CAPÍTULO 1

Introducción

Índice:

1. Motivaciones

La primera pregunta a plantearnos antes de comenzar a desarrollar el proyecto es ¿por qué un Proyecto Fin de Carrera de convertidores analógico-digitales? Pues bien, dando por supuesto mi interés por la Electrónica, el mundo de las Telecomunicaciones está envuelto en un entorno de información tanto analógica como digital. Hoy día tendemos a digitalizarlo todo por sencillez de operación, aumento de resolución, rapidez... pero no podemos olvidarnos de la naturaleza analógica que nos rodea, y por ello, de la importancia de que siempre tendremos que realizar en algún momento en nuestro sistema una conversión.

La segunda pregunta que plantearía sería, ¿por qué una librería de convertidores analógico digitales?. Pues esta nos centra aún más en el proyecto. El estudio de sistemas de conversión varía dependiendo de la aplicación, es decir, dependiendo de la frecuencia de funcionamiento, de la resolución de nuestros convertidores, incluso del área de integración, nos interesará más o menos un determinado tipo de convertidor: sigma delta, pipeline, flash... Nuestro proyecto nos ofrecerá una gama de convertidores para simular sus características y ver su comportamiento ante la diversidad de no idealidades que afecten al sistema.

2. Estado del arte

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 analógica a digital está sujeta a un compromiso entre la velocidad y la resolución en los CA/D. Esto es, una arquitectura válida para alta frecuencia tendrá poca resolución 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 circuitería 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.

3. Objetivos

Los objetivos planteados para este proyecto *"Librerías para simulación de convertidores analógico-digitales"* es diseñar y modelar en Matlab el funcionamiento de los bloques que componen los diversos tipos de convertidores analógico-digitales importantes en el ámbito de las comunicaciones. Para ello, se ha desarrollado una **interfaz de librería dinámica**, que ayuda al usuario a acceder a cada uno de estos bloques y le permite mediante el **enlace virtual a otra interfaz** simular dichos bloques, visualizar ejemplos y crear sus propios diseños a partir de los ya realizados u otros propios del usuario. Para lograr esta librería se ha profundizado en el **estudio de los convertidores analógico-digitales**, desde un punto de vista tanto teórico como práctico, y de esta forma conseguir un **modelo de diseño propio en bloques de Simulink** para cada una de las funciones de los convertidores. Además, se han **diseñado ejemplos** para cada uno de ellos en aplicaciones actuales y se han obtenido resultado de las **simulaciones** realizadas.

Por tanto, este proyecto ha sido muy completo, ya que ha necesitado de conocimientos de varios ámbitos. Por un lado, a **nivel software** se ha desarrollado:

- programación de la interfaz de librería.
- programación de la comunicación virtual entre interfaces.
- programación de la funcionalidad de los bloques.

Y por otro lado conocimientos a **nivel hardware**:

- estudio de convertidores analógico-digitales.
- diseño y modelado de funcionalidades de convertidores analógico-digitales.
- investigación en el desarrollo de aplicaciones electrónicas actuales.
- visión paralela en el diseño con modelos desarrollados en VHDL.

Todos los objetivos alcanzados se los presentamos en los siguientes capítulos de este proyecto, mostrándoles en el siguiente apartado como están organizados.

4. Organización del proyecto

Los capítulos de los que consta este proyecto son los siguientes:

CAPÍTULO 1: INTRODUCCIÓN

Breve repaso por las motivaciones de la realización del proyecto y los objetivos que se han perseguido. Trata de dar una visión general del mismo.

CAPÍTULO 2: CONVERTIDORES A/D SIGMA DELTA

Revisión teórica de los convertidores Sigma Delta. Estudio de su estructura, funcionalidad y aplicaciones. Se revisan los tipos de no idealidades que afectan a este tipo de convertidores.

CAPÍTULO 3: CONVERTIDORES A/D PIPELINE

Estudio de los convertidores pipeline desde un punto de vista teórico y repasando los conceptos de su estructura. Enumeración de las no idealidades que afectan a este tipo de convertidores.

CAPÍTULO 4: CONVERTIDORES A/D FLASH

Repaso teórico del funcionamiento de los convertidores analógico digitales tipo flash. Profundización de los tipos de estructura que nos podemos encontrar y del proceso de calibración del mismo. No idealidades que pueden afectarles dependiendo de su estructura.

CAPÍTULO 5: INTERFAZ SIMCONVERTER

Presentación de la interfaz simconverter. Descripción de sus componentes y explicación del uso de la misma. Introducción de nuevos elementos.

CAPÍTULO 6: INTERFAZ LIBRARY

Presentación de la interfaz library. Descripción de su funcionalidad. Resumen de su estructura y programación.

CAPÍTULO 7: LIBRERÍAS

Descripción de los bloques desarrollados en la librería, modelo de cada uno de ellos y las variables necesarias para su simulación. Estructura organizada de los bloques en la librería para convertidores Sigma Delta, Flash y Pipeline. Desarrollo de una máscara para cada bloque que facilite su utilización. Enumeración de los ejemplos diseñados.

CAPÍTULO 8: SIMULACIONES

Simulaciones de los ejemplos diseñados más relevantes. Obtención de los resultados tras su simulación para convertidores Sigma Delta, Flash y Pipeline.

CAPÍTULO 9: CONCLUSIONES

Visión general del proyecto. Posibles líneas futuras a desarrollar.

CAPÍTULO 10: BIBLIOGRAFÍA

Documentación utilizada en la realización del proyecto. Artículos utilizados, páginas Web consultadas, tutoriales y libros consultados…

ANEXO 1: MANUAL DE USUARIO

Desarrollo de una documentación para que el usuario del programa pueda seguir paso a paso cómo puede utilizar el simulador y la librería.

ANEXO 2: MANUAL DE USUARIO AVANZADO

Desarrollo de una documentación para que un usuario avanzado con conocimiento de programación en Matlab y desarrollos de GUI, pueda ampliar la librería y hacerla más versátil o personal añadiendo nuevos modelos según sus necesidades.

ANEXO 3: CÓDIGO MATLAB DE LA INTERFAZ SIMCONVERTER

Código programado en Matlab de las funciones que han sido modificadas en la interfaz simconverter para mejorarlas y comunicarla con la interfaz library.

ANEXO 4: CÓDIGO MATLAB DE LA INTERFAZ LIBRARY

Código programado en Matlab para el desarrollo de la interfaz dinámica library.

CAPÍTULO 2

Convertidores A/D Sigma Delta

Índice:

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 circuitería analógica en contrapartida de una circuitería digital más compleja. Este intercambio es deseable de cara a tecnologías modernas sub-micra con fuentes de 3.3V, donde la compleja circuitería 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 la baja impedancia de salida del transistor.

- 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. Además, un muestreador y mantenedor (*sample and hold*) no suele ser necesario a la entrada de un convertidor A/D de sobremuestreo.

2. El convertidor Sigma Delta

Pasemos a estudiar en rasgos generales la estructura de un convertidor Sigma Delta (ΣΔ), para ello fijémonos en la Fig. 2.1 en la que presentamos el esquema de un modulador sigma delta:

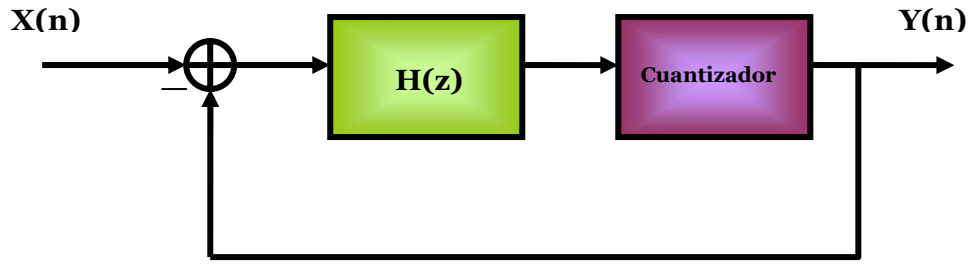


Fig. 3.1. Estructura del modulador Sigma Delta.

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.

Para verlo más claro, mostremos el ejemplo más simple que ilustra el concepto de modulación ΣΔ explicado. Se trata de un modulador ΣΔ de primer orden, paso de baja y tiempo discreto.

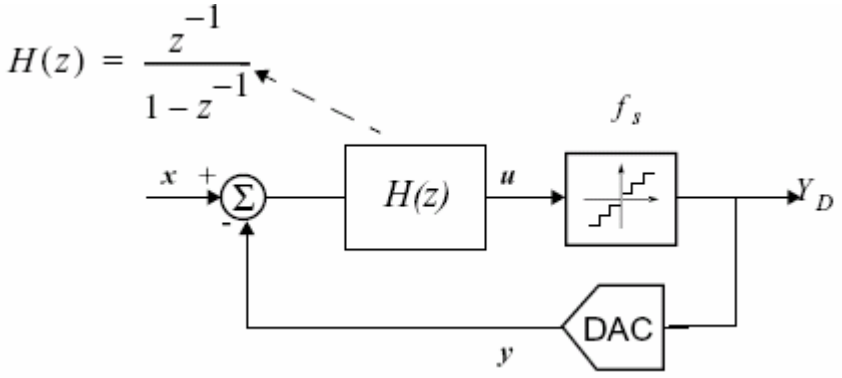


Fig. 3.2. Modulador Sigma Delta, primer orden, LP.

Y el diagrama de bloques del CAD Sigma Delta quedaría según la Fig.3.3 donde podemos comprobar que está formado por tres parte: anti-aliasing, modulador y decimador digital.

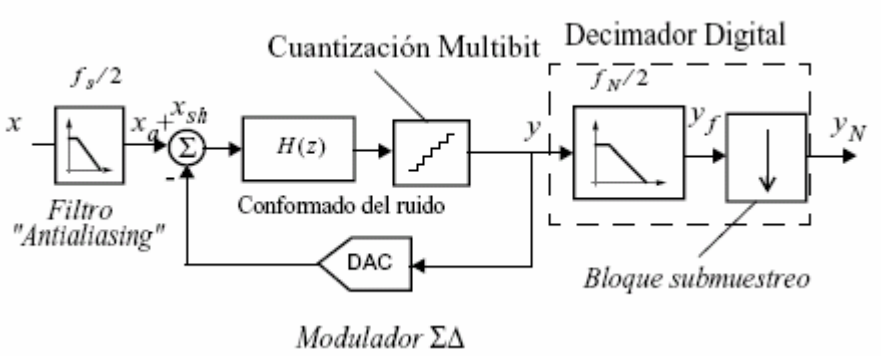


Fig. 3.3. CAD Sigma Delta de tiempo discreto.

El funcionamiento del convertidor es el siguiente: tras filtrarse la señal de entrada (se rechazan las componentes espectrales por encima de fs/2, con fs la frecuencia de muestreo utilizada), la señal es muestreada y cuantizada en el modulador. Este bloque filtra también el error de cuantización, conformando su densidad espectral de potencia de manera que la mayor parte de potencia que de fuera de la banda de la señal, donde es eliminada por el filtrado digital tras el modulador. Por último se decima la señal resultante tras el filtrado digital de forma que se reduce la frecuencia de muestreo a la frecuencia de

Nyquist de la señal de entrada original.

Una manera alternativa de implantar este tipo de CADs es mediante tiempo continuo. De este modo la Fig. 3.4 muestra el esquema de un CAD de sobremuestreo Sigma Delta genérico de tiempo continuo.

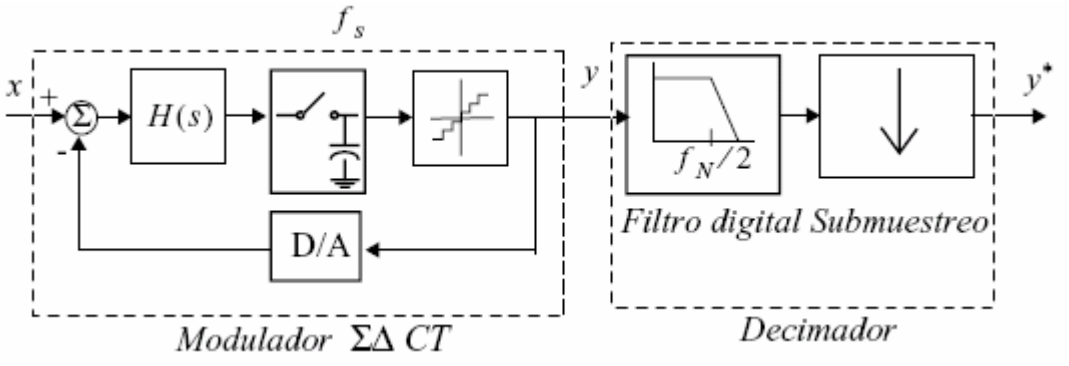


Fig. 3.4. CAD Sigma Delta de tiempo continuo.

En este caso se compone del modulador y decimador, y se prescinde del prefiltrado apareciendo un bloque de muestreo y retención (*sample and hold*) dentro del modulador. Las funciones del modulador y del decimador son idénticas a las explicadas para el caso discreto.

Tras una visión genérica de CAD Sigma Delta, estudiemos a continuación las no idealidades que afectan a estas estructuras, ya que ésta va a ser la base de estudio que vamos a necesitar para generar nuestros modelos de librerías.

3. No idealidades en convertidores A/D Sigma Delta

El diagrama de un modulador de primer orden de capacidades conmutadas Sigma Delta se muestra en la Fig.3.5.

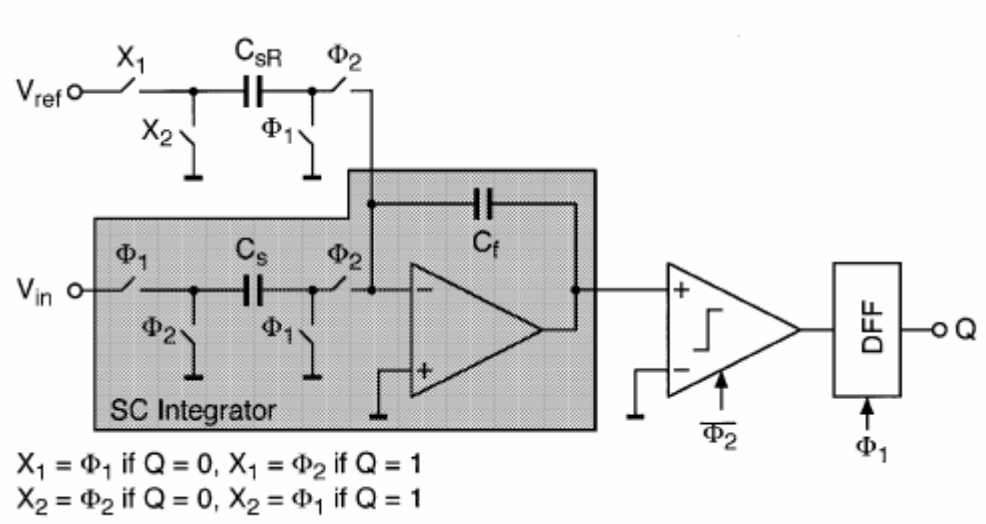


Fig. 3.5. Modulador SC Sigma Delta de primer orden.

Este circuito lo utilizaremos en este apartado para introducir las no idealidades que afectarán a nuestro modelo. Consiste en un integrador de capacidades conmutadas, un cuantizador y un convertidor digital analógico en el lazo de realimentación. Las principales no idealidades de estos circuitos son las siguientes:

1. Ruido térmico en la estructura SC (ruido ktc)
2. Ruido del amplificador operacional
3. Ganancia finita del amplificador operacional
4. Ancho de banda del amplificador operacional
5. Slew-rate del amplificador operacional
6. Saturación de la tensión del amplificador operacional
7. Ruido Jitter
8. Offset de los comparadores
9. Histéresis de los comparadores

3.1. Ruido ktc y ruido del amplificador operacional

Esta primera no idealidad que vamos a comentar es la fuente de ruido más importante que afecta a los moduladores sigma delta de capacidades conmutadas, el cual está asociado a los interruptores de muestreo e intrínsecamente a los amplificadores operacionales.

Estos efectos pueden modelarse utilizando el comportamiento de un integrador ruidoso, que consistiría en sumarle a la entrada del integrador el efecto del ruido.

El cociente de los valores Cs/Cf representa el coeficiente del integrador. Para ello, a

partir de la Fig.3.5 podemos obtener la función de transferencia en dominio z, dada por:

$$H_I(z) = \frac{Cs}{Cf}\frac{z^{-1}}{1-z^{-1}} = b\frac{z^{-1}}{1-z^{-1}} \qquad (2.1)$$

donde b es el coeficiente mencionado y va a ir multiplicando a la señal de entrada.

El modelo del ruido del amplificador operacional se puede considerar como la variación aleatoria de una tensión, Vn, siendo ésta la tensión rms del amplificador operacional referida a la entrada del integrador. Para este modelado vamos a despreciar el ruido flicker (1/f) y el offset de continua, ya que éstos típicamente se cancelan.

3.2. Ganancia finita del amplificador operacional

La ganancia dada de un integrador es idealmente infinita. En la práctica, sin embargo, está limitada. Este límite de la ganancia del integrador incrementa el ruido en la banda de la señal. La función de transferencia del integrador sería para el caso de ganancia finita:

$$H(z) = \frac{z^{-1}}{1-\alpha\cdot z^{-1}} \xrightarrow{dcGain} Ao = H(1) = \frac{1}{1-\alpha} \xrightarrow{despejo\_\alpha} \alpha = 1-\frac{1}{Ao} \qquad (2.2)$$

donde Ao es la ganancia dc del integrador.

El efecto de esta ganancia finita influye también en el siguiente apartado del Slew-rate (SR) y del ancho de banda finito (BW).

3.3. Slew-rate y ancho de banda del amplificador operacional

El efecto del ancho de banda finito (BW) y del slew-rate (SR) puede ser interpretado como una no idealidad en la ganancia. Refiriéndonos al modulador de primer orden mostrado en la Fig.3.5, la evolución de la salida durante el periodo n de integración (cuando φ_2 está encendido) viene dada por:

$$vo(t) = vo(nTs - Ts) + \alpha Vs(1 - e^{-\frac{t}{\tau}}) \text{ en } 0<t<\frac{Ts}{2} \qquad (2.3)$$

donde $Vs = Vin(nTs - \frac{Ts}{2})$, α proviene de la ganancia finita del amplificador operacional, τ es la constante de tiempo del integrador e igual a $\tau = \frac{1}{2\pi GBW}$ y GBW es el producto ganancia ancho de banda.

El valor de esta curva toma el máximo cuando t=0, dando:

$$\frac{d}{dt}vo(t) = \alpha\frac{Vs}{\tau} \qquad (2.4)$$

Consideramos dos casos:

1. El valor de (2.4) es más pequeño que el SR del amplificador operacional. En este caso el SR no limita la evolución de vo(t) durante todo un periodo de reloj (hasta t=Ts/2).
2. El valor de (2.4) es mayor que el SR, en ese caso:

$$\text{t} \le \text{to: } vo(t) = vo(nTs - Ts) + SRt$$

$$\text{t} > \text{to: } vo(t) = vo(to) + (\alpha Vs - SRto)\left(1 - e^{\frac{t-to}{\tau}}\right) \text{ siendo } to = \frac{\alpha Vs}{SR} - \tau \qquad (2.5)$$

3.4. Saturación de la tensión del amplificador operacional

Este efecto no ideal en la saturación de los niveles del amplificador operacional. Es simple de modelar, ya que tan sólo debemos saturar la salida del integrador (dentro del lazo).

3.5. Ruido Jitter

El efecto del ruido jitter en los moduladores Sigma Delta es independiente de la estructura u orden del modulador. El resultado es un muestreo no uniforme en la secuencia temporal y produce un error, el cual incrementa la potencia del ruido a la salida del cuantizador.

El error introducido cuando una señal sinusoidal a la entrada x(t) con amplitud A y frecuencia fsin es muestreada en un instante con error δ está dado por:

$$x(t+\delta) - x(t) = 2\pi f_{\sin}\delta A\cos(2\pi f_{\sin}t) = \delta\frac{d}{dt}x(t) \qquad (2.6)$$

3.6. Histéresis y offset de los comparadores

Los comparadores básicamente tienen dos no idealidades[1], una de ellas es el offset, en la que la salida el comparador se retrasa un cierto tiempo, y la otra es la histéresis, en la que se produce una variación del tiempo del cambio de '0' a '1', y de '1' a '0'. Estos dos efectos podemos representarlos en la siguiente gráfica:

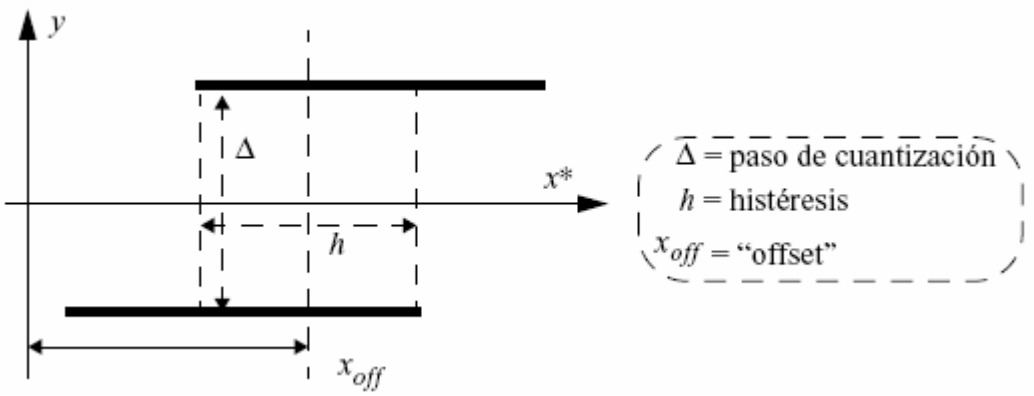


Fig. 3.6. Histéresis y offset en los comparadores.

[1] Estas no idealidades de los comparadores las vamos a tener presentes siempre que haya comparadores en la estructura de nuestro convertidor.

CAPÍTULO 3

Convertidores A/D Pipeline

Índice:

1. Introducción

Los convertidores A/D pipeline son adecuados para aplicaciones donde el ancho de banda es relativamente grande al igual que una resolución también alta. Éstos son similares a los convertidores semi-flash y consiste en varias etapas en cascada con CAD de baja resolución, un amplificador S/H y un CDA. Sus propiedades son similares a los de semi- flash, aunque su principal diferencia como veremos en el capítulo posterior es el circuito S/H entre etapas el cual incrementa el máximo de la tasa de muestreo del convertidor.

Los convertidores pipeline se han convertido en los más utilizados en el diseño de arquitecturas de resolución moderada (8 a 12 bits) y frecuencias de muestreo del orden de decenas de megahercios. Para frecuencias de muestreo menores se suelen utilizar convertidores de aproximaciones sucesivas (bajas resoluciones) o arquitecturas Sigma- Delta (altas resoluciones), mientras que para frecuencias superiores (cientos de megahercios) continúan dominando los convertidores flash y sus variantes.

En este capítulo vamos a hacer una revisión de los fundamentos de los convertidores A/D pipeline y mostraremos como se determina la resolución de este tipo de convertidores así como su salida digital. Existe en estas estructuras un principio de corrección, en el cual no entraremos detalladamente, que se utiliza para relajar los requerimientos de los bloques ADC que adelante comentaremos. En el caso de necesitar resoluciones muy altas será necesaria una calibración.

Nosotros vamos a centrarnos en estudiar la estructura de los convertidores A/D pipeline y posteriormente sus principales no idealidades, que son la base de estudio de este proyecto.

2. El convertidor pipeline

El convertidor A/D pipeline está formado por varias etapas, conteniendo cada una de ellas un convertidor analógico digital (CAD), un convertidor digital analógico (CDA), un restador y un amplificador de residuo. La última etapa necesitará sólo de un CAD.

En ocasiones, se emplea un circuito de muestreo y retención (S&H) a la entrada para evitar errores por desviaciones debidas a retrasos en las dos señales de

entrada al restador. Debido a su simplicidad y rapidez, el CAD suele ser un cuantizador Flash.

Al conjunto del amplificador de residuo, el restador y el CDA se le llama también convertidor digital analógico multiplicador (MCDA) y desempeñará, además, la función de S&H.

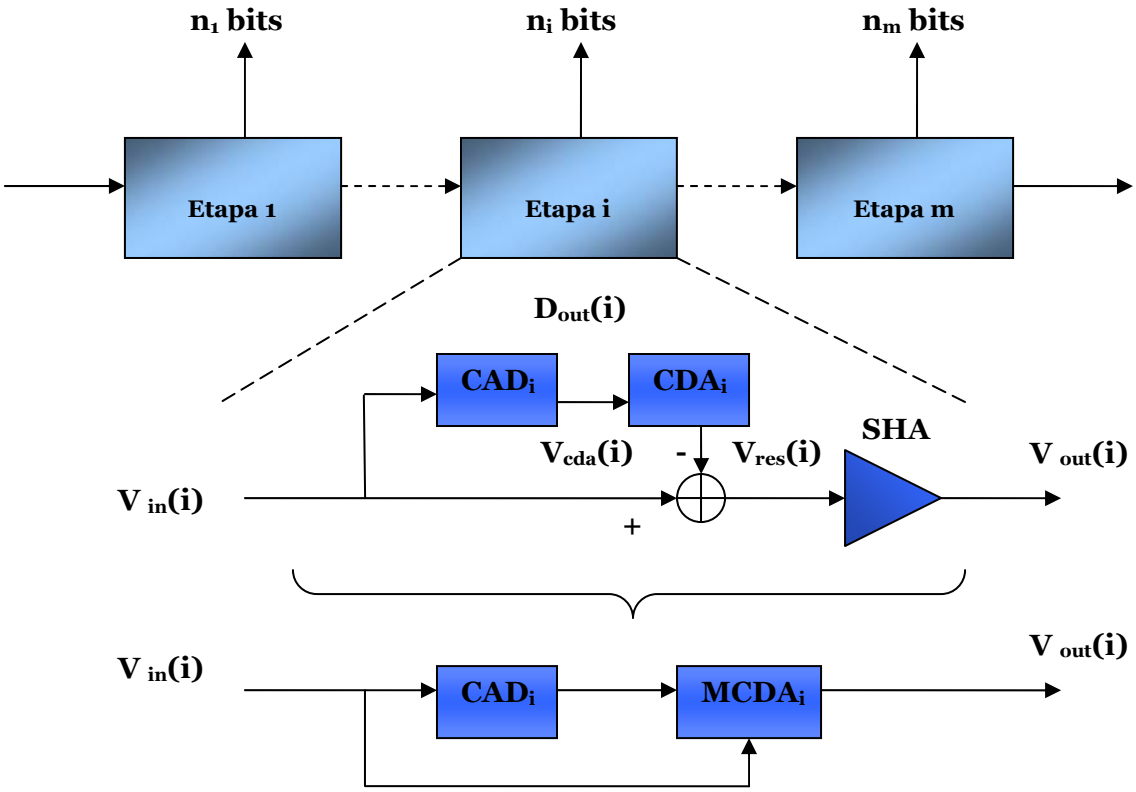


Fig. 3.1. Estructura del convertidor A/D pipeline.

