uno de ellos en la cascada (Ver Capítulo 3).

El archivo *"configurador_g.m*" genera una interfaz de usuario para tomar los coeficientes *'g'* (ganancias a la entrada de los integradores) correspondientes a una determinada arquitectura (seleccionada con anterioridad mediante *"configurador_arquitectura"*).

En la interfaz se muestran de forma ordenada únicamente los coeficientes que es necesario fijar para el modulador en cuestión. Se despliega para cada coeficiente una *caja de texto editable* y un *deslizador* para facilitar al usuario la captura o modificación de valores.

Los coeficientes (g_i,g_i',g_i'') de cada integrador son almacenados por columnas en una matriz G de la forma:

$$G_{(3\times N)} = \begin{bmatrix} entrada_{11}(+) & entrada_{21}(+) & \dots & entrada_{N1}(+) \\ entrada_{12}(-) & entrada_{22}(-) & \dots & entrada_{N2}(-) \\ feedback_1(-) & feedback_2(-) & \dots & feedback_N(-) \end{bmatrix}$$

A modo de ejemplo, para un modulador *sigma-delta 2-2* como el de la figura 5-5

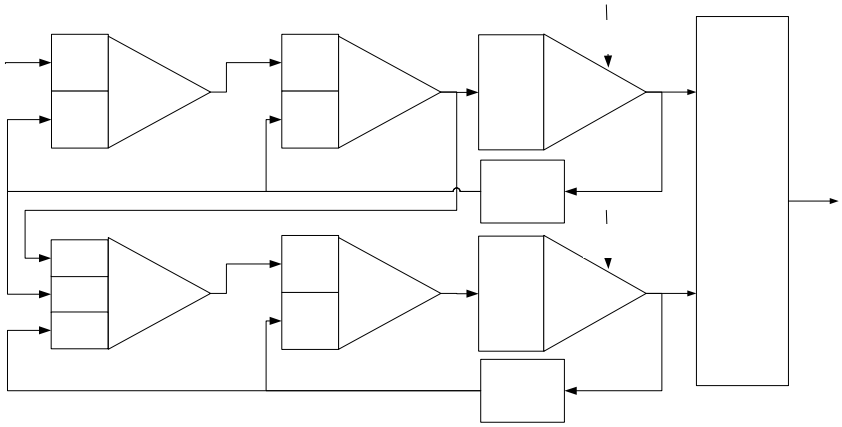


Figura 5-5 Modulador sigma-delta en cascada 2-2

La matriz G quedaría:

$$G_{(3\times4)} = \begin{bmatrix} g_1 & g_2 & g_3 & g_4 \\ 0 & 0 & g_3' & 0 \\ g_1' & g_2' & g_3'' & g_4' \end{bmatrix}$$

Obsérvese que la última fila corresponde siempre con los parámetros *g* que multiplican señales que llegan realimentadas a los integradores. Por ello parece ésta la configuración más lógica para organizar una matriz G que contenga todos los parámetros multiplicativos a la entrada de los integradores. Comprobaremos que esta estructura resulta muy conveniente a la hora de desarrollar el algoritmo de modulación de forma sencilla y eficaz.

La interfaz gráfica para configurar los *"coeficientes g"* hace uso, además, de funciones de control y funciones correspondientes a los "callbacks" de los elementos de dicha interfaz, que son: *acepta_G.m, cancela_G.m, reset_G.m, actualiza_G.m, g_edit.m, g_slider.m*. Vemos, a continuación, cada una de ellas en detalle.

5.6.2.1 acepta_G, cancela_G y reset_G

Estas tres funciones (*acepta_G.m, cancela_G.m, reset_G.m*) se correponden con los "callbacks" de los botones de la interfaz de usuario para configurar los parámetros *'g'*.

- *acepta_G.m*: "Callback" del botón "aceptar" de la interfaz "configurador_g".
  - Actualiza la matriz G y guarda los resultados en sd_matrizG.mat
  - Parámetros de entrada:
    - *orden:* Determina la configuración de la matriz de coeficientes.

        - *h_slider:* Vector con los manejadores de los deslizadores de la interfaz. Son necesarios para tomar el valor de coeficiente seleccionado.
    - Invoca la función *actualiza_g*
- *cancela_G.m,* "Callback" del botón "cancelar" de la interfaz "configurador_g".
    - Cierra el configurador de coeficientes 'g' sin hacer nada mas.
- *reset_G.m:* "Callback" del botón "reset" la interfaz "configurador_g".
    - Resetea el "configurador g" y fija a cero todos los valores
    - Parámetros de entrada:
        - h_slider: vector con los manejadores de los *deslizadores*
        - h_edit: vector con los manejadores de las *cajas de texto editables*

5.6.2.2 actualiza_G

Declaración:

```
function G=actualiza_g(orden,h_slider)
```

Descripción:

Guarda en una matriz *G* los valores de los coeficientes tomados de la interfaz *"configurador_g"*.

Remitirse al *apartado 5.6.2* relativo al *"configurador_g"* para ver la forma en la que son organizados los datos en dicha matriz.

Parámetros de entrada:

- *orden:* Determina la configuración de la matriz de coeficientes G.
- *h_slider*: Vector con los manejadores de los deslizadores de la interfaz. Son necesarios para tomar el valor de coeficiente seleccionado.

Parámetros de salida:

- G: matriz de coeficientes

Código de interés:

A partir de la segunda etapa de la cascada, todos los casos son equivalentes. Sólo hay que considerar como caso especial el primer modulador de la cascada, ya que a diferencia del resto, su primer integrador tendrá dos entradas (dos coeficientes: *g* y *g'*) donde la segunda será la realimentada (*g'*); mientras que en el resto de los moduladores de la cascada, su primer integrador tendrá tres entradas (*g,g',g''*), siendo la de realimentación la tercera de ellas (*g''*). Vemos como se ha resuelto esto por código.

Para el primer modulador de la cascada:

```
primer_mod=orden(1); %capturamos el orden del primer modulador de la cascada
for (i=1:1:primer_mod)
   G(1,i)=get(h_slider(2*i-1),'value'); %coeficiente g
   G(2,i)=0; %cero, no existe este coeficiente
   G(3,i)=get(h_slider(2*i),'value'); %coeficiente g', entrada realimentada
end
```

Para los siguientes moduladores de la cascada:

```
%A partir de la segunda etapa de la cascada, todos los casos son iguales
k=orden(1)+1; %segunda coordenada de G. Indica el numero de integrador
j=2*orden(1)+1; %indice para desplazarnos por el vector de handles (manejadores)
for(i=2:1:length(orden))
   G(1,k)=get(h_slider(j),'value');%coeficiente g
   G(2,k)=get(h_slider(j+1),'value');%coeficiente g'
   G(3,k)=get(h_slider(j+2),'value');%coeficiente g", entrada realimentada
   k=k+1;
   j=j+3;
   if(orden(i)==2)%si esl orden del modulador de la etapa 'i' es 2, añadimos...
      G(1,k)=get(h_slider(j),'value');%coeficiente g
      G(2,k)=0;%cero, no existe este coeficiente
      G(3,k)=get(h_slider(j+1),'value');%coeficiente g', entrada realimentada
      k=k+1;
      j=j+2;
   end
end
```

5.6.2.3 g_edit y g_slider

Las funciones *g_edit* y *g_slider* se corresponden con los "calbacks" de las *cajas de texto editables* y los *deslizadores* de la interfaz de usuario *"configurador_g"* para capturar los valores de los coeficientes de multiplicación a la entrada de los integradores.

Estas dos funciones se encargan de sincronizar ambos controles (*cajas de texto editables* y los *deslizadores*), de forma que marquen un mismo valor para cada coeficiente y al modificar uno de ellos también se modifique el otro.

- *g_edit.m*: "Callback" de *cajas de texto editables* de la interfaz "configurador_g".

  - Actualiza el valor del *deslizador* del parámetro correspondiente, al de la *caja de texto editable* que acaba de ser modificada.
  - Comprueba que el valor introducido cae dentro del rango permitido. Si no es así, fija el valor anterior (el rango de valores es de 0 a 4)[1].
  - Parámetros de entrada:
    - *h_slider*: vector con los manejadores de los deslizadores.
    - *h_edit*: vector con los manejadores de las *cajas editables de texto.*
    - *i*: indice que nos indica a que parámetro concreto nos estamos refiriendo.

- *g_slider.m*: "Callback" de los *deslizadores* de la interfaz "configurador_g".
  - Actualiza el valor de la *caja de texto editable* del parámetro correspondiente, al del *deslizador* que acaba de ser modificado.
  - Parámetros de entrada:
    - *h_slider*: vector con los manejadores de los deslizadores.
    - *h_edit*: vector con los manejadores de las *cajas editables de texto.*
    - *i*: indice que nos indica a que parámetro concreto nos estamos refiriendo.

5.6.3 configurador_nolineal

Esta interfaz permite seleccionar las no-linealidades activas y capturar los datos relativos a las mismas.

Se ofrecen tres no-linealidades:

- Jitter
  - Desviación típica: Se almacena en *dt_jitter* en *sd_nolineal.mat.*
  - Media: Se almacena en *m_jitter* en *sd_nolineal.mat.*
- Variación de los coeficientes de los integradores
  - Desviación típica: Se almacena en *dt_var_g* en *sd_nolineal.mat.*
  - Media: Se almacena en *m_var_g* en *sd_nolineal.mat.*
  - Corregir coeficientes de cancelación de ruido: la variable *corregir_canc* indica si en el cálculo de los coeficientes de cancelación de ruido hemos de tener en cuenta las variaciones de dichos coeficientes debido a la no-linealidad.

[1] Se ha escogido un rango no excesivamente amplio para que los incrementos en el deslizador no sean muy grandes. El rango de valores es fácilmente editable a través del código.

- Pérdidas en los integradores
    - Coeficiente de pérdidas: Se almacena en *coef_perdidas* en *sd_nolineal.mat.*

Al pulsar sobre el botón "aceptar" se guardan la selección y datos introducidos. En el caso del efecto jitter, se llama a GUI_entrada, para que genere y muestre la nueva entrada.

5.7 GUI_salida

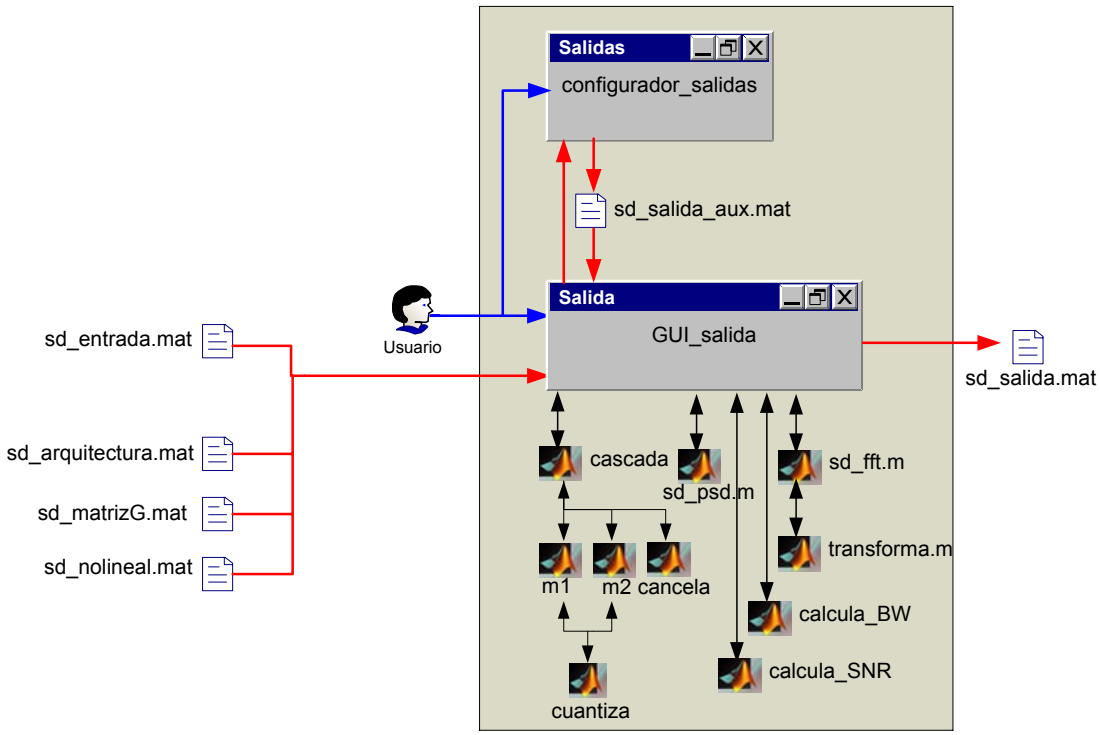


Figura 5-6 Mapa conceptual de la salida

Esta interfaz muestra la salida del modulador (tiempo, PSD, FFT) y las figuras de mérito correspondientes al mismo. Permite, además, seleccionar el enventanado que se desee realizar sobre la salida.

Mediante el botón "salidas" se accede a una nueva interfaz que permite seleccionar las etapas de la cascada que se quieran visualizar.

La simulación del modulador sigma-delta en cascada se realiza al pulsar el botón "simular". En el "callback" correspondiente a dicho control encontramos:

```
%cargamos posibles no linealidades
```

```
load sd_nolineal;
%Simulamos variaciones en los coeficientes de los integradores
G_var=G+dt_var_g*randn(size(G))+m_var_g;
if(corregir_canc==1)%si los coeficientes de cancelacion se modifican con las variaciones
   G=G_var;%la matriz G que pasamos a cancela es igual a G_var
end
%Realizamos la modulacion sigma-delta en cascada
salida_precanc=cascada(orden,x,G_var,cuant,sat,coef_perdidas);%salida pre-cancelacion ruido
%Realizamos la cancelacion del ruido
salida_canc=cancela(salida_precanc,orden,G);%salida despues de la cancelacion de ruido
```

La salida será representada cada vez que modifiquemos un control de la interfaz, de forma que la gráfica estará siempre actualizada. Para ello encontramos en distintas partes del código una estructura de la forma:

```
if(get(handles.s_fft,'value')) %si esta marcada la FFT
      s_fft_Callback(hObject, eventdata, handles); %realiza la FFT de la salida
elseif (get(handles.s_psd,'value'))%si esta marcada la PSD
       s_psd_Callback(hObject, eventdata, handles);%realiza la PSD de la salida
elseif (get(handles.s_t,'value'))%si esta marcada la representacion temporal
       s_t_Callback(hObject, eventdata, handles);%realiza la representacion en tiempo
else
       %Fijamos por defecto la psd
       set(handles.s_t, 'Value', 0);
       set(handles.s_fft, 'Value', 0);
       set(handles.s_psd, 'Value', 1);
       s_psd_Callback(hObject, eventdata, handles);%realiza la PSD de la salida
end
```

De este modo se comprueba que es lo que está seleccionado y pasa a ser representado automáticamente.

Obsérvese que las instrucciones de representación se encuentran en los callbacks de los marcadores para la *FFT*, *PSD* y *dominio temporal (s_fft_Callback, s_psd_Callback, s_t_Callback)*. En este caso, a diferencia de lo que ocurría en la entrada (donde solo representábamos una señal) podemos mostrar más de una señal de salida en la misma gráfica. Dado que el número de señales a representar no es fijo, construimos una cadena de caracteres con la instrucción y pasamos a evaluarla posteriormente.

```
%calculamos el numero total de salidas a representar
total_plots=length(salidas_plot);
%generamos la cadena de caracteres con la instruccion para representar
s_plot=['plot(F,10*log10(salida_psd(',int2str(salidas_plot(1)),',:))'];
```

```
for(i=2:1:total_plots)
   s_plot=[s_plot,',F,10*log10(salida_psd(',int2str(salidas_plot(i)),',:))'];
end
s_plot=[s_plot,');'];
%seleccionamos los ejes apropiados
axes(handles.ejes_salida);
%evaluamos la instruccion de la cadena de caracteres
eval(s_plot);
```

Hemos mostrado el código correspondiente a la PSD. Para la FFT y el caso temporal, el código es similar.

Por lo que respecta a las figuras de mérito, para el cálculo del SNR comprobamos primero cual es la selección para determinar el ancho de banda de la señal a la salida.

- Ancho de banda fijo: En el "callback" correspondiente al selector de "Ancho de banda fijo" calculamos el SNR con dicho ancho de banda
- Caída en dB: En el "callback" correspondiente al selector de "Caida en dB" calculamos el ancho de banda para esa caida y el SNR .

```
if(get(handles.BW_fijo,'value')==1)
   BW_fijo_Callback(hObject, eventdata, handles)
else
   BW_caida_Callback(hObject, eventdata, handles)
end
```

Así distinguimos las funciones:

- BW_fijo_Callback(hObject, eventdata, handles)

```
BW=str2num(get(handles.BW_signal,'string'));%capturamos el ancho de banda introducido
BW_signal=BW*ones(1,length(orden));%el BW sera el mismo para todos los moduladores
Fmin=Fx-BW/2;%limite inferior del BW
Fmax=Fx+BW/2;%limite superior del BW
SNR=calcula_SNR(S_dB,Fx,Fs,B,Fmin,Fmax,f); %Calculamos la SNR
```

- BW_fijo_Callback(hObject, eventdata, handles)

```
caida=str2num(get(handles.dB_caida,'string'));%capturamos la caida en dB
for(i=1:1:length(orden))
```

```
    [BW_signal(i),Fmin,Fmax]=calcula_BW(S_dB(i,:),f, Fx,caida);%calculamos el BW
    SNR(i)=calcula_SNR(S_dB(i,:),Fx,Fs,B,Fmin,Fmax,f);%calculamos el SNR
end
```

Para ambos casos se pasa a "calcula_SNR" la respuesta frecuencial de la salida en decibelios. Dicha respuesta frecuencial será correspondiente a la PSD o FFT según esté seleccionada una u otra.

5.7.1 configurador_salidas

Permite seleccionar aquellas salidas en la cascada del modulador que deseemos representar gráficamente. Activado al pulsar sobre el botón "salidas" en la interfaz de salida.

Se comunica con la interfaz *GUI_sali*das a través del archivo *sd_salida_aux*, donde guarda el vector *salidas_plot* que tendrá una longitud igual al número total de salidas a representar y donde cada escalar del vector indica el numero de etapas de la cascada que comprende dicha salida.

5.7.2 Funciones invocadas

Para la realización de la simulación son invocadas una serie de funciones que invocan a su vez a otras, especialmente diseñadas para el proyecto que tratamos.

Las vemos en detalle a continuación.

5.7.2.1 cascada

Declaración

```
function salidasY=cascada(orden,x,G,cuant_levels,sat_level,perdidas)
```

Descripción

Esta función realiza una modulación en cascada sigma-delta, sin incluir la cancelación de ruido. Para un vector de entrada 'x', de longitud 'N', genera una matriz 'salidasY' 'MxN' ('M', número de etapas total de la cascada) donde cada una de las filas 'i' se corresponde con la modulación de la entrada, hasta la etapa i-ésima, en un modulador sigma-delta cascada, del tipo indicado por 'orden'.

En la cascada, solo de permiten moduladores de primer y segundo orden en serie, para garantizar la estabilidad del sistema (ver Apartado 3.3.4).

La función no limita el número de etapas de la cascada.

Parámetros de entrada:

- *orden*: Vector que determina orden de cada etapa de la cascada del modulador.
- *x*: Entrada a modular.
- *G:* Matriz de coeficientes 'g'
- *cuant_levels:* Vector con el numero de escalones de cuantización para los cuantizadores de cada etapa
- *sat_level*: Vector con el nivel de saturación para los cuantizadores de cada etapa.
- *perdidas:* coeficiente de pérdidas en los integradores de los moduladores.

Parámetros de salida:

- *salidasY:* Matriz con las salidas de cada etapa de modulación de la cascada ordenadas por filas.

Código de interés:

El algoritmo desarrollado va recorriendo un vector "orden" cuya longitud se corresponde con el numero de etapas y donde cada escalar en la posición 'i' indica el orden de modulación de la etapa i-ésima., de forma análoga a la nomenclatura empleada para los moduladores sigma delta en cascada (Ver Apartado 3.3.5). Así, el vector $orden = [2,2,1]$ se corresponde con un modulador 2-2-1.

Se realiza la modulación de primer o segundo orden de cada etapa 'i'. Se almacena la salida en la fila (i+1)-ésima y se guarda además la entrada del cuantizador 'i' en el vector "intermedio". En la etapa la etapa i+1 se cuantiza el error de cuantización de la etapa anterior ('i'). Este error se corresponde con la diferencia entre la fila (i+1)-ésima y el vector "intermedio". El proceso se repite sucesivamente hasta completar el número total de etapas.

La matriz Y guarda por filas la salida de cada modulador de la cascada. Inicializamos la primera fila a cero porque al primer modulador no habrá que restarle la salida de ninguno anterior. Al finalizar tendremos que eliminar esta fila de ceros para que cada fila 'i' se corresponda, en efecto, con la salida de la etapa i-ésima.

Según sea el orden de la etapa en cuestión mediante las funciones 'm1' y 'm2' modulamos con orden 1 ó 2 respectivamente el valor intermedio del modulador anterior (entrada al cuantizador), la salida anterior (se resta dentro del modulador a intermedio para obtener el error de cuantización de la etapa anterior) pasando a dichas funciones los coeficientes de ganancia de los amplificadores almacenados en G. Estas funciones devuelven la salida en una fila de Y y el valor intermedio para que lo emplee el siguiente modulador de la cascada. (ver apartados 5.7.2.2 y 5.7.2.3).

En la figura 5-7 vemos el diagrama de flujo apoyado en código MATLAB correspondiente al algoritmo en cuestión.

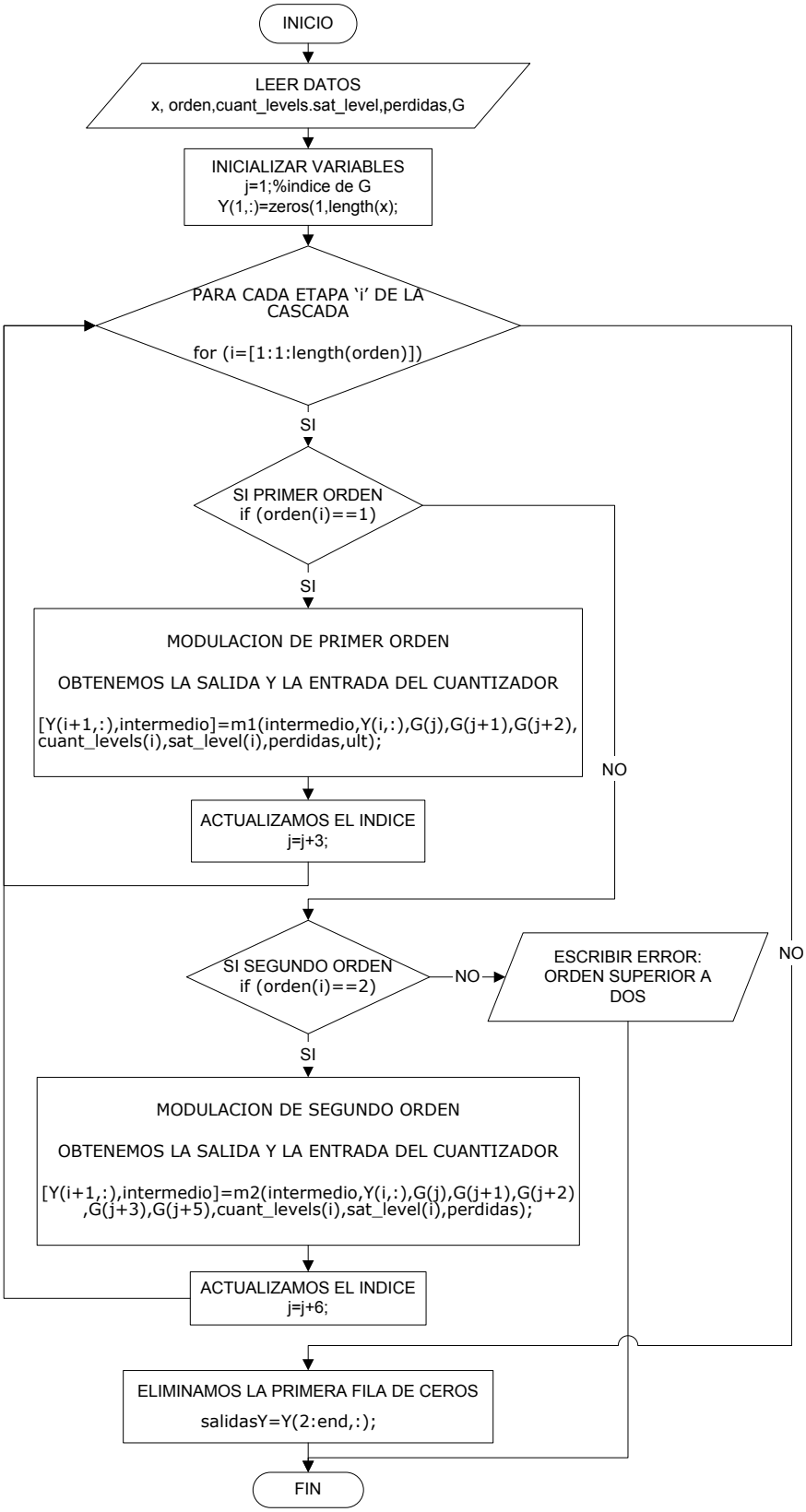


Figura 5-7 Diagrama de flujo del algoritmo incluido en la función cascada

Obsérvese que el algoritmo no limita el número de etapas de la cascada. Este aspecto es especialmente interesante dado que inicialmente se contemplaba el desarrollo de un algoritmo para cada tipo de configuración de la cascada del modulador, mientras que el algoritmo finalmente desarrollado es suficientemente genérico como para abarcar cualquier configuración en cascada de moduladores de primer y segundo orden y lo hace rehuyendo toda complejidad de código.

Por último, un aspecto importante a considerar en la actualización del índice'j' para la obtención de los coeficientes 'g' para cada integrador de los moduladores. Aquí se constata como la configuración escogida para la matriz G (ver Apartado 5.6.2) resulta de sencillo y eficaz manejo cara a las llamadas a función de modulación, paso de parámetros y actualización de índices.

5.7.2.2 m1

Declaración

```
function [modulado,intermedio]=m1(x,z,g11,g12,g1r,levels,sat,coef_perdidas);
```

Descripción

Realiza una modulación sigma-delta de primer orden sobre un vector de entrada.

Permite fijar las características del cuantizador.

Modela las pérdidas en el integrador

Devuelve la señal modulada y la entrada del cuantizador (permite calcular el error de cuantización)

Parámetros de entrada

- *x:* vector de entrada 1 a modular
- *z:* vector de entrada 2 a modular (se resta a la anterior)
- *g11:* coeficiente 'g' que multiplica a la entrada 1
- *g12:* coeficiente 'g' que multiplica a la entrada 2
- *g1r:* coeficiente 'g' que multiplica a la realimentación
- *levels:* número de escalones de cuantización del cuantizador
- *sat:* nivel de saturación del cuantizador
- *coef_perdidas*: coeficiente de pérdidas en el integrador

Parámetros de salida

- *modulado:* vector salida del modulador. Señal modulada.
- *intermedio:* vector intermedio con las entradas al cuantizador.

Código de interés:

El algoritmo de modulación modela el siguiente diagrama de bloques correspondiente a un modulador sigma-delta de primer orden (ver Apartado 3.3.2):

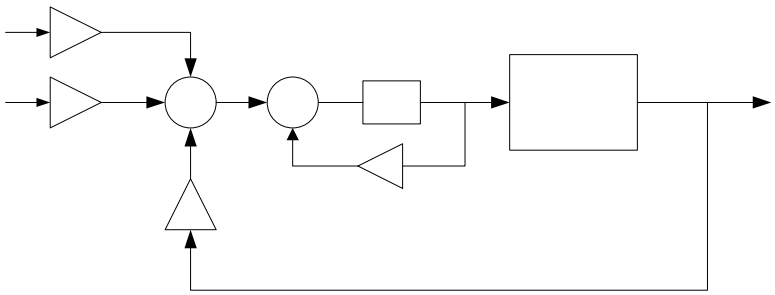


Figura 5-8 Diagrama de bloques correspondiente al algoritmo para el modulador sigma-delta de primer orden 'm1'

Donde, recuérdese que x(n) será la salida del modulador anterior en la cascada y z(n), la entrada del cuantizador del modulador anterior de la cascada (su diferencia se corresponde con el error de cuantización de la etapa anterior).