El significado de cada uno de los elementos es el siguiente:

- **CAD_i:** el convertidor analógico-digital de la etapa i
- **CDA_i:** el convertidor digital-analógico de la etapa i
- **$V_{res}(i)$:** residuo de la etapa i antes de amplificarlo
- **$V_{in}(i)$:** entrada analógica de la etapa i
- **$V_{out}(i)$:** salida analógica de la etapa i. Por lo tanto, $V_{out}(i) = V_{in}(i+1)$
- **$D_{out}(i)$:** salida digital de la etapa i

- **D_{out}:** salida digital total de todo el CAD
- **$V_{cda}(i)$:** salida del CDA en la etapa i
- **FS:** fondo de escala del rango de entrada del CAD de la etapa i, igual para todo i.
- **n_i:** número de bits de salida de la etapa i
- **n_{tot}:** número de bits totales del convertidor pipeline
- **N_i:** número de códigos de salida de la etapa i del CAD
- **N_{tot}:** número de bits totales del convertidor A/Dpipeline
- **G_i:** ganancia del residuo de la etapa i
- **m:** número de etapas

Normalmente, el número de códigos de salida de la etapa i, N_i, es potencia de dos, es decir,

$$N_i = 2^{ni} \tag{3.1}$$

donde n_i es el número de bits de la etapa i.

3. No idealidades en convertidores A/D pipeline

Las no idealidades más importantes en un CA/D pipeline, que estudiaremos a continuación son:

1. Ganancia DC finita del amplificador operacional.
2. Tamaño de las capacidades.
3. Slew Rate y Ancho de Banda del amplificador operacional.
4. Offset en los DACs.
5. Jitter.

3.1. Ganancia DC Finita del amplificador operacional

En este apartado se estudia la influencia de la ganancia finita del amplificador operacional sobre la resolución del convertidor. Para desarrollar el modelo se ha utilizado el circuito de la Fig 3.2, en la fase de cálculo del residuo.

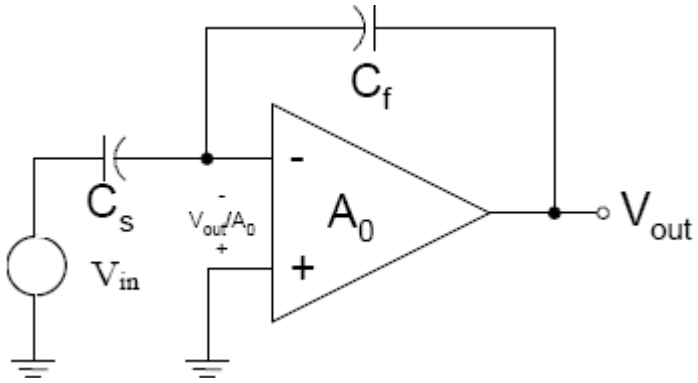


Fig. 3.2. Circuito en fase de cálculo del residuo. Cálculo de la ganancia.

Debido a la ganancia finita del amplificador (A0), la ganancia del amplificador de residuo se ve modificada según la siguiente expresión:

$$Vout = \frac{2}{1+\frac{2}{Ao}} Vin \qquad (3.2)$$

Siendo Vin en este caso la tensión del residuo.

Esta desviación de la ganancia del amplificador de residuo no es corregida por la corrección digital.

3.2. Tamaño de las capacidades

El tamaño de las capacidades utilizadas en el cálculo del residuo es muy importante ya que determinan el consumo del amplificador operacional, y por tanto, del convertidor completo. Por esa razón, las capacidades deben ser lo menores posible para minimizar este consumo. Sin embargo, existen dos factores que limitan el tamaño de estas capacidades y el motivo de que no podamos hacerlas tan pequeñas como queramos: el ruido térmico y el desapareamiento entre capacidades.

3.2.1. Ruido térmico

En un circuito de capacidades conmutadas el ruido introducido durante el

proceso de muestreo es inversamente proporcional al tamaño de la capacidad. Basándonos en este principio, y teniendo en cuenta que el ruido máximo admitido viene dado por la resolución del convertidor, es posible calcular la capacidad de muestreo mínima.

La potencia total de ruido debida al proceso de muestreo viene dada por la siguiente ecuación:

$$\overline{Vout^2} = \frac{kT}{Cs} \tag{3.3}$$

siendo kT una constante de valor 4.14e-21 (k, constante de Boltzmann; T, temperatura ambiente) y Cs la capacidad de muestreo.

Por otra parte, la potencia correspondiente al ruido de cuantización para una entrada senoidal, bajo supuesto de ruido blanco, puede expresarse como:

$$\varepsilon_q^{\,2} = \frac{\Delta^2}{12} \tag{3.4}$$

donde $\Delta = \frac{V_{REF}}{2^n}$ se corresponde con el tamaño del escalón de cuantización de un convertidor de n bits y V_{REF} representa la tensión de fondo de escala.

La relación señal a ruido a la salida para una entrada analógica senoidal de amplitud $\frac{V_{REF}}{2}$, expresada en decibelios, viene dada por la expresión:

$$SNR = 6.02n + 1.76 \tag{3.5}$$

donde n representa, como se ha mencionado, el número de bits de resolución. En el caso, por ejemplo, de tener un convertidor analógico-digital pipeline de 10 bits[2], el SNR será igual a 61.96 dB. Si asumimos una desviación de 1 dB respecto a la relación señal a ruido ideal, la capacidad de carga permitida a la salida del amplificador será de $C_L \geq$ 49.91 fF, valor suficientemente bajo para concluir que el ruido térmico no es un factor determinante a la hora de fijar el tamaño mínimo de las capacidades empleadas en el cálculo del residuo.

3.2.2. Desapareamiento entre capacidades

Cualquier variación en el cociente Cs/Cf (coeficiente del integrador del amplificador operacional) producirá una desviación en la ganancia del residuo, que no se compensaría en la corrección digital. Por otro lado, mientras más pequeña sean

[2] Este es el ejemplo que veremos más adelante en el capítulo 8 de simulaciones.

las capacidades peor es el apareamiento entre ellas. En nuestro caso, el mínimo valor de las capacidades vendrá limitado por este efecto.

Para desarrollar el modelo utilizado se ha tenido en cuenta que $V_{ref} = 2V_{DAC}$, siendo
V_{ref} la tensión de referencia y V_{DAC} la salida del DAC en el modelo ideal.

El desapareamiento entre capacidades se medirá mediante el parámetro ε, que viene dado por la siguiente ecuación:

$$\frac{Cs}{Cf} = 1 + \varepsilon \tag{3.6}$$

La tensión de salida considerando el desapareamiento entre capacidades queda:

$$Vout = 2(Vin - Vdac) + \varepsilon(Vin - 2Vdac) \tag{3.7}$$

3.3. Slew-rate y ancho de banda del amplificador operacional

Ahora veremos la influencia de otra no idealidad que afectará en la velocidad del amplificador operacional para la resolución del convertidor, definiendo las especificaciones dinámicas del opamp como su slew-rate y ancho de banda.

Veremos el efecto del slew-rate y el ancho de banda mediante el circuito de la Fig 3.3 en la fase de amplificación del residuo, en la cual la carga almacenada en Cs pasa a Cf. En la práctica, este traspaso de carga nunca se realiza completamente, no alcanzando la salida del MDAC el valor deseado, debido a las limitaciones dinámicas del amplificador.

El proceso de carga estará limitado por dos efectos. Por un lado, si el cambio de tensión a la salida es muy abrupto el amplificador saldrá de su zona de funcionamiento lineal, estando la pendiente limitada por el slew-rate. Por otro lado, cuando el amplificador está funcionando linealmente (asumimos un comportamiento de primer orden) la carga de la capacidad se producirá siguiendo una característica exponencial.

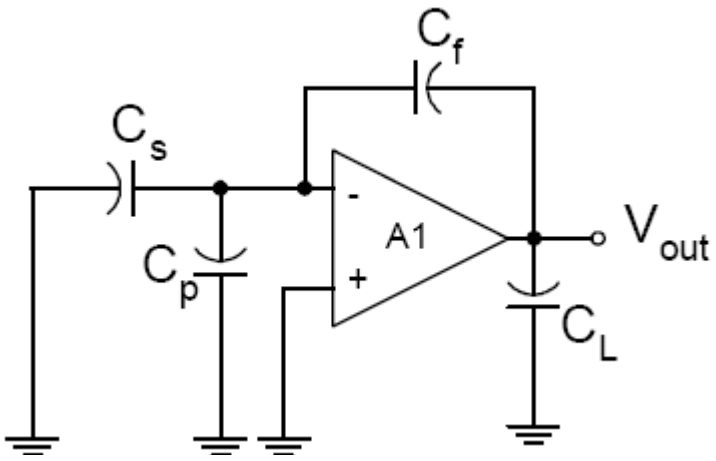


Fig. 3.3. Modelo para calcular la influencia del SR y BW del op en cálculo del residuo.

Asumimos un amplificador con respuesta en frecuencia de primer orden dado por:

$$A(s) = \frac{Ao}{1 - \frac{s}{p_1}} \tag{3.8}$$

donde, *A0* es la ganancia DC del amplificador y *p1* la frecuencia del polo, definidos como:

$$\begin{aligned} Ao &= g_m r_{out} \\ p_1 &= \frac{1}{r_{out} C_{Leff}} \end{aligned} \tag{3.9}$$

donde la capacidad C_{Leff} es la capacidad efectiva que ve el amplificador en bucle abierto a la salida. Es decir:

$$C_{Leff} = C_L + \frac{C_f (C_p + C_s)}{C_f + C_p + C_s} \tag{3.10}$$

El factor de realimentación viene dado por el divisor capacitivo compuesto por *Cf* y*Cs*:

$$\beta = \frac{C_f}{C_f + C_p + C_s} \approx \frac{1}{2} \tag{3.11}$$

Para conocer la respuesta transitoria del sistema calculamos la función de transferencia en bucle cerrado:

$$Acl(s) = \frac{A}{1-A\beta} = \frac{r_{out} g_m}{1 + r_{out} C_{Leff} s - \frac{C_f}{C_f + C_p + C_s} r_{out} g_m} \approx \frac{g_m}{C_{Leff} s - \frac{C_f}{C_f + C_p + C_s} g_m} \tag{3.12}$$

Al ser un sistema de primer orden, la constante de tiempo se corresponde con la inversa de la frecuencia del polo:

$$\tau = 2\frac{C_{Leff}}{g_m} \tag{3.13}$$

Nótese que se corresponde con la constante de tiempo de un sistema con ancho de banda la mitad del producto ganancia ancho de banda del amplificador con una capacidad de carga C_{Leff}.

Es posible hacer un estudio más detallado del proceso de carga del condensador teniendo en cuenta el slew-rate del amplificador. La ecuación que rige el comportamiento del MDAC es la siguiente:

$$Vout = Vin + g(Vin - 2Vdac) \tag{3.14}$$

siendo *g(x)* la función que modela el efecto del *slew-rate* y el ancho de banda del amplificador sobre el proceso de carga, y tiene la siguiente forma:

$$g(x) = \begin{cases} x(1 - e^{-\frac{Ts}{2\tau}}) \xrightarrow{si} |x| \le \tau\zeta \\ x - \mathrm{sgn}(x)\tau\zeta e^{\left(\frac{|x|}{\tau\zeta} - \frac{Ts}{2\tau} - 1\right)} \xrightarrow{si} \tau\zeta < |x| \le (\tau + Ts)\zeta \\ \mathrm{sgn}(x)\zeta\frac{Ts}{2} \xrightarrow{si} (\tau + Ts)\zeta < |x| \end{cases} \tag{3.15}$$

donde τ representa el tiempo de establecimiento, ζ el *slew-rate* del amplificador y Ts el tiempo de muestreo.

3.4. Offset en los DACs

El efecto del offset que está presente en los DACs es un cambio en los niveles de decisión, que ya no estarán equidistribuidos a lo largo del rango que va de –FS/2 a

FS/2 (suponiendo un convertidor simétrico) sino que presentarán una varianza en torno al nivel ideal. Este inconveniente se acentúa al colocarlos en cascada.

3.5. Jitter

El ruido jitter está asociado al proceso de muestreo de señales analógicas para trabajar con señales en tiempo discreto. El mecanismo del jitter no está demasiado claro (se sabe que da como resultado un tiempo de muestreo no uniforme) y, por tanto, no hay muchas técnicas o diseños que traten el problema. Aumentar la resolución y la velocidad de los convertidores depende en gran medida de reducir el jitter.

El ruido jitter produce un error que incrementa la potencia total del error a la salida del cuantizador. La magnitud de este error es función de las propiedades estadísticas del jitter. El error introducido cuando una señal sinusoidal x(t) de amplitud A y frecuencia f_{sin} es muestreada en un instante dado con un error de δ viene dado por:

$$x(t+\delta) - x(t) = 2\pi f_{\sin} \delta A \cos(2\pi f_{\sin} t) = \delta \frac{d}{dt} x(t) \tag{3.16}$$

CAPÍTULO 4

Convertidores A/D Flash

Índice:

1. Introducción

Los convertidores analógico-digitales Flash se utilizan en aplicaciones donde el requerimiento principal es tener una velocidad de conversión muy alta. Estas aplicaciones podrían ser como procesamiento de señales de video, aplicaciones radar...

El convertidor flash alcanza grandes velocidades de conversión debido a que usa una matriz de comparadores en paralelo que muestrean la señal analógica simultáneamente. Como requerimos un comparador por cada nivel de cuantización que tengamos, el número de comparadores se dobla por cada bit adicional de resolución que queramos tener. Así pues, el problema que surge con esta técnica es el significativo incremento de disipación de potencia en comparación con otras técnicas de conversión A/D.

Nosotros vamos a centrarnos en estudiar la estructura de los convertidores A/D flash y posteriormente sus principales no idealidades, que son la base de estudio de este proyecto.

2. El convertidor Flash

El convertidor Flash tiene la topología más simple de todos los convertidores que consiste en la operación de paralelo de comparadores los cuales tienen a su entrada una tensión de referencia. La técnica flash es también conocida como "la aproximación en paralelo".

Un convertidor flash de N bits requiere una matriz de 2^N -1 comparadores. La entrada analógica se conecta a una entrada de la matriz de comparadores, mientras que la otra entrada de cada comparador se conecta a tensiones de referencia fijas. Estas referencias representan 2^N -1 niveles de tensión equidistantes entre FS/2 y –FS/2 (siendo FS el fondo de escala) correspondientes a los 2^N -1 puntos de conmutación entre los extremos del rango del convertidor.

La desventaja de todo esto, evidentemente, es que se necesitan un número muy elevado de comparadores, resistencias para crear las referencias de tensión, e interconexiones, y además que se incrementarán exponencialmente con el aumento de resolución del convertidor.

Veamos en la siguiente figura a modo de ejemplo como sería la estructura de un convertidor analógico-digital flash de 3 bits:

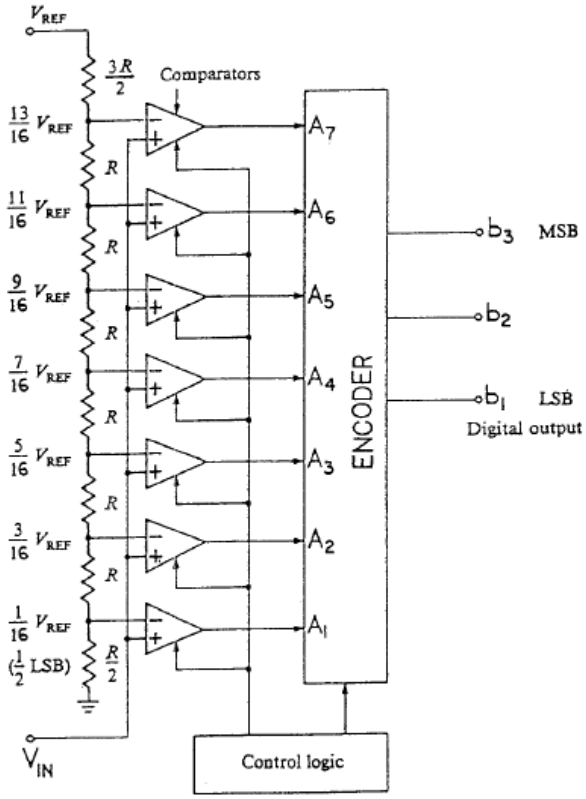


Fig. 4.1. Estructura del convertidor A/D flash de 3bits.

En este esquema podemos distinguir la cadena de resistencia que nos va a dar los distintos valores de referencia a cada uno de los 2^N -1 comparadores (en este caso al ser de 3 bits, necesitamos 7 comparadores). El encoger se encarga de sumar todas las salidas para posteriormente realizar una conversión del código a digital.

Para este tipo de estructuras, se puede también añadir una calibración de los comparadores que explicaremos más adelante en el correspondiente apartado de este capítulo.

La diferencia de tensiones entre 2 resistencias adyacentes es igual a:

$$V_{LBS} = \frac{V_{REF\_TOP} - V_{REF\_BOT}}{2^N} \tag{4.1}$$

donde V_{REF_TOP} y V_{REF_BOT} representan las 2 referencias de tensión proporcionadas al convertidor.

El modo de operación del convertidor es el siguiente: cada comparador cuya entrada analógica esté por debajo de su referencia pone un "0" lógico a su salida, y los comparadores cuya entrada supere la referencia pone un "1". Estas salidas sirven como entrada a un decodificador termométrico, que detecta la transición de ceros lógicos a unos en la matriz de comparadores. Esta transición, en teoría, debe ser única. La siguiente operación ya sería la decodificación de termométrico a binario, obteniéndose la palabra binaria de salida.

3. No idealidades en convertidores A/D Flash

Según la estructura comentada de un convertidor analógico-digital flash, prácticamente está formado por resistencias y comparadores. Por ello las no idealidades se van a limitar a estos componentes. Las resistencias debido a su fabricación presentan una tolerancia que afectará a la generación de la tensión de referencia de entrada a los comparadores. Y los comparadores presentarán sus no idealidades típicas, el offset y la histéresis.

3.1. Offset en la generación de la tensión de referencia

Debido al desapareamiento entre las resistencia (ya que éstas presentan una cierta tolerancia en su fabricación) la tensión de referencia obtenida varía en torno a la ideal. Es como su sumáramos un número aleatorio a la generación de la tensión, dentro el rango de valores disponibles.

Para solucionar este error se utilizan métodos de calibración que ayudan a elegir aquellos comparadores que se acerquen más a cada uno de los cambios que deben producirse según el ideal.

3.2. Comparadores

Para la realización de nuestra estructura flash, vamos a necesitar un número elevado de comparadores, el cual va creciendo cada vez que aumentemos los bits de resolución del convertidor analógico digita. Estos comparadores llevan implícitos una serie de no idealidades, concretamente el offset y la histéresis.

Veamos a continuación estos dos tipos de no idealidades.

3.2.1. Offset

El offset en los comparadores, que provocarán que tenga que haber una diferencia (positiva o negativa) entre la entrada analógica y la tensión de referencia de cada comparador en la comparación (en el caso ideal, se comparaba si era mayor o menor, ahora habrá que tener en cuenta el offset). Este offset, si es suficientemente grande en algún comparador, podría provocar que la transición de ceros a unos en la escalera de comparadores no fuese única, con lo cual el código termométrico y su posterior decodificación podrían fallar.

3.2.2. Histéresis

La histéresis en los comparadores puede hacer que la salida de un comparador en concreto no pase de "0" a "1" y viceversa cuando se espera, cuanto más grande sea esta histéresis con más probabilidad podrá ocurrir este fenómeno.

Este fenómeno lo representamos en la siguiente figura, junto con el offset explicado en el apartado anterior:

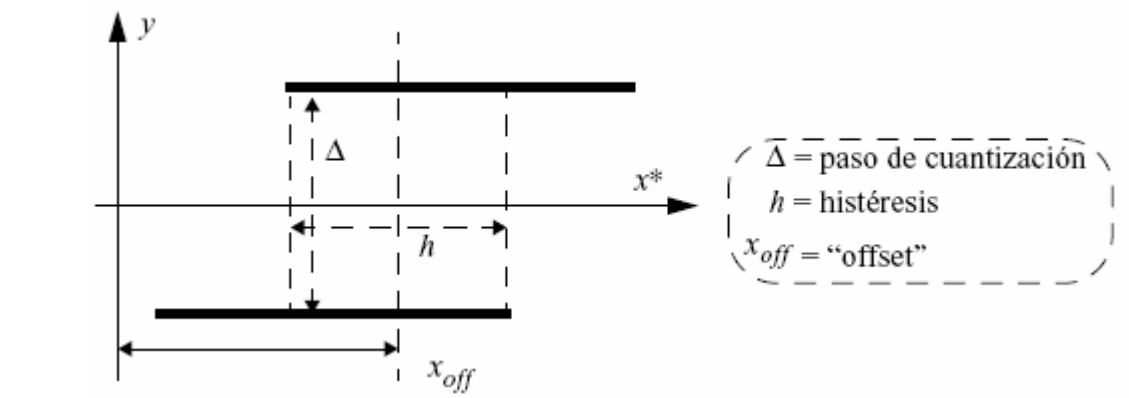


Fig. 4.2. Histéresis y offset en los comparadores.

4. Calibración en convertidores Flash

Para subsanar los problemas ocasionados por las no idealidades explicadas en el apartado anterior, existen técnicas de calibración que permiten ajustar a cada nivel ideal de comparación un comparador que se ajuste en mejor medida. Para ello es necesario repetir la misma estructura de los comparadores vista en la Fig.4.1.

A cada bloque de comparadores le llega por entrada las mismas tensiones de referencia generadas. En el caso de repetir cuatro veces la estructura de comparadores del ejemplo de la Fig.4.1 tendríamos 28 comparadores. De todos éstos vamos a escoger aquél (da igual cuál pero sólo uno) que conmute de '0' a '1' más próximo del caso ideal. Al final tendremos 7 comparadores de los 28, que aunque existe problema de offset, se ajustan en mejor medida al caso ideal.

El inconveniente de este método se puede deducir a simple vista, ya que vamos a cuadriplicar el número de comparadores.

CAPÍTULO 5

Interfaz *simconverter*

Índice:

1. Introducción

En el presente capítulo se describirá brevemente la estructura del programa implementado para la interfaz *simconverter* (funciones que se llaman, etc.). En el anexo 3 se ha reflejado el código de las partes más importes de este programa donde se han realizado modificaciones al mismo para mejorarlo, comunicarlo con la interfaz library y darle mayor funcionalidad.

El nombre que se le ha dado al simulador de convertidores es *simconverter* (ésta será la orden que tendremos que teclear desde la ventana de comandos de Matlab para arrancarlo, ya que el script principal es el archivo simconverter.m).

2. Scripts del programa

Este script principal simconverter contiene las llamadas a los dos scripts que ya contendrán el código del programa. A grandes rasgos, estos scripts son:

- **Initial**: Se encarga de la declaración de las variables globales del programa. Dichas variables globales se han usado para poder disponer de los valores de los parámetros de interés del programa en el WorkSpace externo, de forma que desde el exterior del programa el usuario pueda ver el valor de estas variables, y editarlas desde la ventana de comandos. Este script se encuentra en el archivo initial.m.

- **Guiproyecto**: Este script contiene todo el código que controla el funcionamiento de la GUI. Dicho archivo, denominado guiproyecto.m, inicializa la GUI, contiene todas las callbacks (comandos que se ejecutan cuando un usuario hace clic sobre un componente de la GUI) de la GUI, y realiza todas las llamadas a funciones y scripts auxiliares necesarias para la implementación deseada en cada caso (entraremos en detalle en las funciones auxiliares a las que llama este script a continuación).

Las funciones y scripts auxiliares a las que llamará el script de la GUI son las siguientes:

- **INLDNL_matlab**: Script que calcula INL, DNL y los códigos perdidos.

- **CalcINLDNL**: Función que recibe como parámetros el vector de salida del convertidor y el manejador de la GUI, y ejecuta el script INLDNL_matlab para calcular INL, DNL y los códigos perdidos.

- **CalcSNDR**: Función que recibe como parámetros el vector de salida del convertidor, el ancho de banda, la banda mínima de señal, el número de

muestras que consideramos como banda de señal (del centro a un extremo), el manejador de la GUI y el manejador del objeto figura. Calculará SNDR, SNR y ENOB.

- **SimulacionNoIdeal**: Función que gestiona los análisis simples, es decir, realiza una única simulación para unos valores concretos de los parámetros. Realiza sus cálculos usando las funciones CalcINLDNL y CalcSNDR.

- **Funciones de barrido**: Funciones que gestionan los barridos simples y dobles, tenemos una función por cada barrido simple o doble que se haya implementado. La característica principal de las funciones de barrido es que su nombre siempre comienza por la palabra "Estudio". Usará para realizar sus cálculos las funciones CalcINLDNL y CalcSNDR.