El código resultante es:

```
%inicializamos los parametros
a(1)=g11*x(1)-g12*z(1);
b(1)=0;
y(1)=0;
%modulamos
for(n=2:1:length(x))
  b(n)=a(n-1)+(1-coef_perdidas)*b(n-1);%salida del filtro
  y(n)=cuantiza(b(n),levels,2*sat);
  a(n)=g11*x(n)-g12*z(n)-g1r*y(n);%realimentacion
end
modulado=y;
intermedio=b;%este valor hace falta para el siguiente modulo en cascada
```

x(n)

g11

z(n)

g12

Donde se ha introducido el modelado de pérdidas en los integradores (3.3.6.2) y donde hemos cuantizado b(n), con "levels" niveles, y rango de cuantización igual al doble del nivel de saturación.

5.7.2.3 m2

Declaración

```
function[modulado,intermedio]=m2(x,z,g11,g12,g1r,g21,g22,levels,sat,coef_perdidas);
```

Descripción

Realiza una modulación sigma-delta de segundo orden sobre un vector de entrada.

Permite fijar las características del cuantizador.

Incluye modelado de pérdidas en los integradores.

Devuelve la señal modulada y la entrada del cuantizador (permite calcular el error de cuantización)

Parámetros de entrada

- *x:* vector de entrada 1 a modular
- *z:* vector de entrada 2 a modular (se resta a la anterior)
- *g11:* coeficiente 'g' que multiplica a la entrada 1 del primer integrador
- *g12:* coeficiente 'g' que multiplica a la entrada 2 del primer integrador
- *g1r:* coeficiente 'g' que multiplica a la realimentación del primer integrador
- *g21:* coeficiente 'g' que multiplica a la entrada 1 del segundo integrador
- *g22:* coeficiente 'g' que multiplica a la entrada 2 del segundo integrador (realimentación)
- *levels:* número de escalones de cuantización del cuantizador
- *sat:* nivel de saturación del cuantizador
- *coef_perdidas:* coeficiente de pérdidas en los integradores

Parámetros de salida

- *modulado:* vector salida del modulador. Señal modulada.
- *intermedio:* vector intermedio con las entradas al cuantizador.

Código de interés:

El algoritmo de modulación modela el siguiente diagrama de bloques correspondiente a un modulador sigma-delta de segundo orden (ver Apartado 3.3.3):

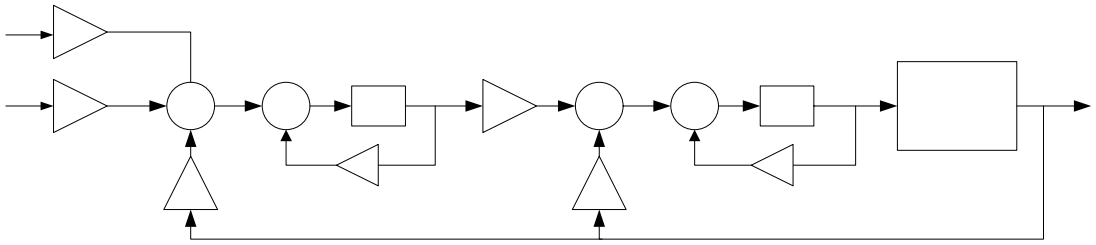


Figura 5-9 Diagrama de bloques correspondiente al algoritmo para el modulador sigma-delta de segundo orden 'm2'

donde, recuérdese que x(n) será la salida del modulador anterior en la cascada y z(n), la entrada del cuantizador del modulador anterior de la cascada (su diferencia se corresponde con el error de cuantización de la etapa anterior).

El código resultante es:

```
%inicializamos los parametros
a1(1)=g11*x(1)-g12*z(1);
a2(1)=0;
b1(1)=0;
b2(1)=0;
y(1)=0;
%modulamos
for(n=2:1:length(x))
  b2(n)=a2(n-1)+(1-coef_perdidas)*b2(n-1);%salida del filtro 2
  y(n)=cuantiza(b2(n),levels,2*sat);
  b1(n)=a1(n-1)+(1-coef_perdidas)*b1(n-1)%salida del filtro 1
  a2(n)=g21*b1(n)-g22*y(n);%realimentacion 2
  a1(n)=g11*x(n)-g12*z(n)-g1r*y(n);%realimentacion 1
end
modulado=y;
intermedio=b2;%este valor hace falta para el siguiente modulo en cascada
```

Como en el caso de 'm1' se ha introducido el modelado de pérdidas en los integradores (apartado 3.3.6.2) y cuantizamos b(n), con "levels" niveles, y rango de cuantización igual al doble del nivel de saturación.

5.7.2.4 cuantiza

Declaración

function cuantizado=cuantiza(entrada,n_levels,rg_cuant)

Descripción

Cuantiza un vector de entrada con un numero determinado de escalones de cuantización y un rango de cuantización fijado.

Esta función opera vectorialmente a excepción de la comprobación de sobrecarga del cuantizador. Para obtener un comportamiento vectorial "comentar" las líneas 23 a 35 (comprobación de saturación)

Parámetros de entrada

- *'entrada'* vector a cuantizar.
- *'n_levels'* niveles digitales posibles a la salida.
- *'rg_cuant'* rango de cuantización, Vmax-Vmin a la entrada.

Parámetros de salida

- *cuantizado* vector de valores cuantizados

Código de interés

La cuantización se realiza vectorialmente, aunque cara a la modulación sigma-delta en 'm1' y 'm2' no es necesario dado que, debido a la realimentación hemos de operar escalarmente (ver Apartados 5.7.2.2 y 5.7.2.3). Es por esto que la comprobación de sobrecarga en el cuantizador se realiza forma escalar (valor a valor).

Inicialmente se preparan dos vectores que se utilizarán con posterioridad:

```
delta=rg_cuant/n_levels; %escalon de cuantizacion
dig=[(-rg_cuant+delta)/2:delta:(rg_cuant-delta)/2];%vector de salidas digitales
```

La cuantización vectorial se resuelve en tres líneas de código:

- Inicialmente desplazamos el eje x para que los valores estén comprendidos entre 0 y rg_din:

```
x=entrada+rg_cuant/2;
```

- A continuación tomamos la parte entera de la división por el escalón de cuantización y le sumamos uno. Esto nos da el numero de la salida digital correspondiente (ordenadas de menor a mayor en *dig)*

```
index=floor(x/delta)+1;
```

- Finalmente tomamos los valores cuantizados marcados por *index*

```
cuantizado=dig(index);
```

5.7.2.5 cancela

Declaración

```
function salida=cancela(entrada,orden,G);
```

Descripción

Esta función simula el bloque de cancelación digital de ruido de cuantización en un convertidor sigma-delta en configuración cascada.

Invoca las siguientes funciones presentes en el mismo fichero "cancela.m":

- H_par: Modela un filtro de cancelación de subíndice par.
- H_impar: Modela un filtro de cancelación de subíndice impar.
- d_par: Modela un coeficiente 'd' de cancelación de subíndice par.
- d_impar: Modela un coeficiente 'd' de cancelación de subíndice impar.

(Ver apartados 3.3.5 y sub-apartados para comprobar la nomenclatura.)

Parámetros de entrada

- *entrada*
  - Matriz correspondiente a la salida del modulador sigma-delta cascada.
  - Cada fila 'i' se corresponde con la salida de la etapa í-esima de modulación .
- *orden*: vector de orden del modulador. Indica el orden de cada etapa de la cascada.
- *G:* Matriz de coeficientes 'g'. Ganancias a la entrada de los integradores.

Parámetros de salida

- *Salida*
  - Matriz de salida del modulador completo (con cancelación de ruido)
  - Cada fila 'i' se corresponde con la salida de la etapa í-esima de modulación con cancelación de ruido.

Código de interés

El algoritmo desarrollado para la cancelación de ruido en las distintas etapas de modulación sigma-delta de la cascada aprovecha la estructura reiterativa de la arquitectura de cancelación como comprobamos a continuación.

Observamos el esquema general de cancelación de ruido de las distintas etapas de la cascada (Ver apartado 3.3.5 y subapartados):

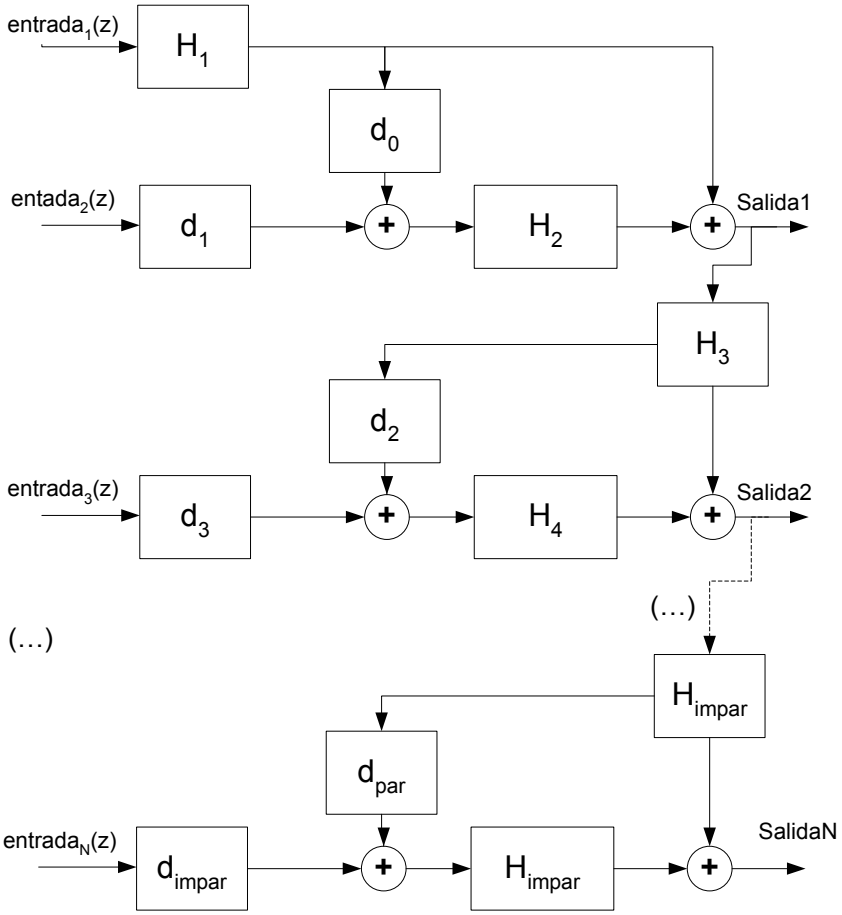


Figura 5-10 Esquema genérico de cancelación de ruido

Comprobamos que se repite el siguiente bloque fundamental:

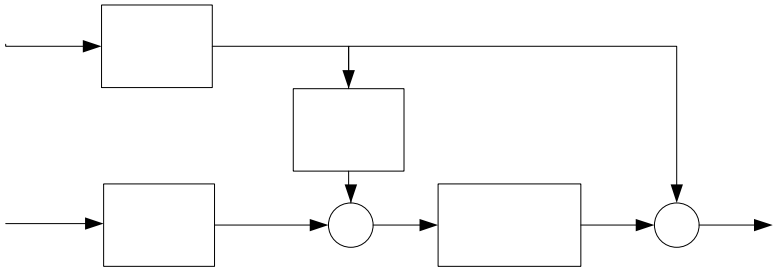


Figura 5-11 Bloque fundamental de cancelación de ruido

Pasamos ahora a estudiar la estructura de los filtros (‘*H*’) y la elección de los coeficientes digitales de cancelación (‘*d*’).

Del estudio de las ecuaciones resultantes para moduladores sigma-delta en cascada[2] , por inducción, se propone como genérica la siguiente elección de filtros y coeficientes para la cancelación de ruido a la salida de las distintas etapas de la cascada (la nomenclatura es la correspondiente al bloque fundamental de la figura 5-11).

- *H_impar*
  - Si el siguiente modulador de la cascada es de primer orden. (Etapa i+2. Recordar que la entrada es de la etapa i+1)
    - $H_{impar}(z) = z^{-1}$
  - Si el siguiente modulador de la cascada es de segundo orden. (Etapa i+2. Recordar que la entrada es de la etapa i+1)
    - $H_{impar}(z) = z^{-2}$
- *H_par*
  - Para ‘α’ igual al número de integradores hasta la etapa anterior. (Etapa i. Recordar que la entrada es la etapa i+1)
    - $H_{par}(z) = (1-z)^{\alpha}$
- *d_impar*
  - $d_{impar} = \dfrac{g_{\beta}^{''}}{g_1 g_2 \cdots g_{\beta}}$, donde
    - Si la esta etapa (i+1) es de primer orden ‘β’ es igual al numero total de integradores hasta esta etapa (incluida)
    - Si la esta etapa (i+1) es de segundo orden ‘β’ es igual al numero total de integradores menos uno hasta esta etapa (incluida)
- *d_par*
  - Para el subíndice igual a cero:
    - $d_0 = \dfrac{g_3^{'}}{g_1 g_2 g_3} - 1$
  - Para el resto de los casos:
    - $d_{par} = 0$

[2] Ver los casos para el modulador 2-2 y 2-1-1 en el Capítulo 3. Para contemplar más casos remitirse a la bibliografía.

Las funciones incluidas en cuantiza.m :*'H_par', 'H_impar', 'd_par', 'd_impar'* se limitan a modelar la casuística desarrollada para la elección y generación de los coeficiente y filtros relativos a la cancelación de ruido.

5.7.2.6 calcula_BW

Declaración:

```
function [BW,Fmin,Fmax]=calcula_BW(signal,f,Fx,caida)
```

Descripción:

Calcula en ancho de banda de señal a la salida a partir de una caída en decibelios sobre el valor de señal en Fx.

Parámetros de entrada:

- signal: señal de salida total en dB
- f: vector de frecuencias de signal
- Fx: Frecuencia central de la señal a la que se quiere calcular el BW
- caida: caida en dB bajo la frecuencia central de la señal en limites del BW

Parámetros de salida:

- BW: ancho de banda de señal
- Fmin: frecuencia inferior del BW
- Fmax: frecuencia superior del BW

5.7.2.7 calcula_SNR

Declaración:

```
function SNR=calcula_SNR(salida,Fx,Fs,B,Fmin,Fmax,f);
```

Descripción:

Calcula la SNR (Signal to Noise Ratio) de una señal a la salida en dB de un modulador.

Puede operar vectorialmente si se le dan las salidas agrupadas por filas en una matriz.

Parámetros de entrada:

- salida: Señal de salida total en dB: Ruido + Señal.
    - Puede ser una matriz con las salidas agrupadas por filas
- Fx: frecuencia central de señal
- Fs: frecuencia de muestreo de la señal
- B: ancho de banda del modulador
- Fmin: frecuencia minima de señal
- Fmax: frecuencia maxima de señal
- f: vector de frecuencias de la salida

Parámetros de salida:

- SNR: vector con los SNR de las sucesivas etapas.

Código de interés

Para calcular la SNR se emplea la fórmula:

```
%calculamos la SNR
SNR(1,i)=10*log10(S/(N1+N2));
```

Donde S, N1 y N2 han sido calculados mediante la siguiente función de integración

```
function I=integra(y,dx)
% Integracion con rectángulos y cuadrados.

n=length(y);
I = (sum(y(1:n-1))+sum(abs(y(2:n)-y(1:n-1))*1/2))*dx;
```

5.8 Menú

En las tres ventanas principales de interfaz de usuario generadas por los archivos GUI_entrada, GUI_modulador y GUI salida, encontramos un menú donde se nos ofrecen opciones de forma ordenada. Han sido agrupadas en*: Archivo, Ventanas, Herramientas y Ayuda*. Analizamos a continuación el código relativo a las mismas.

5.8.1 Archivo

Para cada una de las facilidades referenciamos los "callbacks" invocados para cada caso:

- Cargar
  - Cargar entrada
    - cargar_Callback(hObject, eventdata, handles) en GUI_entrada
  - Cargar modulador
    - m_cargar_Callback(hObject,eventdata,handles), GUI_modulador
  - Cargar simulación: Carga entrada, modulador y la salida simulada.
    - s_cargar_Callback(hObject, eventdata, handles), GUI_salida
- Guardar
  - Guardar entrada
    - guardar_Callback(hObject, eventdata, handles) en GUI_entrada
  - Guardar modulador
    - m_guardar_Callback(hObject,eventdata,handles)
  - Guardar simulación
    - s_cargar_Callback(hObject, eventdata, handles), GUI_salida
- Previsualizar impresión
  - Previsualizar entrada
    - previsualizar_Callback(hObject, eventdata, handles)

```
%abrimos la ventana de previsualizar impresion
printpreview(gcbf);
```

  - Previsualizar salida
    - s_previsualizar_Callback(hObject, eventdata, handles)
- Imprimir
  - Imprimir entrada
    - imprimir_Callback(hObject, eventdata, handles)

```
%abrimos la ventana de impresion
printdlg(gcbf);
```

  - Imprimir modulador
    - m_imprimir_Callback(hObject, eventdata, handles)
  - Imprimir salida
    - s_imprimir_Callback(hObject, eventdata, handles)
- Salir

- Salir (entrada)
  - salir_Callback(hObject, eventdata, handles)

```
%Cerramos todo abriendo una question box
selection = questdlg(['¿Quiere cerrar 'get(handles.GUI_entrada,'Name') '?'],...
                    ['Close ' get(handles.GUI_entrada,'Name') '...'],...
                    'Si','No','Si');
if strcmp(selection,'No')
   return;
end
delete(handles.GUI_entrada)
```

- Salir (modulador)
  - m_salir_Callback(hObject, eventdata, handles)
- Salir (salida)
  - s_salir_Callback(hObject, eventdata, handles)

5.8.2 Ventanas

Llama a la ventana principal seleccionada.

Según la selección, se ejecuta:

- Entrada: GUI_entrada
- Modulador: GUI_modulador
- Salida: GUI_salida

5.8.3 Herramientas

Ofrece herramientas gráficas para trabajar sobre la representación de la salida. Esta opción del menú solo está disponible en las ventanas de entrada y salida, que son las que cuentan con ejes de representación gráfica.

Despliega las siguientes opciones:

- Activar barra de herramientas.
  - activa_herramientas_Callback(hObject, eventdata, handles)

```
if(handles.flag==0)%si no hay barra activada
   fig=gcf;
   htoolbar.htoolbar = uitoolbar('Parent',fig);%hObject);
   % Generamos los botones de impresión (Print, Print Preview)
   htoolbar.hprintbtns = render_sptprintbtns(htoolbar.htoolbar);
```

```
        % Generamos los botones de edición (Edit Plot, Insert Arrow, etc)
        htoolbar.hscribebtns = render_sptscribebtns(htoolbar.htoolbar);
        % Generamos los botones de zoom
        htoolbar.hzoombtns = render_zoombtns(htoolbar.htoolbar);
        %Creamos una clase para almacenar el handles de herramientas
        handles.herramientas=htoolbar.htoolbar;
        handles.flag=1;%indica que hemos activado la barra de herramientas
        guidata(hObject,handles);
    end
```

- Desactivar barra de herramientas
    - desactiva_herramientas_Callback(hObject, eventdata, handles)

```
set (handles.herramientas,'visible','off');%desactivamos
handles.flag=0;%indica que no hay barra de herramientas activa
guidata(hObject,handles);
```

- Zoom On
    - zoomin_Callback(hObject, eventdata, handles)

```
zoom on %activamos el zoom
```

- Zoom Off
    - zoomout_Callback(hObject, eventdata, handles)

```
zoom out %anulamos el zoom
zoom off %desactivamos el zoom
```

5.8.4 Ayuda

Al seleccionar "Documentación de Ayuda", mediante el *callback*, se ejecuta "!ayuda.htm" en MATLAB de forma que se abre el documento HTML correspondiente a la documentación del proyecto con el navegador predeterminado por el usuario.

Todas las funciones desarrolladas cuentan con las primeras líneas de código comentadas con información correspondiente a la función (descripción, parámetros de entrada, de salida, funciones relacionadas, etc.). Podremos acceder a esta información desde la línea de comandos de MATLAB escribiendo "help *nombre_de_función*".

5.9 Archivos de datos ".mat"

La comunicación entre las distintas interfaces se realiza a través de archivos de datos ".mat". Estos son:

- sd_entrada.mat
- sd_arquitectura.mat

- sd_matrizG.mat
- sd nolineal.mat
- sd_salida_aux.mat
- sd_salida.mat

A estos archivos, imprescindibles para el correcto funcionamiento del programa, hemos de añadir otros generados por el usuario y que son también tipo ".mat". Son los formados para guardar entradas, configuración de moduladores y simulaciones completas, cuyo nombre es introducido por el usuario:

- Archivos generados por el usuario
    - Guardar entrada
    - Guardar modulador
    - Guardar simulación

Vemos a continuación la función desarrollada por cada uno de ellos y los datos almacenados en los mismos.

5.9.1 sd_entrada.mat

Este archivo es generado por la interfaz "generar señal de entrada" y guarda los datos relativos a la misma para que puedan ser tomados en el proceso de simulación. Comunica las interfaces GUI_entrada y GUI_simulador.

Almacena los siguientes datos:

- x : vector de entrada generado
- A: Amplitud de la señal sinusoidal de entrada.
- n_ciclos: Número de ciclos de la señal sinusoidal de entrada
- M: Tasa de sobremuestreo.
- Fx: Frecuencia del tono.
- B: Ancho de Banda.
- Fs: Frecuencia de muestreo
- kTs: Vector correspondiente al escalado del eje de tiempos en representación temporal.
- Xw_fft : Transformada de Fourier discreta de la entrada.

- eje_f_fft Vector correspondiente al escalado del eje de frecuencias para la representación gráfica de la FFT
- Xw_psd: Densidad espectral de potencia de la entrada.
- eje_f_psd: Vector correspondiente al escalado del eje de frecuencias para la representación gráfica de la PSD
- NFFT: Tamaño de los intervalos da la PSD

5.9.2 sd_arquitectura.mat

Archivo generado por la interfaz "Configurar arquitectura". Guarda los datos relativos a la misma y comunica dicha interfaz con la interfaz de salida y la correspondiente al "configurador_g".

Almacena los datos siguientes:

- orden: vector de orden. Cada entrada 'i' indica el orden del modulador de la etapa i-ésima (1 ó 2, primer o segundo orden)
- cuant: vector de niveles de cuantización. Cada entrada 'i' indica el número de escalones de cuantización del cuantizador de la etapa i-ésima de la cascada.
- sat: vector de niveles de saturación. Cada entrada 'i' indica el nivel de tensión de saturación del cuantizador de la etapa i-ésima de la cascada.
- m_reset: variable de control. Indica si se ha peseteado el modulador o si es la primera vez que se abre el configurador.

5.9.3 sd_matrizG.mat

Archivo que almacena los datos correspondientes a los coeficientes de ganancia a la entrada de los integradores de la cascada. Comunica el "configurador_g" con la salida ("GUI_salida").

Los datos contenidos en este fichero son:

- G: Matriz de coeficientes 'g'. (Ver Apartado 5.6.2)
- g_reset: Variable de control. Indica si el "configurador_g" ha sido peseteado o es la primera vez que se abre.
- slider_value: Vector de valores de los deslizadores del configurador de forma correlativa. Útil para la actualización de datos en dicho configurador.

5.9.4 sd_nolineal.mat

Archivo de datos correspondientes a las no linealidades. Se crea o edita mediante el configurador no-lineal. Es leído por GUI_entrada (efecto jitter) y GUI_salida (variación de los coeficientes y pérdidas en los integradores)

- jitter_act: variable lógica que indica si el jitter está activo
- var_g_act: variable lógica que indica si las variaciones de coeficientes activas
- perdidas_act: variable lógica que indica si pérdidas de integradores activas
- dt_jitter:desciación típica de jitter
- m_jitter: media de jitter
- dt_var_g: desviación típica de variación de coeficientes
- m_var_g: media de variación de coeficientes
- corregir_canc: variable lógica, indica si los coeficientes de cancelación incluyen o no la variación de los coeficientes de ganancia de los integradores
- coef_perdidas:coeficiente de pérdidas de los integradores

5.9.5 sd_salida_aux.mat

Comunica las interfaces “configurador salidas” y “GUI_salida”.

Almacena:

- salidas_plot: Vector de salidas a representar. Cada entrada indica una salida que ha de ser representada.
- s_reset: Variable de control. Indica si se ha peseteado la salida o si es la primera vez que se abre.

5.9.6 sd_salida.mat

Guarda los datos correspondientes a la salida de la simulación:

- salida_precanc: Matriz de salidas moduladas anterior a la cancelación de ruido. Cada fila ‘i’ se corresponde con la etapa ‘i-ésima’.
- salida_canc: Matriz de salidas moduladas posterior a la cancelación de ruido. Cada fila ‘i’ se corresponde con la etapa ‘i-ésima’

5.9.7 Archivos generados por el usuario

Archivos ".mat" generados por el usuario al guardar una entrada, modulador o simulación completa.

Almacenan toda la información necesaria para poder reconstruir en otro momento dado, la entrada, modulador o simulación mediante la opción *cargar* de los respectivos menús.

5.9.7.1 Guardar entrada

Se origina un fichero ".mat" con nombre introducido por el usuario, que contiene los siguientes datos:

- x : vector de entrada generado
- A: Amplitud de la señal sinusoidal de entrada.
- n_ciclos: Número de ciclos de la señal sinusoidal de entrada
- M: Tasa de sobremuestreo.
- Fx: Frecuencia del tono.
- B: Ancho de Banda.
- Fs: Frecuencia de muestreo
- kTs: Vector correspondiente al escalado del eje de tiempos en representación temporal.

5.9.7.2 Guardar modulador

Crea un fichero ".mat" con el nombre dado por el usuario. Datos contenidos:

- orden: vector de orden. Cada entrada 'i' indica el orden del modulador de la etapa i-ésima (1 ó 2, primer o segundo orden)
- cuant: vector de niveles de cuantización. Cada entrada 'i' indica el número de escalones de cuantización del cuantizador de la etapa i-ésima de la cascada.
- sat: vector de niveles de saturación. Cada entrada 'i' indica el nivel de tensión de saturación del cuantizador de la etapa i-ésima de la cascada.
- G: Matriz de coeficientes 'g'. (Ver Apartado 5.6.2)
- slider_value: Vector de valores de los deslizadores del configurador de forma correlativa. Útil para la actualización de datos en dicho configurador.
- jitter_act: variable lógica que indica si el jitter está activo
- var_g_act: variable lógica que indica si las variaciones de coeficientes activas
- perdidas_act: variable lógica que indica si pérdidas de integradores activas

- dt_jitter: desviación típica de jitter
- m_jitter: media de jitter
- dt_var_g: desviación típica de variación de coeficientes
- m_var_g: media de variación de coeficientes
- corregir_canc: variable lógica, indica si los coeficientes de cancelación incluyen o no la variación de los coeficientes de ganancia de los integradores
- coef_perdidas: coeficiente de pérdidas de los integradores

5.9.7.3 Guardar simulación

Guarda una simulación completa. La salida, y el modulador y entrada para los que fue dada tal salida. Los datos almacenados son:

- salida_precanc: Matriz de salidas moduladas anterior a la cancelación de ruido. Cada fila 'i' se corresponde con la etapa 'i-ésima'.
- salida_canc: Matriz de salidas moduladas posterior a la cancelación de ruido. Cada fila 'i' se corresponde con la etapa 'i-ésima'
- x : vector de entrada generado
- A: Amplitud de la señal sinusoidal de entrada.
- n_ciclos: Número de ciclos de la señal sinusoidal de entrada
- M: Tasa de sobremuestreo.
- Fx: Frecuencia del tono.
- B: Ancho de Banda.
- Fs: Frecuencia de muestreo
- kTs: Vector correspondiente al escalado del eje de tiempos en representación temporal.
- orden: vector de orden. Cada entrada 'i' indica el orden del modulador de la etapa i-ésima (1 ó 2, primer o segundo orden)
- cuant: vector de niveles de cuantización. Cada entrada 'i' indica el número de escalones de cuantización del cuantizador de la etapa i-ésima de la cascada.
- sat: vector de niveles de saturación. Cada entrada 'i' indica el nivel de tensión de saturación del cuantizador de la etapa i-ésima de la cascada.
- G: Matriz de coeficientes 'g'. (Ver Apartado 5.6.2)
- slider_value: Vector de valores de los deslizadores del configurador de forma correlativa. Útil para la actualización de datos en dicho configurador.
- jitter_act: variable lógica que indica si el jitter está activo

- var_g_act: variable lógica que indica si las variaciones de coeficientes activas
- perdidas_act: variable lógica que indica si pérdidas de integradores activas
- dt_jitter: desviación típica de jitter
- m_jitter: media de jitter
- dt_var_g: desviación típica de variación de coeficientes
- m_var_g: media de variación de coeficientes
- corregir_canc: variable lógica, indica si los coeficientes de cancelación incluyen o no la variación de los coeficientes de ganancia de los integradores
- coef_perdidas:coeficiente de pérdidas de los integradores

Capítulo 6
Evolución del proyecto. Test y pruebas.