3. Visualización de la interfaz

La interfaz[3] finalmente desarrollada se muestra a continuación:

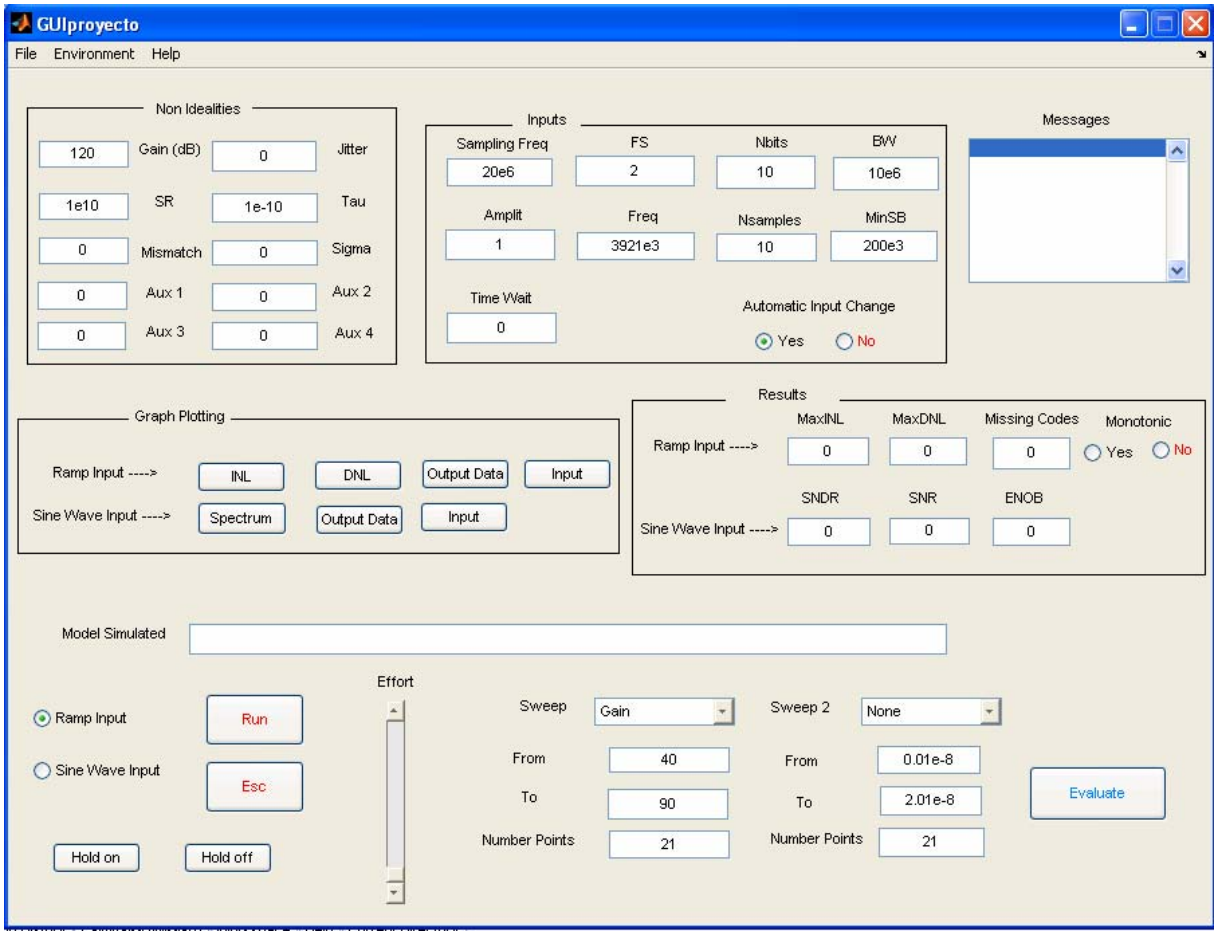


Fig. 5.1. Interfaz simconverter versión 2.0

[3] La descripción de su utilización se realiza en el manual de usuario que se encuentra en el Anexo1.

CAPÍTULO 6

Interfaz *library*

Índice:

1. Introducción

La creación de la interfaz library nace de la necesidad de simplificar el acceso a todos los archivos generados para este proyecto. No sólo es importante dotarla de un gran número de modelos de diseño para simular los convertidores, sino que también hay que facilitar al usuario una herramienta con la que pueda manejar todos estos modelos de manera rápida y sencilla. Y éstos son los objetivos de diseño de la interfaz: SENCILLEZ y RAPIDEZ.

La **sencillez** se ha logradazo gracias al diseño de menús donde el usuario va eligiendo paso a paso las especificaciones del modelo que busca. Se ha investigado en el desarrollo de una interfaz de actualización dinámica que lo permita.

La **rapidez** se ha conseguido por la forma de programación del código, ya que se ha generado una estructura virtual de "directorios", de forma que tan sólo busca en la "carpeta" elegida.

En este capítulo se describirá brevemente como se ha implementado la interfaz library. Para ello haremos un repaso entre los diferentes scripts que la forman y daremos un breve resumen de la funcionalidad de cada uno de ellos. En el anexo 4 tenemos disponible el código diseñado para desarrollar esta interfaz.

El nombre que se le ha dado para llamar a la librería es *library*. Esta orden se puede teclear directamente en la ventana de comandos de Matlab (ya que la función principal es *library.m*), pero lo más usual es que el usuario acceda a ella a partir de la interfaz mostrada en el capítulo 5 por medio del menú. En concreto, se accedería a partir de File → Open → Library.

2. Scripts del programa

Este script principal library contiene el código[4] del todo el programa, el cual además contiene otros que se reparten las diversas funcionalidades de nuestra aplicación.

- **library**: función principal desde dónde se programa la funcionalidad de todos los objetos de la interfaz. Dicho archivo, denominado library.m, inicializa la GUI, contiene todas las callbacks (comandos que se ejecutan cuando un usuario hace clic sobre un componente de la GUI) de la GUI, y realiza todas las llamadas a funciones y scripts auxiliares necesarias para la implementación deseada en cada caso (entraremos en detalle en las funciones auxiliares a las que llama este script a continuación). Otra de sus funciones principales es dotar a la librería de la actualización dinámica de los menús.

[4] El código desarrollado se puede consultar en el Anexo 4 del proyecto.

- **var_global**: se encarga de la declaración de las variables globales del programa y de su inicialización.

- **description**: base de datos con todas las descripciones de los modelos. Se ha creado una estructura de forma que el acceso es directo. Cada bloque o ejemplo presenta una breve descripción de su funcionalidad y características.

- **archive**: función para distinguir entre la apertura de una ventana de librería, o la apertura de un modelo.

- **arch_library_comp**: apertura de la librería de componentes seleccionados. Programada para que abra el modelo una vez que lo selecciones y pulses la tecla Aceptar, o te devuelva el control a la interfaz library en caso de pulsar la tecla de Cancel.

- **arch_model_comp**: apertura del modelo seleccionado. Para ello se mueve en el árbol de directorios y abre el fichero que contiene el bloque deseado.

3. Visualización de la interfaz

La interfaz[5] finalmente desarrollada se muestra a continuación:

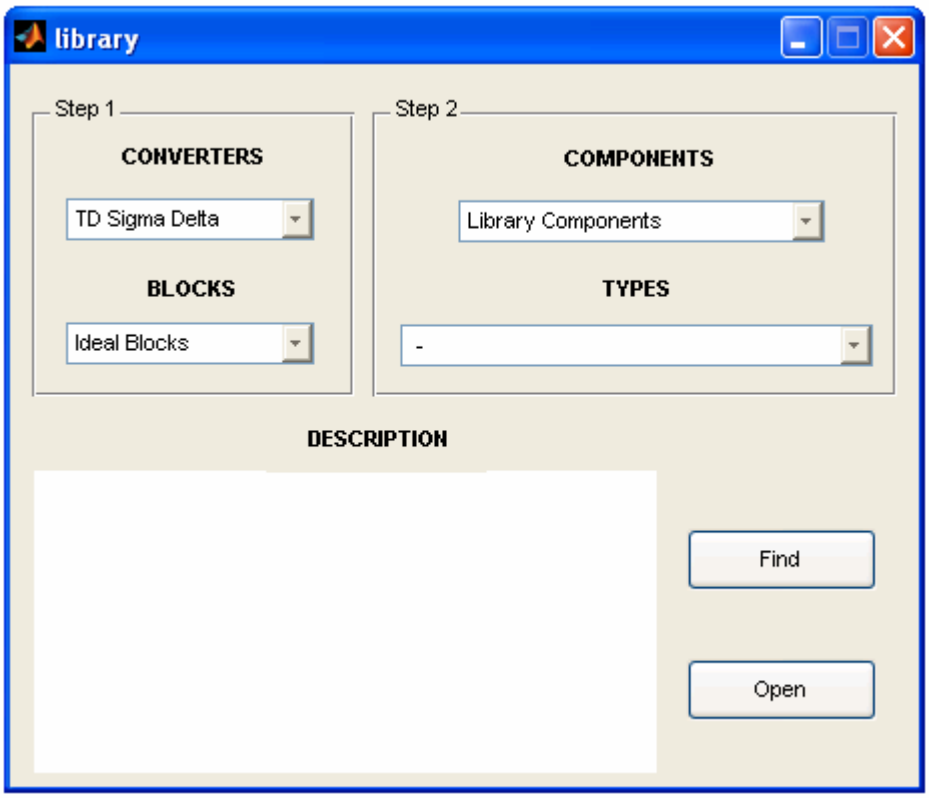


Fig. 6.1. Interfaz library

[5] La descripción de su utilización se realiza en el manual de usuario que se encuentra en el Anexo1.

CAPÍTULO 7

Librerías de simconverter

Índice:

LIBRERÍAS DE SIMCONVERTER

1. Introducción

En este capítulo nos vamos a centrar en visualizar el contenido de la librería de convertidores analógico-digitales que disponemos. Para ello vamos a ir presentando cada uno de los convertidores en estudio, mostrando una visión esquematizada de cada uno.

Los bloques están realizados mediante la herramienta Simulink de MATLAB 7.0. Se ha desarrollado una máscara para cada uno de ellos, de forma que el usuario no necesite conocer la estructura desarrollada para cada modelo, tan sólo las variables necesarias para su simulación.

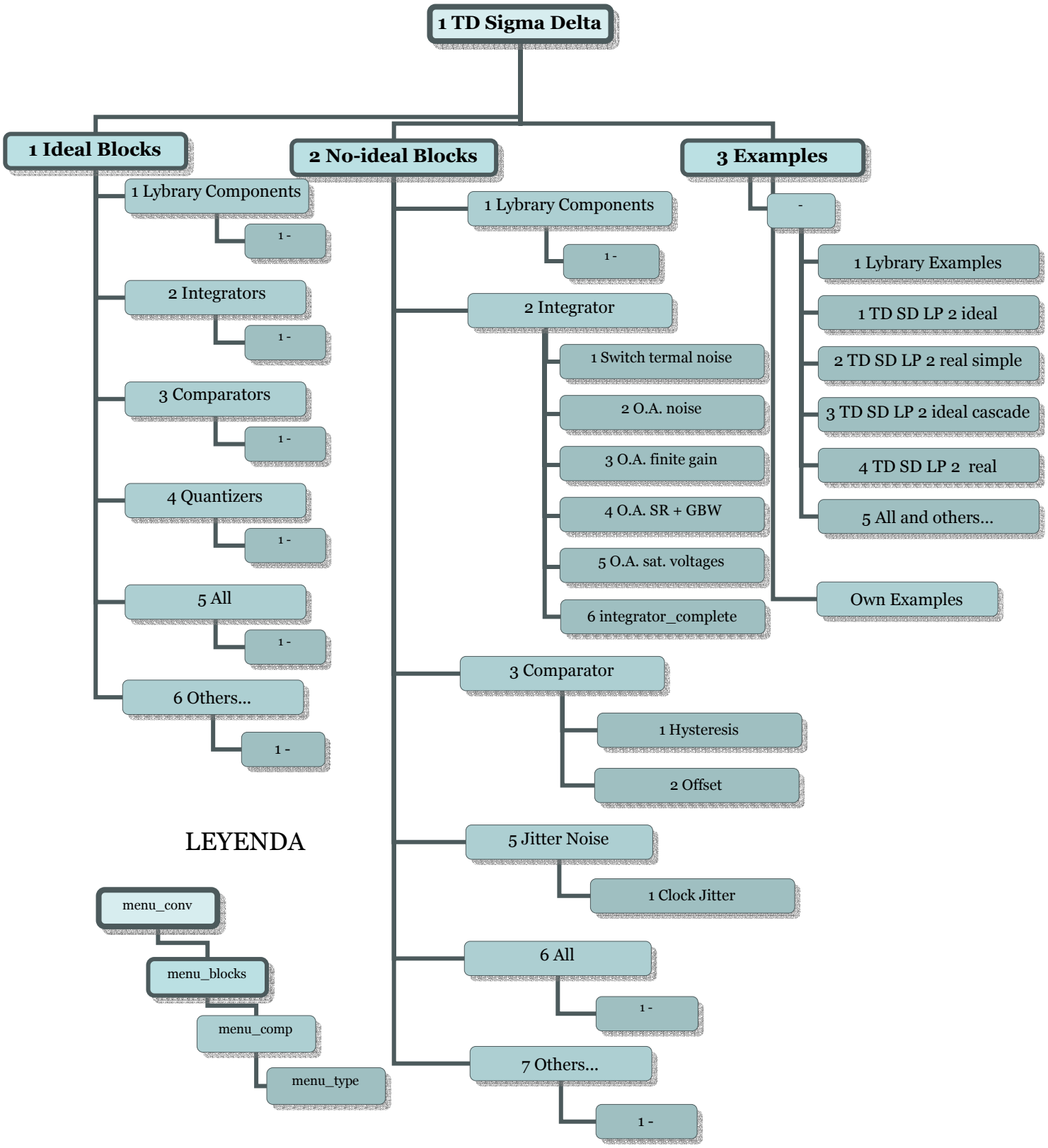
Cada uno de los convertidores estará formado por:

- **Bloques ideales**: Estos bloques no contienen ninguna no-idealidad, y presentaremos varios tipos según el modelo. Para cada convertidor dispondremos de todos los elementos que lo forman, de manera que el usuario disponga de todos los elementos para construir su propio diseño.
- **Bloques no-ideales**: Éstos simularán las no-idealidades a las que pueden estar sometidos los elementos de nuestro diseño. Existe una amplia gama de no-idealidades para cada uno de los bloques de cada convertidor, resultado de un estudio previo de los mismos.
- **Ejemplos**: En este apartado de la librería presentamos algunos diseños ya montados para cada uno de los convertidores. Con ellos pretendemos ayudar al usuario a la comprensión de los convertidores presentando modelos completos en algunos casos particulares. Éstos se podrán utilizar modificando sus especificaciones o sirviendo como base para otros.

Pasemos ahora a mostrar cada una de las partes de la librería desarrollada.

2. Librería de Sigma Delta en Tiempo Discreto

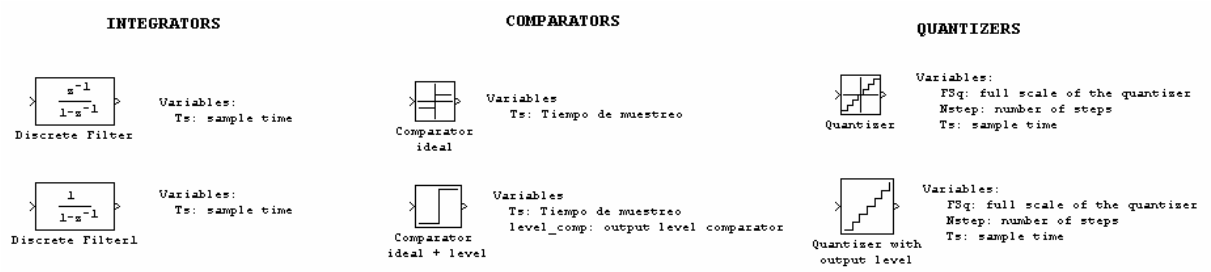
Para tener una visión global de la librería del convertidor Sigma Delta en Tiempo Discreto, presento el siguiente esquema en forma de árbol:


1 TD Sigma Delta
1 Ideal Blocks
1 Lybrary Components
1 -
2 Integrators
1 -
3 Comparators
1 -
4 Quantizers
1 -
5 All
1 -
6 Others...
1 -
2 No-ideal Blocks
1 Lybrary Components
1 -
2 Integrator
1 Switch termal noise
2 O.A. noise
3 O.A. finite gain
4 O.A. SR + GBW
5 O.A. sat. voltages
6 integrator_complete
3 Comparator
1 Hysteresis
2 Offset
5 Jitter Noise
1 Clock Jitter
6 All
1 -
7 Others...
1 -
3 Examples
-
1 Lybrary Examples
1 TD SD LP 2 ideal
2 TD SD LP 2 real simple
3 TD SD LP 2 ideal cascade
4 TD SD LP 2 real
5 All and others...
Own Examples
LEYENDA
menu_conv
menu_blocks
menu_comp
menu_type

2.1. Bloques ideales

Tras el estudio realizado de convertidores Sigma Delta en Tiempo Discreto, llegamos a la conclusión que necesitamos incluir tres tipos de bloques: integradores, comparadores y cuantizadores[6].

A modo de vista global veamos el fichero que contiene todos los bloques diseñados:



Y a continuación los modelos desarrollados por componentes.

2.1.1. Integradores

Los modelos propuestos son:

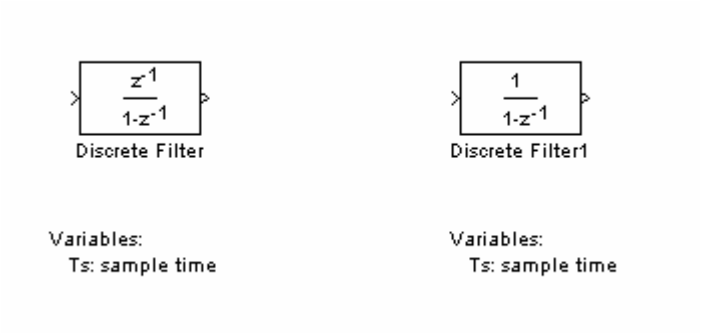


Fig.7.1. Bloques Ideales: Integradores.

Se tratan de dos modelos propuestos de integradores en tiempo discreto cada uno con una función de transferencia visible en el símbolo de cada bloque.

Para este caso no presento máscaras de los bloques, ya que el modelo está

[6] Bloques básicos como sumas, restas y productos, presentes en cualquier modelo de simulación, no los vamos a considerar para esta librería, de forma que el usuario tendrá que acudir a la librería propia de Simulink para hacer uso de ellos.

formado por ese único elemento.

La variable utilizada la vemos en la Tabla 7.1:

Variable simulador	Variable física	Comentario
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.1. Variables: Integradores.

2.1.2. Comparadores

Los modelos propuestos son:

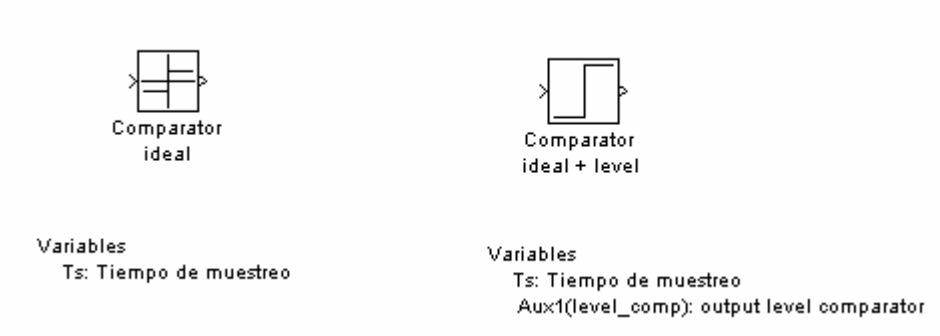


Fig. 7.2. Bloques Ideales: Comparadores.

El primer modelo se trata también un bloque básico tomado de la librería de Simulink. Su salida está dada entre [-1,1]. El segundo de ellos ha sido desarrollado para tener un comparador cuya salida de un cierto nivel distinto del anterior (level_comp).

La máscara para este bloque está formada tan sólo por dos variables, Ts que es el tiempo de muestreo y level_comp, nivel de salida del comparador. Los valores los introduciremos en la interfaz según la Tabla 7.1:

Variable simulador	Variable física	Comentario
Aux1	level_comp	Nivel de salida del comparador
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.2. Variables: Comparadores.

Su diseño viene dado en la Fig. 7.3.

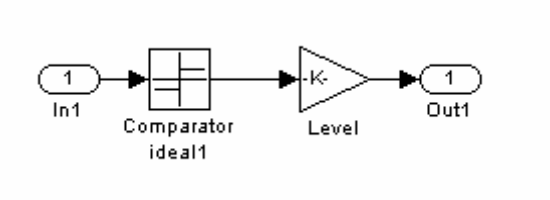
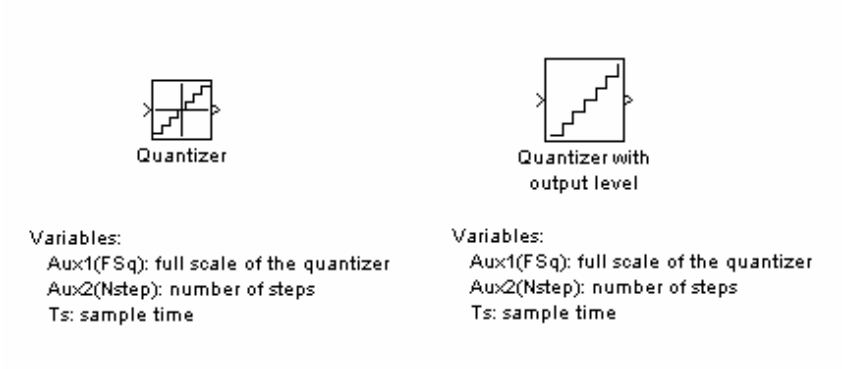


Fig. 7.3. Diseño comparador ideal + level

2.1.3. Cuantizadores

Los modelos propuestos son:



El primer modelo es un bloque básico de cuantizador de la librería de Simulink donde la salida viene dada entre (-∞,∞). El segundo es un cuantizador desarrollado para obtener a la salida la característica propia de un cuantizador, es decir, cuya salida es equivalente al fondo de escala del cuantizador (FSq). Estos límites de la salida viene dados por las fórmulas:

$$\text{Límite superior: } \frac{FSq}{2} - \left(\frac{FSq}{2Nstep}\right) \tag{7.1}$$

$$\text{Límite inferior: } -\frac{FSq}{2} + \left(\frac{FSq}{2Nstep}\right) \tag{7.2}$$

Los valores que introduciremos en la interfaz vienen dados en la Tabla 7.3:

Variable simulador	Variable física	Comentario
Aux 1	FSq	Fondo de escala del cuantizador
Aux2	Nstep	Niveles del cuantizador
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.3. Variables: Cuantizadores.

El diseño de este segundo cuantizador viene dado en la Fig. 7.4:

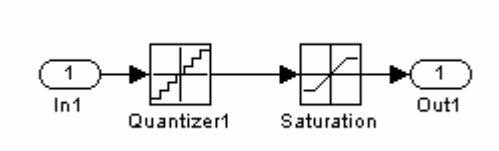


Fig. 7.4. Diseño cuantizador con nivel de salida

2.2. Bloques no ideales

Tal y como comentamos en el Capítulo 2 de convertidores A/D Sigma Delta, éstos presentan una topología para digitalización de señales analógicas con una alta resolución y caracterizadas por tener un ancho de banda (BW) mucho menor que la frecuencia de muestreo (fs).

Centrándonos en convertidores de tiempo discreto que es el punto que nos ocupa, la implementación más común utilizada es mediante capacidades conmutadas (SC, del inglés switched-capacitor). El diagrama de un convertidor sigma delta de primer orden es mostrado en la Fig.7.5:

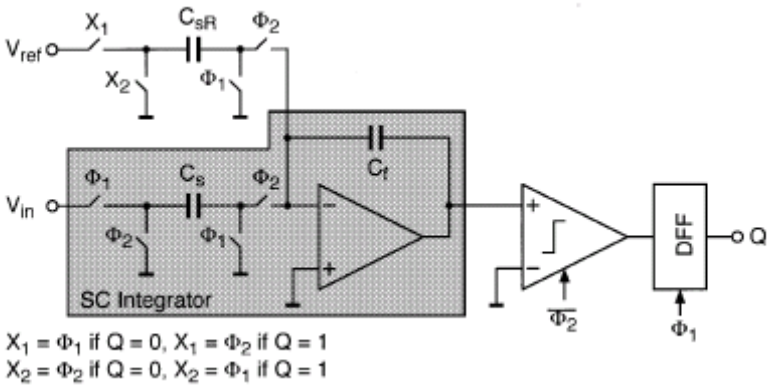


Fig. 7.5. Esquema de un modulador ∑Δ SC de primer orden

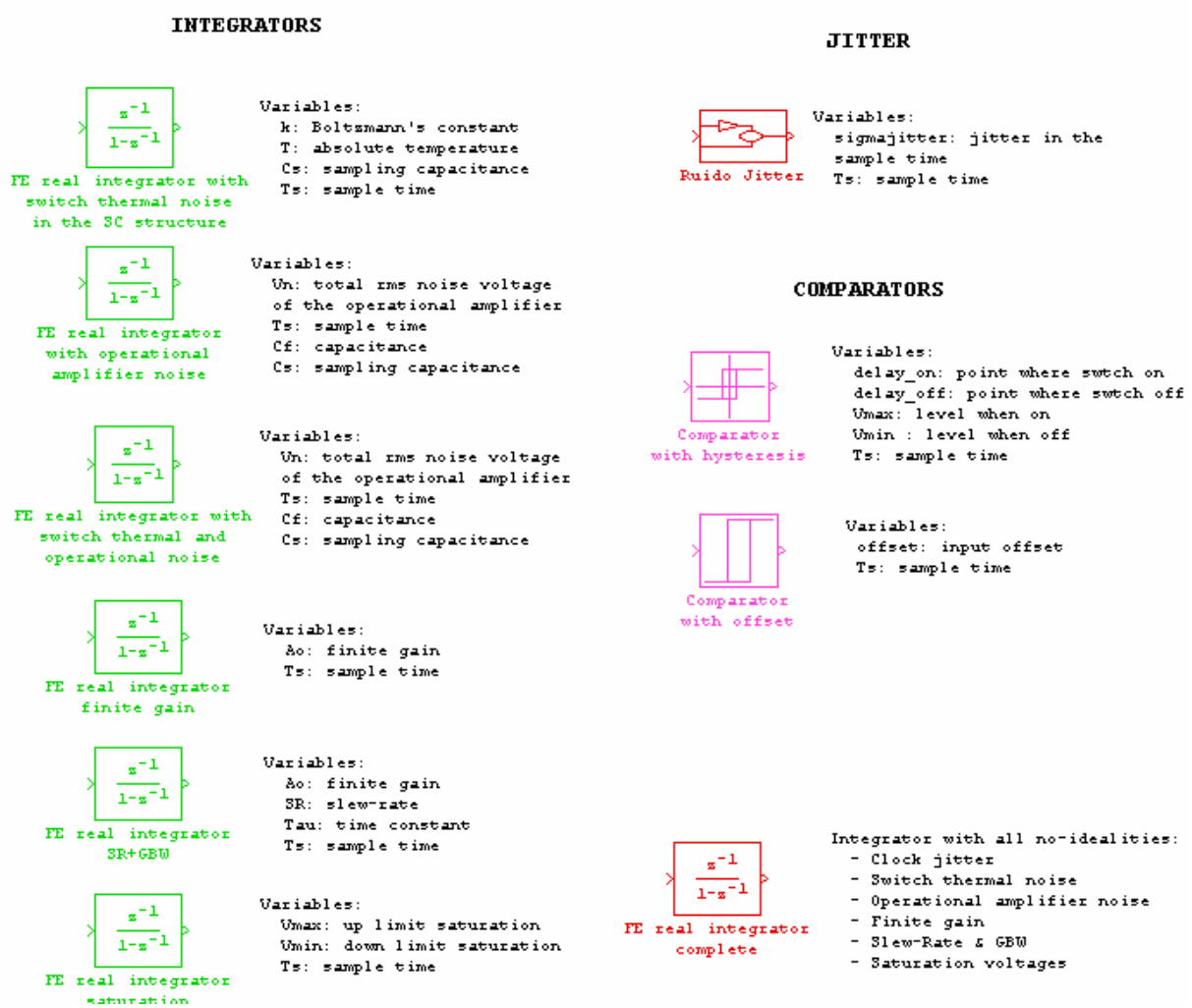
El modulador consiste en una entrada muestreada, un integrador con capacidades conmutadas, un cuantizador y un cuantizador digital-analógico.