6.1 Introducción

En este capítulo describiremos la evolución seguida durante progreso del proyecto. Veremos las pruebas y tests desarrollados a lo largo del mismo, los problemas presentados y soluciones aportadas.

Comprobaremos también el modo en que ha sido organizado el desarrollo del proyecto, sus fases o etapas.

Como hemos comentado en el capítulo de introducción se fija como finalidad del proyecto fin de carrera el desarrollo en MATLAB de un simulador de moduladores sigma-delta en cascada.

Inicialmente, se pretendía realizar tan sólo unas determinadas configuraciones hasta un total de tres etapas de modulación:

- 2-1
- 2-2
- 2-1-1
- 2-2-1
- 2-2-2

Veremos como, fruto del estudio de los moduladores y su arquitectura, finalmente se ha conseguido desarrollar una herramienta mucho más potente (no limitada en número de etapas, ni a determinadas configuraciones).

6.2 Análisis de las necesidades. Situación de partida.

Fijado el objetivo final, se realiza un balance de las necesidades y la situación de partida para acometer aquello que se pretende mejorar, resolver, ampliar.

En este punto situamos lo que es necesario para la correcta comprensión y desarrollo de aquello que es objeto del proyecto; un simulador en MATLAB de moduladores sigma-delta tiempo discreto en cascada.

Se concluye, pues, en la necesidad de profundizar en los conocimientos sobre los convertidores analógico/digital de sobremuestreo y, en particular, en los moduladores sigma-delta tiempo discreto en configuración cascada. A esta etapa de estudio y documentación hemos de añadir, fruto del análisis de las necesidades, que también se desprende la necesaria familiarización con lo que será la herramienta de simulación: MATLAB.

6.3 Fases del proyecto

Distinguimos, tal y como hemos concluido en el apartado anterior, una etapa de documentación y estudio y otra de acercamiento y dominio de MATLAB. Es evidente que para programar determinados algoritmos de simulación en MATLAB habremos de haber adquirido previamente conocimientos acerca de aquello que se quiere modelar y simular, por lo ambas etapas habrán de seguir un desarrollo en paralelo de forma sincronizada.

Finalmente, y aunque también se ha ido desarrollando conforme se iba evolucionando en las etapas anteriores, se ha realizado lo que es la redacción de la memoria, describiendo la solución aportada. En este apartado hemos de considerar la creación de un "estilo" de Word y la realización de bloques descriptivos mediante la herramienta de Microsoft, Visio.

En último lugar quedaría la presentación del proyecto fin de carrera apoyado en la herramienta Powerpoint.

Vemos a continuación las etapas seguidas en el desarrollo del proyecto en los dos aspectos fundamentales que hemos descrito:

- Documentación y estudio.
- Programación. Herramienta de diseño.

6.3.1 Documentación y estudio

Describimos las etapas pertinentes a las fases de documentación y estudio en el desarrollo del proyecto:

- Repaso de conceptos básicos de conversión analógica/digital.
    - En un primer momento, se hace indispensable un repaso de conceptos básicos de conversión analógica/digital ya que en última instancia eso es lo que queremos simular. No entramos aquí en la circutería asociada a los mismos, nos ceñimos tan sólo a bloques fundamentales y teoría de señal de los convertidores.
    - El Capítulo 2 de este documento se corresponde con esta etapa del proyecto.
- Convertidores de sobremuestreo
    - Una vez repasados los conceptos fundamentales, empezamos el estudio de un tipo de convertidores en particular; los convertidores con tasa de muestreo superior a la de Nyquist, convertidores de sobremuestreo.
    - En esta etapa se comprende los beneficios de este tipo de convertidores y sus dificultades de implementación.
- Moduladores Sigma-Delta.
    - Particularizando aún más llegamos a los moduladores sigma-delta, convertidores de sobremuestreo con conformación de ruido.
    - Profundizamos en su modo de funcionamiento y comprendemos sus ventajas y desventajas, así como sus posibles aplicaciones.
- Configuración en cascada de modulación sigma-delta.
    - Estudiamos la configuración concreta que se pretende modelar.
    - Se entiende aquí el interés de dicha arquitectura y las dificultades que entraña.
    - Este, junto a los dos apartados anteriores, se corresponden con el Capítulo 3 de la memoria.
- Obtención de ecuaciones para diferentes configuraciones en cascada
    - Ahondamos aún más en el conocimiento de los moduladores sigma-delta
    - Obtenemos las ecuaciones en el dominio ‘z’ (escogido por modelarse los integradores con un retraso simple) para moduladores de primer y segundo orden y configuraciones en cascada 2-1, 2-2, 2-1-1, 2-2-1 y 2-2-2. Todas para cuantizadores de un bit.
    - Generalizamos las ecuaciones anteriores para el caso de que el último cuantizador sea da más de un bit.
    - Obtenemos las condiciones de realización de cada arquitectura y la salida de cada una de las etapas de modulación.

  - Estos aspectos se desarrollan en los apartados 3.3.5, 5.7.2.1, 5.7.2.2 y 5.7.2.3.

- Desarrollo de bloques de cancelación de ruido
  - Para los casos estudiados en el apartado anterior, desarrollamos la arquitectura de los bloques de cancelación de ruido con las salidas de las distintas etapas de modulación.
  - Seleccionamos los filtros y coeficientes digitales oportunos.
  - A este respecto ver los apartados: 3.3.5 y 5.7.2.5.
- Desarrollo de modelos y algoritmos.
  - Este apartado se realiza en paralelo con la programación en MATLAB.
  - Del estudio de los moduladores en cascada se concluye que se puede realizar un modulador genérico, no limitado a cada configuración en particular. Esto se explica convenientemente en el apartado 5.7.2.1
  - Asimismo del estudio de los bloques de cancelación de ruido se observan ciertas pautas repetitivas de bloques fundamentales que permiten realizar un algoritmo para el modulador genérico del que hablamos. Ver apartado 5.7.2.5

6.3.2 Programación MATLAB. Herramienta de diseño.

Pasamos a describir aspectos relativos a la programación en MATLAB y su evolución a lo largo del proyecto. Se incluyen aquí la elaboración y desarrollo de algoritmos. No debe de entenderse esta parte del proyecto de forma independiente de la anterior, sino considerar ambas de forma alternada y complementaria.

- Familiarización con MATLAB
  - En un primer momento se da una toma de contacto para familiarizarse con el entorno de diseño MATLAB: su forma de trabajo, lenguaje, herramientas y potencial.
- Funciones elementales
  - Se comienza a diseñar funciones elementales para bloques fundamentales del proyecto, como son:
    - muestrea: función para muestrear una señal sinusoidal.
    - cuantiza: cuantiza un determinado vector de entrada.
    - sd_psd, sd_fft y transforma: Relativas al estudio frecuencial.
  - Las funciones anteriores se van depurando a lo largo de todo el proyecto, incluyendo nuevas salidas y entradas y un código más limpio, elegante y eficiente.
- Definición de la matriz G

- Se define como será la matriz de coeficientes de ganancias a la entrada de los integradores. Inicialmente parece conveniente la configuración:

$$G_{(3\times N)} = \begin{bmatrix} g_1 & g_2 & \cdots & g_N \\ g_1' & g_2' & \cdots & g_N' \\ g_1'' & g_2'' & \cdots & g_N'' \end{bmatrix}$$

- Pero posteriormente se comprueba que no es la configuración más oportuna para desarrollar los algoritmos de modulación. Un estudio más profundo de los moduladores $\Sigma\Delta$ revela que es más apropiado (ver Apartados 5.6.2 5.7.2) :

$$G_{(3\times N)} = \begin{bmatrix} entrada_{11}(+) & entrada_{21}(+) & \cdots & entrada_{N1}(+) \\ entrada_{12}(-) & entrada_{22}(-) & \cdots & entrada_{N2}(-) \\ feedback_1(-) & feedback_2(-) & \cdots & feedback_N(-) \end{bmatrix}$$

- Bloques de modulación de primer y segundo orden (‘m1’ y ‘m2’)
  - Se realizan las funciones ‘m1’ y ‘m2’ que simulan un modulador $\Sigma\Delta$ de primer y segundo orden respectivamente (Apartados 5.7.2.2 y 5.7.2.3)
  - Posteriormente estas funciones se van depurando para acoplarlas a otros bloques funcionales de simulación.
- Algoritmo cascada
  - Inicialmente se contempla desarrollar un algoritmo para cada configuración en cascada de modulación $\Sigma\Delta$.
  - Fruto del estudio de los moduladores $\Sigma\Delta$ en cascada, se concluye la realización de un algoritmo genérico que permite la simulación de cualquier modulador en cascada con un numero no acotado[3] de etapas de primer y segundo orden.
  - Este algoritmo se vale de las funciones elementales diseñadas anteriormente.
- Algoritmo cancela
  - Como en el caso anterior, se plantea de inicio desarrollar un algoritmo para cada bloque de cancelación de ruido correspondiente a una determinada configuración en cascada de modulación $\Sigma\Delta$.
  - Nuevamente, el estudio de los moduladores $\Sigma\Delta$ en cascada, posibilita la realización de un algoritmo genérico que permite la generación de bloques de cancelación de ruido para cualquier modulador en cascada con un numero no acotado[3] de etapas de primer y segundo orden.
- Simulaciones y depuración de código y algoritmos.

[3] Realmente, el número de etapas será finito y acotado por la memoria del programa, pero la limitación no vendrá dada por el algoritmo.

- Se realiza una batería de simulaciones que permiten depurar el código y los algoritmos desarrollados.
- Se observó que para una entrada cualquiera, por ejemplo:
  - Frecuencia del tono, F_x=1000
  - Ancho de banda, B=5000
  - Tasa de sobremuestreo, M=16
  - 10 ciclos
  - Amplitud=0.1

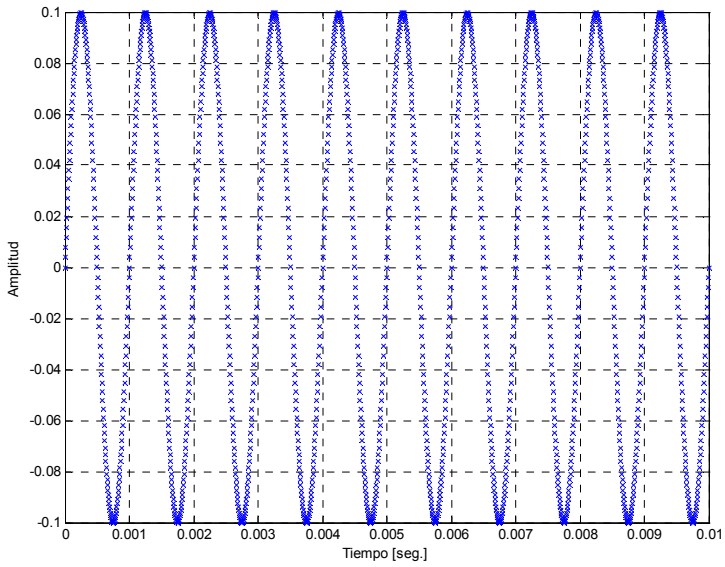


Figura 6-1 Representación de la entrada en el dominio temporal

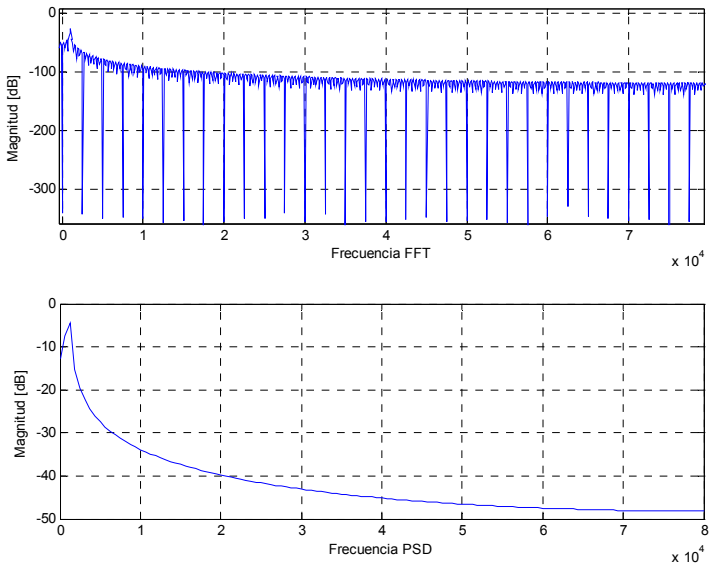


Figura 6-2 Representación de la entrada en dominio frecuencial, FFT y PSD (NFFT=256).

- Y, considerando múltiples configuraciones en cascada, se observa que la salida en todos los casos era de la forma:

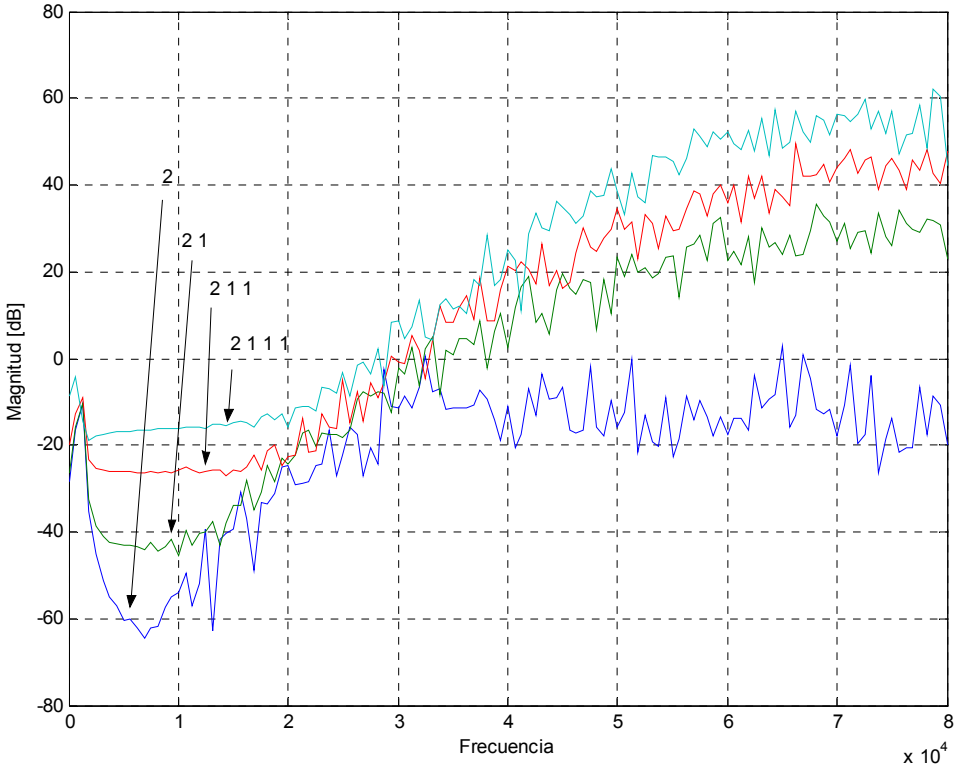


Figura 6-3 Señales de salida de las distintas etapas de la cascada sin enventanar

- Donde se ha considerado, a modo de ejemplo un modulador 2-1-1-1 con coeficientes:

Etapa 1						*Etapa 2*			*Etapa 3*			*Etapa 4*		
g_1	g_1'	g_1''	g_2	g_2'	g_2''	g_3	g_3'	g_3''	g_4	g_4'	g_4''	g_5	g_5'	g_5''
0.25	0.25	--	1	0.5	--	1	0.5	0.5	1	0.5	0.5	1	0.5	0.5

- Es decir, parece que la modulación de las etapas sucesivas de la cascada va siendo la apropiada (mayor pendiente) salvo en las proximidades de la frecuencia de interés (frecuencia de la entrada), donde paradójicamente comprobamos que incluso se empeoran los resultados, siendo mejor un modulador ΣΔ de segundo orden que un 2-1, 2-1-1, o incluso 2-1-1-1

- Necesidad de enventanado

- Recurrimos al estudio y la documentación y concluimos en la necesidad de un enventanado de la señal de salida. Para una ventana de los resultados para la misma entrada y modulador son

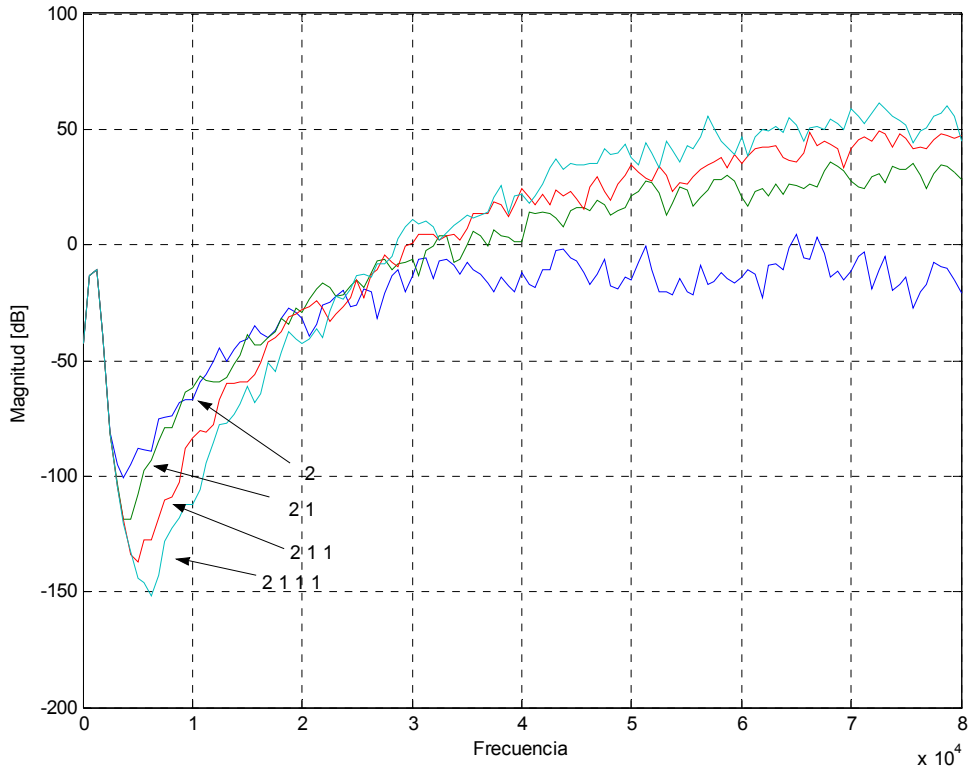


Figura 6-4 Señales de salida de las distintas etapas de la cascada con ventana de Hanning

- Observamos ahora el correcto funcionamiento del modulador y los algoritmos. En proximidades de la frecuencia de interés, con las sucesivas etapas de la cascada se obtienen mejores resultados, tal y como era de esperar.
- Por lo que respecta al dominio temporal, la figura 6-5 muestra como podrían ser la salidas de las distintas etapas de modulación en el tiempo.

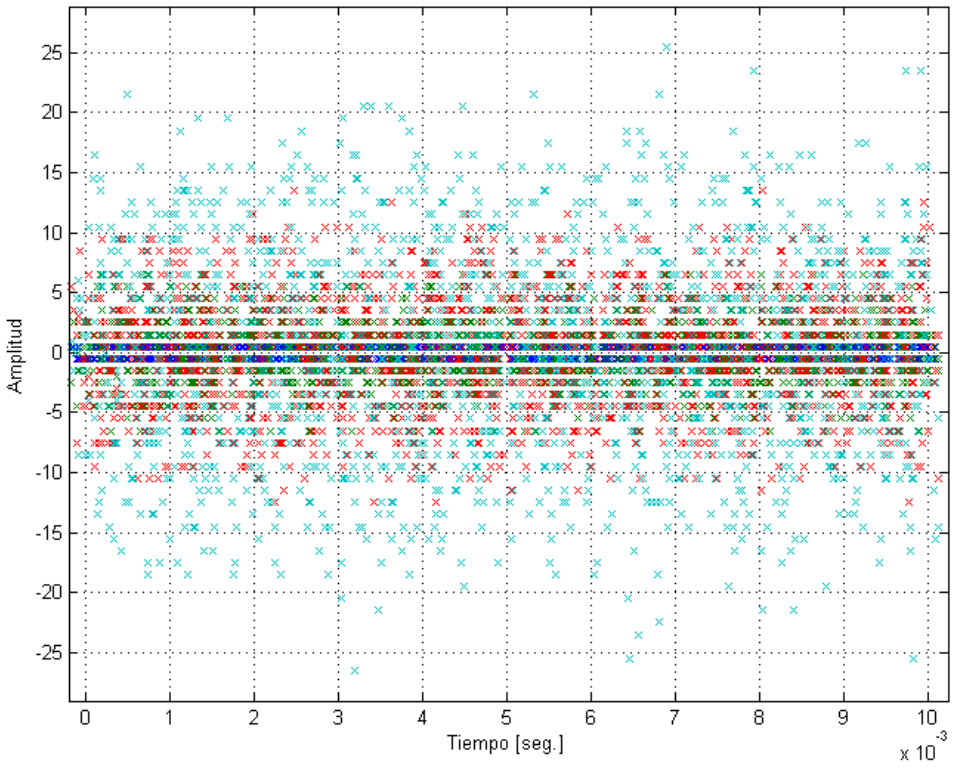


Figura 6-5 Salida de todas las etapas en el dominio temporal

- Diseño de interfaces de usuario. Herramienta GUIDE.
  - Familiarización con las interfaces de usuario de MATLAB y la herramienta GUIDE.
  - Diseño y organización de las ventanas de interfaz. (ver capítulos 4 y 5)
- Configurador_g
  - Esta herramienta ha sido desarrollada sin la ayuda de GUIDE debido al carácter no estático de su configuración: número y tipo de parámetros a mostrar, que dependen de la arquitectura previamente diseñada.
  - Se implementan asimismo las funciones correspondientes a los "callbacks" de los controles de esta interfaz (Ver 5.6.2)
- Desarrollo de menús en las interfaces.
  - Se programan nuevas opciones en las interfaces presentándolas de forma ordenada a través de menús. (Ver apartados 4.6 y 5.8)
- Figuras de mérito
  - Se desarrollan las funciones correspondientes a las figuras de mérito de los moduladores.
  - Dichas funciones son integradas en la interfaz de usuario.

- Consideremos la entrada y el modulador dados anteriormente.
- Hacemos un zoom sobre FFT en la zona de interés de la salida

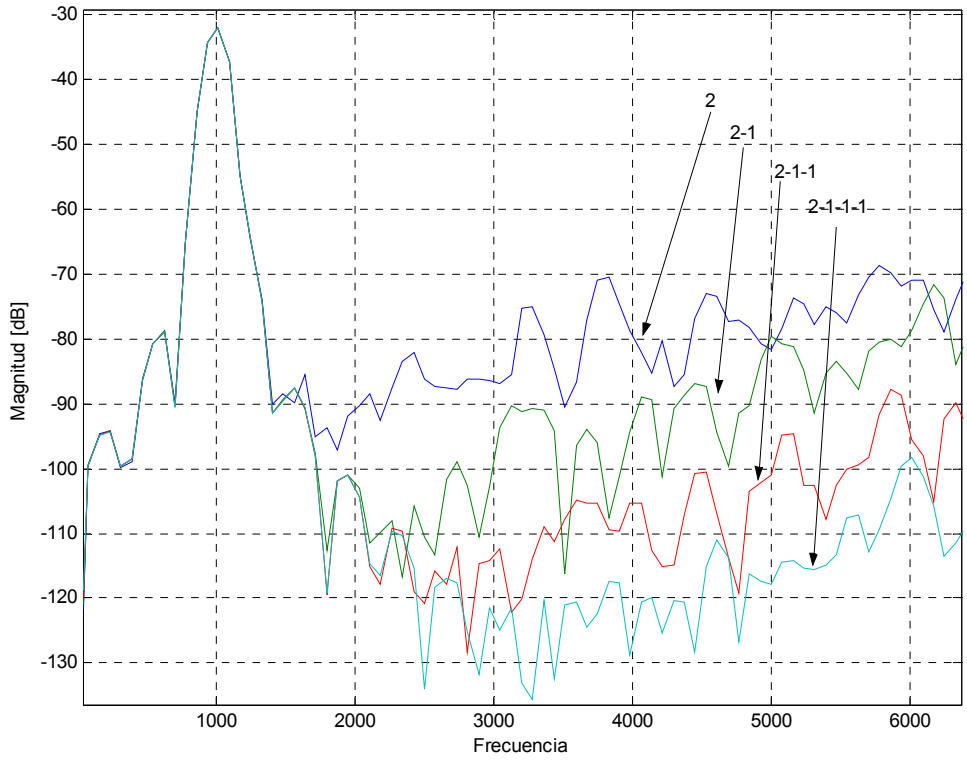


Figura 6-6 Salida en dominio frecuencial. Frecuencias de interés.

- Obtenemos las siguientes figuras de mérito fijando el ancho de banda de señal a 2000Hz en la FFT.:

Modulador	*SNR*	*BW de señal*
2	34.0534 dB	2000 Hz
2 1	48.8338 dB	2000 Hz
2 1 1	63.5952 dB	2000 Hz
2 1 1 1	70.9649 dB	2000 Hz

- No linealidades
  - Se modifican algunos aspectos de funciones de bloques básicos para integrar no linealidades al modelo del modulador.
  - Son integradas en la interfaz.
  - Observamos los efectos producidos por las no linealidades

- Teníamos la entrada de la figura 6-1 .Si introducimos el efecto Jitter con desviación típica de 0.0002), la nueva entrada será:

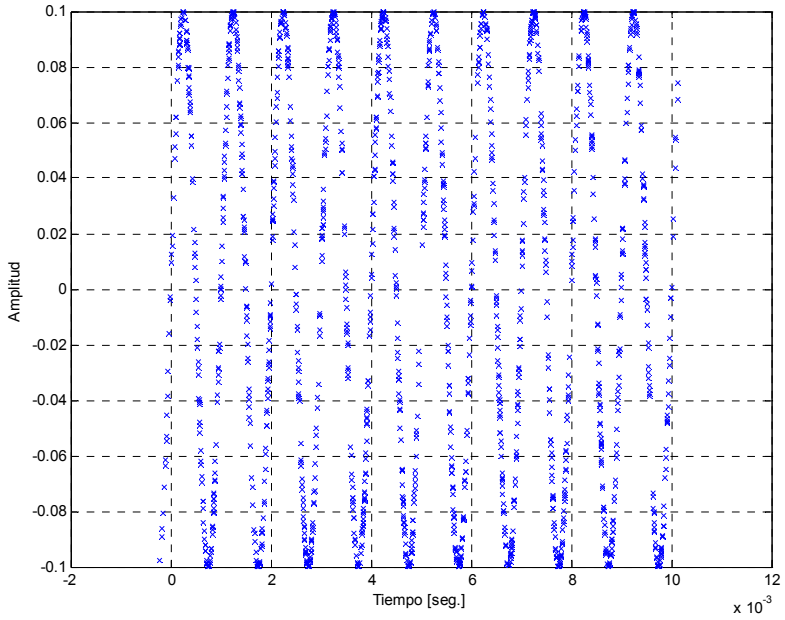


Figura 6-7 Entrada con efecto Jitter. Dominio temporal.