Las principales no idealidades de estos circuitos son las siguientes:

1. Jitter en el reloj de muestreo de la entrada
2. Ruido térmico en la estructura SC
3. Ruido en el amplificador operacional
4. Ganancia finita del amplificador operacional
5. Ancho de banda del amplificador operacional

6. Slew-rate del amplificador operacional
7. Tesión de saturación del amplificador operacional
8. Histéresis en el comparador
9. Offset en el comparador

Una visión global de los diseños realizados:



Veamos el estudio de cada uno de ellos, y el modelo de simulación encontrado para cada uno.

2.2.1. Integradores

2.2.1.1. Ruido térmico en la estructura SC

El modelo propuesto es el siguiente:

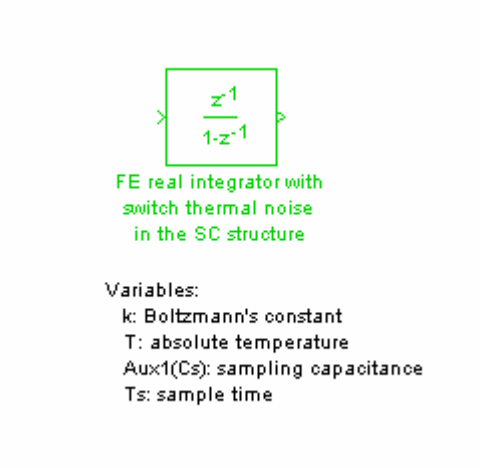


Fig. 7.6. Bloques no ideales: Integradores. Ruido térmico SC.

Los valores que necesitaremos introducir en la interfaz para su simulación son:

Variable simulador	Variable física	Comentario
Aux1	Cs	Capacidad de muestreo
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.4. Variables: Integrador. Ruido térmico SC.

La máscara que nos encontramos al pulsar sobre este tipo de no idealidad es el siguiente:

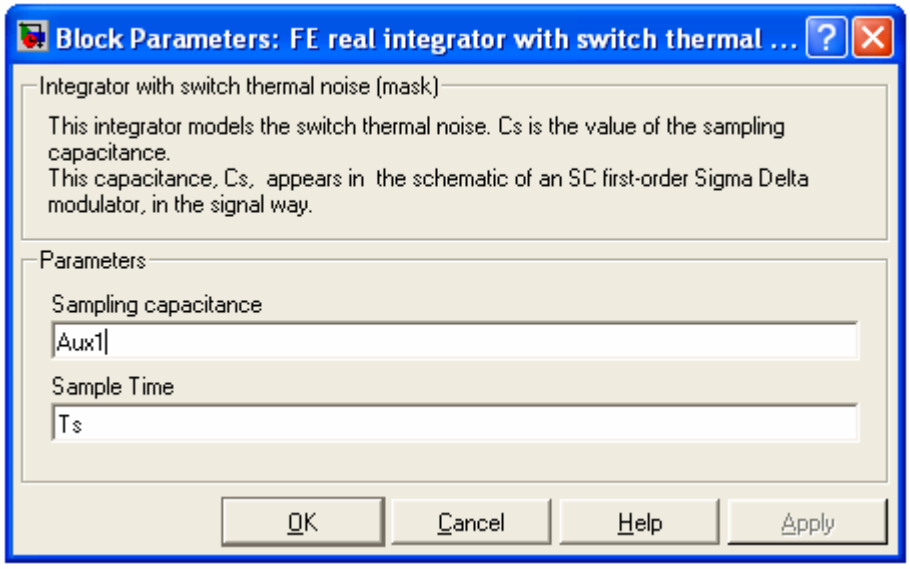


Fig. 7.7. Bloques no ideales: Máscara. Ruido térmico SC.

El diseño correspondiente para esta no idealidad ha sido diseñada bajo la siguiente estructura de bloques:

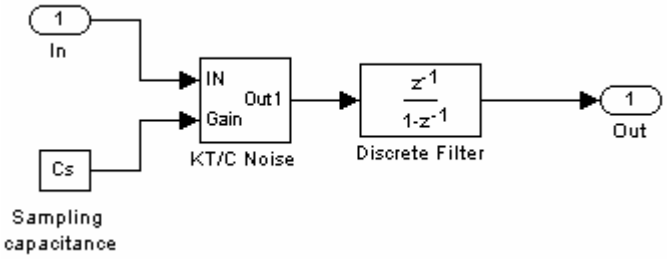


Fig. 7.8. Bloques no ideales: Diseño. Ruido térmico SC.

El bloque KT/C Noise de la Fig.7.8 está representado en la Fig.7.9

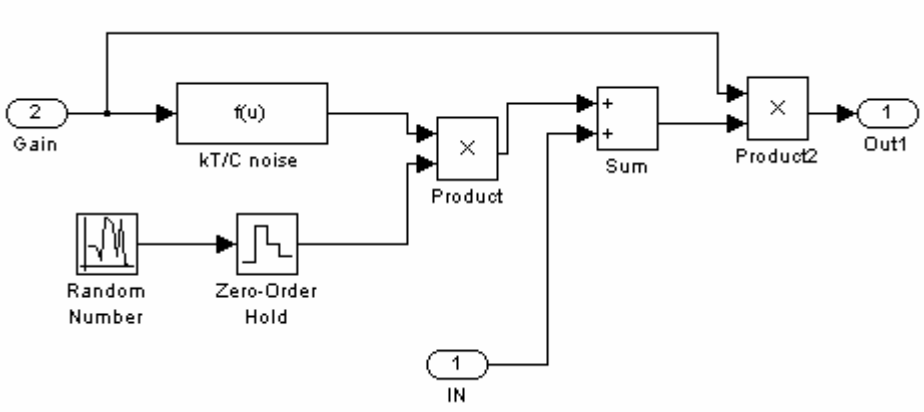


Fig. 7.9. Bloques no ideales: Bloque KT/C Noise.

2.2.1.2. Ruido en el amplificador operacional

El modelo para esta no idealidad lo representamos con el bloque:

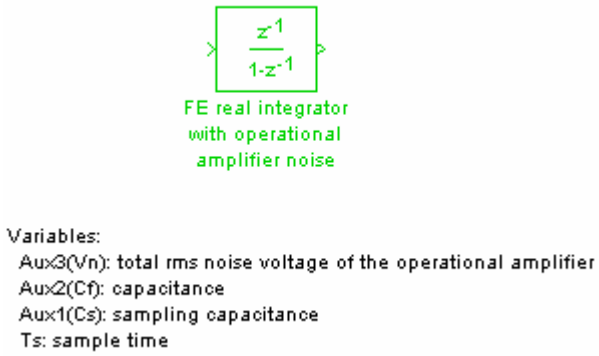


Fig. 7.10. Bloques no ideales: Integradores. Ruido amplificador operacional.

Los valores que necesitaremos introducir en la interfaz para su simulación son tres, y cada uno de ellos lo hacemos corresponder con una variable del simulador. Esta equivalencia la mostramos en la siguiente Tabla 7.5, donde además podemos ver el significado físico de cada uno, así como una breve explicación de a que se refiere cada variable en nuestro modelo.

Variable simulador	Variable física	Comentario
Aux1	Cs	Capacidad de muestreo
Aux2	Cf	Capacidad del amplificador
Aux3	Vn	Ruido total rms del A.O.
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.4. Variables: Integrador. Ruido amplificador operacional.

El cociente de los valores Cs/Cf representa el coeficiente del integrador. Para ello, a partir de la Fig.x.5 podemos obtener la función de transferencia en dominio z, dada por:

$$H_I(z) = \frac{Cs}{Cf}\frac{z^{-1}}{1-z^{-1}} = b\frac{z^{-1}}{1-z^{-1}} \qquad (x.3)$$

siendo b el coeficiente del integrador antes mencionado.

La máscara que nos encontramos al pulsar sobre este tipo de no idealidad es el siguiente:

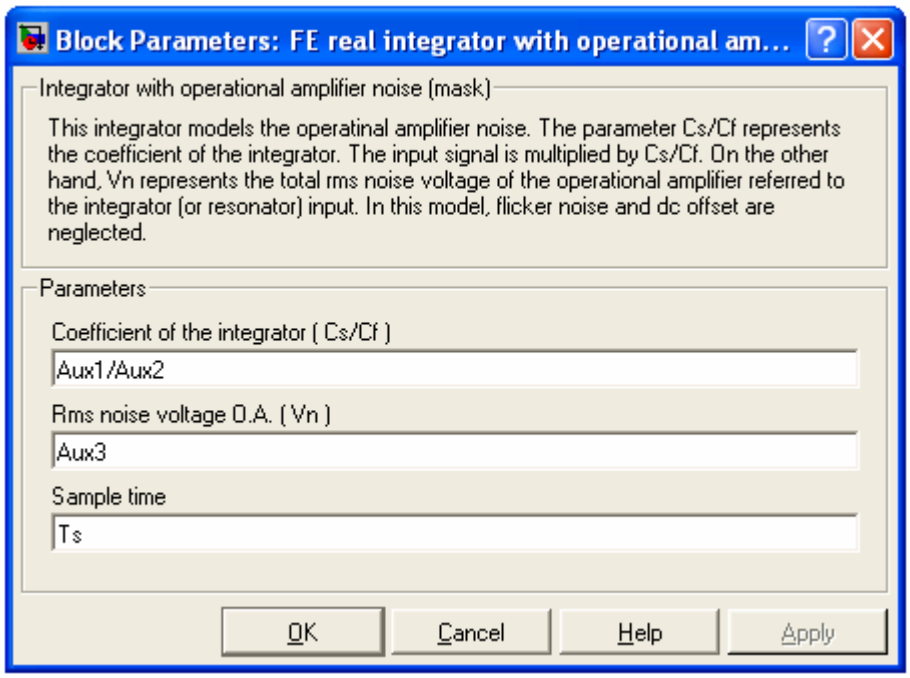


Fig. 7.11. Bloques no ideales: Máscara. Ruido amplificador operacional.

El diagrama de bloques al que corresponde es al siguiente:

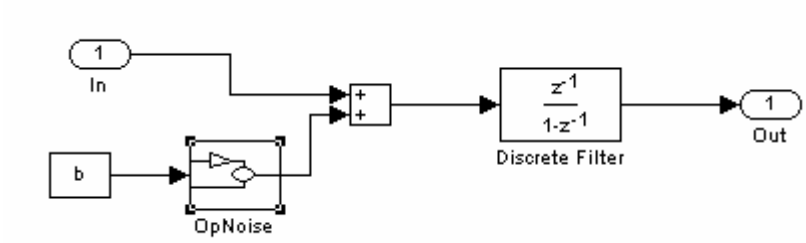


Fig. 7.12. Bloques no ideales: Diseño. Ruido amplificador operacional.

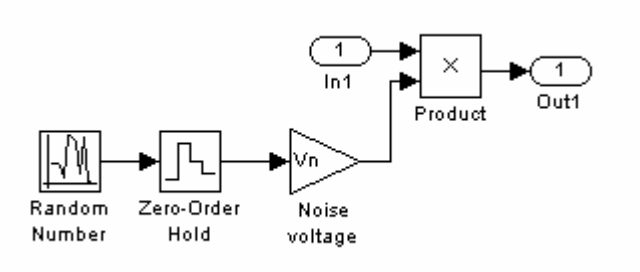


Fig. 7.13. Bloques no ideales: Bloque OpNoise.

2.2.1.3. Ganancia finita del amplificador operacional

El bloque que presentamos para este caso es el representado en la Fig.7.14:

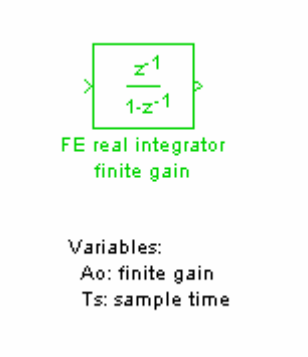


Fig. 7.14. Bloques no ideales: Integradores. Ganancia finita.

Para este caso ambas variables coinciden con entradas propias del simulador, ya que Ts es la inversa de la entrada Sampling Freq, y Ao es la entrada de Gain en la zona de no idealidades.

La máscara realizada para este caso contiene por tanto las dos variables Ao y Ts, sin ser necesaria ninguna variable auxiliar:

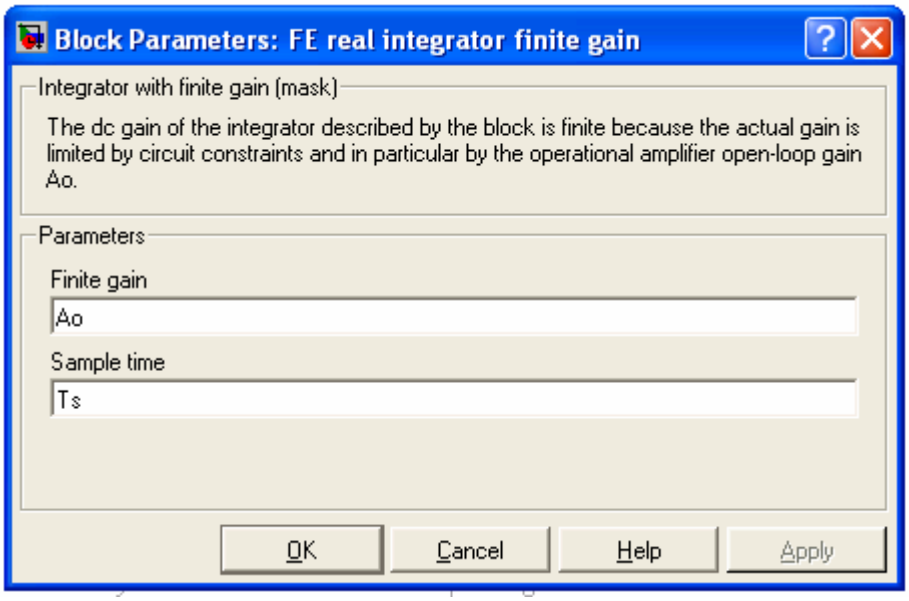


Fig. 7.15. Bloques no ideales: Máscara. Ganancia finita.

El diagrama de bloques diseñado para el caso de simular la ganancia finita procede de:

$$H(z)=\frac{z^{-1}}{1-\alpha\cdot z^{-1}}\xrightarrow{dcGain} Ao=H(1)=\frac{1}{1-\alpha}\xrightarrow{despejo\_\alpha}\alpha=1-\frac{1}{Ao} \quad (7.4)$$

Por tanto consiste en añadir α al lazo de realimentación del integrador de la siguiente forma:

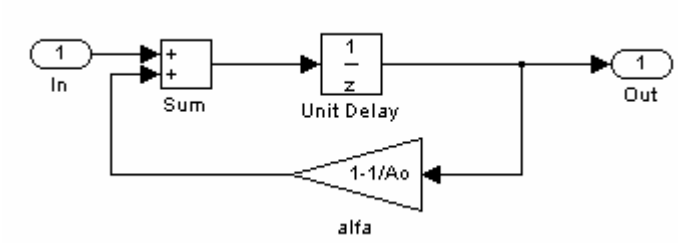


Fig. 7.16. Bloques no ideales: Diseño. Ganancia finita.

2.2.1.4. Ancho de banda del amplificador operacional y Slew-Rate

El modelo viene dado por:

Fig. 7.17. Bloques no ideales: Integradores. SR+GBW.

Todas las variables forman parte de entradas de la interfaz del simulador. Ao (Gain), SR y Tau están en la zona de no idealidades y Ts (la inversa de Sampling Freq) en la zona de entradas.

La máscara hecha para este caso queda de la forma:

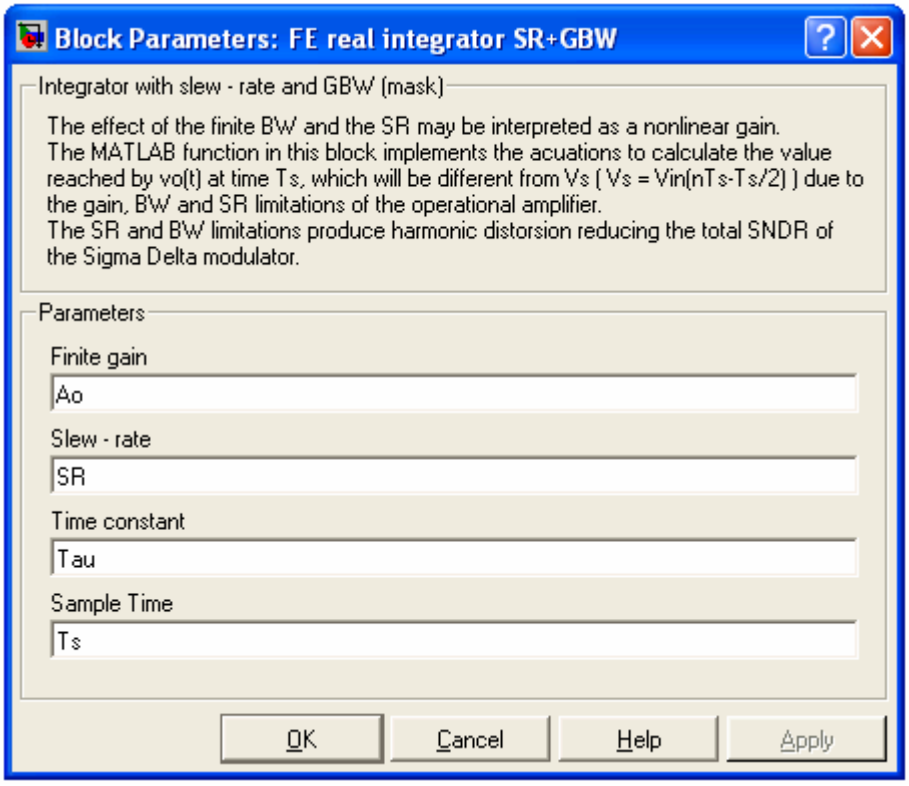


Fig. 7.18. Bloques no ideales: Máscara. SR+GBW.

El efecto del ancho de banda finito (BW) y del slew-rate (SR) puede ser interpretado como una no idealidad en la ganancia. Refiriéndonos al modulador de primer orden mostrado en la Fig.7.5, la evolución de la salida durante el periodo n de integración (cuando φ_2 está encendido) viene dada por:

$$vo(t) = vo(nTs - Ts) + \alpha Vs(1 - e^{-\frac{t}{\tau}}) \text{ en } 0<t<\frac{Ts}{2} \qquad (7.5)$$

donde $Vs = Vin(nTs - \frac{Ts}{2})$, α proviene de la ganancia finita del amplificador operacional, τ es la constante de tiempo del integrador e igual a $\tau = \frac{1}{2\pi GBW}$ y GBW es el producto ganancia ancho de banda.

El valor de esta curva toma el máximo cuando t=0, dando:

$$\frac{d}{dt}vo(t) = \alpha\frac{Vs}{\tau} \qquad (7.6)$$

Consideramos dos casos:

3. El valor de (7.6) es más pequeño que el SR del amplificador operacional. En

este caso el SR no limita la evolución de vo(t) durante todo un periodo de reloj (hasta t=Ts/2).

4. El valor de (7.6) es mayor que el SR, en ese caso:

$$t \le to:\ vo(t) = vo(nTs - Ts) + SRt$$

$$t > to:\ vo(t) = vo(to) + (\alpha Vs - SRto)\left(1 - e^{\frac{t-to}{\tau}}\right) \text{ siendo } to = \frac{\alpha Vs}{SR} - \tau \qquad (7.7)$$

Según todo esto, el modelo que tomamos la simular el efecto del SR y del ancho de banda, es el siguiente:

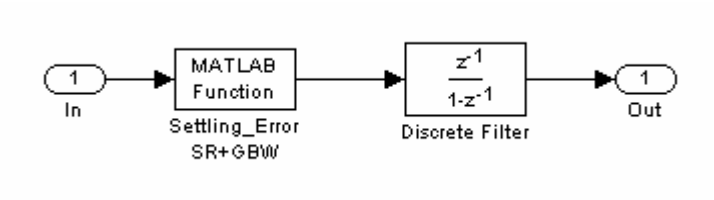


Fig. 7.19. Bloques no ideales: Diseño. SR+GBW.

A continuación mostramos la función matlab que cumple el desarrollo matemático expuesto anteriormente que modela los efectos de SR y del GBW en amplificadores operacionales:

```
% Calculates the gain error due to settling and slew-rate

function y = settlinggain(x,Ao,SR,Tau,Ts)

% output = settlinggain(input,Av,SR,Tau,Ts)

alfa=1-1/Ao;
to=alfa*x/SR-Tau;

if x<(Tau*SR/alfa)
    y=alfa*x*(1-exp(-Ts/2/Tau));
end

if (x>=(Tau*SR/alfa))&(to>=ts/2)
    y=SR*Ts/2;
end

if (x>(Tau*SR/alfa))&(to<Ts/2)
    y=SR*to+(alfa*x-SR*to)*(1-exp(-1*(Ts/2-to)/Tau));
end
```

2.2.1.5. Tensión de saturación del amplificador operacional

El modelo utilizado es el siguiente:

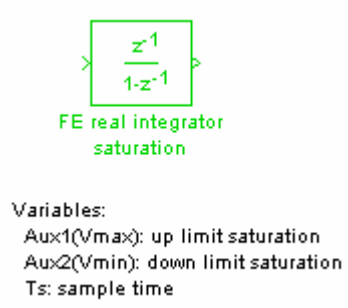


Fig. 7.20. Bloques no ideales: Integradores. Saturación.

Las variables que tomaremos para su simulación y que deberemos introducir en la interfaz de simconverter son las siguientes:

Variable simulador	Variable física	Comentario
Aux1	Vmax	Limite de saturación superior
Aux 2	Vmin	Limite de saturación inferior
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.5. Variables: Integrador. Saturación.

La máscara estará formada por estas variables:

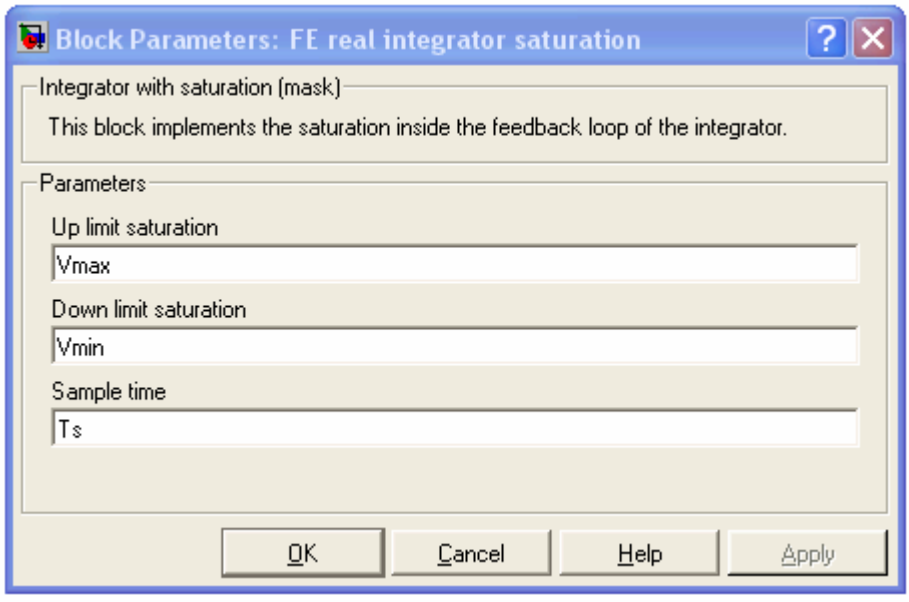


Fig. 7.21. Bloques no ideales: Máscara. Saturación.

El modelo de bloques desarrollado para este caso es muy sencillo, y consiste en

saturar la salida del integrador entre unos ciertos niveles (Vmax y Vmin).

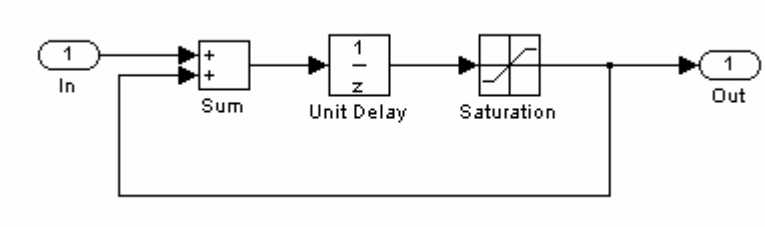


Fig. 7.22. Bloques no ideales: Diseño. Saturación.

2.2.1.6. Integrador con todas las no idealidades

Presentamos un modelo de integrador en el que hemos unido todas las no idealidades vistas hasta ahora para integradores de convertidores analógicos digitales Sigma Delta. El modelo en cuestión, que se encuentra dentro de la librería de integradores de los bloques no ideales, es el siguiente:

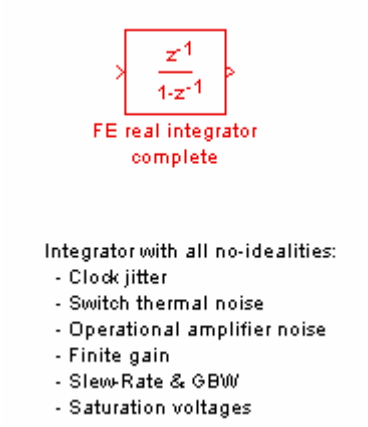


Fig. 7.23. Bloques no ideales: Integrador no ideal completo.

En este caso en el que el número de no idealidades es superior[7] al valor de variables auxiliares que dispone por ello hay que tener especial cuidado con dar valor

[7] Que el número de no idealidades sea superior al de variables auxiliares no hace que nuestra interfaz sea limitada, ya que si recordamos, las variables auxiliares nos permiten dar valores a no idealidades pero tienen como objetivo realizar un barrido de los valores de éstas respecto a otras. En caso de tener más no idealidades que variables auxiliares, una opción es que el usuario se cree un archivo con extensión .m con las variables que quiere modelar y sus valores, haciendo coincidir el nombre de las variables con el de la máscara. Otra forma también válida es sustituir directamente el valor en la máscara del bloque del que vamos a estudiar la no idealidad de aquellas variables en las que no vamos a necesitar hacer un barrido de las mismas.

a todas las variables que no correspondan con las variables auxiliares y no sean entradas de nuestra interfaz, por ejemplo para el caso que dispongo el valor b, que es el coeficiente del integrador.

Para indicar cada una de estas variables que visualizaremos más adelante en la máscara, detengámonos primero en la Tabla 7.6 que nos muestra cada una de ellas, tanto su nombre físico como el que debemos buscar el la interfaz del simulador para sustituir su valor.