En frecuencia, como es lógico aparecerá más ruido en comparación a la figura 6-2

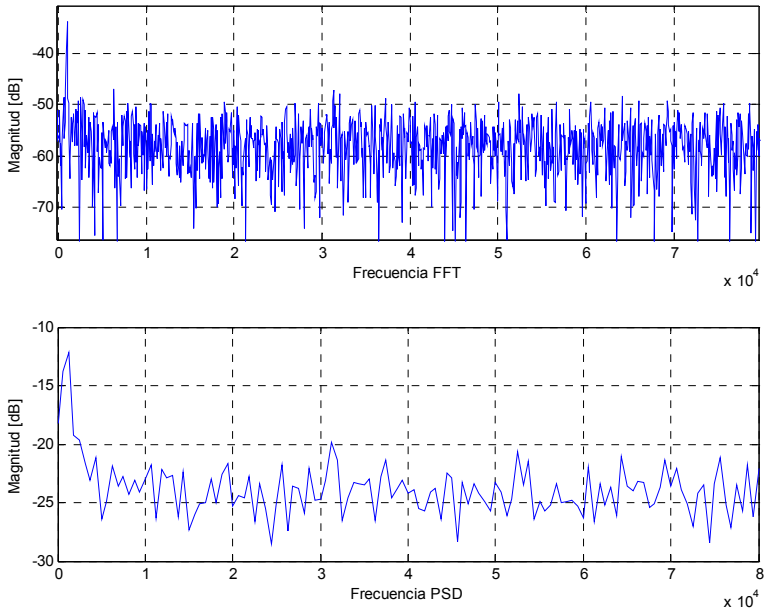


Figura 6-8 Entrada con efecto Jitter. Dominio frecuencial.

- Y a la salida tendremos

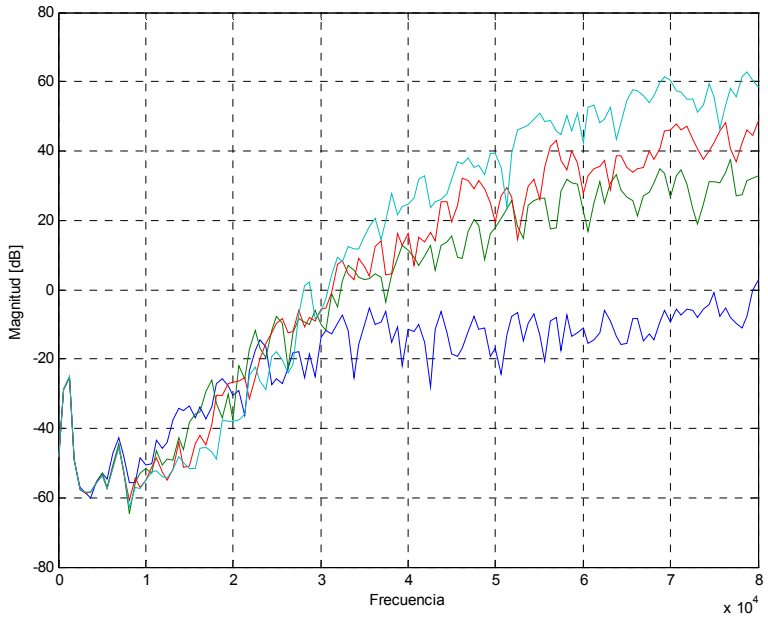


Figura 6-9 Salida con efecto Jitter.

- Si incluimos las variaciones en los coeficientes de ganancia de los integradores (desviación típica de 1%) podemos llegar a perder los beneficios de la cascada:

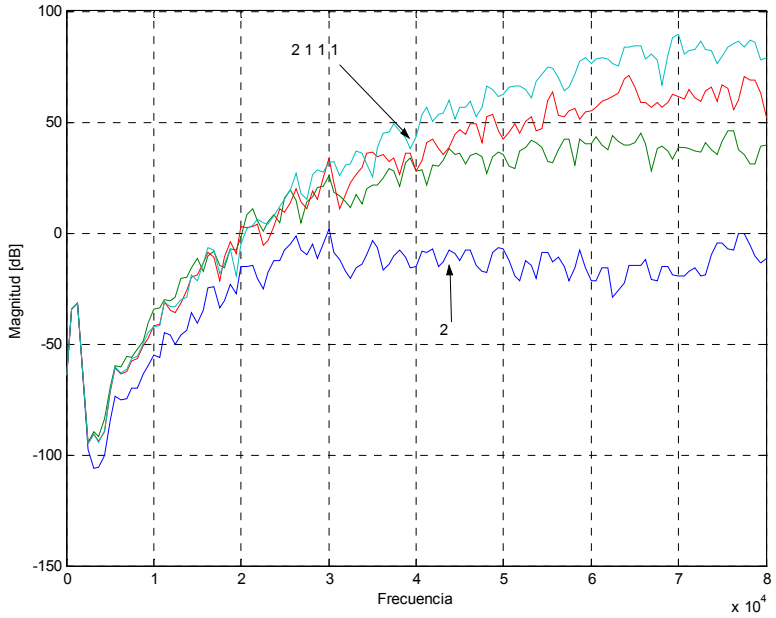


Figura 6-10 Salida con variación de coeficientes de los integradores

- Por último, las pérdidas en los integradores. Observamos como para unas pérdidas del 20% aparecen tonos en frecuencias espúreas.

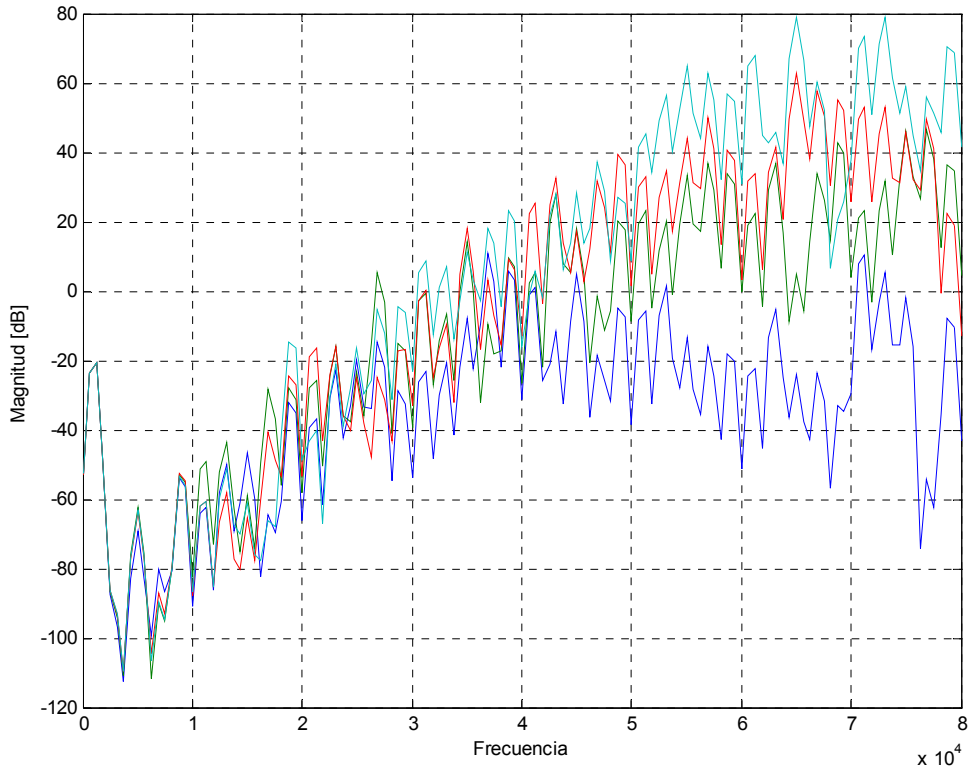


Figura 6-11 Salida con pérdidas en los integradores

- Hemos visto cada una de las no linealidades por separado. También podemos incluirlas todas a la vez. Una posible interfaz de salida para este caso sería la de la figura 6.12.

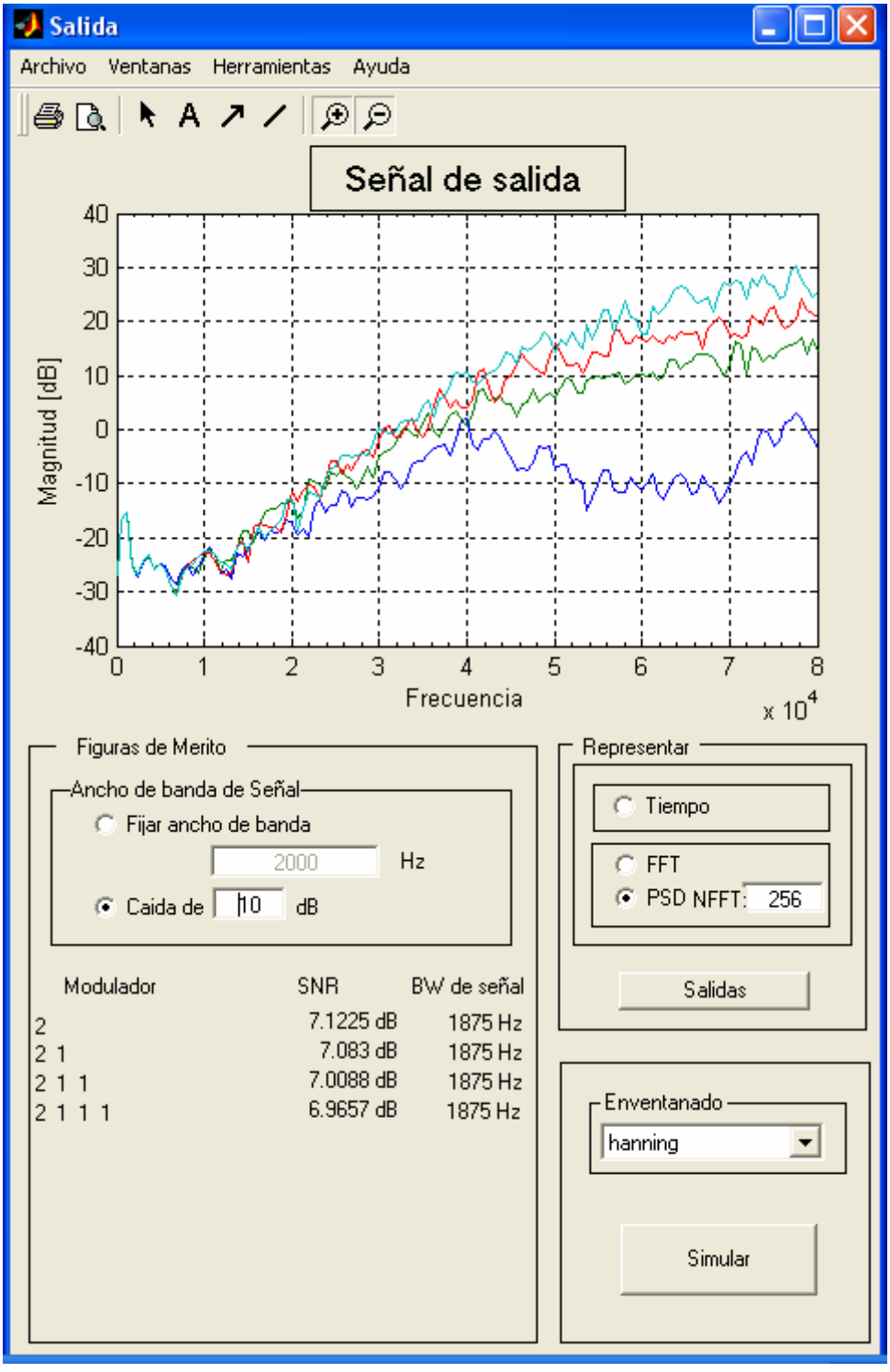


Figura 6-12 Interfaz de salida con las tres no-linealidades incluidas.

- Depuración
  - Depuración del código y algoritmos: Depuramos el código atendiendo a las últimas modificaciones introducidas.
- Test y pruebas

  - Batería de test y pruebas para comprobar posibles errores
- Depuración final.
  - Corrección de errores detectados en el apartado anterior
  - Correcciones de estilo y presentación del código.
  - Limpieza de funciones y código "auto-generado" por MATLAB que no resulta útil.
- Test y pruebas finales.
  - Comprobación final de correcto funcionamiento.

Capítulo 7
Conclusión y futuras líneas de desarrollo

7.1 Conclusión

Se ha desarrollado una herramienta de simulación de modulación sigma-delta en cascada en tiempo discreto. Este proyecto no pretende mostrar una herramienta definitiva sino establecer una base sobre la que poder construir nuevas ampliaciones y mejoras.

Se había fijado como objetivo aportar una herramienta que ayude a simular el diseño de moduladores $\Sigma\Delta$, dado que estos se encuentran en la interfaz analógica-digital y carecen de metodologías semiautomáticas para su diseño, como ocurre con los circuitos puramente digitales. Se ha pretendido satisfacer esta necesidad de simuladores de comportamiento para convertidores $\Sigma\Delta$, mediante una interfaz amigable, versátil, de sencillo e intuitivo manejo y que posibilitara a su vez futuras líneas de desarrollo.

Es por esto que se ha organizado la interfaz en ventanas por bloques diferenciados, agrupando controles relativos a tareas similares, mostrando de forma uniforme aspectos de diferente implementación de código (Ej: FFT, PSD…) y haciendo transparentes al usuario elementos complejos de la simulación (selección de filtros y coeficientes de cancelación de ruido…). La toma de datos (selectores, deslizadores, cajas de texto…) se ha facilitado en la medida de lo posible ofertando más de un modo de captura.

Se ha hecho especial incidencia en la versatilidad de modo que la organización en ventanas permite trabajar independientemente con la entrada, el modulador o la salida posibilitando, asimismo, guardar/cargar entradas, moduladores y simulaciones, mediante las opciones de menú. Estos menús, han sido organizados de forma similar a otros programas incluyendo diversas opciones de configuración de impresión, acceso a

otras ventanas, activación/desactivación de una barra de herramientas de edición grafica y acceso a documentación de ayuda.

Como ya hemos indicado en el desarrollo del proyecto, un aspecto especialmente interesante de la herramienta desarrollada es que permite simular cualquier configuración en cascada de moduladores $\Sigma\Delta$ de primer y/o segundo orden sin límite en cuanto al número de etapas[4].

Se ha tenido un especial cuidado en elaborar una herramienta flexible frente a futuras modificaciones o ampliaciones, lo que se consigue gracias a su estructura modular, con bloques básicos (cuantiza, muestrea, m1,m2…). Las funciones se han implementado, además, con vocación generalista, estos es, que puedan funcionar de forma independiente del conjunto cara a otros objetivos generales.

La herramienta desarrollada está integrada en MATLAB, lo que conlleva el aprovechamiento de multitud de herramientas que ofrece MATLAB, la posibilidad de acceder a distintas variables a lo largo del proceso y el poder reutilizar las funciones desarrolladas fuera del programa. Este aspecto potencia la flexibilidad de la que hablamos cara a la implementación de futuras líneas de desarrollo.

Por lo que respecta al modelado de los diversos bloques, se ha aprovechando la modularidad de las funciones y algoritmos desarrollados de modo que se permite configurar de manera independiente cada uno de los coeficientes de ganancia a la entrada de los integradores y fijar, también de modo independiente, las características de cada uno de los cuantizadores de cada etapa (número de escalones de cuantización y nivel de saturación).

En la medida de lo posible se ha optado por soluciones eficientes, esto es, algoritmos vectorizados cuando ha sido viable, valores por defecto potencia de 2 que agilizan algunos algoritmos y operar matricialmente (por filas) con las salidas y entradas de los distintos módulos (modulación, cancelación de ruido, cálculo de SNR…).

Se aprovecha además el carácter incremental de la arquitectura de modulación sigma-delta en cascada que hace que una determinada modulación incluya otras de orden inferior (un modulador 2-1-1-1, incluye una modulación 2, 2-1 y 2-1-1). Para ello, al realizar una simulación de una determinada arquitectura, se guardan los datos correspondientes a la salida de todas las contenidas en las etapas anteriores. De este modo, en una misma simulación se pueden presentar los resultados de cualquier etapa de modulación de la cascada y ser comparados tanto gráficamente como mediante figuras de mérito.

Por último, se han incluido algunas no-linealidades para acentuar la posible aplicación práctica de la herramienta desarrollada.

[4] En la interfaz gráfica se ha limitado el número de etapas en cascada por cuestiones de manejabilidad. No obstante, los algoritmos de simulación desarrollados no están limitados a este respecto.

7.2 Futuras líneas de desarrollo

Como hemos señalado a lo largo del proyecto, se ha incidido especialmente en la flexibilidad del código para permitir futuras ampliaciones del mismo. A continuación se proponen algunas de las posibles líneas de desarrollo que pueden surgir de este proyecto fin de carrera:

- Estudio de sensibilidad
    - Incluir herramientas que posibiliten el estudio de la sensibilidad de la salida a distintos parámetros del modulador.
    - Herramientas que vayan variando los distintos coeficientes del sistema y determinen la sensibilidad de la salida.
- Módulo de diezmado
    - Modelar el módulo de diezmado incluyendo el filtrado previo (Apartado 3.3).
- Introducir moduladores de orden superior a 2. Estudio de estabilidad.
    - De forma análoga a la que han sido desarrollados los algoritmos 'm1' y 'm2' para moduladores de primer y segundo orden (Apartados 5.7.2.2 y 5.7.2.3) desarrollar moduladores en orden superior m3, m4… que podrían ser integrados fácilmente en la función 'cascada' (Ver 5.7.2.1)
    - Incluir en estos casos estudios de estabilidad, ya que ésta no estará garantizada.
- Introducir nuevas figuras de merito.
    - Rango dinámico
    - Resolución equivalente
- Fijar una figura de merito y buscar arquitectura.
    - Habilitar herramientas que, para un determinado objetivo de diseño, simulen diversas arquitecturas y configuraciones en busca de la solución óptima, o más cercana.
- Incluir herramientas búsqueda de coeficientes de ganancia adecuados.
    - Fijar determinados coeficientes de ganancia y barrer valores de otros coeficientes.
- Incluir nuevas no-linealidades
    - Establecer un coeficiente de pérdidas en los integradores independiente para cada etapa de modulación.
    - No linealidades de cuantizadores de más de 1bit.
    - Desplazamientos en los niveles de cuantización.

  - Histéresis en cuantizadores de 1-bit, que pueden degradar el comportamiento del sistema, aumentando el fondo de ruido.
  - Ruido térmico en elementos circuitales
  - Ancho de banda finito y slew-rate
  - Saturación
- Adaptar el modulador a moduladores multitasa
  - Incluir bloques de diezmado e interpolación para bajar/subir la tasa de muestreo en determinados puntos del sistema.

Bibliografía

- James A. Cherry, W. Martin Snelgrove, **"Continuous-Time delta-sigma modulators for high-speed A/D conversion. Theory, practice and fundamental performance limits" ,** Kluwer Academic Publishers, 2000
- F. Colodro, A. Torralba, **"Multirate Sigma-Delta Modulators"**, Universidad de Sevilla.
- F. Colodro, A. Torralba, A. P. Vegeleal, L.G. Franquelo, **"Multirate-Multibit Sigma-Delta Modulators"**
- Javier García de Jalón, Jose Ignacio Rodríguez, Alfonso Brazales **"Aprenda MATLAB 5.3 como si estuviera en primero"**, Universidad Politécnica de Madrid, 2001
- David A. Johns, Ken Martin. **"Analog Integrated Circuit Design"**. John Wiley &Sons, Inc.1997
- María Dolores López Chica, **"Estudio de las no-linealidades de los moduladores sigma-delta multitasa"**
- F. Medeiro, B. Pérez- Verdú , A. Rodríguez-Vázquez, **"Top-Down design of high performance sigma-delta modulators",** Kluwer Academic Publisher, 1999
- Fernando Muñoz Chavero, **"Aportaciones al diseño de circuitos para comunicaciones de baja tensión de alimentación y consumo"**, Tesis doctoral, Universidad de Sevilla, 2002
- Alan V. Oppenheim, Ronald W. Schafer, John R. Buck. "**Tratamiento de señales en tiempo discreto".** Prentice&Hall, 2000.

- Antonio Parras Díaz, "**Diseño, realización y prototipazo de un filtro de diezmado para un convertidor A/D Sigma-Delta en la Virtex800 HQ240"**, Universidad de Sevilla, 2001
- John. G. Proakis, Dimitris G. Manolakis; traducción, Verónica Santalla del Río, Jose Luis Alba Castro. **"Tratamiento digital de señales. Principios Algoritmos y aplicaciones"**. Prentice&Hall, 2000.
- R.del Río, F. Medeiro, J. de la Rosa, B. Pérez- Verdú , A. Rodríguez-Vázquez, **"Correction-Free Multi-Bit Sigma-Delta Modulators for ADSL"**
- Luis Miguel Rivas Asensio, **"Diseño de convertidores sigma-delta cascada multibit en tiempo discreto"**, Universidad de Sevilla
- Andrés Felipe Talero Alvarado, **"Modelado y simulación de moduladores sigma-delta SC mediante lenguajes de descripción de hardware"**, Universidad de Sevilla, 2002
- The Math Works Inc. **"Creating Graphical User Interfaces",** 2000
- The Math Works Inc. **"Getting Started with MATLAB",** 2000
- The Math Works Inc. **"MATLAB User Guide",** 1997

Anexo 1: Listado de archivos

El proyecto fin de carrera comprende los siguientes archivos:

- Archivos de función y comandos de MATLAB *("M-files")*
  - acepta_G.m
  - actualiza_G.m
  - calcula_BW.m
  - calcula_SNR.m
  - cancela.m
  - cancela_G.m
  - cascada.m
  - configurador_arquitectura.m
  - configurador_g.m
  - configurador_nolineal.m
  - configurador_salidas.m
  - cuantiza.m
  - g_edit.m
  - g_slider.m
  - GUI_entrada.m
  - GUI_modulador.m

  - GUI_salida.m
  - m1.m
  - m2.m
  - muestrea.m
  - reset_G.m
  - sd_cascada.m
  - sd_fft.m
  - sd_psd.m
  - transforma.m
  - ventana.m
- Archivos de figura de MATLAB *("figure-files")*[5]
  - configurador_arquitectura.fig
  - configurador_nolineal.fig
  - configurador_salidas.fig
  - GUI_entrada.fig
  - GUI_modulador.fig
  - GUI_salida.fig
- Archivos de datos de MATLAB *("data-files")*[6]
  - sd_entrada.mat
  - sd_matrizG.mat
  - sd_nolineal.mat
  - sd_salida.mat
  - sd_salida_aux.mat
- Archivo HTML
  - ayuda.htm
- Archivos de gráficos para documentación HTML (carpeta "ayuda_archivos")