Variable simulador	Variable física	Comentario
Aux1	Vmax	Límite de saturación superior
Aux2	Vmin	Límite de saturación inferior
Aux3	Cs	Capacidad de muestreo
Aux4	Vn	Ruido total rms del A.O.
-	b	Coeficiente del integrador
Gain	Ao	Ganancia finita
SR	SR	Slew-rate
Tau	Tau	Constante de tiempo
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.6. Variables: Integrador no ideal completo.

La máscara realizada es la siguiente:

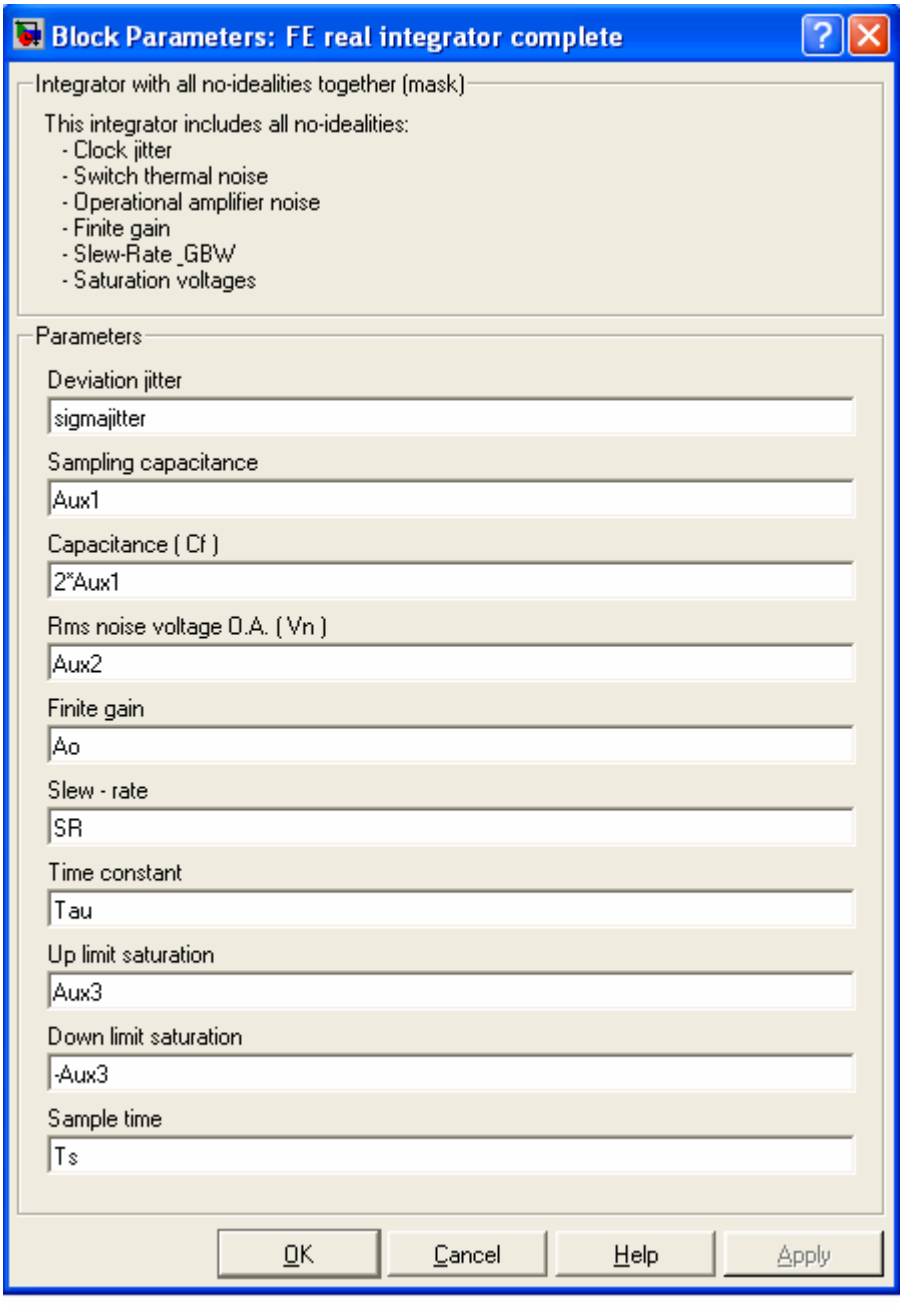


Fig. 7.24. Máscara. Integrador no ideal completo.

Consideramos que Cf va a ser siempre el doble que Cs para que el coeficiente de integración sea ½. Por simplificación también consideramos que Vmax y Vmin tienen los mismos valores absolutos y por ello tomamos la misma variable pero negada.

2.2.2. Comparadores

2.2.2.1. Histéresis

El modelo que proponemos para el caso de la histéresis de los comparadores:

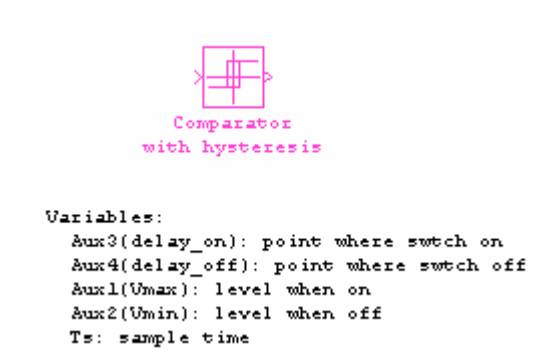


Fig. 7.25. Bloques no ideales: Comparadores. Histéresis.

Este modelo pertenece a la librería de simulink, por tanto tan sólo debemos adaptar las variables a las entradas de la interfaz.

2.2.2.2. Offset

El modelo que hemos diseñado es el siguiente:

Fig. 7.26. Bloques no ideales: Comparadores. Offset.

Las variables que utilizaremos para su situación, y que deberemos introducir en la interfaz de simconverter son los mostrados en la Tabla 7.6

Variable simulador	Variable física	Comentario
Aux1	offset	Offset del comparador
Sampling Freq	Ts	Tiempo de muestreo

Tabla 7.7. Variables: Comparador. Offset.

Según esto, la máscara realizada para este bloque queda:

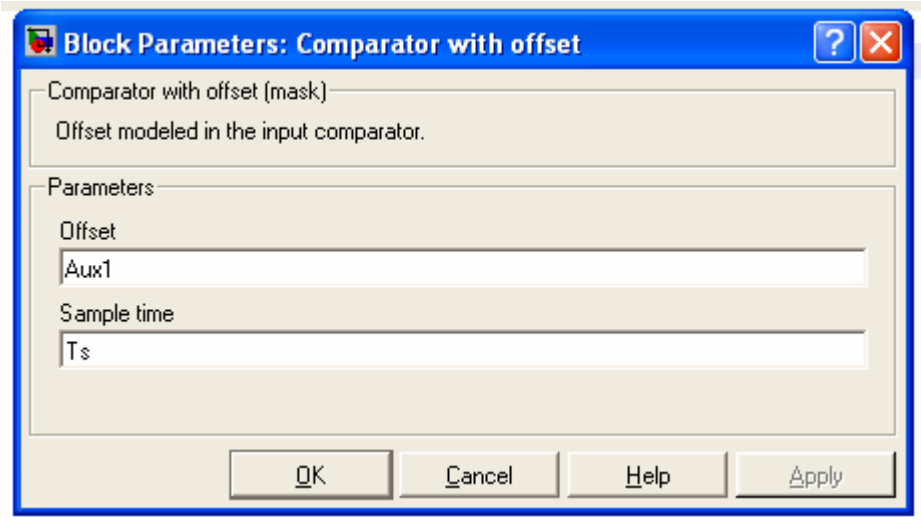


Fig. 7.27. Bloques no ideales: Máscara. Offset.

El diseño implementado para la simulación del offset, consiste sencillamente en sumarle una constante antes de realizar la comparación.

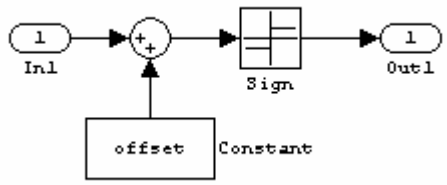


Fig. 7.28. Bloques no ideales: Diseño. Offset.

2.2.3. Ruido Jitter

El modelo que hemos diseñado viene dado por el siguiente bloque:

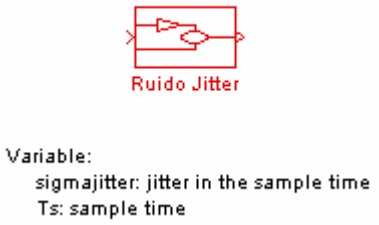


Fig. 7.29. Bloques no ideales: Ruido Jitter.

Ambas variables, sigmajitter (desviación del ruido jitter) y Ts (tiempo de muestreo), son entradas de la interfaz del simulador, la primera dentro de la zona de no idealidades y la segunda dentro de la zona de entradas.

La máscara para este caso aparecería una ventana como la de la Fig.7.30:

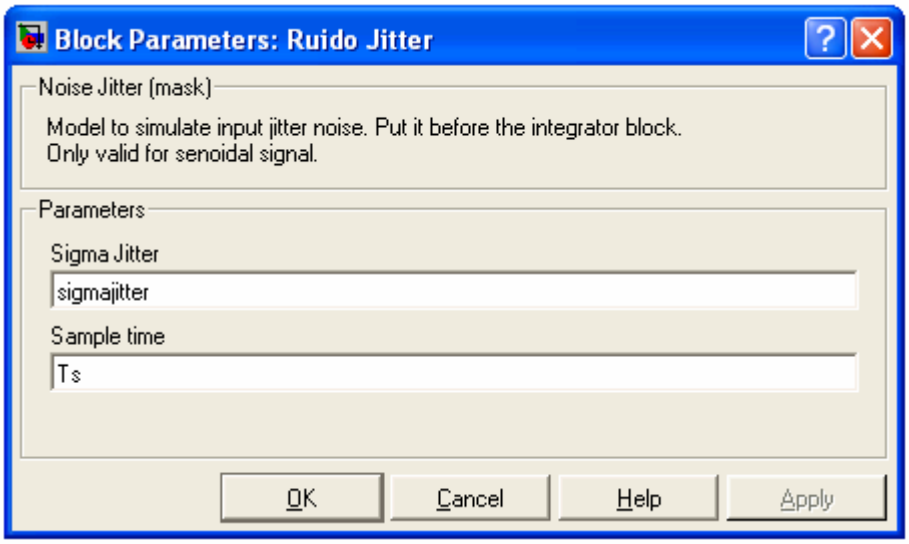


Fig. 7.30. Bloques no ideales: Ruido Jitter.

El efecto del ruido jitter en los moduladores Sigma Delta es independiente de la estructura u orden del modulador. El resultado es un muestreo no uniforme en la secuencia temporal y produce un error, el cual incrementa la potencia del ruido a la salida del cuantizador.

El error introducido cuando una señal sinusoidal a la entrada x(t) con

amplitud A y frecuencia fsin es muestreada en un instante con error δ está dado por:

$$x(t+\delta) - x(t) = 2\pi f_{\sin} \delta A \cos(2\pi f_{\sin} t) = \delta \frac{d}{dt} x(t) \qquad (7.8)$$

El diagrama de bloques diseñado para emular el ruido jitter a la entrada siguiendo la fórmula (1), vendría dado por:

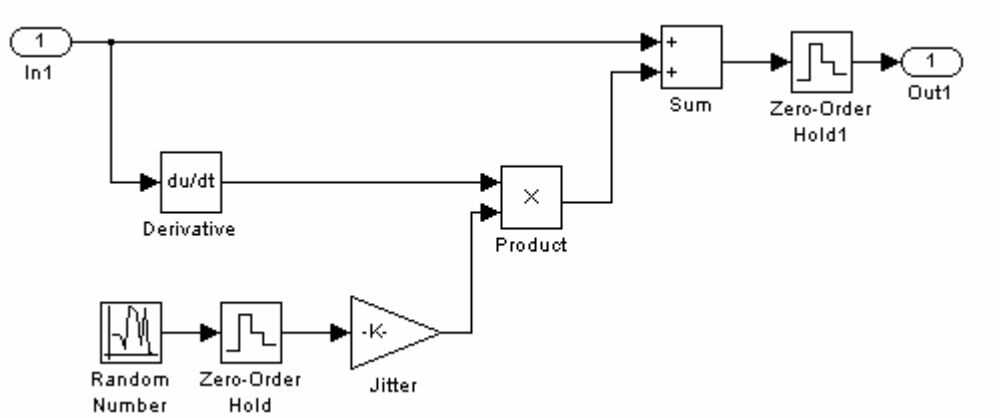


Fig. 7.31. Bloques no ideales: Diseño. Ruido Jitter.

2.3. Ejemplos

En este apartado vamos a mostrar algunos ejemplos de convertidores Sigma Delta en Tiempo Discreto. En concreto hemos diseñado cuatro que cito a continuación:

1. ∑Δ de 2º orden, paso de baja, lazo simple y componentes ideales.
2. ∑Δ de 2º orden, paso de baja, lazo simple y componentes no ideales.
3. ∑Δ de 2º orden, paso de baja, cascada y componentes ideales.
4. ∑Δ de 2º orden, paso de baja, cascada y componentes no ideales.

Estos ejemplos nos ayudarán a la compresión de este tipo de convertidores o nos podrán servir de base para nuestro modelo.

Pasamos a realizar una descripción a nivel de bloques de estos ejemplos diseñados, y en el Capítulo 8 mostraremos los resultados obtenidos tras la simulación de algunos de estos ejemplos. Pero antes, mostremos todos los ejemplos realizados para sigma delta, y así tener una visión general de esta parte de la librería.

Sampling Freq: 1/Ts =
FS: FS =
Nbits: N =
Amplit: A =
Frec: fsin =
sigmajitter: sigmajitter =
Values in the interface of the simulator:
Inl
Outl
TD Sigma Delta LP 2° order simple ideal
TD Sigma Delta LP 2° order - simple real
Cs (sampling capacitance): Aux1=
Gain: AodB =
TD Sigma Delta LP 2° order simple - real - switch thermal
TD Sigma Delta LP 2° order simple - real - gain
Un(total rms noise voltage of the operational amplifier): Aux3 =
Cs/Cf (coefficient of the integrator): Aux1/Aux2 =
Gain: AodB =
SR: SR =
Tau: Tau =
TD Sigma Delta LP 2° order simple - real - op noise
TD Sigma Delta LP 2° order simple - real - SR & GBW
Cs (sampling capacitance): Aux1=
Cf (capacitance): Aux2=
Un(total rms noise voltage of the operational amplifier): Aux3 =
Up limit saturation: Aux1=
Down limit saturation: Aux2 =
TD Sigma Delta LP 2° order simple - real - op noise & switch thermal
TD Sigma Delta LP 2° order simple - real - sat
Input
Output
ADC to ADSL+ aplication: 14 bits and BW 2.2 MHz ideal
ADC to ADSL+ aplication: 14 bits and BW 2.2 MHz real
Sampling Freq: 1/Ts =
FS: FS =
Nbits: N =
Amplit: A =
Frec: fsin =
sigmajitter: sigmajitter =
Values in the interface of the simulator:

2.3.1. ∑Δ de 2º orden, paso de baja, lazo simple y componentes ideales

El modelo en estudio viene definido por el bloque:

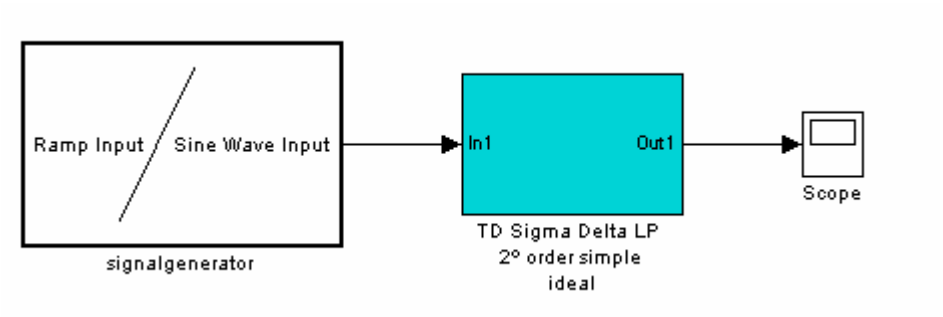


Fig. 7.32. Ejemplo: ∑Δ de 2º orden, paso de baja, lazo simple y componentes ideales.

Los valores de simulación[8] de la entrada para este ejemplo son:

- Sampling Freq: 1/Ts = 1600 Hz
- FS: FS = 2 V
- Nbits: N = 8
- Amplit: A = 0.5 V
- Frec: fsin = 70 Hz

O también otra forma de cargar estos valores es mediante la opción Load del menú Environment de la interfaz de simconverter. Para ello cargamos el archivo que contenga el mismo nombre que el ejemplo.

El diseño considerado para este ejemplo lo mostramos a continuación:

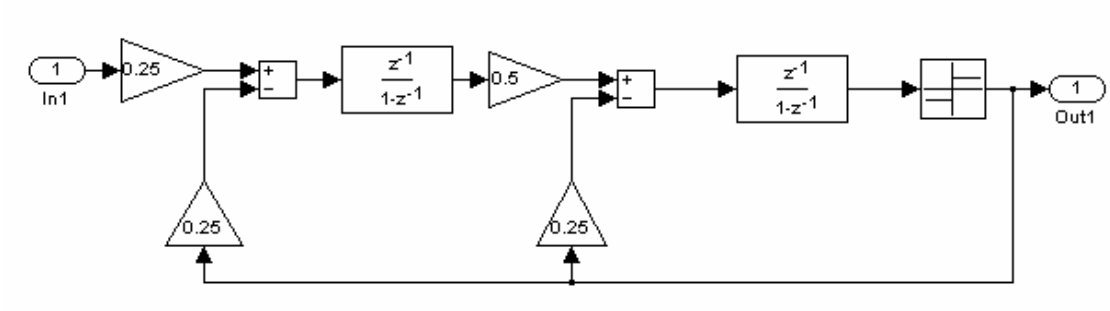


Fig. 7.33. Diseño: ∑Δ de 2º orden, paso de baja, lazo simple y componentes ideales.

[8] Estos valores de simulación son orientativos y pueden ser modificado por el usuario en la interfaz.

2.3.2. ∑Δ de 2º orden, paso de baja, lazo simple y componentes no ideales

Este modelo es similar al anterior, con la salvedad que hemos introducido componente no ideales al diseño. El bloque correspondiente a este ejemplo:

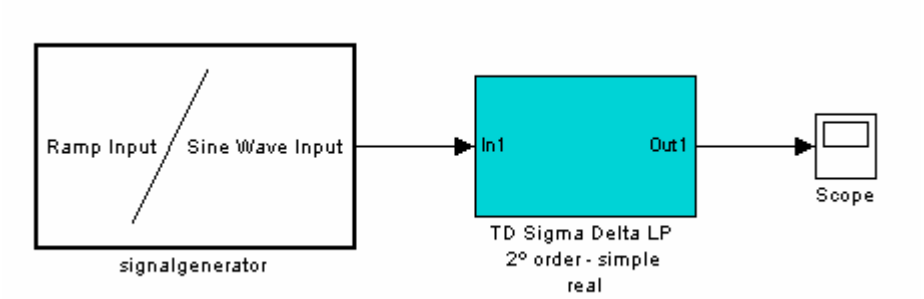


Fig. 7.34. Ejemplo: ∑Δ de 2º orden, paso de baja, lazo simple y componentes no ideales.

Para este diseño los valores de simulación[9] son:

- Sampling Freq: 1/Ts = 1600 Hz
- FS: FS = 2 V
- Nbits: N = 8
- Amplit: A = 0.5 V
- Frec: fsin = 70 Hz
- Gain: AodB = 22 dB
- Cs[10]: Aux1 = 1e-12
- Vn: Aux2 = 10e-3
- Vmax[11]: Aux3 = 0.65
- Sigmajitter = 3e-5

O también otra forma, al igual que el caso anterior, es cargar estos valores es mediante la opción Load del menú Environment de la interfaz de simconverter.

El diseño considerado para este ejemplo, es a partir de modelo real, añadir las no idealidades de jitter a la entrada de la señal y ruido en los amplificadores[12].

[9] Estos valores de simulación son orientativos y se cargan automáticamente. El usuario podría modificarlos por otros en la interfaz.
[10] El valor de Cf es el doble de Cs para tener un coeficiente de integración igual a ½.
[11] Tomaremos el valor de Vmin igual al negado del valor absoluto de Vmax.
[12] Los modelos de jitter noise y del integrador real ya han sido mostrados en el apartado 2.2.4 y 2.2.1.2 de este mismo capítulo.

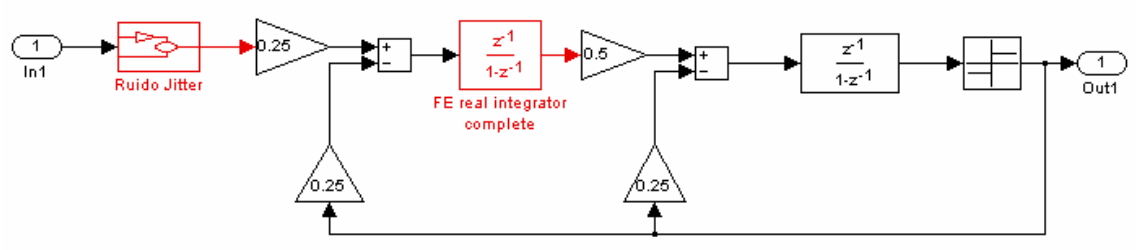


Fig. 7.35. Diseño: ∑Δ de 2º orden, paso de baja, lazo simple y componentes no ideales.

2.3.3. ∑Δ de 2º orden, paso de baja, cascada y componentes ideales

Presentamos un bloque que nos va a simular en mismo caso anterior, pero para ello vamos a utilizar un modelo en cascada. El bloque que nos representará este ejemplo es el siguiente:

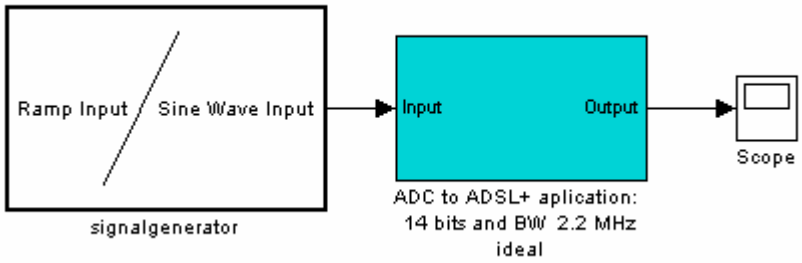


Fig. 7.36. Ejemplo: ∑Δ de 2º orden, paso de baja, cascada y componentes ideales.

Los valores de simulación que tomaremos para el caso de una aplicación ADSL+ son los siguientes:

- Sampling Freq: 1/Ts = 704e6 Hz
- FS: FS = 2 V
- Nbits: N = 14
- Amplit: A = 0.5 V
- Frec: fsin = 550 Hz

Estos valores también podremos cargarlos a partir de la opción Load del menú Environment de la interfaz.

El diseño desarrollado queda representado en la siguiente ilustración:

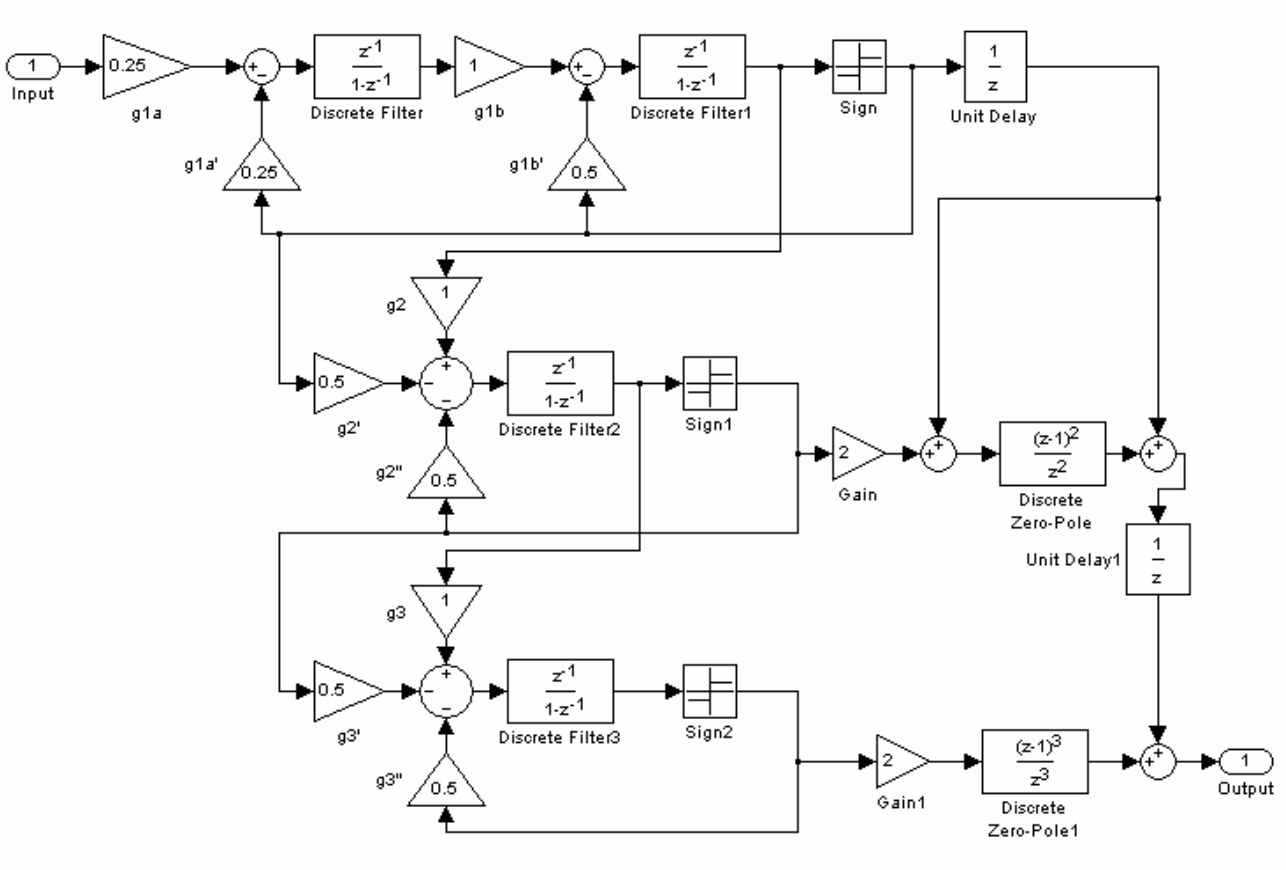


Fig. 7.37. Diseño: ∑Δ de 2º orden, paso de baja, cascada y componentes ideales.