[5] Generados automáticamente mediante la herramienta GUIDE de MATLAB

[6] Estos archivos son generados automáticamente al ejecutar el programa

Anexo 2: Código MATLAB

```
function acepta_G(orden,h_slider)
% "Callback del boton "aceptar" la interfaz "configurador_g"
% Actualiza la matriz G y guarda los resultados en "sd_matrizG.mat"
% PARAMETROS DE ENTRADA:
%   orden: Determina la configuracion de la matriz de coeficientes.
%         h_slider:   Vector con los manejadores de los deslizadores de la interfaz.
%               Son necesarios para tomar el valor de coeficiente seleccionado.
%
% Invoca la función actualiza_g
% Ver tambien: actualiza_g, cancela_G.m, reset_G.m, actualiza_G.m, g_edit.m, g_slider.m

G=actualiza_g(orden,h_slider); %actualizamos la matriz G
load sd_matrizG %cargamos los datos de control almacenados en sd_matrizG
save sd_matrizG G g_reset slider_value; %salvamos la matriz y las variable de control
close(gcf); %cerramos el configurador de coeficientes 'g'
```

```
function G=actualiza_g(orden,h_slider)
%DESCRIPCION:
%   Guarda en una matriz G los valores de los coeficientes tomados de la
%   interfaz "configurador_g"
%   Los coeficientes (g,g',g") de cada integrador son almacenados en una
%   matriz G de la forma:
%
```

```
%          Integrador1             Integrador2               ...
%
%          entrada 1 (+)           entrada 1 (+)             ...
%   G(3xN)= entrada 2 (-)           entrada 2 (-)             ...
%          entrada realimentada (-) entrada realimentada (-)   ...
%
%   Ejemplo1: Para un modulador 2-1-1, G tendra la forma
%       g1  g2  g3  g4
%    G=  0   0   g3' g4'
%       g1' g2' g3" g4"
%   Ejemplo2: Para un modulador 2-2, G tendra la forma
%       g1  g2  g3  g4
%    G=  0   0   g3' 0
%       g1' g2' g3" g4'
%
% PARAMETROS DE ENTRADA:
%   orden: Determina la configuracion de la matriz de coeficientes.
%         h_slider:   Vector con los manejadores de los deslizadores de la interfaz.
%              Son necesarios para tomar el valor de coeficiente seleccionado.
%
% PARAMETROS DE SALIDA
%   G: matriz de coeficientes
%
%Ver: acepta_G.m, cancela_G.m, reset_G.m, configurador_g.m, g_edit.m, g_slider.m

%capturamos el orden del primer modulador de la cascada
primer_mod=orden(1);
%calculamos el numero total de integradores de la cascada
num_integradores=sum(orden);
%Obtenemos la matriz G, reservamos memoria rellenandola de unos
G=ones(3,num_integradores);

%capturamos los coeficientes del primer modulador de la cascada
%este es un caso especial: los ingradores tendran siempre solo 2 entradas
for (i=1:1:primer_mod)
   G(1,i)=get(h_slider(2*i-1),'value'); %coeficiente g
   G(2,i)=0; %cero, no existe este coeficiente
   G(3,i)=get(h_slider(2*i),'value'); %coeficiente g', entrada realimentada
end

%A partir de la segunda etapa de la cascada, todos los casos son iguales
k=orden(1)+1; %segunda coordenada de G. Indica el numero de integrador
j=2*orden(1)+1; %indice para desplazarnos por el vector de handles (manejadores)
```

```
for(i=2:1:length(orden))
   G(1,k)=get(h_slider(j),'value');%coeficiente g
   G(2,k)=get(h_slider(j+1),'value');%coeficiente g'
   G(3,k)=get(h_slider(j+2),'value');%coeficiente g", entrada realimentada
   k=k+1;
   j=j+3;
   if(orden(i)==2)%si esl orden del modulador de la etapa 'i' es 2, añadimos...
      G(1,k)=get(h_slider(j),'value');%coeficiente g
      G(2,k)=0;%cero, no existe este coeficiente
      G(3,k)=get(h_slider(j+1),'value');%coeficiente g', entrada realimentada
      k=k+1;
      j=j+2;
   end
end

%guardamos en el vector "slider_value" los valores de todos los deslizadores
for(i=1:1:length(h_slider))
  slider_value(i)=get(h_slider(i),'value');
end

g_reset=0; %variable de control de reset a cero
save sd_matrizG g_reset slider_value; %salvamos la variable de control y los valores de todos los deslizadores
```

```
function [BW,Fmin,Fmax]=calcula_BW(signal,f,Fx,caida)
%Calcula el ancho de banda de caida en dB
%PARAMETROS DE ENTRADA:
%   signal:señal de salida total en dB
%   f: vector de frecuencias de signal
%   Fx: Frecuencia centrl de la señal a la que se quiere calcular el BW
%   caida: caida en dB bajo la frec central de la señal en limites del BW
%
%PARAMETROS DE SALIDA
%   BW: ancho de banda de señal
%   Fmin: frecuencia inferior del BW
%   Fmax: frecuencia superior del BW
%
% Ver tambien: calcula_SNR

%calculamos el indice en f de Fx

if(f(1)>=Fx)
```

```
   iFx=1 ;
else
   %cogemos la frecuencia mas cercana a Fx
   ind=find(f<Fx);
   if((Fx-f(ind(end)))<(f(ind(end)+1))-Fx)
      iFx=length(ind);
   else
      iFx=length(ind)+1;
   end
end


s_max=signal(iFx);%valor maximo de señal


%calculamos el indice en f de Fmax


iFmax=iFx;%partimos del incide de Fx
while((signal(iFmax)>(s_max-caida))&(iFmax<length(signal)))
   iFmax=iFmax+1; %aumentamos mientras tengamos frecuencias superiores y no superemos la caida en dB
end


%calculamos el indice en f para Fmin


iFmin=iFx;%partimos del incide de Fx
while((signal(iFmin)>(s_max-caida))&(iFmin>1)&(f(iFmin)>0))
   iFmin=iFmin-1;%disminuimos mientras tengamos frecuencias inferiores y no superemos la caida en dB
end
if(f(iFmin)<0)
   iFmin=iFmin+1;
end


%asignamos los valores a las salidas
Fmax=f(iFmax);
Fmin=f(iFmin);
BW=Fmax-Fmin;
```

```
function SNR=calcula_SNR(salida,Fx,Fs,B,Fmin,Fmax,f);
%Calcula la SNR (Signal to Noise Ratio) de una señal a la salida en dB de un modulador
%Puede operar vectorialmente si se le dan las salidas agrupadas por filas en
%una matriz
%PARAMETROS DE ENTRADA
%   salida: Señal de salida total en dB: Ruido + Señal. Salidas agrupadas
```

```
%   por filas en una matriz
%   Fx: frecuencia central de señal
%   Fs: frecuencia de muestreo de la señal
%   B: ancho de banda del modulador
%   Fmin: frecuencia minima de señal
%   Fmax: frecuencia maxima de señal
%   f: vector de frecuencias de la salida
%PARAMETROS DE SALIDA
%   SNR: vector con los SNR de las sucesivas etapas.
%
%OBSERVACION:
%   Incluye la funcion 'integra'
%
% Ver tambien calcula_SNR


%calculamos la SNR para cada fila de salida
for(i=1:1:size(salida,1))
   ydB=salida(i,:);%tomamos la salida i-esima
   y=10.^(ydB/20);%pasamos a naturales
   y=y.^2;%elevamos al cuadrado para calcular potencia
   %buscamos el indice de la frecuencia inferior de señal
   if(Fmin>=f(end))
      imin=length(f);
   else
      ind=find(f>Fmin);
      if(ind(1)>1)
         if((f(ind(1))-Fmin)>Fmin-f(ind(1)-1))
            imin=ind(1)-1;
         else
           imin=ind(1);
         end
      else
         imin=1;
      end
   end

   %calculamos el indice de la frecuencia superior de señal
   ind=find(f<Fmax);
   if((Fmax-f(ind(end)))>(f(ind(end)+1)-Fmax))
      imax=ind(end)+1;
   else
      imax=ind(end);
```

```
    end


    % el paso en frecuencia es:
    df=f(imin+1)-f(imin);
    %Potencia de señal
    S=integra(y(imin:imax),df);

    % el primer intervalo de ruido va desde 0 hasta fmin
    % N1: potencia del primer intervalo de ruido
    fminr=0;
    ind=find(f>fminr);
    iminr=ind(1);
    if(imin<=iminr)
       N1=0;
    else
       N1=integra(y(iminr:imin),df);
    end

    % el segundo intervalo de ruido va desde fmax hasta B
    % N2: Potencia del segundo intervalo de ruido
    ind=find(f<B);
    imaxr=ind(length(ind));
    N2=integra(y(imax:imaxr),df);


    %calculamos la SNR
    SNR(1,i)=10*log10(S/(N1+N2));

end

function I=integra(y,dx)
% Integracion con rectángulos y cuadrados.

n=length(y);
I = (sum(y(1:n-1))+sum(abs(y(2:n)-y(1:n-1))*1/2))*dx;

%I=0;
%for i=1:n-1
%   I=I+y(i)*dx+dx/2*abs(y(i)-y(i+1));
%end
```

```
function salida=cancela(entrada,orden,G);
% Esta función simula el bloque de cancelación digital de ruido de cuantización en un convertidor sigma-delta en configuración cascada.
% Invoca las siguientes funciones presentes en el mismo fichero "cancela.m":
%       H_par: Modela un filtro de cancelación de subíndice par.
%       H_impar: Modela un filtro de cancelación de subíndice impar.
%       d_par: Modela un coeficiente 'd' de cancelación de subíndice par.
%       d_impar: Modela un coeficiente 'd' de cancelación de subíndice impar.
%  (Ver la documentacion para comprobar nomenclatura)
%
%PARAMETROS DE ENTRADA
%       entrada
%       Matriz correspondiente a la salida del modulador sigma-delta cascada.
%       Cada fila 'i' se corresponde con la salida de la etapa í-esima de modulación .
%       orden: vector de orden del modulador. Indica el orden de cada etapa de la cascada.
%       G: Matriz de coeficientes 'g'. Ganancias a la entrada de los integradores.
%
%PARAMETROS DE SALIDA
%       Salida
%       Matriz de salida del modulador completo (con cancelación de ruido)
%       Cada fila 'i' se corresponde con la salida de la etapa í-esima de modulación  con cancelación de ruido.
%
% Ver tambien configurador_g, cascada, m1, m2, cuantiza


Y=entrada;
%recorremos cada salida del modulador y anulamos el reudo de cuantizacion
for(i=1:1:length(orden)-1)
   aux1=H_impar(Y(i,:),orden,i); %aux1 es la salida del primer filtro H_impar
   aux2=d_impar(orden,i,G)*Y(i+1,:)+d_par(orden,i,G)*aux1; %aux2 es la entrada al segundo filtro H_par
   Y(i+1,:)=H_par(aux2,orden,i)+aux1;%la salida del filtro H_par es la correspondiente a la etapa i+1 por lo que la guardamos en dicha fila en una matriz que contendra todas las salidas por filas Y
end
salida=Y; %la salida sera una matriz que almacene por filas las correspondientes salidas


function b=H_impar(a,orden,i) %filto de subindice impar
tam=length(a); %el tamaño del vector de entrada
b=zeros(1,tam);%Inicializamos a cero los vectores intermedios que vamos a utilizar
if(orden(i+1)==1)% si el orden del siguiente modulador en la cascada es uno...
   b(2:tam)=a(1:tam-1);% entonces el filtro es 1/z que equivale a un retraso
else if(orden(i+1)==2) % si el orden del siguiente modulador en la cascada es dos...
      b(3:tam)=a(1:tam-2);% entonces el filtro es (1/z)*(1/z) que equivale a un retraso doble
```

```
    else disp('Error: Modulador de orden superior a 2 en serie'); %si no ERROR
    end
end



function b=H_par(a,orden,i) %filto de subindice par
orden_f=sum(orden(1:i)); %el orden del filtro es la suma de los moduladores hasta la etapa i. Recordar que estamos en la i+1.
filtro=poly(ones(1,orden_f)); %calculamos el polinomio caracteristico del filtro pasando un vectro con tantos unos como orden sea el filtro. El filto sera de la forma H=(1-z^(-1))^orden_f
tam=length(a);
suma=zeros(1,tam);%Inicializamos a cero los vectores intermedios que vamos a utilizar
for(j=1:1:length(filtro)) %recorremos el vector con los coeficientes del filtro
    aux=zeros(1,tam); %vector auxiliar inicializado a ceros
    aux(j:tam)=a(1:tam-j+1); %retrasamos tantas posiciones como posicion en el vector de coeficientes del filtro
    aux=filtro(j)*aux; %multiplicamos por el correspondiente coeficiente para ese retraso
    suma=suma+aux; %sumamos lo obtenido a lo calculado anteriormente
end
b=suma; %la salida sera la suma total



function d=d_par(orden,i,G) %coeficiente 'd' de subindice par
if (i==1)%d0
    d=G(2,3)/(G(1,1)*G(1,2)*G(1,3))-1; %d=g3'/(g1*g2*g3) - 1
else%d2,d4,d6...
    d=0;
end



function d=d_impar(orden,i,G) %coeficiente 'd' de subindice par
k=sum(orden(1:i))+1;%k es el subindice max que hemos de considerar
prod=1;
for(j=1:1:k)
    prod=prod*G(1,j); %multiplicamos g1*g2*g3*...*gk
end
d=G(3,k)/prod; % d=gk''/(g1*g2*g3*...*gk)
```

```
function cancela_G
% "Callback del boton "cancelar" la interfaz "configurador_g"
% Cierra el configurador de coeficientes 'g' sin hacer nada mas.
%
% % Ver tambien: actualiza_g, acepta_G.m, reset_G.m, actualiza_G.m, g_edit.m, g_slider.m
```

```
close(gcf);
```

```
function salidasY=cascada(orden,x,G,cuant_levels,sat_level,perdidas)
% Esta función realiza una modulación en cascada sigma-delta, sin incluir la cancelación de ruido.
% Para un vector de entrada 'x', de longitud 'N', genera una matriz 'salidasY' 'MxN' ('M', número de etapas total de la cascada)
% Cada una de las filas 'i' se corresponde con la modulación de la entrada,
% hasta la etapa i-ésima, en un  modulado sigma-delta cascada
%
% En la cascada, solo de permiten moduladores de primer y segundo orden en serie, para garantizar la estabilidad del sistema (ver Apartado 3.3.4).
%
% La función no limita el número de etapas de la cascada.
%
% PARAMETROS DE ENTRADA
%         orden: Vector que determina orden de cada etapa de la cascada del modulador.
%         x: Entrada a modular.
%         G: Matriz de coeficientes 'g'
%         cuant_levels: Vector con el numero de escalones de cuantización para los cuantizadores de cada etapa
%         sat_level: Vector con el nivel de saturación para los cuantizadores de cada etapa.
%    perdidas: coeficiente de perdidas en integradores. Comprendido entre 0 y 1
%
% PARAMETROS DE SALIDA
%         salidasY: Matriz con las salidas de cada etapa de modulación de la cascada ordenadas por filas.
%
% Ver tambien configurador_g, m1, m2, cuantiza, cancela



%si solo se especifican 3 argumentos de entrada
if (nargin==3)
    cuant_levels=2*ones(1,length(orden)); % si no se especifica como entrada, todos los cuantizadores son de dos niveles
    sat_level=ones(1,length(orden)); % si no se especifica como entrada, todos los cuantizadores saturan en 1V
    perdidas=0;%si no se especifica como entrada, no habra perdidas
end


intermedio=x;%la entrada de los moduladores sera intermedio (valor intermedio del modulador anterior). Inicializamos a x para el primer modulador
Y(1,:)=zeros(1,length(x));%la matriz Y guarda por filas cada salida de cada modulador de la cascada. Inicializamos a cero porque al primer modulador no habra que restarle la salida de ninguno anterior
j=1;%inicializamos el indice de la matriz de coeficientes G. Una columna para cada integrador.(columnas de tres filas ya que cada integrador como max tendra 3 coeficientes)
```

```
for i=[1:1:length(orden)];%por cada modulador serie de la cascada...
   if (orden(i)==1)%si modulador de primer orden

[Y(i+1,:),intermedio]=m1(intermedio,Y(i,:),G(j),G(j+1),G(j+2),cuant_levels(i),sat_level(i),perdidas);% modulamos con orden1 el valor intermedio del mod anterior, la salida anterior (se resta dentro del mod) e intoducimos coeficientes de G. Devuelve en una fila de Y la salida y el valor intermedio para qu lo emplee el siguiente mod de la cascada
      j=j+3;%actualizamos el indice de G
   else
      if (orden(i)==2)%si modulador de segundo orden

[Y(i+1,:),intermedio]=m2(intermedio,Y(i,:),G(j),G(j+1),G(j+2),G(j+3),G(j+5),cuant_levels(i),sat_level(i),perdidas);%modulamos con orden 2 el valor intermedio del mod anterior, la salida enterior (se resta dentro del mod) e intoducimos coeficientes de G. Devuelve en una fila de Y la salida y el valor intermedio para qu lo emplee el siguiente mod de la cascada
         j=j+6;%actualizamos el indice de G
      else %si el modulador serie no es de orden 1 o 2 entonces...
         disp('Error: Modulador de orden superior a 2 en serie');
      end
   end
end
salidasY=Y(2:end,:);%extraemos la primera fila de ceros para que cada fila se corresponda con las salida correspondiente (Y1 con fila1...);
```

```
function varargout = configurador_arquitectura(varargin)
% CONFIGURADOR_ARQUITECTURA M-file for configurador_arquitectura.fig
%      CONFIGURADOR_ARQUITECTURA, by itself, creates a new CONFIGURADOR_ARQUITECTURA or raises the existing
%      singleton*.
%
%      H = CONFIGURADOR_ARQUITECTURA returns the handle to a new CONFIGURADOR_ARQUITECTURA or the handle to
%      the existing singleton*.
%
%      CONFIGURADOR_ARQUITECTURA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONFIGURADOR_ARQUITECTURA.M with the given input arguments.
%
%      CONFIGURADOR_ARQUITECTURA('Property','Value',...) creates a new CONFIGURADOR_ARQUITECTURA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before configurador_arquitectura_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to configurador_arquitectura_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help configurador_arquitectura

% Last Modified by GUIDE v2.5 10-Nov-2003 10:35:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @configurador_arquitectura_OpeningFcn, ...
                   'gui_OutputFcn',  @configurador_arquitectura_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before configurador_arquitectura is made visible.
function configurador_arquitectura_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to configurador_arquitectura (see VARARGIN)

% Choose default command line output for configurador_arquitectura
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes configurador_arquitectura wait for user response (see UIRESUME)
% uiwait(handles.configurador_arquitectura);
```

```
%Creamos una clase en la estructura para almacenar el orden
handles.orden=0;
guidata(hObject,handles);

%Creamos una clase en la estructura para ver si ha cambiado la arquitectura
handles.cambio=0;
guidata(hObject,handles);

%Inicializamos a cero los marcadores de orden
for(i=1:1:10)
   clear_orden=['set(handles.mod',int2str(i),'_1, ''Value'', 0);'];
   eval(clear_orden); %Inicializamos a cero los marcadores de primer orden
   clear_orden=['set(handles.mod',int2str(i),'_2, ''Value'', 0);'];
   eval(clear_orden); %Inicializamos a cero los marcadores de segundo orden
end

%desactivamos todos los marcadores menos el correspondiente al primer modulador
for(i=2:1:10)
   desactiva_mod_1=['set(handles.mod',int2str(i),'_1',',''Enable'',''off'');'];
   eval (desactiva_mod_1); %desactiva marcador de primer orden
   desactiva_mod_2=['set(handles.mod',int2str(i),'_2',',''Enable'',''off'');'];
   eval (desactiva_mod_2); %desactiva marcador de segundo orden
end

%desactivamos todos las casillas de niveles de cuantizacion
for(i=1:1:10)
   desactiva_cuant_nivel=['set(handles.cuant',int2str(i),',''Enable'',''off'');'];
   eval (desactiva_cuant_nivel);
end

%desactivamos todos las casillas de nivel de saturacion del cuantizador
for(i=1:1:10)
   desactiva_cuant_sat=['set(handles.sat',int2str(i),',''Enable'',''off'');'];
   eval (desactiva_cuant_sat);
end

%Cargamos los posibles valores de una arquitectura previamente configurada
load sd_arquitectura;

if(m_reset==0) %si no es la primera vez que se abre la ventana
   %activamos el orden de los moduladores correspondientes y el siguiente al ultimo
   for(i=1:1:length(orden)+1)
      if(i<11)%si no estamos en el caso limite de 10 etapas
```

```
            activa_mod_1=['set(handles.mod',int2str(i),'_1',','Enable'','on'');'];
            eval (activa_mod_1);
            activa_mod_2=['set(handles.mod',int2str(i),'_2',','Enable'','on'');'];
            eval (activa_mod_2);
        end
    end

    for(i=1:1:length(orden))
        %activamos los niveles de los cuantizadores y su saturacion
        activa_cuant_nivel=['set(handles.cuant',int2str(i),','Enable'','on'');'];
        eval (activa_cuant_nivel);
        activa_cuant_sat=['set(handles.sat',int2str(i),','Enable'','on'');'];
        eval (activa_cuant_sat);
        %fijamos el orden de cada etapa en los marcadores
        if(orden(i)==1) %si primer orden
            %fijamos el marcador correspondiente
            set_orden=['set(handles.mod',int2str(i),'_1, ''Value'', 1);'];
            eval(set_orden);
            %guardamos el orden en la estructura
            orden_struct=['handles.orden(',int2str(i),')=1;'];
            eval(orden_struct);
            guidata(hObject,handles);
        elseif(orden(i)==2)%si segundo orden
            %fijamos el marcador correspondiente
            set_orden=['set(handles.mod',int2str(i),'_2, ''Value'', 1);'];
            eval(set_orden);
            %guardamos el orden en la estructura
            orden_struct=['handles.orden(',int2str(i),')=2;'];
            eval(orden_struct);
            guidata(hObject,handles);
        end
        %fijamos el numero de escalones y niveles de saturacion de cada etapa
        set_cuant_nivel=['set(handles.cuant',int2str(i),','string'',',int2str(cuant(i)),');'];
        eval (set_cuant_nivel);
        %fijamos el nivel de saturacion del cuantizador de cada etapa
        set_cuant_sat=['set(handles.sat',int2str(i),','string'',',num2str(sat(i)),');'];
        eval (set_cuant_sat);
    end
    %Mostramos el modulador mediante texto no editable
    set(handles.texto_orden,'String',int2str(handles.orden));

end
```

```
% --- Outputs from this function are returned to the command line.
function varargout = configurador_arquitectura_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in mod3_1.
function mod3_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod3_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod3_1

handles.orden(3)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod3_1, 'Value', 1);
set(handles.mod3_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod4_1,'Enable','on');
set(handles.mod4_2,'Enable','on');
set(handles.cuant3,'Enable','on');
set(handles.sat3,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod3_2.
function mod3_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod3_2
```

```
handles.orden(3)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod3_2, 'Value', 1);
set(handles.mod3_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod4_1,'Enable','on');
set(handles.mod4_2,'Enable','on');
set(handles.cuant3,'Enable','on');
set(handles.sat3,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant3_Callback(hObject, eventdata, handles)
% hObject    handle to cuant3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant3 as text
%        str2double(get(hObject,'String')) returns contents of cuant3 as a double


% --- Executes during object creation, after setting all properties.
```

```
function sat3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat3_Callback(hObject, eventdata, handles)
% hObject    handle to sat3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat3 as text
%        str2double(get(hObject,'String')) returns contents of sat3 as a double


% --- Executes on button press in mod4_1.
function mod4_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod4_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod4_1

handles.orden(4)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod4_1, 'Value', 1);
set(handles.mod4_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod5_1,'Enable','on');
set(handles.mod5_2,'Enable','on');
set(handles.cuant4,'Enable','on');
set(handles.sat4,'Enable','on');
```

```
%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod4_2.
function mod4_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod4_2

handles.orden(4)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod4_2, 'Value', 1);
set(handles.mod4_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod5_1,'Enable','on');
set(handles.mod5_2,'Enable','on');
set(handles.cuant4,'Enable','on');
set(handles.sat4,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function cuant4_Callback(hObject, eventdata, handles)
% hObject    handle to cuant4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant4 as text
%        str2double(get(hObject,'String')) returns contents of cuant4 as a double


% --- Executes during object creation, after setting all properties.
function sat4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat4_Callback(hObject, eventdata, handles)
% hObject    handle to sat4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat4 as text
%        str2double(get(hObject,'String')) returns contents of sat4 as a double


% --- Executes on button press in mod5_1.
function mod5_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod5_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod5_1
```

```
handles.orden(5)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod5_1, 'Value', 1);
set(handles.mod5_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod6_1,'Enable','on');
set(handles.mod6_2,'Enable','on');
set(handles.cuant5,'Enable','on');
set(handles.sat5,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod5_2.
function mod5_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod5_2

handles.orden(5)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod5_2, 'Value', 1);
set(handles.mod5_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod6_1,'Enable','on');
set(handles.mod6_2,'Enable','on');
set(handles.cuant5,'Enable','on');
set(handles.sat5,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant5 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant5_Callback(hObject, eventdata, handles)
% hObject    handle to cuant5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant5 as text
%        str2double(get(hObject,'String')) returns contents of cuant5 as a double


% --- Executes during object creation, after setting all properties.
function sat5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat5_Callback(hObject, eventdata, handles)
% hObject    handle to sat5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of sat5 as text
%        str2double(get(hObject,'String')) returns contents of sat5 as a double


% --- Executes on button press in mod6_1.
function mod6_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod6_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod6_1

handles.orden(6)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod6_1, 'Value', 1);
set(handles.mod6_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod7_1,'Enable','on');
set(handles.mod7_2,'Enable','on');
set(handles.cuant6,'Enable','on');
set(handles.sat6,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function cuant6_Callback(hObject, eventdata, handles)
% hObject    handle to cuant6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant6 as text
%        str2double(get(hObject,'String')) returns contents of cuant6 as a double


% --- Executes during object creation, after setting all properties.
function sat6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat6_Callback(hObject, eventdata, handles)
% hObject    handle to sat6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat6 as text
%        str2double(get(hObject,'String')) returns contents of sat6 as a double


% --- Executes on button press in mod7_1.
function mod7_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod7_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod7_1
```

```
handles.orden(7)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod7_1, 'Value', 1);
set(handles.mod7_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod8_1,'Enable','on');
set(handles.mod8_2,'Enable','on');
set(handles.cuant7,'Enable','on');
set(handles.sat7,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod7_2.
function mod7_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod7_2

handles.orden(7)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod7_2, 'Value', 1);
set(handles.mod7_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod8_1,'Enable','on');
set(handles.mod8_2,'Enable','on');
set(handles.cuant7,'Enable','on');
set(handles.sat7,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant7 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant7_Callback(hObject, eventdata, handles)
% hObject    handle to cuant7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant7 as text
%        str2double(get(hObject,'String')) returns contents of cuant7 as a double


% --- Executes during object creation, after setting all properties.
function sat7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat7_Callback(hObject, eventdata, handles)
% hObject    handle to sat7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of sat7 as text
%        str2double(get(hObject,'String')) returns contents of sat7 as a double


% --- Executes on button press in mod8_1.
function mod8_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod8_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod8_1

handles.orden(8)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod8_1, 'Value', 1);
set(handles.mod8_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod9_1,'Enable','on');
set(handles.mod9_2,'Enable','on');
set(handles.cuant8,'Enable','on');
set(handles.sat8,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function cuant8_Callback(hObject, eventdata, handles)
% hObject    handle to cuant8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant8 as text
%        str2double(get(hObject,'String')) returns contents of cuant8 as a double


% --- Executes during object creation, after setting all properties.
function sat8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat8_Callback(hObject, eventdata, handles)
% hObject    handle to sat8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat8 as text
%        str2double(get(hObject,'String')) returns contents of sat8 as a double


% --- Executes on button press in mod9_1.
function mod9_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod9_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod9_1
```

```
handles.orden(9)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod9_1, 'Value', 1);
set(handles.mod9_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod10_1,'Enable','on');
set(handles.mod10_2,'Enable','on');
set(handles.cuant9,'Enable','on');
set(handles.sat9,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod9_2.
function mod9_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod9_2

handles.orden(9)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod9_2, 'Value', 1);
set(handles.mod9_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod10_1,'Enable','on');
set(handles.mod10_2,'Enable','on');
set(handles.cuant9,'Enable','on');
set(handles.sat9,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant9 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant9_Callback(hObject, eventdata, handles)
% hObject    handle to cuant9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant9 as text
%        str2double(get(hObject,'String')) returns contents of cuant9 as a double


% --- Executes during object creation, after setting all properties.
function sat9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat9_Callback(hObject, eventdata, handles)
% hObject    handle to sat9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of sat9 as text
%        str2double(get(hObject,'String')) returns contents of sat9 as a double


% --- Executes on button press in mod10_1.
function mod10_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod10_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod10_1

handles.orden(10)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod10_1, 'Value', 1);
set(handles.mod10_2, 'Value', 0);

set(handles.cuant10,'Enable','on');
set(handles.sat10,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod10_2.
function mod10_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod10_2

handles.orden(10)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod10_2, 'Value', 1);
set(handles.mod10_1, 'Value', 0);

set(handles.cuant10,'Enable','on');
set(handles.sat10,'Enable','on');

%ha habido un cambio en la arquitectura
```

```
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant10_Callback(hObject, eventdata, handles)
% hObject    handle to cuant10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant10 as text
%        str2double(get(hObject,'String')) returns contents of cuant10 as a double


% --- Executes during object creation, after setting all properties.
function sat10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function sat10_Callback(hObject, eventdata, handles)
% hObject    handle to sat10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat10 as text
%        str2double(get(hObject,'String')) returns contents of sat10 as a double


% --- Executes on button press in mod1_1.
function mod1_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod1_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod1_1

handles.orden(1)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod1_1, 'Value', 1);
set(handles.mod1_2, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod2_1,'Enable','on');
set(handles.mod2_2,'Enable','on');
set(handles.cuant1,'Enable','on');
set(handles.sat1,'Enable','on');

m_reset=0;
save sd_arquitectura m_reset;

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod1_2.
function mod1_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of mod1_2

handles.orden(1)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod1_1, 'Value', 0);
set(handles.mod1_2, 'Value', 1);

%activamos los controles del siguiente modulador
set(handles.mod2_1,'Enable','on');
set(handles.mod2_2,'Enable','on');
set(handles.cuant1,'Enable','on');
set(handles.sat1,'Enable','on');

m_reset=0;
save sd_arquitectura m_reset;

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant1_Callback(hObject, eventdata, handles)
% hObject    handle to cuant1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant1 as text
```

```
%        str2double(get(hObject,'String')) returns contents of cuant1 as a double


% --- Executes during object creation, after setting all properties.
function sat1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat1_Callback(hObject, eventdata, handles)
% hObject    handle to sat1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat1 as text
%        str2double(get(hObject,'String')) returns contents of sat1 as a double


% --- Executes on button press in mod2_1.
function mod2_1_Callback(hObject, eventdata, handles)
% hObject    handle to mod2_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod2_1

handles.orden(2)=1;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod2_1, 'Value', 1);
set(handles.mod2_2, 'Value', 0);

%activamos los controles del siguiente modulador
```

```
set(handles.mod3_1,'Enable','on');
set(handles.mod3_2,'Enable','on');
set(handles.cuant2,'Enable','on');
set(handles.sat2,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod2_2.
function mod2_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod2_2

handles.orden(2)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod2_2, 'Value', 1);
set(handles.mod2_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod3_1,'Enable','on');
set(handles.mod3_2,'Enable','on');
set(handles.cuant2,'Enable','on');
set(handles.sat2,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function cuant2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cuant2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cuant2_Callback(hObject, eventdata, handles)
% hObject    handle to cuant2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cuant2 as text
%        str2double(get(hObject,'String')) returns contents of cuant2 as a double


% --- Executes during object creation, after setting all properties.
function sat2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sat2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sat2_Callback(hObject, eventdata, handles)
% hObject    handle to sat2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sat2 as text
%        str2double(get(hObject,'String')) returns contents of sat2 as a double


% --- Executes on button press in m_cancelar_pushbutton.
function m_cancelar_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to m_cancelar_pushbutton (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


load sd_arquitectura;
m_reset=0;
save sd_arquitectura orden cuant sat m_reset;

close(handles.configurador_arquitectura);

% --- Executes on button press in m_aceptar_pushbutton.
function m_aceptar_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to m_aceptar_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load sd_arquitectura
%recuperamos el orden
orden=handles.orden;
%recuperamos el numero escalones de cuantizacion y niveles de saturacion de los cuantizadores
for(i=1:1:length(orden))
   guarda_cuant=['cuant(',int2str(i),')=str2double(get(handles.cuant',int2str(i),',''String''));'];
   eval(guarda_cuant);
   guarda_sat=['sat(',int2str(i),')=str2double(get(handles.sat',int2str(i),',''String''));'];
   eval(guarda_sat);
end

%guardamos el orden del modulador, y los niveles y rangos de los cuantizadores
save sd_arquitectura orden cuant sat m_reset;
%cerramos la ventana del configurador de arquitectura
close(handles.configurador_arquitectura);
%volvemos a la interfaz de configuracion del modulador
GUI_modulador

if(handles.cambio==1)
   %reseteamos el configurador_G ya que cambiara al cambiar el modulador
   g_reset=1;
   save sd_matrizG g_reset;
end

%reseteamos el configurador_salidas ya que cambiara al cambiar el modulador
salidas_plot=1;
s_reset=1;
```

```
save sd_salida_aux salidas_plot s_reset;


% --- Executes on button press in m_reset_pushbutton.
function m_reset_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to m_reset_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

m_reset=1;
save sd_arquitectura m_reset;
close(handles.configurador_arquitectura);
configurador_arquitectura

% --- Executes on button press in mod6_2.
function mod6_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod6_2

handles.orden(6)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod6_2, 'Value', 1);
set(handles.mod6_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod7_1,'Enable','on');
set(handles.mod7_2,'Enable','on');
set(handles.cuant6,'Enable','on');
set(handles.sat6,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);

% --- Executes on button press in mod8_2.
function mod8_2_Callback(hObject, eventdata, handles)
% hObject    handle to mod8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of mod8_2

handles.orden(8)=2;
guidata(hObject,handles);
set(handles.texto_orden,'String',int2str(handles.orden));
set(handles.mod8_2, 'Value', 1);
set(handles.mod8_1, 'Value', 0);

%activamos los controles del siguiente modulador
set(handles.mod9_1,'Enable','on');
set(handles.mod9_2,'Enable','on');
set(handles.cuant8,'Enable','on');
set(handles.sat8,'Enable','on');

%ha habido un cambio en la arquitectura
handles.cambio=1;
guidata(hObject,handles);
```

```
%FUNCION: "configurador_g"
%DESCRIPCION:
%   Genera una interfaz de usuario para tomar los coeficientes 'g'
%   correspondientes a la arquitectura previamente seleccionada mediante
%   "configurador_arquitectura" para el programa "sd_cascada".
%   Los coeficientes (g,g',g") de cada integrador son almacenados en una
%   matriz G de la forma:
%
%          Integrador1            Integrador2              ...
%
%          entrada 1 (+)          entrada 1 (+)            ...
%   G(3xN)= entrada 2 (-)          entrada 2 (-)            ...
%          entrada realimentada (-) entrada realimentada (-)   ...
%
%   Ejemplo1: Para un modulador 2-1-1, G tendra la forma
%       g1  g2  g3  g4
%    G=  0   0   g3' g4'
%       g1' g2' g3" g4"
%   Ejemplo2: Para un modulador 2-2, G tendra la forma
%       g1  g2  g3  g4
%    G=  0   0   g3' 0
%       g1' g2' g3" g4'
%
%PARAMETROS:
```

```
%   No tiene parametros propiamente dichos.
%   Toma la arquitectura del modulador cargado el archivo "sd_arquitectura"
%
%Ver: acepta_G.m, cancela_G.m, reset_G.m, actualiza_G.m, g_edit.m, g_slider.m

clear all;
load sd_arquitectura %cargamos la arquitectura previamente seleccionada

%Ajustamos el tamaño y distribucion de la ventana dependiendo del numero de
%moduladores de la cascada
primer_mod=orden(1);
num_moduladores=sum(orden);
if(num_moduladores<11)
   tam_x=400;
   tam_y=(num_moduladores+1)*50;
else if (num_moduladores<21)
       tam_x=800;
       tam_y=11*50;
   else
       disp ('Demasiados moduladores. Introducir G Matricialmente');
   end
end
%Ajustamos la posicion de la ventana
ind_x=800-tam_x;
ind_y=580-tam_y;

%creamos la ventana y fijamos sus propiedades
fig=figure(1);
set(1,'units','pixels','Menubar','none','Name','Configurador de coeficientes "g"','Resize','off','position',[ind_x ind_y tam_x tam_y]);
set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

%Implementamos los botones de control "aceptar", "cancelar" y un marco
h_frame=uicontrol(fig,'Style','frame','units','pixels','position',[ 255 tam_y-20-50*1-5  102+10 71 ]);
h_reset=uicontrol(fig,'Style','pushbutton','units','pixels','position',[ 260 tam_y-26 (102) (17) ], 'String','Reset','Callback','reset_G(h_slider,h_edit)');
h_aceptar=uicontrol(fig,'Style','pushbutton','units','pixels','position',[ 260 tam_y-48 (102) (17) ], 'String','Aceptar','Callback','acepta_G(orden,h_slider)');
h_cancelar=uicontrol(fig,'Style','pushbutton','units','pixels','position',[ 260 tam_y-20-50*1 (102) (17) ], 'String','Cancelar','Callback','cancela_G');

%Implementamos los controles para el primer modulador que no contendra g"
for(i=1:1:primer_mod)
   nombre_var=['g',int2str(i)];
```

```
    h_text(2*i-1)=uicontrol(fig,'Style','text','units','pixels','position',[ 20 tam_y-50*i 20 20 ], 'String',nombre_var);

    h_edit(2*i-1)=uicontrol(fig,'Style','edit','position',[ 70 tam_y-i*50 50 20 ], 'String','0','BackgroundColor','white');

    h_slider(2*i-1)=uicontrol(fig,'Style','slider','position',[ 20 (tam_y-20-i*50) (102) (15) ], 'Min',0,'Max',4);

    nombre_var=['g',int2str(i),''''];

    h_text(2*i)=uicontrol(fig,'Style','text','position',[ 140 tam_y-50*i (20) (20) ], 'String',nombre_var);

    h_edit(2*i)=uicontrol(fig,'Style','edit','position',[ 190 tam_y-i*50 (50) (20) ], 'String','0','BackgroundColor','white');

    h_slider(2*i)=uicontrol(fig,'Style','slider','position',[ 140 (tam_y-20-i*50) (102) (15) ], 'Min',0,'Max',4);

end


%Implementamos los controles para el resto de moduladores de la cascada. Contendra g,g',g"

desplaza_x=0;

desplaza_y=0;

index=0;

for(j=2:1:length(orden))

    i=sum(orden(1:j));

    if(orden(j)==2)

        i=i-1;

    end

    if (i==11) %si tenemos mas de 10 moduladores distribuiremos los coeficientes en 2 columnas

        desplaza_x=400;

        desplaza_y=10*50;

    end

    for(k=1:1:orden(j))

        nombre_var=['g',int2str(i)];

        h_text(3*i-2-primer_mod+index)=uicontrol(fig,'Style','text','position',[ 20+desplaza_x tam_y-50*i+desplaza_y (25) (20) ], 'String',nombre_var);

        h_edit(3*i-2-primer_mod+index)=uicontrol(fig,'Style','edit','position',[ 70+desplaza_x tam_y-50*i+desplaza_y (50) (20) ], 'String','0','BackgroundColor','white');

        h_slider(3*i-2-primer_mod+index)=uicontrol(fig,'Style','slider','position',[ 20+desplaza_x (tam_y-20-i*50)+desplaza_y (102) (15) ], 'Min',0,'Max',4);

        nombre_var=['g',int2str(i),''''];

        h_text(3*i-1-primer_mod+index)=uicontrol(fig,'Style','text','position',[ 140+desplaza_x tam_y-50*i+desplaza_y (25) (20) ], 'String',nombre_var);

        h_edit(3*i-1-primer_mod+index)=uicontrol(fig,'Style','edit','position',[ 190+desplaza_x tam_y-50*i+desplaza_y (50) (20) ], 'String','0','BackgroundColor','white');

        h_slider(3*i-1-primer_mod+index)=uicontrol(fig,'Style','slider','position',[ 140+desplaza_x (tam_y-20-i*50)+desplaza_y (102) (15) ], 'Min',0,'Max',4);

        if(k==1) %si es de primer orden habra g"

            nombre_var=['g',int2str(i),''''];

            h_text(3*i-primer_mod+index)=uicontrol(fig,'Style','text','position',[ 260+desplaza_x tam_y-50*i+desplaza_y (25) (20) ], 'String',nombre_var);

            h_edit(3*i-primer_mod+index)=uicontrol(fig,'Style','edit','position',[ 310+desplaza_x tam_y-50*i+desplaza_y (50) (20) ], 'String','0','BackgroundColor','white');
```

```
        h_slider(3*i-primer_mod+index)=uicontrol(fig,'Style','slider','position',[ 260+desplaza_x
(tam_y-20-i*50)+desplaza_y (102) (15) ], 'Min',0,'Max',4);
        i=i+1;
      else
      index=index-1; %para ajestar el indice en el caso de que solo hubiera g y g'
      end
   end
end

%Ajustamos los callback de los 'slider' y 'edit'
for(i=1:1:length(h_text))
   set(h_slider(i),'Callback',['g_slider(h_slider,h_edit,',num2str(i),')'])%,',orden)'])
   set(h_edit(i),'Callback',['g_edit(h_slider,h_edit,',num2str(i),')'])%,',orden)'])
end

%cargamos la matriz G peviamente guardada
load sd_matrizG
%si no ha sido reseteada, representamos los valores almacenados
if(g_reset==0)
   for(i=1:1:length(h_slider))
      set(h_slider(i),'value',slider_value(i));%para cada delizador, marcamos su valor almacenado
      g_slider (h_slider,h_edit,i); %sincronizamos el deslizador y la caja de texto para que marque el
mismo valor
   end
end
```

```
function varargout = configurador_nolineal(varargin)
% CONFIGURADOR_NOLINEAL M-file for configurador_nolineal.fig
%       CONFIGURADOR_NOLINEAL, by itself, creates a new CONFIGURADOR_NOLINEAL or raises the
existing
%      singleton*.
%
%       H = CONFIGURADOR_NOLINEAL returns the handle to a new CONFIGURADOR_NOLINEAL or the
handle to
%      the existing singleton*.
%
%      CONFIGURADOR_NOLINEAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONFIGURADOR_NOLINEAL.M with the given input arguments.
%
%       CONFIGURADOR_NOLINEAL('Property','Value',...) creates a new CONFIGURADOR_NOLINEAL or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before configurador_nolineal_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
```

```
%      stop.  All inputs are passed to configurador_nolineal_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help configurador_nolineal

% Last Modified by GUIDE v2.5 09-Dec-2003 19:29:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @configurador_nolineal_OpeningFcn, ...
                   'gui_OutputFcn',  @configurador_nolineal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before configurador_nolineal is made visible.
function configurador_nolineal_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to configurador_nolineal (see VARARGIN)

% Choose default command line output for configurador_nolineal
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes configurador_nolineal wait for user response (see UIRESUME)
% uiwait(handles.figure1);

load sd_nolineal

if (jitter_act)%si esta activado el Jitter
    set (handles.jitter, 'value',1);%Marcador
    set (handles.dt_jitter, 'string',num2str(dt_jitter));%cargamos la desviacion tipica
    set (handles.m_jitter, 'string',num2str(m_jitter));%cargamos la media
else%si no esta activado
    set (handles.dt_jitter, 'enable', 'off')%inhabilitamos controles de desviacion tipica
    set (handles.m_jitter, 'enable', 'off')%inhabilitamos controles de media
end

if (var_g_act)%si estan activadas las variaciones de ganancia
    set (handles.variacion_g, 'value',1);%Marcador
    set (handles.dt_var_g, 'string',num2str(dt_var_g));%cargamos la desviacion tipica
    set (handles.m_var_g, 'string',num2str(m_var_g));%cargamos la media
    set (handles.corregir_canc,'value',corregir_canc);
else
    set (handles.dt_var_g, 'enable', 'off')%inhabilitamos controles de desviacion tipica
    set (handles.m_var_g, 'enable', 'off')%inhabilitamos controles de media
    set (handles.corregir_canc,'enable','off');
end

if (perdidas_act)%si estan activadas las perdidas
    set (handles.perdidas, 'value',1);%Marcador
    set (handles.coef_perdidas, 'string',num2str(coef_perdidas));%cargamos el coeficiente de perdidas
    set (handles.slider_perdidas, 'value',coef_perdidas);%tambien en el deslizador
else%si no esta activado
    set (handles.coef_perdidas, 'enable', 'off')%inhabilitamos control de coeficiente
    set (handles.slider_perdidas, 'enable', 'off')%inhabilitamos deslizador
end


% --- Outputs from this function are returned to the command line.
function varargout = configurador_nolineal_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function dt_jitter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dt_jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function dt_jitter_Callback(hObject, eventdata, handles)
% hObject    handle to dt_jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dt_jitter as text
%        str2double(get(hObject,'String')) returns contents of dt_jitter as a double


% --- Executes during object creation, after setting all properties.
function m_jitter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m_jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function m_jitter_Callback(hObject, eventdata, handles)
% hObject    handle to m_jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m_jitter as text
%        str2double(get(hObject,'String')) returns contents of m_jitter as a double



% --- Executes during object creation, after setting all properties.
function dt_var_g_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dt_var_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function dt_var_g_Callback(hObject, eventdata, handles)
% hObject    handle to dt_var_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dt_var_g as text
%        str2double(get(hObject,'String')) returns contents of dt_var_g as a double


% --- Executes during object creation, after setting all properties.
function m_var_g_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m_var_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function m_var_g_Callback(hObject, eventdata, handles)
% hObject    handle to m_var_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m_var_g as text
%        str2double(get(hObject,'String')) returns contents of m_var_g as a double


% --- Executes during object creation, after setting all properties.
function coef_perdidas_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coef_perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function coef_perdidas_Callback(hObject, eventdata, handles)
% hObject    handle to coef_perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of coef_perdidas as text
%        str2double(get(hObject,'String')) returns contents of coef_perdidas as a double
```

```
NewStrVal = get(handles.coef_perdidas, 'String'); %capturamos el nuevo valor de la caja de texto editable
NewVal = str2double(NewStrVal); %convertimos la cadena de caracteres en un double
% Comprobamos que el valor introducido cae dentro del rango permitido
if  isempty(NewVal) | (NewVal< 0) | (NewVal>1),
   % Regresar al valor anterior
   OldVal = get(handles.slider_perdidas,'Value'); %capturamos el valor anterior del deslizador
   set(handles.coef_perdidas, 'String',OldVal) %fijamos en la caja de texto editable el valor anterior
else, % Usar el nuevo valor
   % Fija en el deslizador el nuevo valor
   set(handles.slider_perdidas,'Value',NewVal)
end


% --- Executes on button press in aceptar_nolin.
function aceptar_nolin_Callback(hObject, eventdata, handles)
% hObject    handle to aceptar_nolin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load sd_nolineal

if(get(handles.jitter,'value')==1)
   jitter_act=1;
   dt_jitter=str2double(get(handles.dt_jitter, 'string'));
   m_jitter=str2double(get(handles.m_jitter, 'string'));
   save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g coef_perdidas;
   GUI_entrada;
else
   jitter_act=0;
   dt_jitter=0;
   m_jitter=0;
   save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g coef_perdidas;
   GUI_entrada;
end

if(get(handles.variacion_g,'value')==1)
   var_g_act=1;
   dt_var_g=str2double(get(handles.dt_var_g, 'string'));
   m_var_g=str2double(get(handles.m_var_g, 'string'));
   corregir_canc=get(handles.corregir_canc,'value');
else
```

```
    var_g_act=0;
    dt_var_g=0;
    m_var_g=0;
    corregir_canc=0;
end

if(get(handles.perdidas,'value')==1)
    perdidas_act=1;
    coef_perdidas=str2double(get(handles.coef_perdidas, 'string'));
else
    perdidas_act=0;
    coef_perdidas=0;
end



save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g corregir_canc
coef_perdidas;
close (configurador_nolineal)

% --- Executes on button press in cancelar_nolin.
function cancelar_nolin_Callback(hObject, eventdata, handles)
% hObject    handle to cancelar_nolin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close (configurador_nolineal)

% --- Executes during object creation, after setting all properties.
function slider_perdidas_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider_perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes on slider movement.
function slider_perdidas_Callback(hObject, eventdata, handles)
% hObject    handle to slider_perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

NewVal = get(handles.slider_perdidas, 'Value'); %tomamos el nuevo valor del deslizador
set(handles.coef_perdidas,'String',NewVal); %fijamos el nuevo valor en la caja de texto editable



% --- Executes on button press in jitter.
function jitter_Callback(hObject, eventdata, handles)
% hObject    handle to jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of jitter

if (get (handles.jitter, 'value')==1)
   set (handles.jitter, 'value',1);
   set (handles.dt_jitter, 'enable', 'on')
   set (handles.m_jitter, 'enable', 'on')
else
   set (handles.jitter, 'value',0);
   set (handles.dt_jitter, 'enable', 'off')
   set (handles.m_jitter, 'enable', 'off')
end

% --- Executes on button press in variacion_g.
function variacion_g_Callback(hObject, eventdata, handles)
% hObject    handle to variacion_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of variacion_g

if (get (handles.variacion_g, 'value')==1)
   set (handles.variacion_g, 'value',1);
   set (handles.dt_var_g, 'enable', 'on')
   set (handles.m_var_g, 'enable', 'on')
```

```
    set (handles.corregir_canc,'enable','on')
else
    set (handles.variacion_g, 'value',0);
    set (handles.dt_var_g, 'enable', 'off')
    set (handles.m_var_g, 'enable', 'off')
    set (handles.corregir_canc,'enable','off')
end

% --- Executes on button press in perdidas.
function perdidas_Callback(hObject, eventdata, handles)
% hObject    handle to perdidas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of perdidas

if (get (handles.perdidas, 'value')==1)
    set (handles.perdidas, 'value',1);
    set (handles.coef_perdidas, 'enable', 'on')
    set (handles.slider_perdidas, 'enable', 'on')
else
    set (handles.perdidas, 'value',0);
    set (handles.coef_perdidas, 'enable', 'off')
    set (handles.slider_perdidas, 'enable', 'off')
end



% --- Executes on button press in corregir_canc.
function corregir_canc_Callback(hObject, eventdata, handles)
% hObject    handle to corregir_canc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of corregir_canc
```

```
function varargout = configurador_salidas(varargin)
% CONFIGURADOR_SALIDAS M-file for configurador_salidas.fig
%      CONFIGURADOR_SALIDAS, by itself, creates a new CONFIGURADOR_SALIDAS or raises the existing
%      singleton*.
%
%      H = CONFIGURADOR_SALIDAS returns the handle to a new CONFIGURADOR_SALIDAS or the handle to
```

```
%      the existing singleton*.
%
%      CONFIGURADOR_SALIDAS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CONFIGURADOR_SALIDAS.M with the given input arguments.
%
%          CONFIGURADOR_SALIDAS('Property','Value',...) creates a new CONFIGURADOR_SALIDAS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before configurador_salidas_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to configurador_salidas_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help configurador_salidas

% Last Modified by GUIDE v2.5 15-Sep-2003 10:59:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @configurador_salidas_OpeningFcn, ...
                   'gui_OutputFcn',  @configurador_salidas_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before configurador_salidas is made visible.
function configurador_salidas_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to configurador_salidas (see VARARGIN)

% Choose default command line output for configurador_salidas
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes configurador_salidas wait for user response (see UIRESUME)
% uiwait(handles.figure1);

%cargamos datos
load sd_arquitectura;
load sd_salida_aux; %este archivo nos servira para comunicarnos con la interfaz GUI_salida

tam=length(orden); %numero de etapas del modulador

%Mostramos las posibles salidas correspondientes a cada etapa de la cascada
for(i=1:1:10)
   if(i<tam)
      escribe_orden=['set(handles.s_m',int2str(i),'_t,''string'',''',int2str(orden(1:i)),''');'];
      eval(escribe_orden);
      fija_cero=['set (handles.s_m',int2str(i),','value'',0);'];
      eval(fija_cero);
   elseif(i==tam)%la ultima salida, modulador con todas las etapas de la cascada
      escribe_orden=['set(handles.s_m',int2str(i),'_t,''string'',''',int2str(orden(tam+1-i:tam)),''');'];
      eval(escribe_orden);
      if(s_reset==1)%la ultima salida la activamos por defecto ya que es el modulador total
         fija_uno=['set (handles.s_m',int2str(i),','value'',1);'];
         eval(fija_uno);
      end
   else %desactivamos los controles que no necesitamos
       desactiva=['set(handles.s_m',int2str(i),','enable'',''off'');'];
       eval(desactiva);
   end
end

if(s_reset==0)% si no esta activado el reset
   for(i=1:1:length(salidas_plot)) %marcamos las salidas previamente seleccionadas
```

```
        act_sal=['set (handles.s_m',int2str(salidas_plot(i)),',''value'',1);'];
        eval(act_sal);
    end
end

s_reset=0; %desactivamos el reset
save sd_salida_aux salidas_plot s_reset;% guardamos datos

% --- Outputs from this function are returned to the command line.
function varargout = configurador_salidas_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in s_m1.
function s_m1_Callback(hObject, eventdata, handles)
% hObject    handle to s_m1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m1


% --- Executes on button press in s_m2.
function s_m2_Callback(hObject, eventdata, handles)
% hObject    handle to s_m2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m2


% --- Executes on button press in s_m3.
function s_m3_Callback(hObject, eventdata, handles)
% hObject    handle to s_m3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of s_m3


% --- Executes on button press in s_m4.
function s_m4_Callback(hObject, eventdata, handles)
% hObject    handle to s_m4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m4


% --- Executes on button press in s_m7.
function s_m7_Callback(hObject, eventdata, handles)
% hObject    handle to s_m7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m7


% --- Executes on button press in s_m9.
function s_m9_Callback(hObject, eventdata, handles)
% hObject    handle to s_m9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m9


% --- Executes on button press in s_m5.
function s_m5_Callback(hObject, eventdata, handles)
% hObject    handle to s_m5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m5


% --- Executes on button press in s_m10.
function s_m10_Callback(hObject, eventdata, handles)
% hObject    handle to s_m10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% Hint: get(hObject,'Value') returns toggle state of s_m10



% --- Executes on button press in s_m8.
function s_m8_Callback(hObject, eventdata, handles)
% hObject    handle to s_m8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m8



% --- Executes on button press in s_m6.
function s_m6_Callback(hObject, eventdata, handles)
% hObject    handle to s_m6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_m6


% --- Executes on button press in s_aceptar_pushbutton.
function s_aceptar_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to s_aceptar_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%cargamos datos
load sd_arquitectura
load sd_salida_aux
tam=length(orden);

%capturamos si la salida de cada etapa ha sido seleccionada para
%representarse
i=1;%inicializamos el indice del vector salidas_plot
if(get (handles.s_m1,'value'))%si la primera etapa esta seleccionada
   salidas_plot(i)=1; %añadimos al vector salidas_plot un 1
   i=i+1;%incrementamos el indice del vector
```

```
end
if(get (handles.s_m2,'value'))%si la segunda etapa esta seleccionada
   salidas_plot(i)=2; %añadimos al vector salidas_plot un 2
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m3,'value'))%si la tercera etapa esta seleccionada
   salidas_plot(i)=3; %añadimos al vector salidas_plot un 3
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m4,'value'))%si la cuarta etapa esta seleccionada
   salidas_plot(i)=4; %añadimos al vector salidas_plot un 4
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m5,'value'))%si la quinta etapa esta seleccionada
   salidas_plot(i)=5; %añadimos al vector salidas_plot un 5
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m6,'value'))%si la sexta etapa esta seleccionada
   salidas_plot(i)=6; %añadimos al vector salidas_plot un 6
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m7,'value'))%si la septima etapa esta seleccionada
   salidas_plot(i)=7; %añadimos al vector salidas_plot un 7
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m8,'value'))%si la octava etapa esta seleccionada
   salidas_plot(i)=8; %añadimos al vector salidas_plot un 8
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m9,'value'))%si la novena etapa esta seleccionada
   salidas_plot(i)=9; %añadimos al vector salidas_plot un 9
   i=i+1;%incrementamos el indice del vector
end
if(get (handles.s_m10,'value'))%si la decima etapa esta seleccionada
   salidas_plot(i)=10; %añadimos al vector salidas_plot un 10
   i=i+1;%incrementamos el indice del vector
end

%salvamos los datos
salidas_plot=salidas_plot(1:(i-1));
save sd_salida_aux salidas_plot s_reset;

close(configurador_salidas) %cerramos el configurador de salidas
```

```
GUI_salida;%volvemos a la interfaz de salida

% --- Executes on button press in s_cancelar_pushbutton.
function s_cancelar_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to s_cancelar_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close(configurador_salidas)%cierra la ventana activa sin hacer nada
```

```
function cuantizado=cuantiza(entrada,n_levels,rg_cuant)
% Cuantiza un vector de entrada con un numero determinado de escalones de cuantización y un rango de cuantización fijado.
% OBSERVACION:
%   Esta funcion opera vectorialmente a excepcion de la comprobacion de
%   sobrecarga del cuantizador.
%   Para obtener un comportamiento vectorial comentar las lineas 23 a 35
%   (comprobacion de saturacion)
%
% PARAMETROS DE ENTRADA:
%   entrada: vector a cuantizar.
%   n_levels: niveles digitales posibles a la salida.
%   rg_cuant: rango de cuantizacion, Vmax-Vmin a la entrada.
%
% PARAMETROS DE SALIDA
%         cuantizado vector de valores cuantizados
%
% Ver tambien configurador_g, cascada, m1, m2, cancela