2.3.4. ∑Δ de 2º orden, paso de baja, cascada y componentes no ideales

El bloque que mostramos en este apartado es similar al anterior, con la salvedad que algunos elementos del mismo no son ideales.

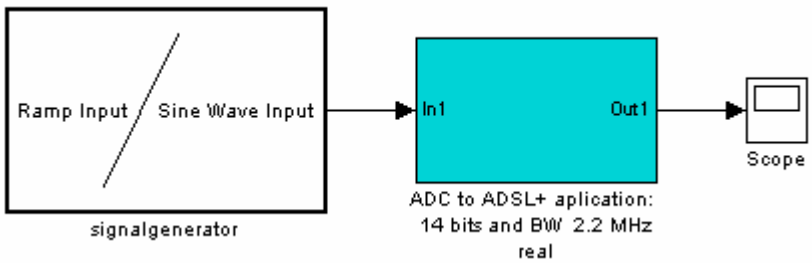


Fig. 7.38. Ejemplo: ∑Δ de 2º orden, paso de baja, cascada y componentes no ideales.

Los valores que tomaremos para la simulación serán los correspondientes a la señal sinusoidal de entrada, así como a las no idealidades de ruido jitter a la entrada y de ruido de los amplificadores operacionales:

- Sampling Freq: 1/Ts = 704e6 Hz
- FS: FS = 2 V
- Nbits: N = 14
- Amplit: A = 0.5 V
- Frec: fsin = 550 Hz
- Gain: AodB = 30 dB
- Cs: Aux1 = 6pF
- Vn: Aux2 = 1.7e-5
- Vmax: Aux3 = 0.7
- Sigmajitter = 40 ps

Estos valores también podremos cargarlos por medio del menú Environment de la interfaz.

El diseño para este caso, es similar al de el caso ideal, sustituyendo los elementos reales:

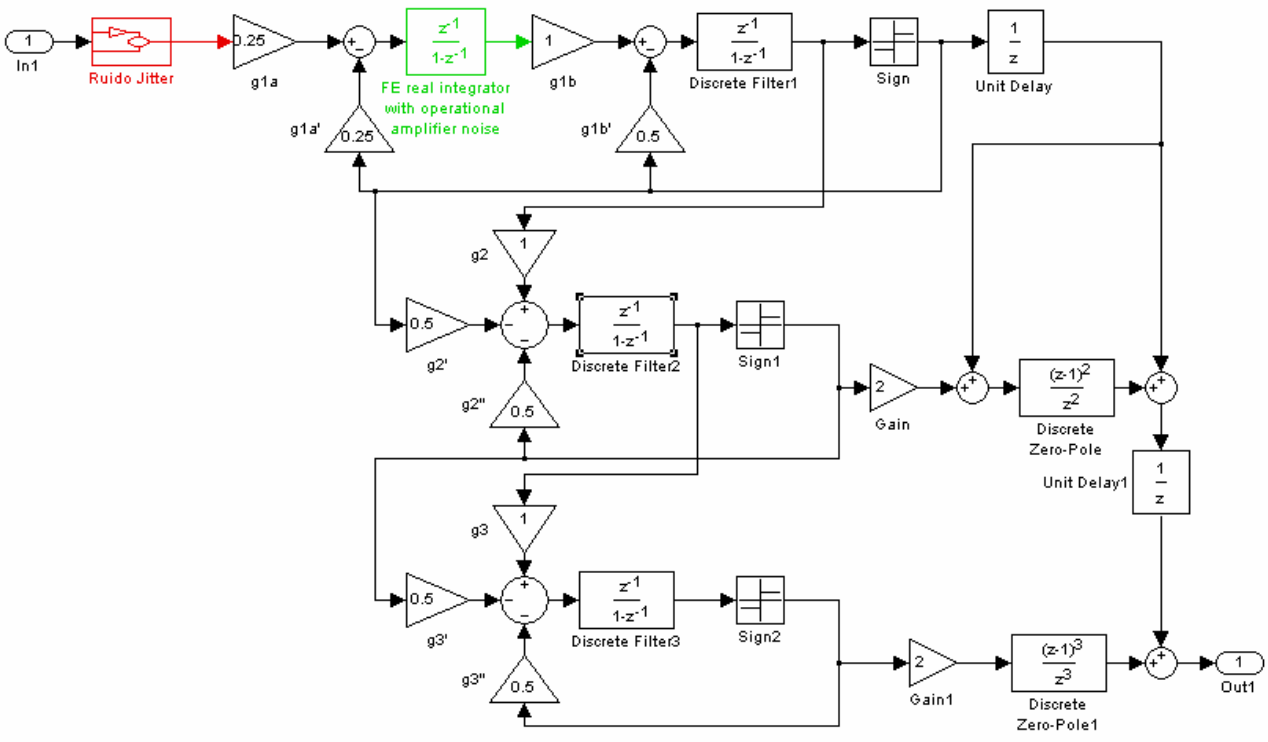
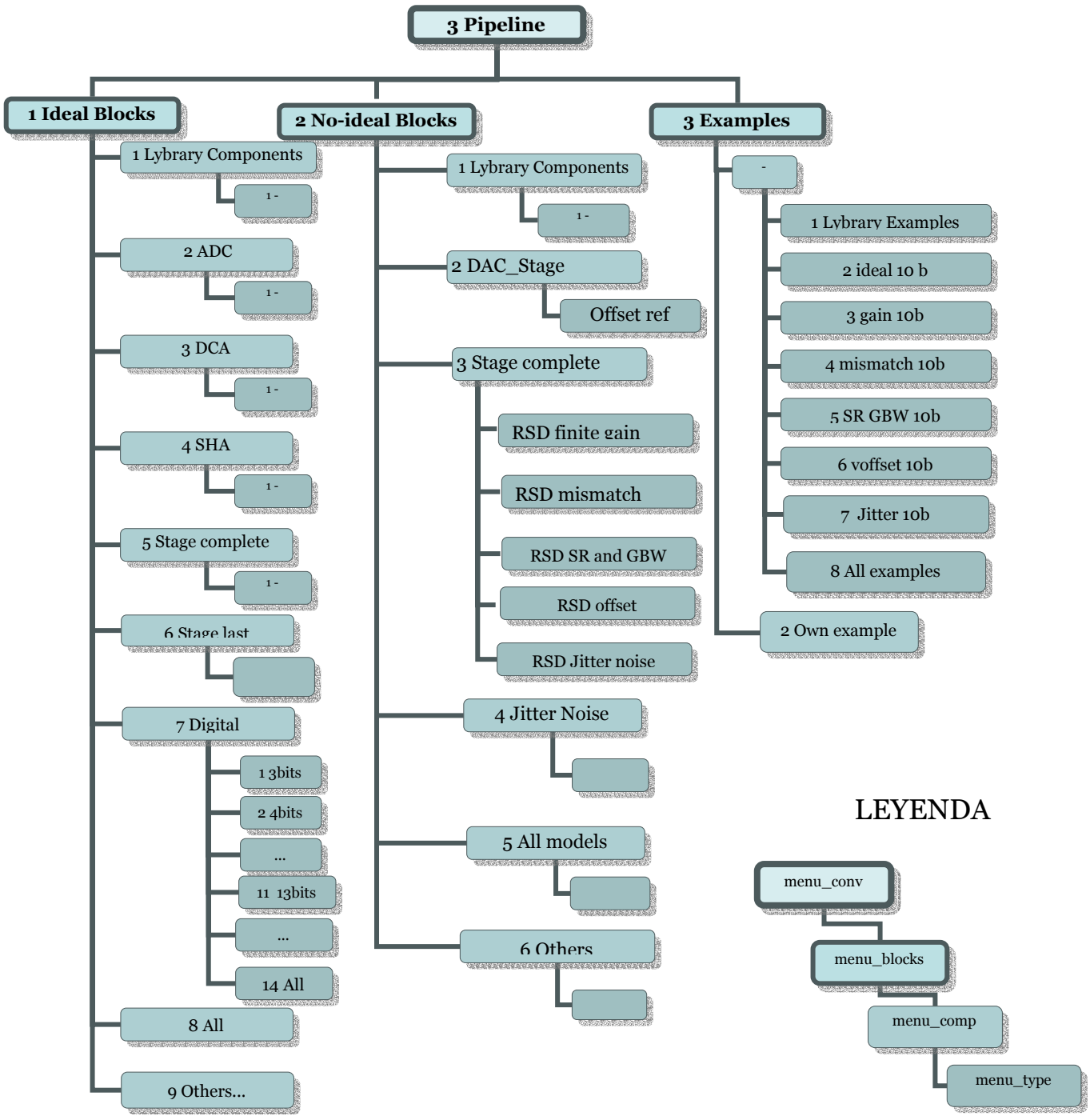


Fig. 7.39. Diseño: ∑Δ de 2º orden, paso de baja, cascada y componentes no ideales.

3. Librería de Pipeline

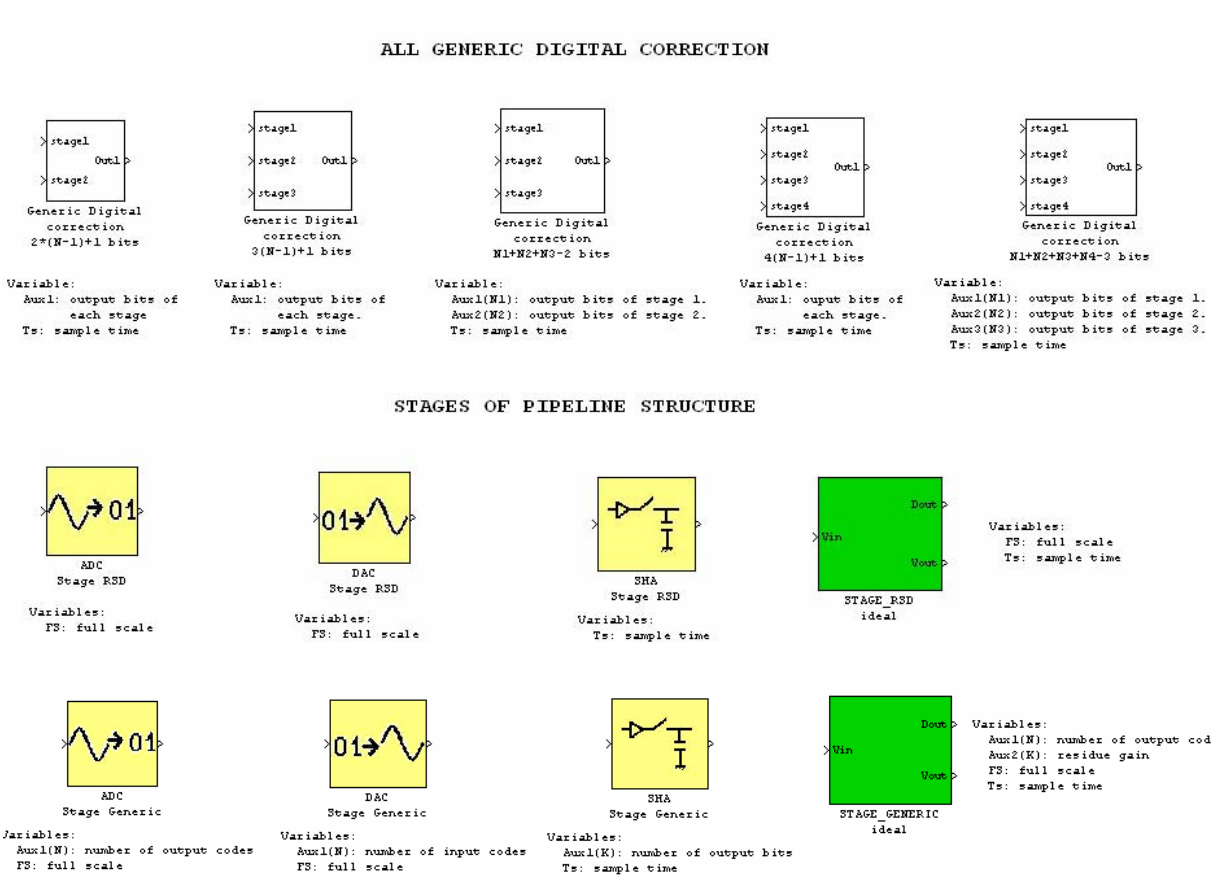
Para tener una visión completa de la librería, muestro a continuación el siguiente esquema en forma de árbol que incluye a grande rasgos los modelos diseñados para convertidores pipeline:

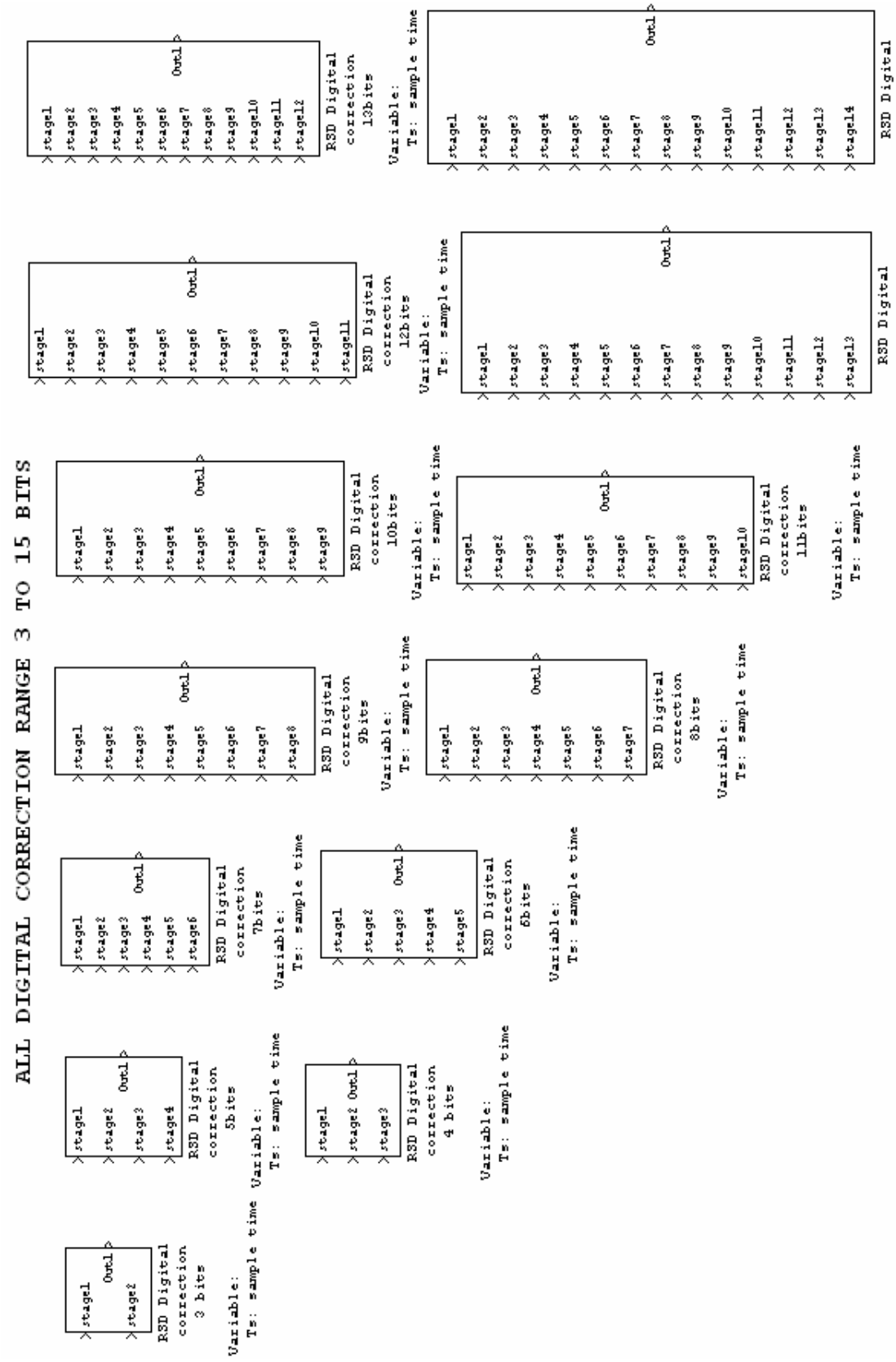


3.1. Bloques ideales

Tras el estudio realizado de convertidores Pipeline, llegamos a la conclusión que necesitamos incluir tres tipos de bloques: ADC, DAC, SHA, Stage completo y último de la cadena, corrección digital. Para cada uno de estos bloques veremos el caso RSD, explicado en el capítulo 3, y el caso genérico.

Una visión global de los bloques generados es la siguiente:




ALL DIGITAL CORRECTION RANGE 3 TO 15 BITS
stage1
stage2
Out1
RSD Digital correction 3 bits
Variable:
Ts: sample time
stage3
stage4
RSD Digital correction 5bits
stage5
stage6
RSD Digital correction 7bits
stage7
stage8
RSD Digital correction 9bits
stage9
RSD Digital correction 10bits
stage10
stage11
RSD Digital correction 12bits
stage12
stage13
stage14
RSD Digital correction 13bits
RSD Digital correction 4 bits
RSD Digital correction 6bits
RSD Digital correction 8bits
RSD Digital correction 11bits
RSD Digital

Veamos a continuación los modelos desarrollados para cada uno de ellos.

3.1.1. ADC_Stage

Los modelos desarrollados son:

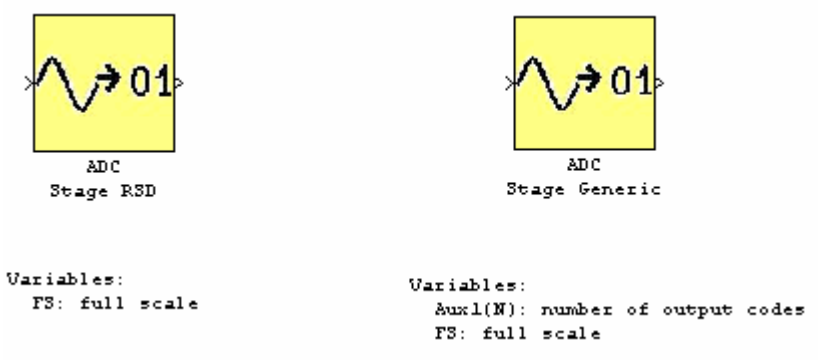


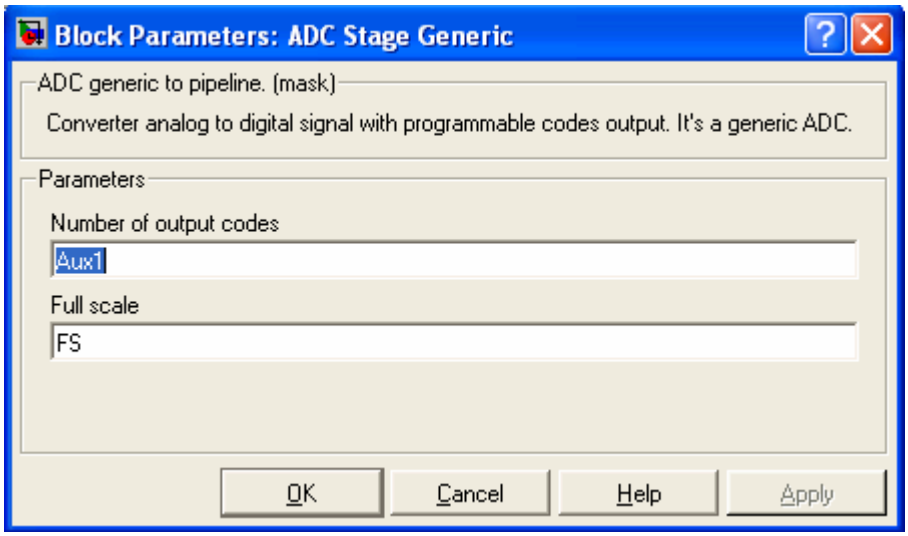
Fig.7.40. Bloques ideales. ADC_Stage.

El primer bloque modela un ADC cuya salida tiene 2 códigos de salida. El segundo, esta variable es programable.

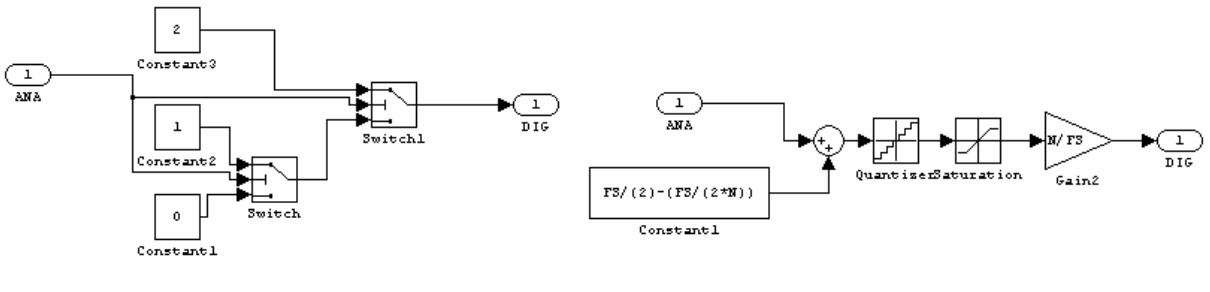
Variable simulador	Variable física	Comentario
Aux1	N	Número de códigos de salida
FS	FS	Fondo de escala

Tabla.7.8. Variables: Bloques ideales. ADC_Stage.

Al pinchar sobre ellos nos aparece la máscara creada. Por ejemplo la del ADC genérico es:



Los diseños de cada uno son los siguientes:



(a) **(b)**

Fig.7.41. (a) Diseño ADC_Stage RSD (b) Diseño ADC_Stage genérico

3.1.2. DAC_Stage

Para el caso del DAC, los modelos son:

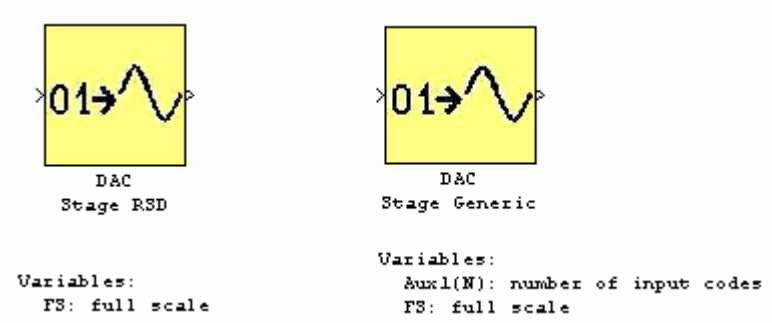


Fig.7.42. Bloques ideales. DAC_Stage.

Las variables son como las del ADC pero con la conversión contraria, FS para el fondo de escale, y ahora N, como el número de códigos de entrada (igual a la N del DAC).

El diseño en bloques Simulink desarrollado es el siguiente:

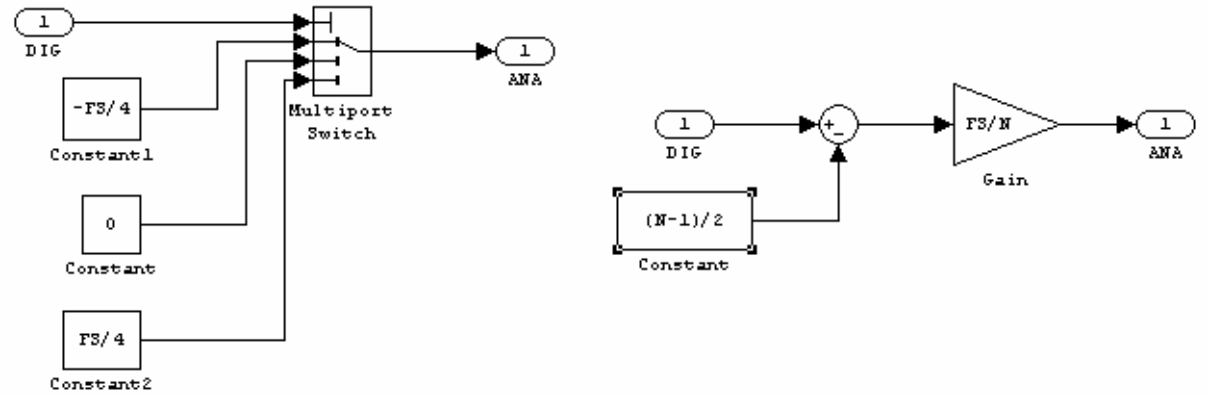


Fig.7.43. (a) Diseño DAC_Stage RSD (b) Diseño DAC_Stage genérico

3.1.3. SHA_Stage

Los modelos para la función de sample and hold con ganancia son:

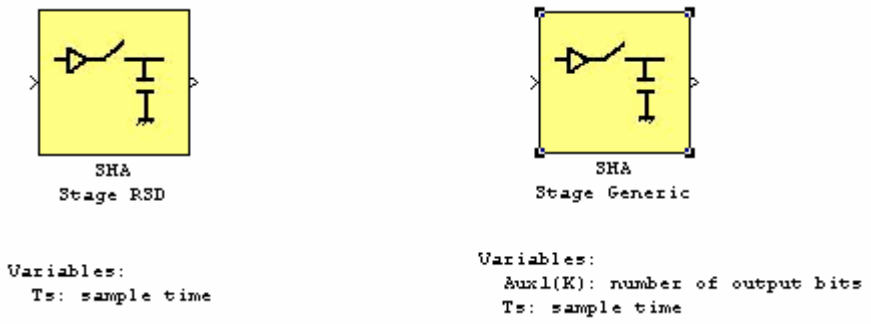


Fig.7.44. Bloques ideales. SHA_Stage.

Las variables son el tiempo de muestro y la ganancia que coincide con el número de códigos de salida, N, que vimos en el apartado del ADC.