delta=rg_cuant/n_levels; %escalon de cuantizacion
dig=[(-rg_cuant+delta)/2:delta:(rg_cuant-delta)/2];%vector de salidas digitales posibles

%comprobamos si la entrada sobrecarga el cuantizador
for(i=1:1:length(entrada))
   if(entrada(i)>=(rg_cuant/2))%comprobamos si hay sobrecarga superior en el cuantizador
      disp ('Warning: Sobrecarga superior en el cuantizador');
      disp ('x>(rg_cuant/2)')
      entrada(i)=dig(n_levels);%no obstante damos como salida el valor digital mas alto
   else%si no, seguimos...
   if(entrada(i)<=(-rg_cuant/2))%comprobamos si hay sobrecarga inferior en el cuantizador
      disp ('Warning: Sobrecarga inferior en el cuantizador');
      disp ('x<(-rg_cuant/2)')
```

```
        entrada(i)=dig(1);%no obstante damos como salida el valor digital mas bajo
    end
    end
end


%ahora cuantizamos propiamente:
x=entrada+rg_cuant/2;%desplazamos el eje x para que los valores esten comprendidos entre 0 y rg_din
index=floor(x/delta)+1;%tomamos la parte entera de la division por el escalon de cuantizacion y le sumamos uno. Esto nos da el numero de la salida digital correspondiente (ordenadas de menor a mayor en dig)
cuantizado=dig(index);%tomamos los valores cuantizados marcados por index
```

```
function g_edit (h_slider,h_edit,i)
% "Callback" de las cajas de texto editables de la interfaz "configurador_g".
% Actualiza el valor del deslizador del parámetro correspondiente, al de la caja de texto editable que acaba de ser modificada.
% Comprueba que el valor introducido cae dentro del rango permitido. Si no
% es asi, fija el valor anterior.
% PARAMETROS DE ENTRADA
%       h_slider: vector con los manejadores de los deslizadores.
%       h_edit: vector con los manejadores de las cajas editables de texto.
%       i: indice que nos indica a que parámetro concreto nos estamos refiriendo.
%
% Ver tambien: actualiza_g, acepta_G.m, cancela_G.m, actualiza_G.m, g_slider.m, reset_g.m

NewStrVal = get(h_edit(i), 'String'); %capturamos el nuevo valor de la caja de texto editable
NewVal = str2double(NewStrVal); %convertimos la cadena de caracteres en un double
% Comprobamos que el valor introducido cae dentro del rango permitido
if  isempty(NewVal) | (NewVal< 0) | (NewVal>4),
    % Regresar al valor anterior
    OldVal = get(h_slider(i),'Value'); %capturamos el valor anterior del deslizador
    set(h_edit(i), 'String',OldVal) %fijamos en la caja de texto editable el valor anterior
else, % Usar el nuevo valor
    % Fija en el deslizador el nuevo valor
    set(h_slider(i),'Value',NewVal)
end
```

```
function g_slider (h_slider,h_edit,i);
% "Callback" de los deslizadores de la interfaz "configurador_g".
% Actualiza el valor de la caja de texto editable del parámetro correspondiente, al del deslizador que acaba de ser modificado.
% PARAMETROS DE ENTRADA
```

```
%        h_slider: vector con los manejadores de los deslizadores.
%        h_edit: vector con los manejadores de las cajas editables de texto.
%        i: indice que nos indica a que parámetro concreto nos estamos refiriendo.
%
% Ver tambien: actualiza_g, acepta_G.m, cancela_G.m, actualiza_G.m, g_edit.m, reset_g.m

NewVal = get(h_slider(i), 'Value'); %tomamos el nuevo valor del deslizador
set(h_edit(i),'String',NewVal); %fijamos el nuevo valor en la caja de texto editable
```

```
function varargout = GUI_entrada(varargin)
% GUI_ENTRADA M-file for GUI_entrada.fig
%      GUI_ENTRADA, by itself, creates a new GUI_ENTRADA or raises the existing
%      singleton*.
%
%      H = GUI_ENTRADA returns the handle to a new GUI_ENTRADA or the handle to
%      the existing singleton*.
%
%      GUI_ENTRADA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_ENTRADA.M with the given input arguments.
%
%      GUI_ENTRADA('Property','Value',...) creates a new GUI_ENTRADA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_entrada_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_entrada_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_entrada

% Last Modified by GUIDE v2.5 13-Nov-2003 20:49:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_entrada_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_entrada_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before GUI_entrada is made visible.
function GUI_entrada_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_entrada (see VARARGIN)

% Choose default command line output for GUI_entrada
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI_entrada wait for user response (see UIRESUME)
% uiwait(handles.GUI_entrada);

%Inicializamos los radio buttons
set(handles.fft, 'Value', 0);
set(handles.psd, 'Value', 1);%activamos la PSD por defecto

handles.flag=0; %indica que no hay barra de herramientas activa
guidata(hObject,handles);

load sd_entrada;%cargamos los valores de las variables de entrada
%fijamos en la ventana los valores correspondientes
set(handles.frecuencia,'String',Fx);
set(handles.sobremuestreo,'String',M);
set(handles.ciclos,'String',n_ciclos);
set(handles.ancho_de_banda,'String',B);
```

```
set(handles.amplitud,'String',A);
set(handles.e_nfft,'String',256);
%generamos la entrada propiamente dicha
entrada_Callback(hObject, eventdata, handles);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_entrada_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function frecuencia_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function frecuencia_Callback(hObject, eventdata, handles)
% hObject    handle to frecuencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of frecuencia as text
%        str2double(get(hObject,'String')) returns contents of frecuencia as a double


% --- Executes during object creation, after setting all properties.
function ancho_de_banda_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ancho_de_banda (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function ancho_de_banda_Callback(hObject, eventdata, handles)
% hObject    handle to ancho_de_banda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ancho_de_banda as text
%        str2double(get(hObject,'String')) returns contents of ancho_de_banda as a double



% --- Executes during object creation, after setting all properties.



function sobremuestreo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sobremuestreo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function sobremuestreo_Callback(hObject, eventdata, handles)
% hObject    handle to sobremuestreo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of sobremuestreo as text
%        str2double(get(hObject,'String')) returns contents of sobremuestreo as a double


% --- Executes during object creation, after setting all properties.
function ciclos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ciclos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function ciclos_Callback(hObject, eventdata, handles)
% hObject    handle to ciclos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ciclos as text
%        str2double(get(hObject,'String')) returns contents of ciclos as a double


% --- Executes during object creation, after setting all properties.
function amplitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function amplitud_Callback(hObject, eventdata, handles)
% hObject    handle to amplitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of amplitud as text
%        str2double(get(hObject,'String')) returns contents of amplitud as a double


% --- Executes on button press in entrada.
function entrada_Callback(hObject, eventdata, handles)
% hObject    handle to entrada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Tomamos datos de entrada del usuario a traves de la interfaz GUI
Fx = str2double(get(handles.frecuencia,'String')); %frecuencia de la señal sinusoidal de entrada
M = str2double(get(handles.sobremuestreo,'String')); %tasa de sobremuestreo
n_ciclos = str2double(get(handles.ciclos,'String')); %numero de ciclos de la señal sinusoidal de entrada
B = str2double(get(handles.ancho_de_banda,'String')); %ancho de banda
A = str2double(get(handles.amplitud,'String')); %amplitud de la señal sinusoidal de entrada

load sd_nolineal%cargamosposibles no linealidades
%Muestreamos la señal
[x Fs kTs]=muestrea(Fx,M,n_ciclos,B,A,m_jitter,(dt_jitter)^2);

%Mostramos la frecuencia de muestreo
set(handles.Fs, 'String', Fs);

%Calculamos la fft
[Xw_fft,eje_f_fft]=sd_fft(x,Fs);

%Calculamos la psd
NFFT= str2double(get(handles.e_nfft,'string'));%tomamos nfft de la GUI
%Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
if (NFFT>length(x))%si se señala una NFFT mayor que la longitud de las secuencias
   disp('Warning: max(NFFT)=length(x)')
   NFFT=length(x)%en ese caso fijamos la particion al tamaño de la señal
   set(handles.e_nfft,'string',NFFT)%fijamos el valor maximo permitido
end
[Xw_psd,eje_f_psd]=sd_psd(x,NFFT,Fs,eval(['boxcar(',int2str(NFFT),')']));
```

```
%Representamos en frecuencia
if (get(handles.fft, 'Value'))% si esta marcada la fft en la GUI
    %dibujamos en frecuencia la fft
    axes(handles.ejes_frecuencia) % seleccionamos los ejes adecuados
    %al ser real la señal en el tiempo, en frecuencia su espectro sera par
    %dibujamos solo el espectro correspondiente a "frecuencias positivas"
    plot(eje_f_fft,20*log10(abs(Xw_fft)));
    %fijamos los ejes para mostrar solo las frecuencias positivas
    axis([0,eje_f_fft(end),min(20*log10(abs(Xw_fft))), max(20*log10(abs(Xw_fft)))]);
    xlabel('Frecuencia');
    ylabel('20*log()');
    set(handles.ejes_frecuencia,'XMinorTick','on')
    grid on
else % Si esta marcado en la GUI la PSD
    %dibujamos en frecuencia la psd
    axes(handles.ejes_frecuencia) % seleccionamos en frecuencia los ejes adecuados
    plot(eje_f_psd,10*log10(abs(Xw_psd)));
    xlabel('Frecuencia');
    ylabel('Magnitud [dB]');
    set(handles.ejes_frecuencia,'XMinorTick','on')
    grid on
end

% Dibujamos la señal en el tiempo
axes(handles.ejes_tiempo) % seleccionamos los ejes apropiados
%stem(kTs,x);
plot(kTs,x,'X');
xlabel('Tiempo [seg.]');
ylabel('Amplitud');
set(handles.ejes_tiempo,'XMinorTick','on')
grid on

%guardamos los datos obtenidos en un archivo ".mat"
save sd_entrada  x A n_ciclos M Fx B Fs kTs Xw_fft eje_f_fft Xw_psd eje_f_psd  NFFT;


% --- Executes on button press in fft.
function fft_Callback(hObject, eventdata, handles)
% hObject    handle to fft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of fft
```

```
%Fijamos el selector de la fft
set(handles.fft, 'Value', 1);
set(handles.psd, 'Value', 0);

% Create frequency "fft" plot
load sd_entrada; %Cargamos los datos de la entrada
%dibujamos en frecuencia la fft
axes(handles.ejes_frecuencia) % seleccionamos los ejes adecuados
%al ser real la señal en el tiempo, en frecuencia su espectro sera par
%dibujamos solo el espectro correspondiente a "frecuencias positivas"
plot(eje_f_fft,20*log10(abs(Xw_fft)));
%fijamos los ejes para mostrar solo las frecuencias positivas
axis([0,eje_f_fft(end),min(20*log10(abs(Xw_fft))), max(20*log10(abs(Xw_fft)))]);
xlabel('Frecuencia');
ylabel('Magnitud [dB]');
set(handles.ejes_frecuencia,'XMinorTick','on')
grid on


% --- Executes on button press in psd.
function psd_Callback(hObject, eventdata, handles)
% hObject    handle to psd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of psd

%Fijamos el selector de la psd
set(handles.psd, 'Value', 1);
set(handles.fft, 'Value', 0);

% Create frequency "psd" plot

load sd_entrada; %cargamos los datos de la entrada
%por si se ha modificado la NFFT recalculamos la psd
NFFT= str2double(get(handles.e_nfft,'string'));%tomamos nfft de la GUI
%Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
if (NFFT>length(x))%si se señala una NFFT mayor que la longitud de las secuencias
   disp('Warning: max(NFFT)=length(x)')
   NFFT=length(x)%en ese caso fijamos la particion al tamaño de la señal
   set(handles.e_nfft,'string',NFFT)%fijamos el valor maximo permitido
end
```

```
[Xw_psd,eje_f_psd]=sd_psd(x,NFFT,Fs,eval(['boxcar(',int2str(NFFT),')']));

axes(handles.ejes_frecuencia) % Select the proper axes
plot(eje_f_psd,10*log10(abs(Xw_psd)));
xlabel('Frecuencia');
ylabel('Magnitud [dB]');
set(handles.ejes_frecuencia,'XMinorTick','on')
grid on


% --------------------------------------------------------------------
function archivo_Callback(hObject, eventdata, handles)
% hObject    handle to archivo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function imprimir_Callback(hObject, eventdata, handles)
% hObject    handle to imprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%abrimos la ventana de impresion
printdlg(gcbf);

function previsualizar_Callback(hObject, eventdata, handles)
% hObject    handle to imprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%abrimos la ventana de previsualizar impresion
printpreview(gcbf);

% --------------------------------------------------------------------
function cargar_Callback(hObject, eventdata, handles)
% hObject    handle to cargar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file = uigetfile('*.mat','Cargar entrada');%recogemos es archivo a cargar
if ~isequal(file, 0)%si se ha seleccionado un archivo
   datos=open (file);%cargamos los valores de las variables
```

```
    %fijamos en la ventana los valores correspondientes
    set(handles.frecuencia,'String',datos.Fx);
    set(handles.sobremuestreo,'String',datos.M);
    set(handles.ciclos,'String',datos.n_ciclos);
    set(handles.ancho_de_banda,'String',datos.B);
    set(handles.amplitud,'String',datos.A);
    set(handles.e_nfft,'String',256);
    %generamos la entrada propiamente dicha
    entrada_Callback(hObject, eventdata, handles);
end

% --------------------------------------------------------------------
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file=uiputfile('*.mat','Guardar entrada');%tomamos el nombre del fichero
% Get user input from GUI
Fx = str2double(get(handles.frecuencia,'String'));
M = str2double(get(handles.sobremuestreo,'String'));
n_ciclos = str2double(get(handles.ciclos,'String'));
B = str2double(get(handles.ancho_de_banda,'String'));
A = str2double(get(handles.amplitud,'String'));
%Muestreamos la señal
[x Fs kTs]=muestrea(Fx,M,n_ciclos,B,A);
%Grabamos las variables de interes
save (file, 'x', 'Fs', 'kTs', 'Fx', 'M', 'n_ciclos', 'B', 'A');

% --------------------------------------------------------------------
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Cerramos todo abriendo una question box
selection = questdlg(['¿Quiere cerrar ' get(handles.GUI_entrada,'Name') '?'],...
                     ['Close ' get(handles.GUI_entrada,'Name') '...'],...
                     'Si','No','Si');
if strcmp(selection,'No')
    return;
end
```

```
delete(handles.GUI_entrada)


% --------------------------------------------------------------------
function ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --------------------------------------------------------------------
function documentacion_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

!ayuda.htm

% --------------------------------------------------------------------
function ventana_entrada_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_entrada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function ventana_modulador_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_modulador (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

GUI_modulador;%activamos el modulador

% --------------------------------------------------------------------
function ventana_simulador_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_simulador (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

GUI_salida;%activamos la salida
```

```
% --- Executes during object creation, after setting all properties.
function e_nfft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to e_nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function e_nfft_Callback(hObject, eventdata, handles)
% hObject    handle to e_nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of e_nfft as text
%        str2double(get(hObject,'String')) returns contents of e_nfft as a double


% --------------------------------------------------------------------
function herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function activa_herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to activa_herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if(handles.flag==0)%si no hay barra activada
    fig=gcf;
    htoolbar.htoolbar = uitoolbar('Parent',fig);%hObject);
    % Render Print buttons (Print, Print Preview)
    htoolbar.hprintbtns = render_sptprintbtns(htoolbar.htoolbar);
```

```
    % Render the annotation buttons (Edit Plot, Insert Arrow, etc)
    htoolbar.hscribebtns = render_sptscribebtns(htoolbar.htoolbar);
    % Render the zoom buttons
    htoolbar.hzoombtns = render_zoombtns(htoolbar.htoolbar);
    %Creamos una clase en la estructura para almacenar el handles de herramientas
    handles.herramientas=htoolbar.htoolbar;
    handles.flag=1;%indica que hemos activado la barra de herramientas
    guidata(hObject,handles);
end

% --------------------------------------------------------------------
function desactiva_herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to desactiva_herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set (handles.herramientas,'visible','off');%desactivamos
handles.flag=0;%indica que no hay barra de herramientas activa
guidata(hObject,handles);



% --------------------------------------------------------------------
function zoomin_Callback(hObject, eventdata, handles)
% hObject    handle to zoomin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

zoom on %activamos el zoom

% --------------------------------------------------------------------
function zoomout_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

zoom out %anulamos el zoom
zoom off %desactivamos el zoom



% --------------------------------------------------------------------
function editar_Callback(hObject, eventdata, handles)
% hObject    handle to editar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
sigsetappdata(gcbf,'siggui','ZoomState', 'none');setzoomstate(gcbf);putdowntext('select',gcbo)

% --------------------------------------------------------------------
function insertar_texto_Callback(hObject, eventdata, handles)
% hObject    handle to insertar_texto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%sigsetappdata(gcbf,'siggui','ZoomState', 'none');setzoomstate(gcbf);
plotedit
putdowntext(gcbo);%,'textstart'

% --------------------------------------------------------------------
function insertar_flecha_Callback(hObject, eventdata, handles)
% hObject    handle to insertar_flecha (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sigsetappdata(gcbf,'siggui','ZoomState', 'none');setzoomstate(gcbf);putdowntext('arrowstart',gcbo);

% --------------------------------------------------------------------
function insertar_linea_Callback(hObject, eventdata, handles)
% hObject    handle to insertar_linea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sigsetappdata(gcbf,'siggui','ZoomState', 'none');setzoomstate(gcbf);putdowntext('linestart',gcbo);
```

```
function varargout = GUI_modulador(varargin)
% GUI_MODULADOR M-file for GUI_modulador.fig
%      GUI_MODULADOR, by itself, creates a new GUI_MODULADOR or raises the existing
%      singleton*.
%
%      H = GUI_MODULADOR returns the handle to a new GUI_MODULADOR or the handle to
%      the existing singleton*.
%
%      GUI_MODULADOR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_MODULADOR.M with the given input arguments.
%
%      GUI_MODULADOR('Property','Value',...) creates a new GUI_MODULADOR or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_modulador_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_modulador_OpeningFcn via varargin.
```

```
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_modulador

% Last Modified by GUIDE v2.5 03-Dec-2003 23:49:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_modulador_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_modulador_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_modulador is made visible.
function GUI_modulador_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_modulador (see VARARGIN)

% Choose default command line output for GUI_modulador
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes GUI_modulador wait for user response (see UIRESUME)
% uiwait(handles.GUI_modulador);

%Cargamos y mostramos el orden configurado
load sd_arquitectura;
if(m_reset==1)%si no se habia configurado ninguna arquitectura previamente
    %mostramos cedenas vacias
    set (handles.muestra_orden,'String','');
    set (handles.muestra_cuant,'String','');
    set (handles.muestra_sat,'String','');
    set (handles.G_pushbutton,'enable','off');
    set (handles.no_lineal_pushbutton,'enable','off');
else%si se habia configurado ninguna arquitectura previamente; la cargamos
    set (handles.muestra_orden,'String',int2str(orden));
    %Cargamos y mostramos el cuantizador configurado
    set (handles.muestra_cuant,'String',int2str(cuant));
    %Cargamos y mostramos el cuantizador configurado
    set (handles.muestra_sat,'String',num2str(sat));
    set (handles.G_pushbutton,'enable','on');
    set (handles.no_lineal_pushbutton,'enable','on');
end

% --- Outputs from this function are returned to the command line.
function varargout = GUI_modulador_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes during object creation, after setting all properties.
function orden_CreateFcn(hObject, eventdata, handles)
% hObject    handle to orden (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in G_pushbutton.
function G_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to G_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%abrimos el configurador de coeficientes 'g'
configurador_g

% --- Executes on button press in no_lineal_pushbutton.
function no_lineal_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to no_lineal_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Abrimos el configurador de no-linealidades
configurador_nolineal

% --- Executes on button press in arquitectura_push_button.
function arquitectura_push_button_Callback(hObject, eventdata, handles)
% hObject    handle to arquitectura_push_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%abrimos el configurador de orden
configurador_arquitectura


% --------------------------------------------------------------------
function m_cargar_Callback(hObject, eventdata, handles)
% hObject    handle to m_cargar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.mat','Cargar modulador');%recogemos es archivo a cargar
if ~isequal(file, 0)%si se ha seleccionado un archivo
    datos=open (file);%cargamos los valores de las variables
    %fijamos en la ventana los valores correspondientes
    cuant=datos.cuant;
    orden=datos.orden;
```

```
    sat=datos.sat;
    G=datos.G;
    slider_value=datos.slider_value;
    m_reset=0;
    save sd_arquitectura orden cuant sat m_reset;
    g_reset=0;
    save sd_matrizG G g_reset slider_value;
    s_reset=1;
    salidas_plot=length(orden);
    save sd_salida_aux s_reset salidas_plot;
    jitter_act=datos.jitter_act;
    var_g_act=datos.var_g_act;
    perdidas_act=datos.perdidas_act;
    dt_jitter=datos.dt_jitter;
    m_jitter=datos.m_jitter;
    dt_var_g=datos.dt_var_g;
    m_var_g=datos.m_var_g;
    corregir_canc=datos.corregir_canc;
    coef_perdidas=datos.coef_perdidas;
    save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g corregir_canc
coef_perdidas;
    GUI_modulador('varargin')
    if(jitter_act==1)
        GUI_entrada%si esta activado el jitter la entrada ha de cambiar
    end
end


% --------------------------------------------------------------------
function m_guardar_Callback(hObject, eventdata, handles)
% hObject    handle to m_guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%Regogemos el nombre que el usuario desee para el archivo
file=uiputfile('*.mat','Guardar modulador');
%Cargamos los valores del modulador configurados
load sd_arquitectura;
load sd_matrizG;
load sd_nolineal;
%los salvamos
save(file,'orden','cuant','sat','G','slider_value','jitter_act','var_g_act','perdidas_act','dt_jitter','m_jitter','dt
_var_g','m_var_g','corregir_canc','coef_perdidas');


% --------------------------------------------------------------------
```

```
function m_imprimir_Callback(hObject, eventdata, handles)
% hObject    handle to m_imprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

printdlg(handles.GUI_modulador)

% --------------------------------------------------------------------
function m_salir_Callback(hObject, eventdata, handles)
% hObject    handle to m_salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Cerramos todo abriendo una question box
selection = questdlg(['¿Quiere cerrar ' get(handles.GUI_modulador,'Name') '?'],...
                  ['Close ' get(handles.GUI_modulador,'Name') '...'],...
                  'Si','No','Si');
if strcmp(selection,'No')
   return;
end

delete(handles.GUI_modulador)

% --------------------------------------------------------------------
function m_ventanas_Callback(hObject, eventdata, handles)
% hObject    handle to m_ventanas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function m_entrada_Callback(hObject, eventdata, handles)
% hObject    handle to m_entrada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GUI_entrada

% --------------------------------------------------------------------
function m_modulador_Callback(hObject, eventdata, handles)
% hObject    handle to m_modulador (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GUI_modulador
```

```
% --------------------------------------------------------------------
function m_salida_Callback(hObject, eventdata, handles)
% hObject    handle to m_salida (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GUI_salida

% --------------------------------------------------------------------
function m_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to m_ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function documentacion_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to documentacion_ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

!ayuda.htm
```

```
function varargout = GUI_salida(varargin)
% GUI_SALIDA M-file for GUI_salida.fig
%      GUI_SALIDA, by itself, creates a new GUI_SALIDA or raises the existing
%      singleton*.
%
%      H = GUI_SALIDA returns the handle to a new GUI_SALIDA or the handle to
%      the existing singleton*.
%
%      GUI_SALIDA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_SALIDA.M with the given input arguments.
%
%      GUI_SALIDA('Property','Value',...) creates a new GUI_SALIDA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_salida_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_salida_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_salida

% Last Modified by GUIDE v2.5 03-Dec-2003 23:49:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_salida_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_salida_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before GUI_salida is made visible.
function GUI_salida_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_salida (see VARARGIN)

% Choose default command line output for GUI_salida
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI_salida wait for user response (see UIRESUME)
% uiwait(handles.GUI_salida);
```

```
%cargamos sd_salida_aux para comprobar el estado de s_reset
load sd_salida_aux
if(s_reset==0); % si no ha sido reseteada
   if(get(handles.s_fft,'value')) %si esta marcada la FFT
      s_fft_Callback(hObject, eventdata, handles); %realiza la FFT de la salida
   elseif (get(handles.s_psd,'value'))%si esta marcada la PSD
      s_psd_Callback(hObject, eventdata, handles);%realiza la PSD de la salida
   elseif (get(handles.s_t,'value'))%si esta marcada la representacion temporal
      s_t_Callback(hObject, eventdata, handles);%realiza la representacion en tiempo
   else
      %Fijamos por defecto la psd
      set(handles.s_t, 'Value', 0);
      set(handles.s_fft, 'Value', 0);
      set(handles.s_psd, 'Value', 1);
      s_psd_Callback(hObject, eventdata, handles);%realiza la PSD de la salida
   end

   load sd_salida
   load sd_entrada
   load sd_arquitectura

   %calculamos SNR

   if(get(handles.BW_fijo,'value')==1)
      BW_fijo_Callback(hObject, eventdata, handles);
   else
      BW_caida_Callback(hObject, eventdata, handles);
   end
end

handles.flag=0;%indica que no hay barra de herramientas activa
guidata(hObject,handles);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_salida_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in simular_pushbutton.
function simular_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to simular_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%cargamos los valores configurados en la entrada
load sd_entrada;
%cargamos el modulador
load sd_arquitectura;%arquitectura del modulador
load sd_matrizG;%valor de los coeficientes
%cargamos el archivo de variables auxiliares
load sd_salida_aux;
%cargamos posibles no linealidades
load sd_nolineal;

%Simulamos variaciones en los coeficientes de los integradores
G_var=G+dt_var_g*randn(size(G))+m_var_g;
if(corregir_canc==1)%si queremos que los coeficientes de cancelacion se modifiquen con las variaciones
   G=G_var;%los coeficientes de cancelacion tendran en cuenta las variaciones ya que a cancela le pasamos G
end



%Realizamos la modulacion sigma-delta en cascada
salida_precanc=cascada(orden,x,G_var,cuant,sat,coef_perdidas);%salida antes de la cancelacion de ruido
%Realizamos la cancelacion analogica del ruido
salida_canc=cancela(salida_precanc,orden,G);%salida despues de la cancelacion de ruido


%salvamos ambas salidas
save sd_salida salida_canc salida_precanc;


%calculamos SNR
%procede a enventanar
n_vent=get(handles.s_vent,'value'); %captura la ventana seleccionada
salida=ventana(salida_canc,n_vent);
%realizamos ahora la FFT
[salida,f]=sd_fft(salida,Fs);
S_dB=20*log10(abs(salida));
```

```
%al simular por defecto solo mostramos la salida del modulador completo
salidas_plot=length(orden);%salidas_plot indica que salidas se representaran
%salvamos las variables auxiliares
save sd_salida_aux salidas_plot s_reset;

if(get(handles.BW_fijo,'value')==1)
   BW_fijo_Callback(hObject, eventdata, handles)
else
   BW_caida_Callback(hObject, eventdata, handles)
end

%Representamos graficamente la simulacion
if(get(handles.s_fft,'value')) %si esta marcada la FFT
   s_fft_Callback(hObject, eventdata, handles); %realizamos la FFT
elseif (get(handles.s_psd,'value'))%si esta marcada la PSD
   s_psd_Callback(hObject, eventdata, handles);%realizamos la PSD
elseif (get(handles.s_t,'value'))%si esta marcada la representacion temporal
   s_t_Callback(hObject, eventdata, handles);%representamos en el dominio del tiempo
else
   %Fijamos por defecto la psd
   set(handles.s_t, 'Value', 0);
   set(handles.s_fft, 'Value', 0);
   set(handles.s_psd, 'Value', 1);
   s_psd_Callback(hObject, eventdata, handles);%realizamos la PSD
end