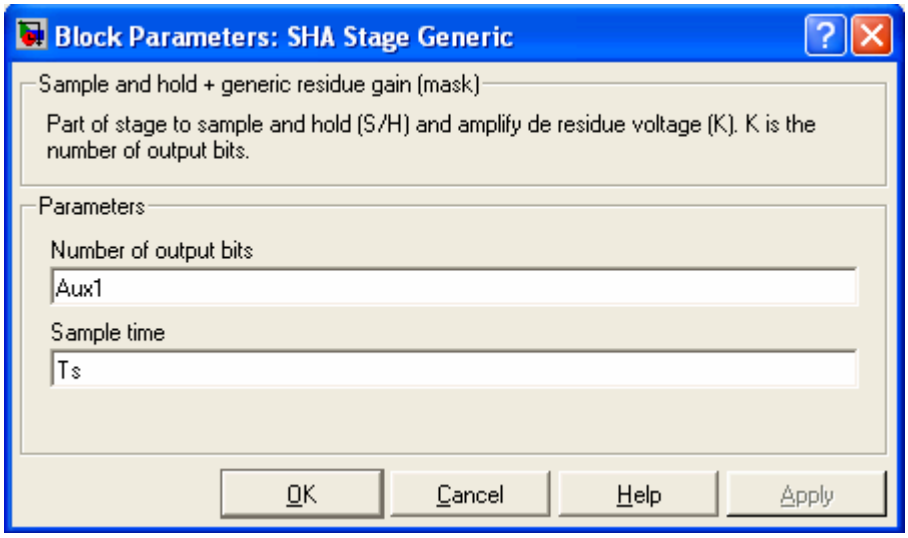
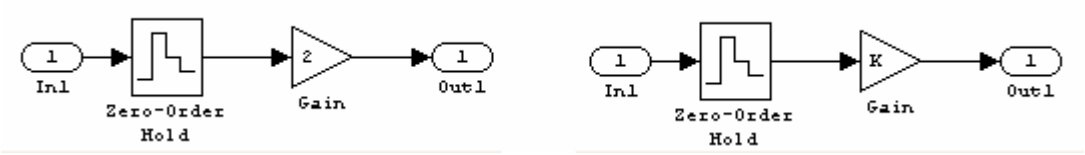


Fig.7.45. Variables: Bloques ideales. SHA_Stage.

Los diseños lo mostramos en la Fig. 7.45:



(a) **(b)**

Fig.7.46. (a) Diseño SHA_Stage RSD (b) Diseño SHA_Stage genérico

3.1.4. Stage_complete

Los modelos para RSD y genérico son:

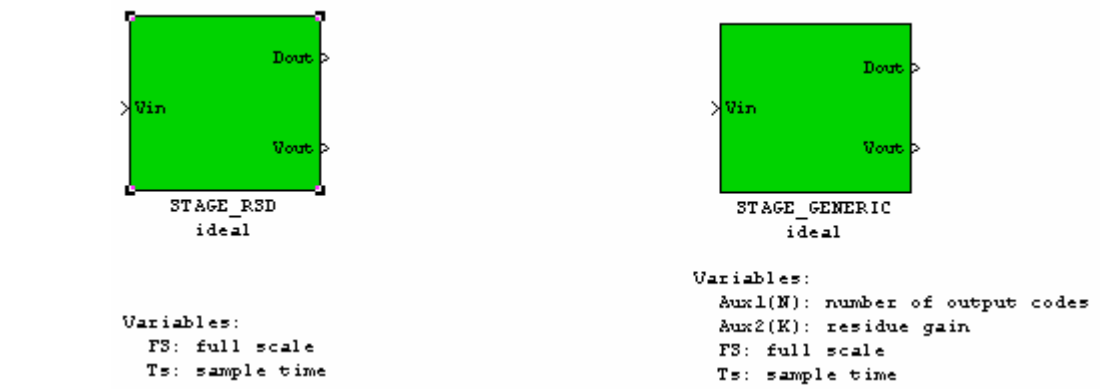


Fig.7.47. Bloques ideales. Stage_complete.

En la leyenda de cada bloque podemos ver las variables que necesitan. Pasemos ahora a mostrar los diseños a nivel e bloques de cada uno de ellos:

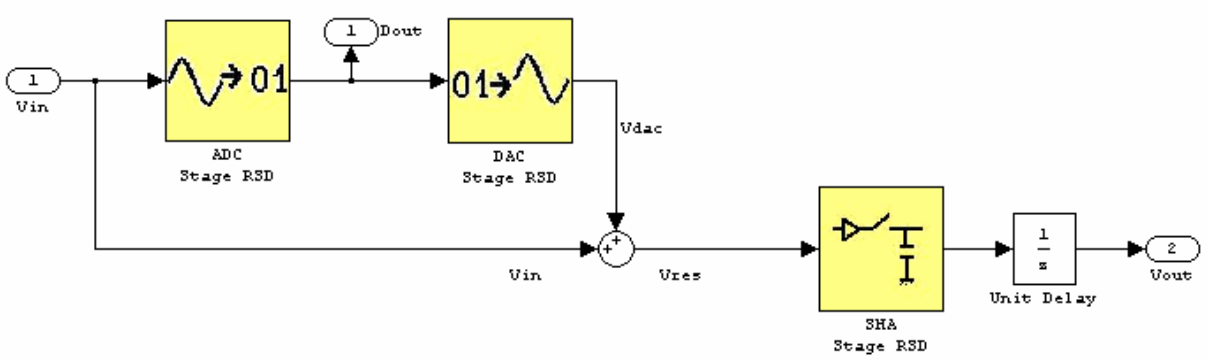


Fig.7.48. Diseño. Stage_complete. RSD.

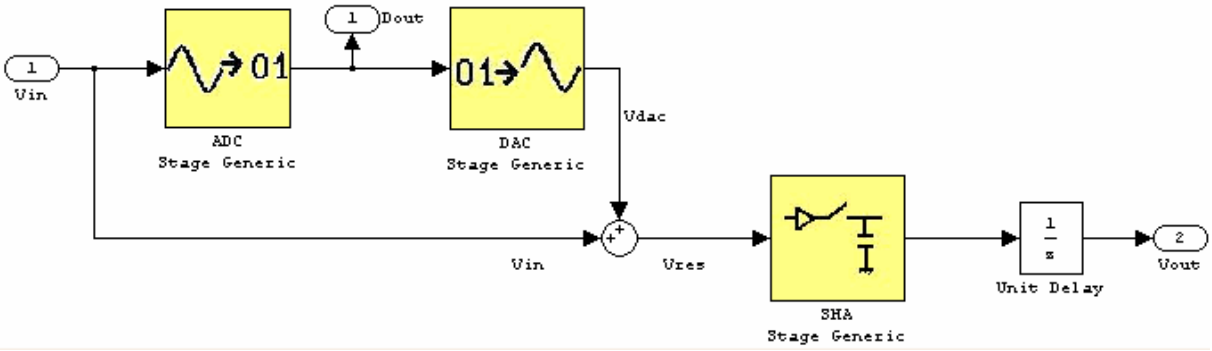


Fig.7.49. Diseño. Stage_complete. Generic.

El esquema en ambos es similar, tan sólo hay que cambiar bloques RSD por genéricos.

3.1.5. Stage last

Se ha desarrollado un modelo distinto para la última etapa, ya que en convertidores analógico-digitales pipeline, la última etapa de la cadena es un SDC genérico.

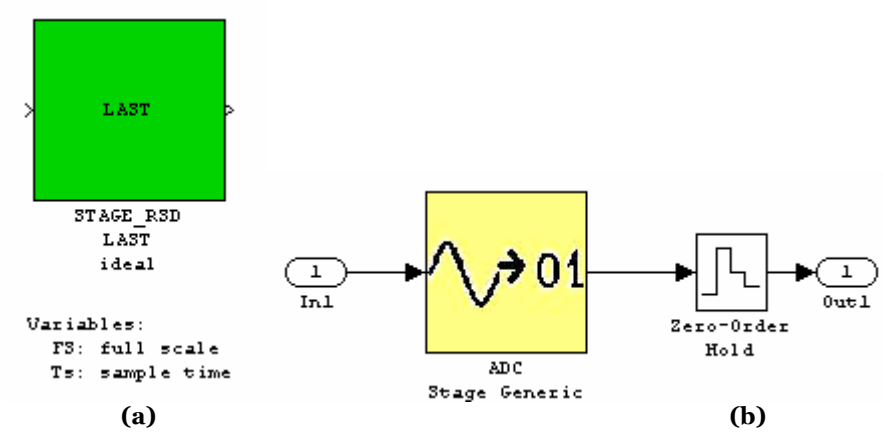


Fig.7.50. (a) Bloques ideales. Last_stage. (b) Diseño

3.1.6. Corrección digital

La corrección digital, posterior a la conversión, puede ser también para el caso de genérico o para RSD. Veamos por separado estos modelos.

3.1.6.1. Corrección digital RSD

Para no extendernos mostrando cada uno de los bloques realizados para la corrección digital RSD, mostraremos uno de ellos como ejemplo.

Se han realizado bloques de corrección de 3 a 15 bits. Cada uno de ellos tiene tan sólo como parámetro el tiempo de muestreo Ts. Estos bloques toman de entrada la salida de cada etapa RSD. La retrasan tantas veces como posiciones se encuentren en la cadena de pipeline para no haya retraso entre ellos y se multiplica por una ganancia. Esta ganancia es potencia de dos, 2^p , siendo p el número de retrasos.

Para el ejemplo de corrección digital de 12 bits, el modelo de diseño es el siguiente:

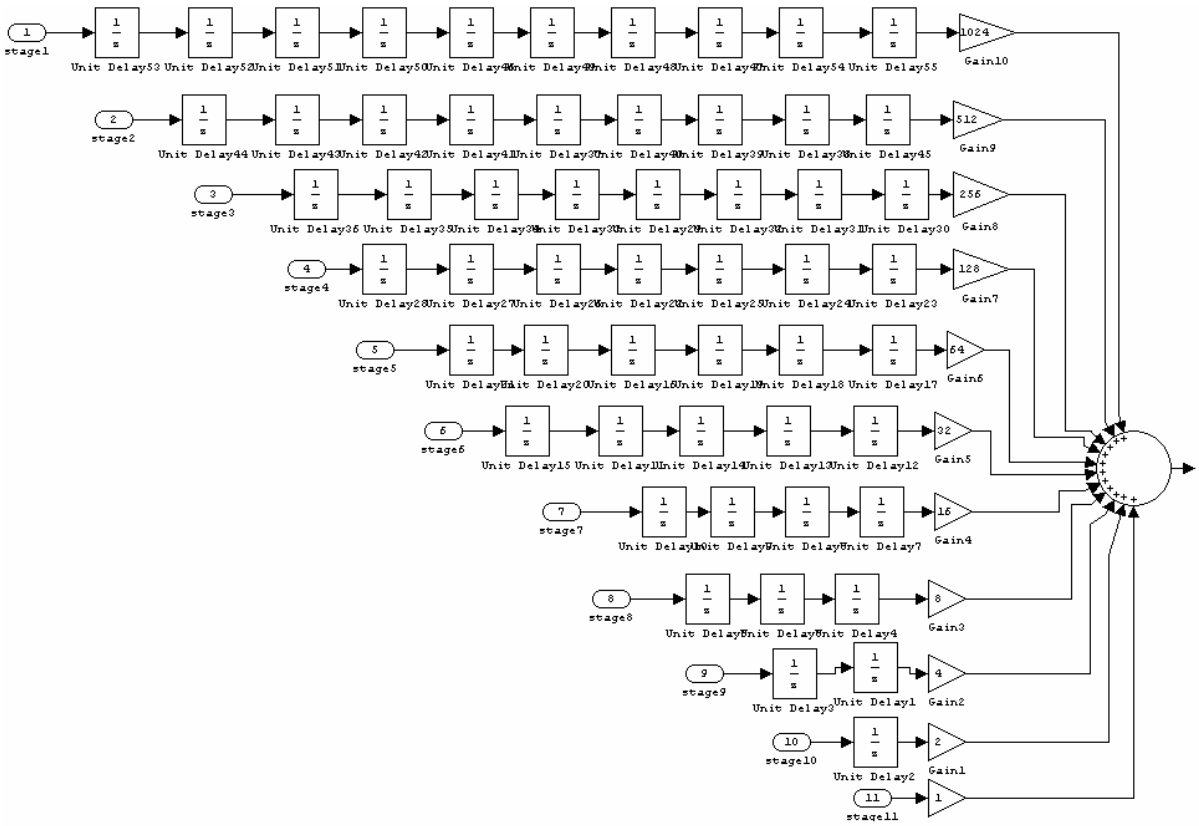


Fig.7.51. Bloques ideales. Corrección Digital 12 bits.

3.1.6.2. Corrección digital genérica

La forma de operación es igual a la anterior, pero en este caso la ganancia depende del número de códigos a la salida de cada etapa. Vamos a distinguir dos casos:

- todos los bloques tengan igual numero de códigos de salida
- los bloques tengan códigos distintos de salida

Para el primer caso, si tuviéramos 3 bloques con salida N códigos cada uno, las ganancias serían $2^{N-1}, 2^{N}, 2^{N+1}$. El número total de bits a la salida sería Nb·(N-1)+1, siendo N el número de códigos a la salida y Nb el número de bloques.

Para el segundo caso, suponiendo también 3 bloques con N_1, N_2 y N_3 códigos de salida, las ganancias serían $2^{N1-1}, 2^{N2-1}, 2^{N3-1}$. El número total de bits a la salida serán $\sum_{i=1}^{Nb} Ni - Nb + 1$, siendo Ni el número de códigos a la salida de la etapa i, y Nb el número de bloques.

Sabiendo esto, presentamos los siguientes bloques de corrección digital genérica.

Modelos para bloques con el mismo número de códigos de salida:

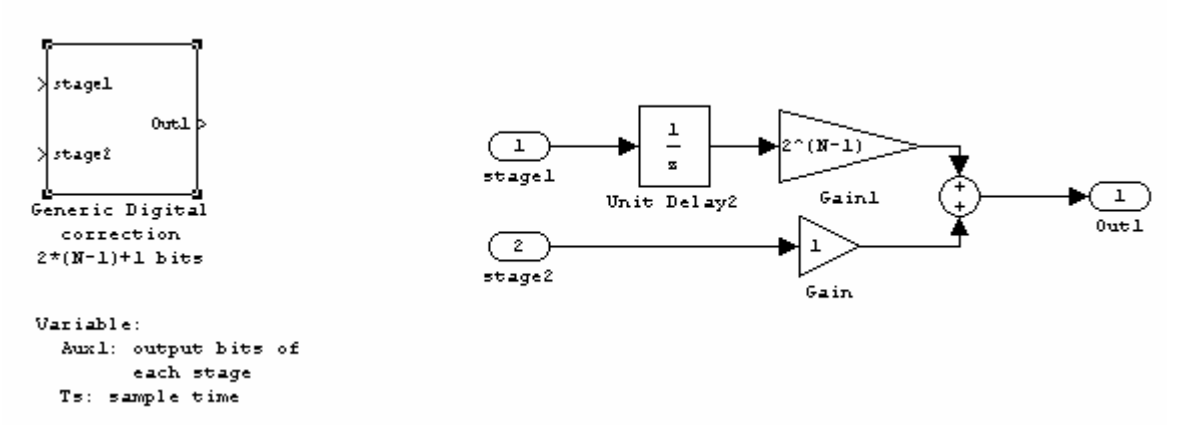


Fig.7.52. Bloques ideales. Corrección Digital 2 stage.

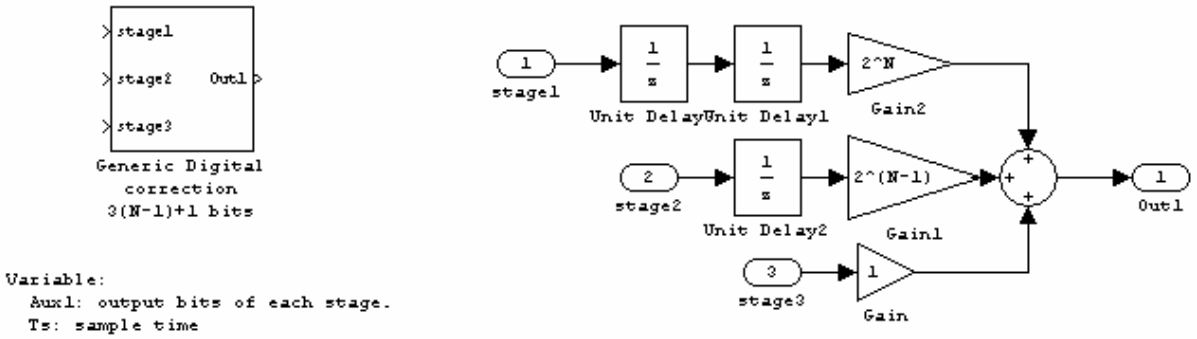


Fig.7.53. Bloques ideales. Corrección Digital 3 stage.

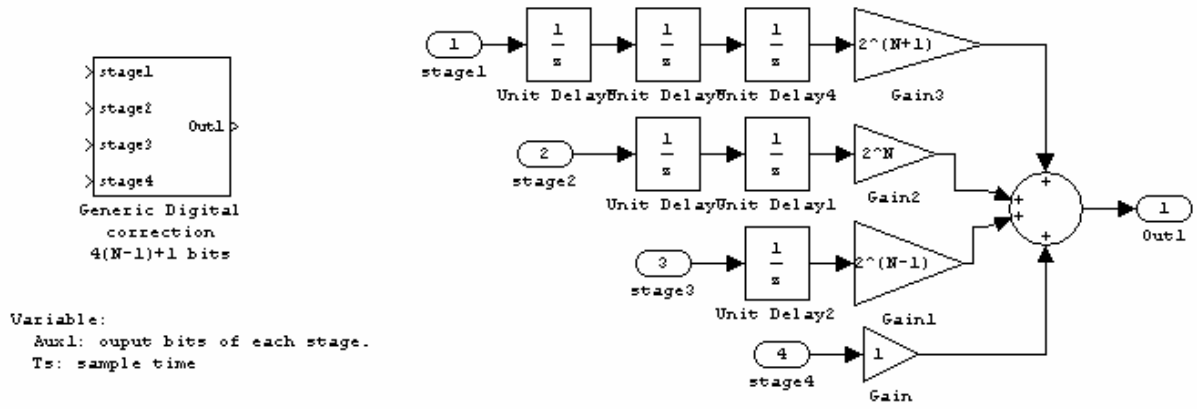


Fig.7.54. Bloques ideales. Corrección Digital 4 stage.

Y para el caso de que el número de código de salida de los bloques sea distinto:

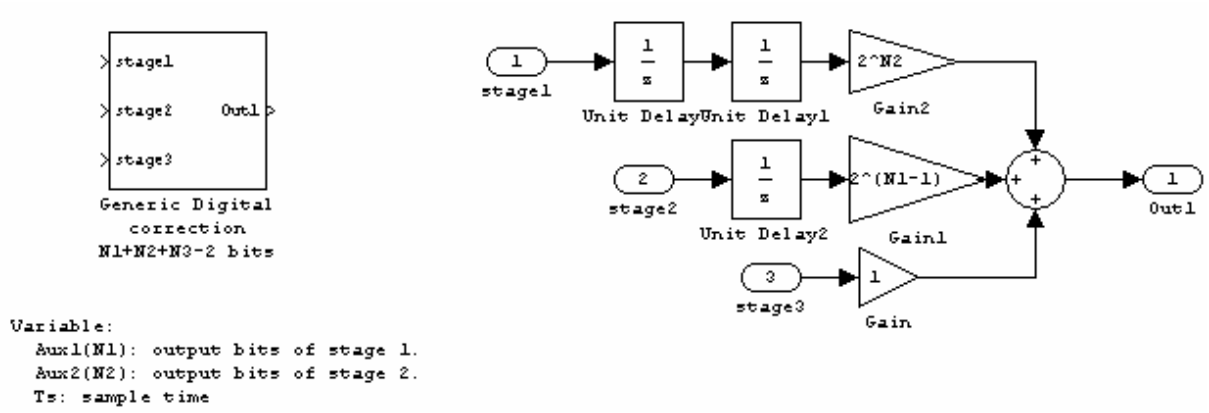


Fig.7.55. Bloques ideales. Corrección Digital 3 stage.

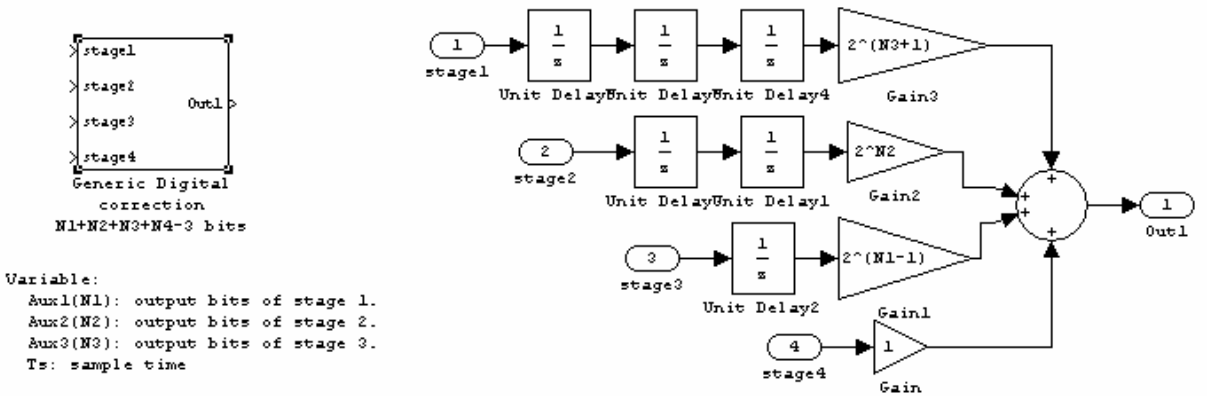


Fig.7.56. Bloques ideales. Corrección Digital 4 stage.

3.2. Bloques no ideales

Ya vimos en el capítulo 3 los diferentes tipos de no idealidades que podían a afectar a un convertidor analógico digital pipeline. Pues ahora, en este apartado mostraremos los modelos de diseño que hemos realizado para simularlas.

3.2.1. DAC Stage

El modelo diseñado para el caso del DAC lo mostramos en la Fig.7.57. Este muestra como no idealidad el offset el la tensión de referencia.

3.2.1.1. Offset en la tensión de referencia

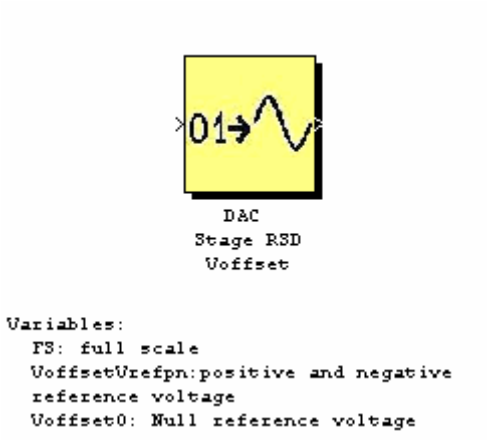


Fig.7.57. Bloques no ideales. DAC.

Este bloque presenta los siguientes parámetros:

Variable simulador	Comentario
VoffsetVrefpn	Offset en la referencia positiva y negativa
Voffset0	Offset en la referencia nula
FS	Fondo de escala

Tabla.7.9. Variables: Bloques no ideales. DAC.

El diseño:

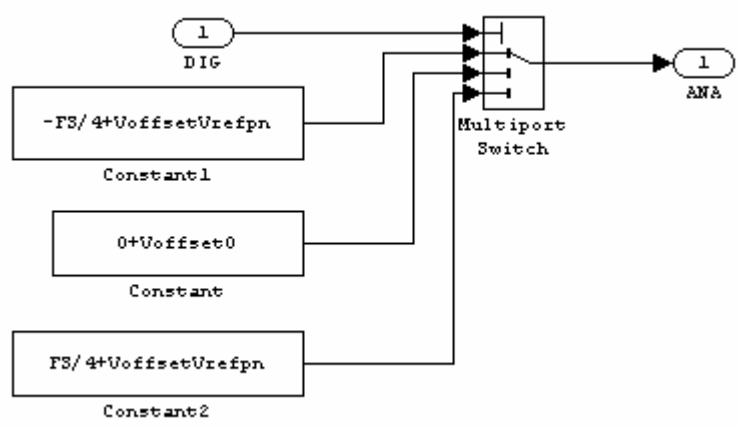


Fig.7.58. Diseño: Bloques no ideales. DAC.

3.2.2. Stage Complete

Veamos a continuación diferentes no idealidades para una etapa concreta del convertidor pipeline.

3.2.2.1. Stage RSD. Ganacia finita.

El modelo para este caso:

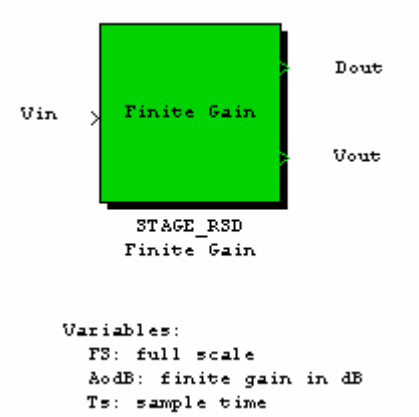


Fig.7.59. Bloques no ideales. Stage. Gain Finite.

El diseño lo mostramos en la siguiente gráfica:

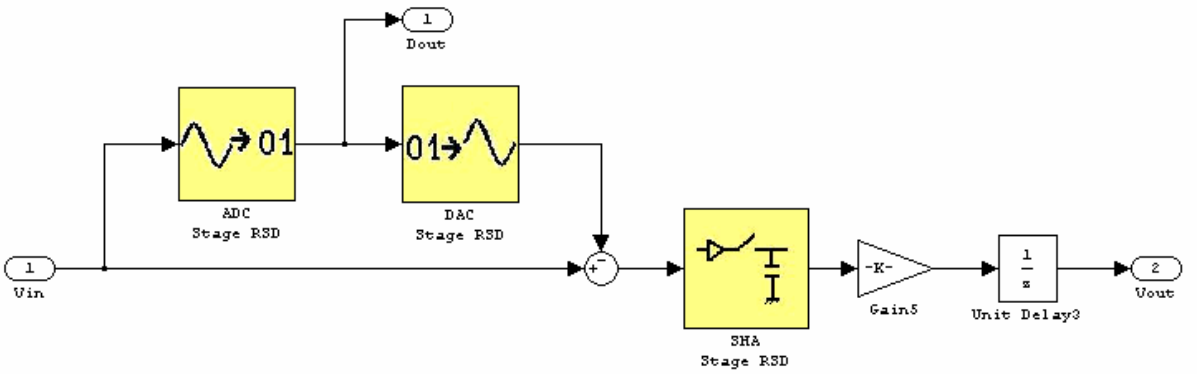


Fig.7.60. Diseño: Bloques no ideales. Stage. Gain Finite.

El valor de la ganancia finita viene dado por la constante K en el modelo, que tiene el valor: 1/(1+(2/(10^(AodB/20)))).

Para el caso genérico bastaría con sustituir los bloques anteriores por bloques genéricos, tal y como mostramos en el apartado 3.1.4.

3.2.2.2. Stage RSD. Mismatch.

El bloque que representa la no idealidad de mismatch[13] lo representamos en la figura siguiente:

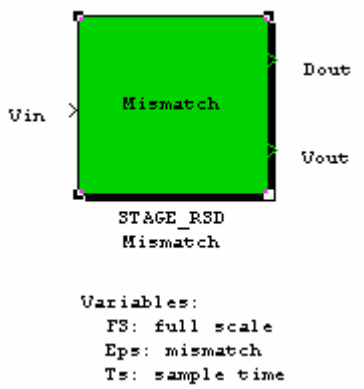


Fig.7.61. Bloques no ideales. Stage. Mismatch.