% --- Executes during object creation, after setting all properties.
function s_vent_CreateFcn(hObject, eventdata, handles)
% hObject    handle to s_vent (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes on selection change in s_vent.
function s_vent_Callback(hObject, eventdata, handles)
% hObject    handle to s_vent (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns s_vent contents as cell array
%        contents{get(hObject,'Value')} returns selected item from s_vent

%Cuando se da algun cambio en el enventanado, representamos la salida
%automaticamente
if(get(handles.s_fft,'value')) %si esta marcada la FFT
   s_fft_Callback(hObject, eventdata, handles); %realiza la FFT
elseif (get(handles.s_psd,'value'))%si esta marcada la PSD
   s_psd_Callback(hObject, eventdata, handles); %realiza la PSD
elseif (get(handles.s_t,'value')) %si esta marcado el dominio temporal
   s_t_Callback(hObject, eventdata, handles); %representa en el tiempo
end

load sd_salida
load sd_salida_aux
load sd_entrada
load sd_arquitectura
%calculamos SNR
if(get(handles.BW_fijo,'value')==1)
   BW_fijo_Callback(hObject, eventdata, handles);
else
   BW_caida_Callback(hObject, eventdata, handles);
end

% --- Executes on button press in s_fft.
function s_fft_Callback(hObject, eventdata, handles)
% hObject    handle to s_fft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_fft

%Ajustamos los marcadores mutuamente excluyentes
set(handles.s_t, 'Value', 0);
set(handles.s_fft, 'Value', 1); %marcamos la FFT
set(handles.s_psd, 'Value', 0);
```

```
%Calculamos la fft

%cargamos datos
load sd_entrada
load sd_salida
load sd_salida_aux
%realizamos el enventanado
n_vent=get(handles.s_vent,'value'); %captura la ventana seleccionada
salida_vent=ventana(salida_canc,n_vent); %procede a enventanar
%realizamos ahora la FFT
[salida_fft,F]=sd_fft(salida_vent,Fs);

%representamos la FFT

total_plots=length(salidas_plot); %numero total de salidas a representar
%generamos la cadena de caracteres con instruccion para representar las salidas
s_plot=['plot(F,20*log10(abs(salida_fft(',int2str(salidas_plot(1)),',:)))'];
for(i=2:1:total_plots)
   s_plot=[s_plot,',F,20*log10(abs(salida_fft(',int2str(salidas_plot(i)),',:)))'];
end
s_plot=[s_plot,');'];
%seleccionamos los ejes adecuados
axes(handles.ejes_salida);
%calculamos los limites verticales de representacion
y_max=0;
y_min=0;
for(i=1:1:total_plots)
   temp1=max(20*log10(abs(salida_fft(salidas_plot(i),:))));
   if(temp1>y_max)
      y_max=temp1;
   end
   temp2=min(20*log10(abs(salida_fft(salidas_plot(i),:))));
   if(temp2<y_min)
      y_min=temp2;
   end
end
%evaluamos la cadena generada anteriormente
eval(s_plot);%dibuja las salidas
%fijamos los ejes para mostrar solo las frecuencias positivas
axis([0,F(end),y_min, y_max]);
xlabel('Frecuencia');
ylabel('Magnitud [dB]');
set(handles.ejes_salida,'XMinorTick','on')
```

```
grid on

%Calculamos la SNR
if(get(handles.BW_fijo,'value')==1)
   BW_fijo_Callback(hObject, eventdata, handles);
else
   BW_caida_Callback(hObject, eventdata, handles);
end

% --- Executes on button press in s_psd.
function s_psd_Callback(hObject, eventdata, handles)
% hObject    handle to s_psd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_psd

set(handles.s_t, 'Value', 0);
set(handles.s_fft, 'Value', 0);
set(handles.s_psd, 'Value', 1); %marcamos la PSD

%Calculamos la psd

%cargamos datos
load sd_entrada; %es necesario conocer la frecuencia de muestreo Fs
load sd_salida
load sd_salida_aux %para ver las que tenemos que representar "salidas_plot"

%capturamos la NFFT, temaño de los intervalos de la PSD
NFFT=str2double(get(handles.s_nfft,'string'));
%Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
if (NFFT>length(salida_canc))%si se señala una NFFT mayor que la longitud de las secuencias
   disp('Warning: max(NFFT)=length(salida_canc)')
   NFFT=length(salida_canc)%en ese caso fijamos la particion al tamaño de la señal
   set(handles.s_nfft,'string',NFFT)%fijamos el valor maximo permitido
end

%comprobamos que ventana es la seleccionada
n_vent=get(handles.s_vent,'value');
if (n_vent==1), %calculamos la ventana correspondiente
   ventana = ['boxcar(',int2str(NFFT),')'];
elseif (n_vent==2),
   ventana = ['blackman(',int2str(NFFT),')'];
```

```
elseif (n_vent==3),
   ventana = ['hanning(',int2str(NFFT),')'];
elseif (n_vent==4),
   ventana = ['hamming(',int2str(NFFT),')'];
end;

%realizamos la PSD con los datos obtenidos
[salida_psd,F]=sd_psd(salida_canc,NFFT,Fs,eval(ventana));

%Representamos la PSD graficamente

%calculamos el numero total de salidas a representar
total_plots=length(salidas_plot);
%generamos la cadena de caracteres con la instruccion para representar
s_plot=['plot(F,10*log10(salida_psd(',int2str(salidas_plot(1)),',:))'];
for(i=2:1:total_plots)
   s_plot=[s_plot,',F,10*log10(salida_psd(',int2str(salidas_plot(i)),',:))'];
end
s_plot=[s_plot,');'];
%seleccionamos los ejes apropiados
axes(handles.ejes_salida);
%evaluamos la instruccion de la cadena de caracteres
eval(s_plot);
xlabel('Frecuencia');
ylabel('Magnitud [dB]');
set(handles.ejes_salida,'XMinorTick','on')
grid on

%Calculamos la SNR
if(get(handles.BW_fijo,'value')==1)
   BW_fijo_Callback(hObject, eventdata, handles);
else
   BW_caida_Callback(hObject, eventdata, handles);
end

% --- Executes on button press in s_t.
function s_t_Callback(hObject, eventdata, handles)
% hObject    handle to s_t (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of s_t
```

```
set(handles.s_t, 'Value', 1); %marcamos la representacion temporal
set(handles.s_fft, 'Value', 0);
set(handles.s_psd, 'Value', 0);

%cargamos datos
load sd_salida
load sd_entrada
load sd_salida_aux

%calculamos el numero de salidas a representar
total_plots=length(salidas_plot);
%generamos la cedena de caracteres con la instruccion para representar
s_plot=['plot(kTs,salida_canc(',int2str(salidas_plot(1)),',:),''X'''];
for(i=2:1:total_plots)
    s_plot=[s_plot,',kTs,salida_canc(',int2str(salidas_plot(i)),',:),''X'''];
end
s_plot=[s_plot,');'];
%seleccionamos los ejes adecuados
axes(handles.ejes_salida);
%evaluamos la intruccion de la cadena de caracteres
eval(s_plot);
xlabel('Tiempo [seg.]');
ylabel('Amplitud');
set(handles.ejes_salida,'XMinorTick','on')
grid on

% --- Executes on button press in salidas_pushbutton.
function salidas_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to salidas_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%abrimos el configurador que permite seleccionar las salidas a representar
configurador_salidas

% --------------------------------------------------------------------
function s_archivo_Callback(hObject, eventdata, handles)
% hObject    handle to s_archivo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
```

```
function s_ventanas_Callback(hObject, eventdata, handles)
% hObject    handle to s_ventanas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --------------------------------------------------------------------
function s_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to s_ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function ventana_s_entrada_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_s_entrada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GUI_entrada

% --------------------------------------------------------------------
function ventana_s_modulador_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_s_modulador (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
GUI_modulador

% --------------------------------------------------------------------
function ventana_s_salida_Callback(hObject, eventdata, handles)
% hObject    handle to ventana_s_salida (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function s_nfft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to s_nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function s_nfft_Callback(hObject, eventdata, handles)
% hObject    handle to s_nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of s_nfft as text
%        str2double(get(hObject,'String')) returns contents of s_nfft as a double



% --------------------------------------------------------------------
function s_cargar_Callback(hObject, eventdata, handles)
% hObject    handle to c_cargar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file = uigetfile('*.mat','Cargar simulacion');%recogemos es archivo a cargar
if ~isequal(file, 0)%si se ha seleccionado un archivo
    datos=open (file);%cargamos los valores de las variables
    %fijamos en la ventana los valores correspondientes
    salida_canc=datos.salida_canc;
    salida_precanc=datos.salida_precanc;
    Fs=datos.Fs;
    kTs=datos.kTs;
    save sd_salida salida_canc salida_precanc Fs kTs; %las dos ultimas las necesitamos para
representar;


    %modulador
    cuant=datos.cuant;
    orden=datos.orden;
    sat=datos.sat;
    G=datos.G;
    slider_value=datos.slider_value;
    m_reset=0;
    save sd_arquitectura orden cuant sat m_reset;
    g_reset=0;
```

```
    save sd_matrizG G g_reset slider_value;
    s_reset=1;
    salidas_plot=length(orden);
    save sd_salida_aux s_reset salidas_plot;
    %no linealidades
    jitter_act=datos.jitter_act;
    var_g_act=datos.var_g_act;
    perdidas_act=datos.perdidas_act;
    dt_jitter=datos.dt_jitter;
    m_jitter=datos.m_jitter;
    dt_var_g=datos.dt_var_g;
    m_var_g=datos.m_var_g;
    corregir_canc=datos.corregir_canc;
    coef_perdidas=datos.coef_perdidas;
    save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g corregir_canc
coef_perdidas;
    GUI_modulador('varargin')
    %entrada
    Fx=datos.Fx;
    M=datos.M;
    n_ciclos=datos.n_ciclos;
    B=datos.B;
    A=datos.A;
    save sd_entrada  A n_ciclos M Fx B;
    %generamos la entrada propiamente dicha


    GUI_entrada
    %GUI_entrada('entrada_Callback',hObject, eventdata, handles);



    %representamos la salida
    if(get(handles.s_fft,'value'))
       s_fft_Callback(hObject, eventdata, handles);
    elseif (get(handles.s_psd,'value'))
       s_psd_Callback(hObject, eventdata, handles);
    elseif (get(handles.s_t,'value'))
       s_t_Callback(hObject, eventdata, handles);
    else
       %Fijamos por defecto la psd
       set(handles.s_t, 'Value', 0);
       set(handles.s_fft, 'Value', 0);
       set(handles.s_psd, 'Value', 1);
       s_psd_Callback(hObject, eventdata, handles);
```

```
    end
end

% --------------------------------------------------------------------
function s_guardar_Callback(hObject, eventdata, handles)
% hObject    handle to s_guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file=uiputfile('*.mat','Guardar simulacion');
% Get user input from GUI
load sd_entrada;
load sd_arquitectura;
load sd_matrizG;
load sd_salida ;
load sd_nolineal;

%Grabamos las variables de interes
save (file, 'salida_canc', 'salida_precanc','x', 'Fs', 'kTs', 'Fx', 'M', 'n_ciclos', 'B', 'A','orden','cuant','sat','G','slider_value','jitter_act','var_g_act','perdidas_act','dt_jitter','m_jitter','dt_var_g','m_var_g','corregir_canc','coef_perdidas'); %, 'NFFT'

% --------------------------------------------------------------------
function s_imprimir_Callback(hObject, eventdata, handles)
% hObject    handle to s_imprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

printdlg(gcbf)

function s_previsualizar_Callback(hObject, eventdata, handles)
% hObject    handle to s_imprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

printpreview(gcbf)

% --------------------------------------------------------------------
function s_salir_Callback(hObject, eventdata, handles)
% hObject    handle to s_salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Cerramos todo abriendo una question box
```

```
selection = questdlg(['¿Quiere cerrar ' get(handles.GUI_salida,'Name') '?'],...
                     ['Close ' get(handles.GUI_salida,'Name') '...'],...
                     'Si','No','Si');
if strcmp(selection,'No')
   return;
end

delete(handles.GUI_salida)


% --------------------------------------------------------------------
function s_herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to s_herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function activa_herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to activa_herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if(handles.flag==0)%si no hay barra activada
   fig=gcf;
   htoolbar.htoolbar = uitoolbar('Parent',fig);%hObject);
   % Render Print buttons (Print, Print Preview)
   htoolbar.hprintbtns = render_sptprintbtns(htoolbar.htoolbar);
   % Render the annotation buttons (Edit Plot, Insert Arrow, etc)
   htoolbar.hscribebtns = render_sptscribebtns(htoolbar.htoolbar);
   % Render the zoom buttons
   htoolbar.hzoombtns = render_zoombtns(htoolbar.htoolbar);
   %Creamos una clase en la estructura para almacenar el handles de herramientas
   handles.herramientas=htoolbar.htoolbar;
   handles.flag=1;%indica que hemos activado la barra de herramientas
   guidata(hObject,handles);
   %herramientas=1;
   %save aux_herramientas htoolbar herramientas;
end

% --------------------------------------------------------------------
function desactiva_herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to desactiva_herramientas (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set (handles.herramientas,'visible','off');
handles.flag=0;%indica que no hay barra de herramientas activa
guidata(hObject,handles);


% --------------------------------------------------------------------
function zoomin_Callback(hObject, eventdata, handles)
% hObject    handle to zoomin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
zoom on

% --------------------------------------------------------------------
function zoomout_Callback(hObject, eventdata, handles)
% hObject    handle to zoomout (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

zoom out
zoom off


% --- Executes during object creation, after setting all properties.
function BW_signal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BW_signal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function BW_signal_Callback(hObject, eventdata, handles)
% hObject    handle to BW_signal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of BW_signal as text
%        str2double(get(hObject,'String')) returns contents of BW_signal as a double

BW_fijo_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function dB_caida_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dB_caida (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function dB_caida_Callback(hObject, eventdata, handles)
% hObject    handle to dB_caida (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dB_caida as text
%        str2double(get(hObject,'String')) returns contents of dB_caida as a double

BW_caida_Callback(hObject, eventdata, handles)

% --- Executes on button press in BW_fijo.
function BW_fijo_Callback(hObject, eventdata, handles)
% hObject    handle to BW_fijo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of BW_fijo
set(handles.BW_fijo,'value',1);
set(handles.BW_caida,'value',0);
```

```
set(handles.BW_signal,'enable','on');
set(handles.dB_caida,'enable','off');

load sd_salida_aux
load sd_salida
load sd_entrada
load sd_arquitectura
%calculamos SNR

%procede a enventanar
n_vent=get(handles.s_vent,'value'); %captura la ventana seleccionada
if(get(handles.s_fft,'value')==1)
   salida=ventana(salida_canc,n_vent);
   [salida,f]=sd_fft(salida,Fs);
   S_dB=20*log10(abs(salida));
else
   %capturamos la NFFT, temaño de los intervalos de la PSD
   NFFT=str2double(get(handles.s_nfft,'string'));
   %Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
   if (NFFT>length(salida_canc))%si se señala una NFFT mayor que la longitud de las secuencias
      disp('Warning: max(NFFT)=length(salida_canc)')
      NFFT=length(salida_canc)%en ese caso fijamos la particion al tamaño de la señal
      set(handles.s_nfft,'string',NFFT)%fijamos el valor maximo permitido
   end

   %comprobamos que ventana es la seleccionada
   if (n_vent==1), %calculamos la ventana correspondiente
      ventana_selecc = ['boxcar(',int2str(NFFT),')'];
   elseif (n_vent==2),
      ventana_selecc = ['blackman(',int2str(NFFT),')'];
   elseif (n_vent==3),
      ventana_selecc = ['hanning(',int2str(NFFT),')'];
   elseif (n_vent==4),
      ventana_selecc = ['hamming(',int2str(NFFT),')'];
   end;

  [salida,f]=sd_psd(salida_canc,NFFT,Fs,eval(ventana_selecc));
  S_dB=10*log10(abs(salida));
end

BW=str2num(get(handles.BW_signal,'string'));%capturamos el ancho de banda introducido
BW_signal=BW*ones(1,length(orden));%el ancho de banda sera el mismo para todos los moduladores
Fmin=Fx-BW/2;%limite inferior del BW
```

```
Fmax=Fx+BW/2;%limite superior del BW
SNR=calcula_SNR(S_dB,Fx,Fs,B,Fmin,Fmax,f); %Calculamos la SNR

%representa, modulador, SNR y ancho de banda de señal
for(i=1:1:length(salidas_plot))
    cadena=['set(handles.m_',int2str(i),',''string'',''',int2str(orden(1:salidas_plot(i))),''');'];
    eval(cadena);
    cadena=['set(handles.SNR_',int2str(i),',''string'',''',num2str(SNR(salidas_plot(i))),' dB'');'];
    eval(cadena);
    cadena=['set(handles.BW_',int2str(i),',''string'',''',num2str(BW_signal(salidas_plot(i))),' Hz'');'];
    eval(cadena);
end
for(i=length(salidas_plot)+1:1:10)
   cadena=['set(handles.m_',int2str(i),',''string'','''');'];
   eval(cadena);
   cadena=['set(handles.SNR_',int2str(i),',''string'','''');'];
   eval(cadena);
   cadena=['set(handles.BW_',int2str(i),',''string'','''');'];
   eval(cadena);
end

% --- Executes on button press in BW_caida.
function BW_caida_Callback(hObject, eventdata, handles)
% hObject    handle to BW_caida (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of BW_caida

set(handles.BW_fijo,'value',0);
set(handles.BW_caida,'value',1);
set(handles.BW_signal,'enable','off');
set(handles.dB_caida,'enable','on');

load sd_salida
load sd_salida_aux
load sd_entrada
load sd_arquitectura
%calculamos SNR

%procede a enventanar
n_vent=get(handles.s_vent,'value'); %captura la ventana seleccionada
if(get(handles.s_fft,'value')==1)
```

```
    salida=ventana(salida_canc,n_vent);
    [salida,f]=sd_fft(salida,Fs);
    S_dB=20*log10(abs(salida));
else
    %capturamos la NFFT, temaño de los intervalos de la PSD
    NFFT=str2double(get(handles.s_nfft,'string'));
    %Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
    if (NFFT>length(salida_canc))%si se señala una NFFT mayor que la longitud de las secuencias
        disp('Warning: max(NFFT)=length(salida_canc)')
        NFFT=length(salida_canc)%en ese caso fijamos la particion al tamaño de la señal
        set(handles.s_nfft,'string',NFFT)%fijamos el valor maximo permitido
    end

    %comprobamos que ventana es la seleccionada
    if (n_vent==1), %calculamos la ventana correspondiente
        ventana_selecc = ['boxcar(',int2str(NFFT),')'];
    elseif (n_vent==2),
        ventana_selecc = ['blackman(',int2str(NFFT),')'];
    elseif (n_vent==3),
        ventana_selecc = ['hanning(',int2str(NFFT),')'];
    elseif (n_vent==4),
        ventana_selecc = ['hamming(',int2str(NFFT),')'];
    end;

   [salida,f]=sd_psd(salida_canc,NFFT,Fs,eval(ventana_selecc));
   S_dB=10*log10(abs(salida));
end

caida=str2num(get(handles.dB_caida,'string'));%capturamos la caida en dB
for(i=1:1:length(orden))
     [BW_signal(i),Fmin,Fmax]=calcula_BW(S_dB(i,:),f, Fx,caida);%calculamos el BW
     SNR(i)=calcula_SNR(S_dB(i,:),Fx,Fs,B,Fmin,Fmax,f);%calculamos el SNR
end

%representa, modulador, SNR y ancho de banda de señal
for(i=1:1:length(salidas_plot))
    cadena=['set(handles.m_',int2str(i),',''string'',''',int2str(orden(1:salidas_plot(i))),''');'];
    eval(cadena);
    cadena=['set(handles.SNR_',int2str(i),',''string'',''',num2str(SNR(salidas_plot(i))),' dB'');'];
    eval(cadena);
    cadena=['set(handles.BW_',int2str(i),',''string'',''',num2str(BW_signal(salidas_plot(i))),' Hz'');'];
    eval(cadena);
end
```

```
for(i=length(salidas_plot)+1:1:10)
   cadena=['set(handles.m_',int2str(i),',''string'','''');'];
   eval(cadena);
   cadena=['set(handles.SNR_',int2str(i),',''string'','''');'];
   eval(cadena);
   cadena=['set(handles.BW_',int2str(i),',''string'','''');'];
   eval(cadena);
end


% --------------------------------------------------------------------
function documentacion_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to documentacion_ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

!ayuda.htm
```

```
function [modulado,intermedio]=m1(x,z,g11,g12,g1r,levels,sat,coef_perdidas,ult);
% Realiza una modulación sigma-delta de primer orden sobre un vector de entrada.
% Permite fijar las características del cuantizador.
% Devuelve la señal modulada y la entrada del cuantizador (permite calcular el error de cuantización)
%
%PARAMETROS DE ENTRADA
%         x: vector de entrada 1 a modular
%         z: vector de entrada 2 a modular (se resta a la anterior)
%         g11: coeficiente 'g' que multiplica a la entrada 1
%         g12: coeficiente 'g' que multiplica a la entrada 2
%    g1r: coeficiente 'g' que multiplica a la realimentación
%         levels: número de escalones de cuantización del cuantizador
%         sat: nivel de saturación del cuantizador
%    coef_perdidas:coeficiente de perdidas en integradores. Comprendido entre 0 y 1
%
%PARAMETROS DE SALIDA
%         modulado: vector salida del modulador. Señal modulada.
%         intermedio: vector intermedio con las entradas al cuantizador.
%
% Ver tambien configurador_g, cascada, m2, cuantiza, cancela
%

%si solo nos dan la entrada a modular (un unico parametro)
if nargin==1
```

```
    z=zeros(1,length(x));
    g12=0
    g11=1;
    g1r=g11;
    coef_perdidas=0;
    ult=0;
end

%Modulacion sigma-delta de primer orden

%inicializamos los parametros
a(1)=g11*x(1)-g12*z(1);
b(1)=0;
y(1)=0;
%modulamos
for(n=2:1:length(x))
   b(n)=a(n-1)+(1-coef_perdidas)*b(n-1);%salida del filtro
   y(n)=cuantiza(b(n),levels,2*sat);%cuantizamos b(n), con "levels" niveles, y rango de cuantizacion igual al doble del nivel de saturacion
   a(n)=g11*x(n)-g12*z(n)-g1r*y(n);%realimentacion
end
modulado=y;
intermedio=b;%este valor es necesario para el siguiente modulo en cascada. Entrada al cuantizador.
```

```
function [modulado,intermedio]=m2(x,z,g11,g12,g1r,g21,g22,levels,sat,coef_perdidas,ult);
% Realiza una modulación sigma-delta de segundo orden sobre un vector de entrada.
% Permite fijar las características del cuantizador.
% Incluye modelado de perdidas en los integradores
% Devuelve la señal modulada y la entrada del cuantizador (permite calcular el error de cuantización)
%
%PARAMETROS DE ENTRADA
%         x: vector de entrada 1 a modular
%         z: vector de entrada 2 a modular (se resta a la anterior)
%         g11: coeficiente 'g' que multiplica a la entrada 1 del primer integrador
%         g12: coeficiente 'g' que multiplica a la entrada 2 del primer integrador
%    g1r: coeficiente 'g' que multiplica a la realimentación
%    g21: coeficiente 'g' que multiplica a la entrada 1 del segundo integrador
%         g22: coeficiente 'g' que multiplica a la entrada 2 del segundo
%              integrador (realimentada)
%         levels: número de escalones de cuantización del cuantizador
%         sat: nivel de saturación del cuantizador
%    coef_perdidas:coeficiente de perdidas en integradores. Comprendido entre 0 y 1
```

```
%
%PARAMETROS DE SALIDA
%       modulado: vector salida del modulador. Señal modulada.
%       intermedio: vector intermedio con las entradas al cuantizador.
%
% Ver tambien configurador_g, cascada, m1, cuantiza, cancela
%

%si solo nos dan la entrada a modular (un unico parametro)
if nargin==1
   z=zeros(1,length(x));
   g12=0
   g11=1;
   g1r=g11;
   g21=1;
   g22=2*g1r*g21;
   coef_perdidas=0;
end



%Modulacion sigma-delta de segundo orden

%inicializamos los parametros
a1(1)=g11*x(1)-g12*z(1);
a2(1)=0;
b1(1)=0;
b2(1)=0;
y(1)=0;
%modulamos
for(n=2:1:length(x))
  b2(n)=a2(n-1)+(1-coef_perdidas)*b2(n-1);%salida del filtro 2
  y(n)=cuantiza(b2(n),levels,2*sat);%cuantizamos b(n), con "levels" niveles, y rango de cuantizacion igual al doble del nivel de saturacion
  b1(n)=a1(n-1)+(1-coef_perdidas)*b1(n-1);%salida del filtro 1
  a2(n)=g21*b1(n)-g22*y(n);%realimentacion 2
  a1(n)=g11*x(n)-g12*z(n)-g1r*y(n);%realimentacion 1
end
modulado=y;
intermedio=b2;%este valor hace falta para el siguiente modulo en cascada
```

```
function [muestreada,Fs,kTs] =muestrea (Fx,M,n_ciclos,B,A,m,v)
%DESCRIPCION: muestrea una señal sinusoidal para los parametros de entrada
```

```
%PARAMETROS ENTRADA:
%       Fx: Frecuencia del tono.
%       M: Tasa de sobremuestreo.
%       n_ciclos: Número de periodos del tono.
%       B: Ancho de banda.
%       A: Amplitud.
%   m: media de efecto jitter
%   v: varianza de efecto jitter
%PARAMETROS SALIDA:
%       Muestreada: secuencia resultante.
%       Fs: Frecuencia de muestreo.
%       kTs: Eje para representación temporal.
%
%OBSERVACION: si no se pasa la media y la varianza no hay efecto jitter

if (nargin==5)
   m=0;
   v=0;
end

Fn=2*B; %Frecuencia de Niquist respecto a B
Fs=M*Fn; %Frecuencia de sobremuestreo
Ts=(1/Fs);%Periodo de sobremuestreo
kTs=[0:Ts:(n_ciclos*(1/Fx))];%generamos el vector correspondiente al eje x en representacion temporal
kTs=kTs+sqrt(v)*randn(size(kTs))+m;%efecto jitter
muestreada=A*sin(2*pi*Fx*kTs);%generamos la frecuencia para el tono deseado
```

```
function reset_G(h_slider,h_edit)
% "Callback" del botón "reset" la interfaz "configurador_g".
% Resetea el configurador g y fija a cero todos los valores
% PARAMETROS DE ENTRADA:
%       h_slider: vector con los manejadores de los deslizadores
%       h_edit: vector con los manejadores de las cajas de texto editables
%
% Ver tambien: actualiza_g, acepta_G.m, cancela_G.m, actualiza_G.m, g_edit.m, g_slider.m

g_reset=1 %activamos la variable de control de reset
save sd_matrizG g_reset; %guardamos la variable de control en sd_matrizG
for(i=1:1:length(h_slider))
   set(h_slider(i),'value',0); %fijamos a 0 todos los deslizadores
   set(h_edit(i),'string','0'); %fijamos a 0 todas las cajas de texto editables
```

```
end
```

```
disp('Simulador de convertidores Sigma-Delta cascada tiempo discreto')
disp('Para obtener ayuda escribir: "!ayuda.htm" en la linea de comandos de MATLAB')

warning off MATLAB:divideByZero;

%Inicializamos variables de entrada
Fx = 0;%frecuencia de la señal sinusoidal de entrada
M = 0;%tasa de sobremuestreo
n_ciclos =0;%numero de ciclos de la señal sinusoidal de entrada
B = 0;%ancho de banda
A = 0;%amplitud de la señal sinusoidal de entrada
save sd_entrada A n_ciclos M Fx B;

%reseteamos el configurador de ganancias de los amplificadores
g_reset=1;
save sd_matrizG g_reset;

%reseteamos la salida
s_reset=1;
save sd_salida_aux s_reset;

%reseteamos el configurador de arquitectura
m_reset=1;
save sd_arquitectura m_reset;

%inicializamos las no-linealidades
jitter_act=0;
var_g_act=0;
perdidas_act=0;
dt_jitter=0;
m_jitter=0;
dt_var_g=0;
m_var_g=0;
coef_perdidas=0;
save sd_nolineal jitter_act var_g_act perdidas_act dt_jitter m_jitter dt_var_g m_var_g coef_perdidas;

%Abrimos las ventanas de interfaz de usuario
GUI_entrada %Interfaz de entrada
GUI_modulador %interfaz de modulador
GUI_salida %interfaz de salida
```

```
function [Yw,F]=sd_fft(Y,Fs)
%Aplica la FFT a las filas de una matriz de entrada
%PARAMETROS DE ENTRADA:
%       Y: matriz de vectores por filas.
%       Fs: frecuencia de muestreo.
%PARAMETROS DE SALIDA:
%       Yw: matriz de transformadas PSD por filas.
%       F: eje de frecuencias.
%
%Ver tambien: transforma.m

aux=size(Y);
n_etapas=aux(1); %comprobamos el numero de etapas de la cascada
%realizamos la FFT por filas
for (i=1:1:n_etapas)
Yw(i,:)=transforma(Y(i,:)); %transformamos cada salida y las almacenamos por filas
end
%calculamos el eje x de frecuencias para la reresentacion
expo=ceil(log2(length(Y)));%Por defecto tomamos N como la potencia de 2 >= que la longitud de la secuencia para agilizar la fft
N=2^expo;
F=[-Fs/2:(Fs/N):(Fs/2)*((N-1)/N)];%de -pi a pi
```

```
function [Yw,F]=sd_psd(Y,NFFT,Fs,ventana)
%Aplica la PSD a las filas de una matriz de entrada
%PARAMETROS DE ENTRADA:
%       Y: matriz de vectores por filas.
%       NFFT
%       Fs: frecuencia de muestreo.
%       Ventana: enventanado a realizar.
%PARAMETROS DE SALIDA:
%       Yw: matriz de transformadas PSD por filas.
%       F: eje de frecuencias.

aux=size(Y);
n_etapas=aux(1); %comprobamos el numero de etapas o vectores

%Comprobamos primero si las partes en que vamos a dividir la señal son mayores que la propia señal
if (NFFT>aux(2))%si se señala una NFFT mayor que la longitud de las secuencias
   NFFT=aux(2);%en ese caso fijamos la particion al tamaño de la señal
```

```
end


%Aplicamos la PSD por filas
for (i=1:1:n_etapas)
[Yw(:,i),F]=psd(Y(i,:),NFFT,Fs,ventana); %transformamos cada fila de entrada y las almacenamos por filas en la salida
end
Yw=Yw';% trasponemos para organizar las salidas por filas
F=F';% trasponemos para devolver un vector fila
```

```
function transformada=transforma(secuencia,N);
%Realiza la transformada de Fourier discreta FFT de N puntos de un vector.
%  Si sólo se pasa un argumento, toma N como la potencia de 2 mayor o igual que la longitud de la secuencia para agilizar la FFT.
%  Si N es menor que la longitud de la secuencia, se trunca la secuencia.
%  Si N es mayor que la longitud de la secuencia; se rellena con ceros.
%  Se devuelve la FFT centrada en cero.
%PARAMERTOS DE ENTRADA
%       secuencia: vectores al que realizar la FFT.
%       N: Número de puntos de los que se desea realizar la transformada.
%PARAMETROS DE SALIDA
%       transformada: Transformada de Fourier discreta FFT de N puntos de secuencia.
%
% Ver tambien: sd_fft.m


if nargin==1
   expo=ceil(log2(length(secuencia)));%Por defecto tomamos N como la potencia de 2 >= que la longitud de la secuencia para agilizar la fft
   N=2^expo;
end
if(N<length(secuencia));
   disp('Warning: Se esta introduciendo efecto aliasing');
   disp('N<length(secuencia)');
else if (N>length(secuencia));
      disp('Warning: Rellenando con ceros. Zero padding');
      disp('N>length(secuencia)');
   end
end
transformada=(1/length(secuencia))*fftshift(fft(secuencia,N));%modulo de la fft de N puntos, centrando en w=0
%hemos escalado en magnitud para que no sea funcion de la longitud de 'secuencia'
```

```
function salida=ventana(entrada,v);
%Aplica enventanado por filas a una matriz
%
%PARAMETROS DE ENTRADA_
%   entrada: matriz de vectores ordenados por filas
%   v: seleccion de ventana
%       v=1: rectangular
%       v=2: blackman
%       v=3: hanning
%       v=4: hamming
%
%PARAMETROS DE SALIDA
%   salida: matriz de vectores enventanados ordenados por filas
%

aux=size(entrada);
n_etapas=aux(1); %comprobamos el numero de etapas de la cascada

tam=length(entrada);
if (v==1), %calculamos la ventana correspondiente
   window = boxcar(tam);
elseif (v==2),
   window = blackman(tam);
elseif (v==3),
   window = hanning(tam);
elseif (v==4),
   window = hamming(tam);
end;

for (i=1:1:n_etapas)
salida(i,:)=entrada(i,:).*window'; %aplicamos la ventana a cada fila y lo almacenamos tb por filas
end
```