Las variables que utiliza el modelo las podemos observar en la máscara[14]:

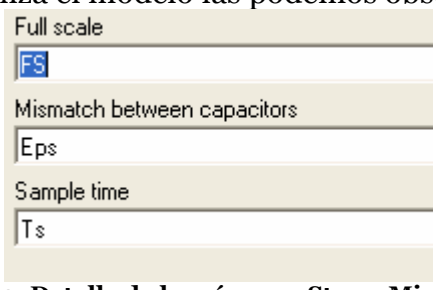


Fig.7.62. Detalle de la máscara. Stage. Mismatch.

El diseño para este bloque:

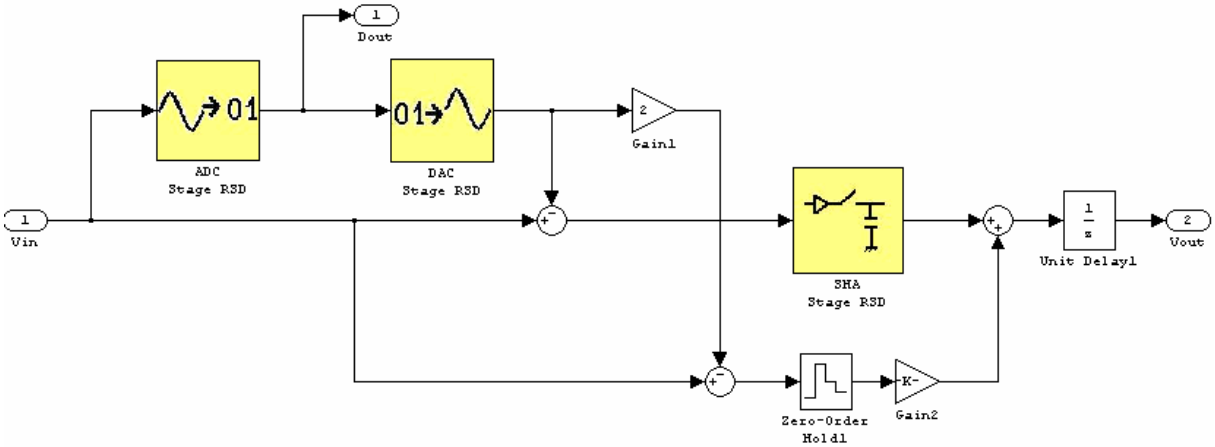


Fig.7.63. Diseño: Bloque no ideal. Stage. Mismatch.

[13] Para el caso genérico sería igual cambiando los bloques del diseño RSD por genéricos.
[14] Es bueno recordar, que para cada modelo se ha diseñado una máscara que contiene una descripción del mismo y las variables utilizadas.

3.2.2.3. Stage RSD. SR y GBW.

En este caso, el bloque diseñado:

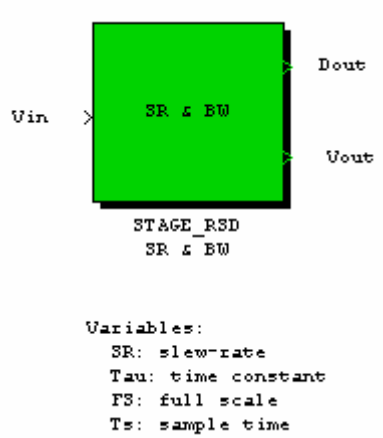


Fig.7.64. Bloque no ideal. Stage. SR y GBW.

Las variables utilizadas las podemos ver debajo del bloque: SR, Tau, FS y Ts. El diseño realizado:

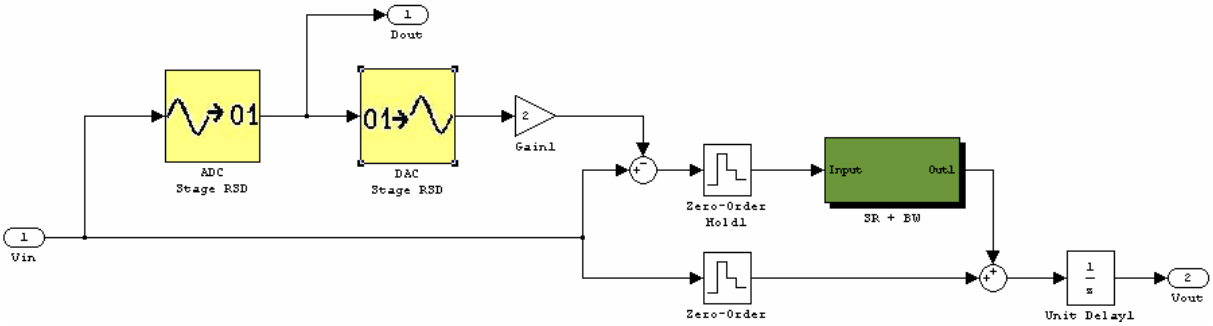


Fig.7.65. Bloque no ideal. Stage. SR y GBW.

Donde el bloque SR + BW tiene como diseño:

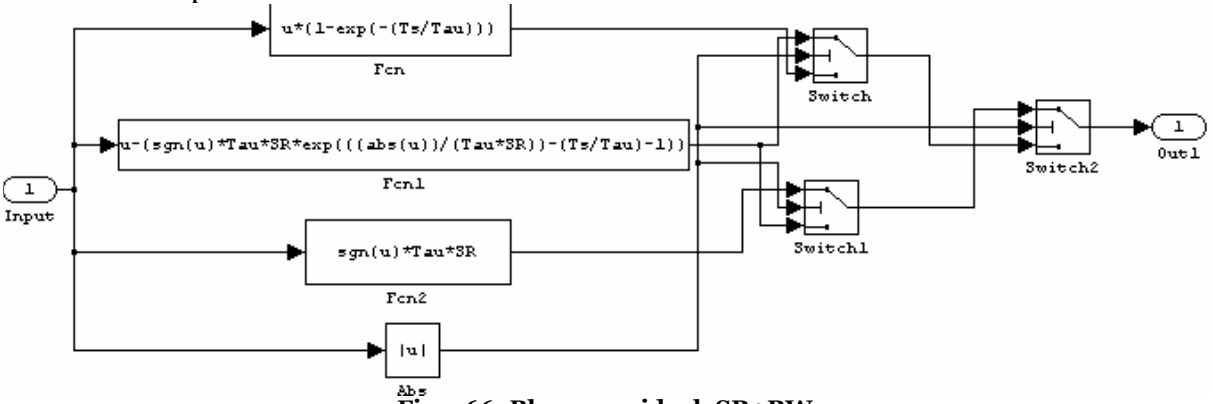


Fig.7.66. Bloque no ideal. SR+BW.

3.2.2.4. Stage RSD. Offset.

En este caso el modelo:

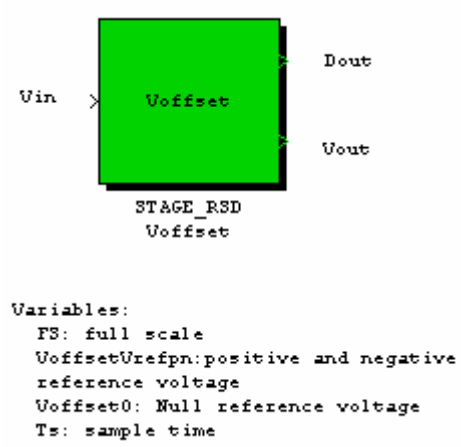


Fig.7.67. Bloque no ideal. Offset.

El diseño consiste en añadir al caso ideal el DAC con características no ideales de offset. Nos queda la siguiente composición:

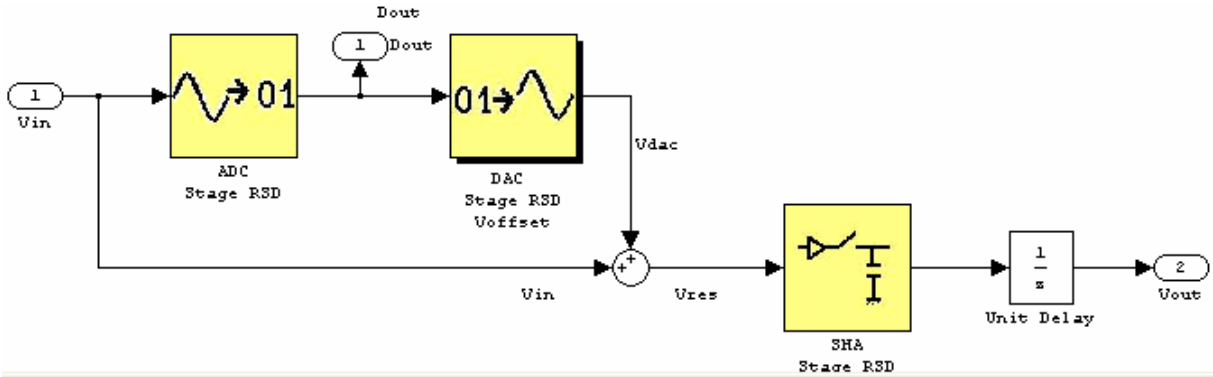


Fig.7.68. Diseño: Bloque no ideal. Offset.

3.2.2.5. Stage RSD. Jitter noise.

En este bloque lo que hacemos es añadir al bloque ideal el bloque de ruido jitter a la entrada. Este bloque deberá ir al comienzo.

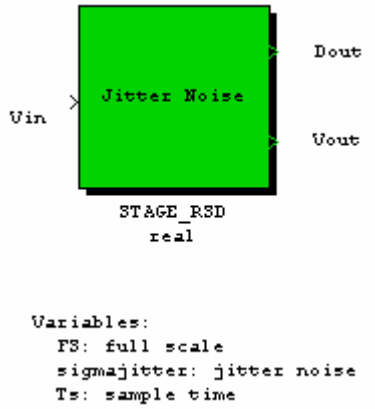


Fig.7.69. Bloque no ideal. Stage. Jitter.

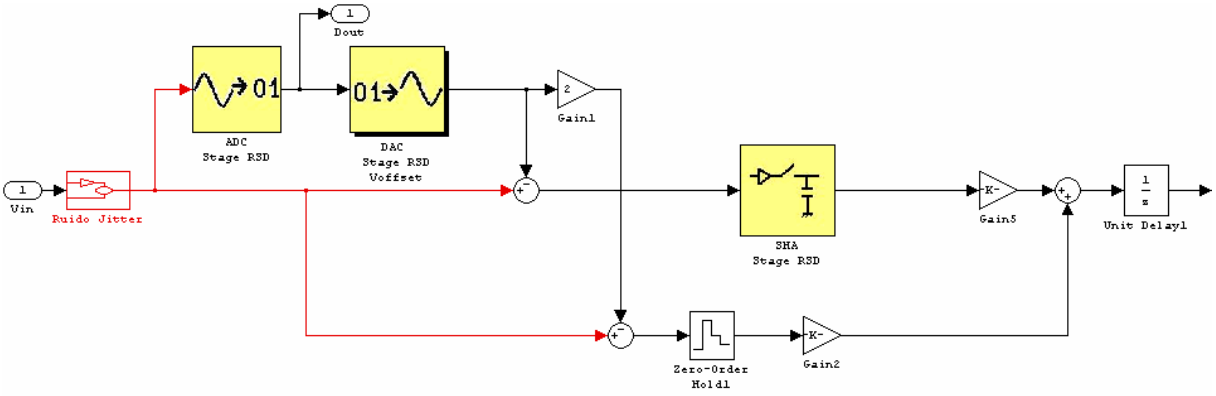


Fig.7.70. Diseño: Bloque no ideal. Stage. Jitter.

El bloque del Jitter lo vemos en el apartado 3.2.3.

3.2.3. Jitter noise

El modelo del Jitter, visto ya en otro tipo de convertidores es el siguiente:

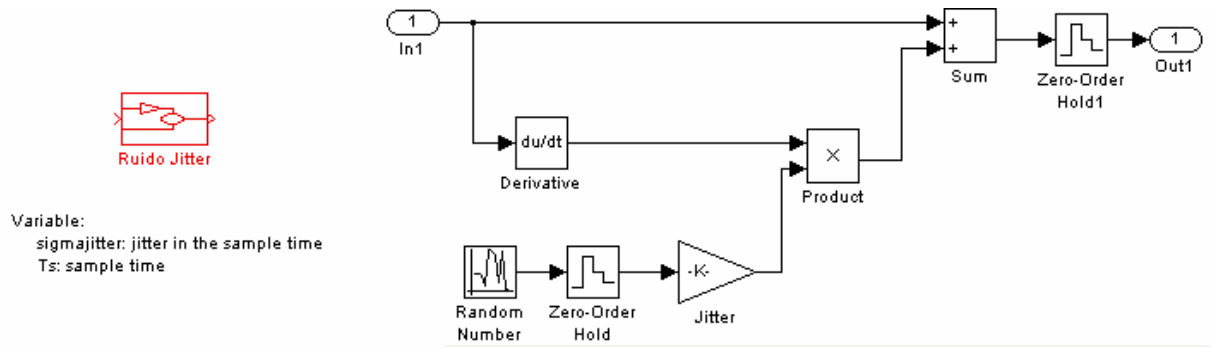


Fig.7.71. Diseño: Bloque no ideal. Jitter.

A modo de resumen, presentamos todos los modelos de bloques no ideales desarrollados en el apartado 3.2.

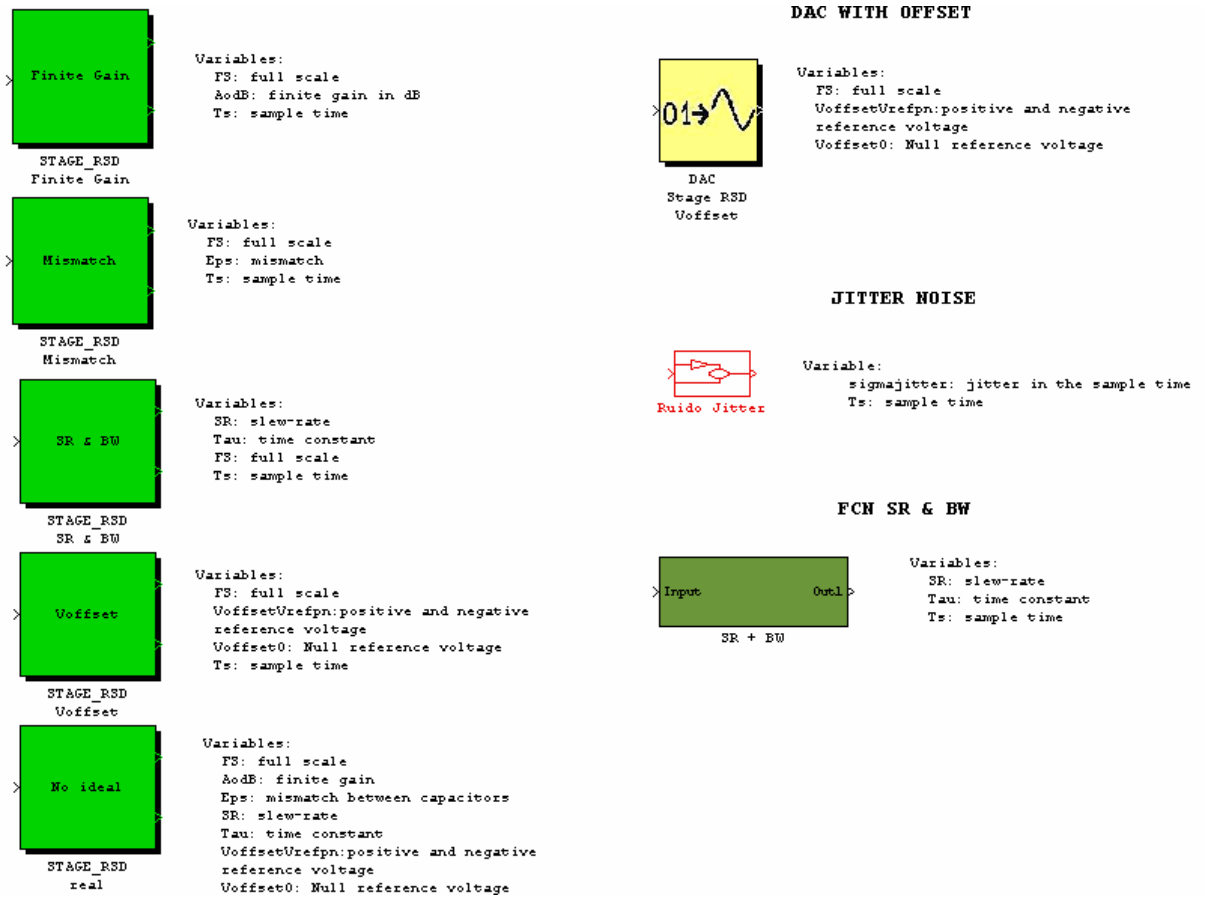


Fig.7.72. Bloques no ideales.

3.3. Ejemplos

En este apartado nos centramos en presentar los ejemplos desarrollados. Cada uno de los bloques de los que están compuestos ya los hemos descrito en apartados anteriores. Mostraremos el esquema general para cada uno de ellos.

En el capítulo 8 mostraremos la simulación de estos ejemplos, de forma que este apartado nos limitamos a mostrarlos estructuralmente.

Para tener una visión global de todos los ejemplos desarrollados para el

pipeline, les mostramos a continuación la agrupación de todos ellos:

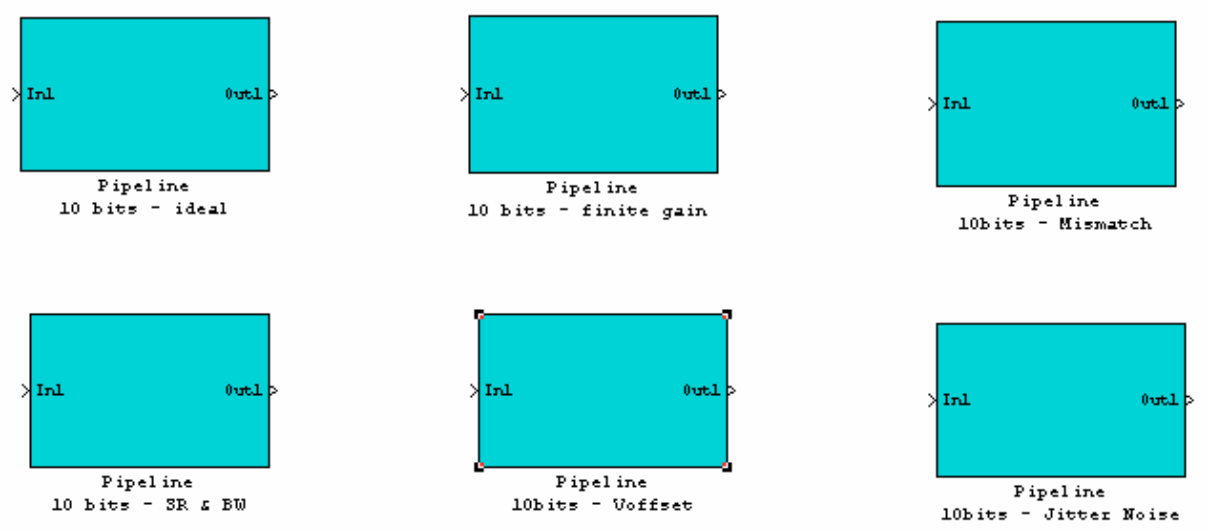


Fig.7.72. Ejemplos pipline.

Todos los ejemplos siguen la estructura que mostramos en la Fig.7.74. Lo que varía es el primer bloque que dependiendo de la no idealidad que queramos simular, tendremos el bloque correspondiente a eso no idealidad. En la Fig. 7.74 presentamos como ejemplo el caso del mismatch. El resto de ejemplo aunque están montados no lo mostramos por nos extendernos, en caso de querer consultarlos pueden acceder al CD de instalación del programa.

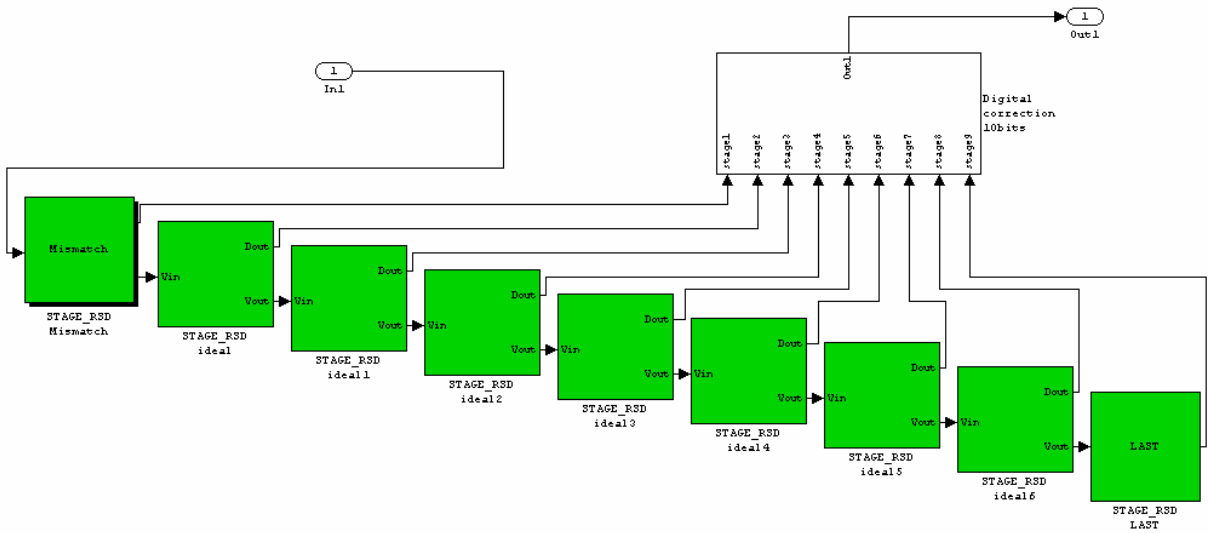
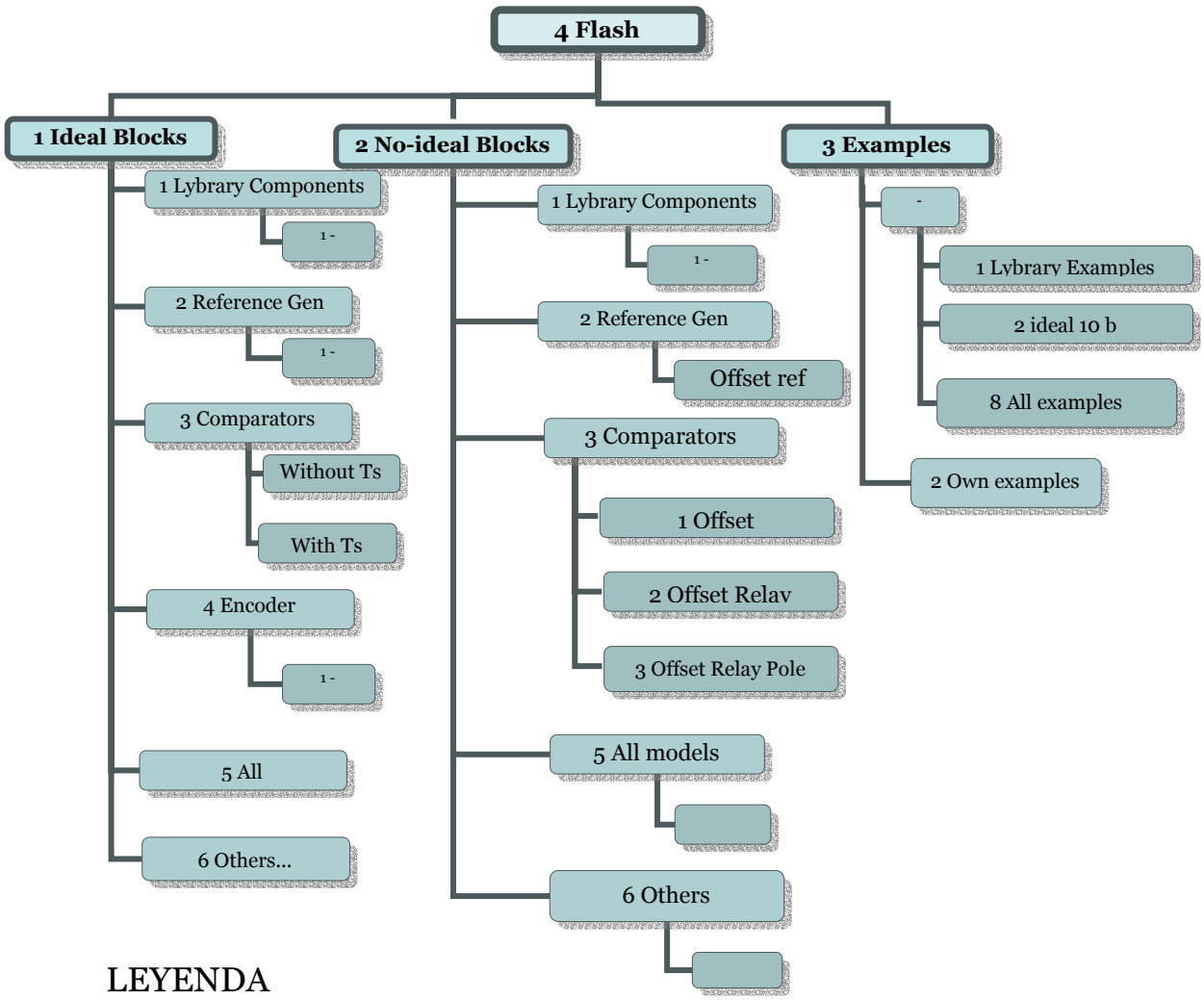


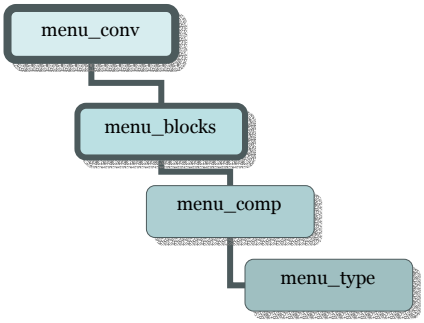
Fig.7.74. Ejemplos pipeline. Mismatch.

4. Librería del Flash

Para tener una visión completa de la librería, muestro a continuación el siguiente esquema en forma de árbol que incluye a grande rasgos los modelos diseñados para convertidores flash:



LEYENDA



4.1. Bloques ideales

Tras el estudio realizado de convertidores Flash, llegamos a la conclusión que necesitamos incluir bloques como comparadores, generadores de tensión de referencia para la entrada a los comparadores, encoger...

Una visión global de los bloques generados es la siguiente:

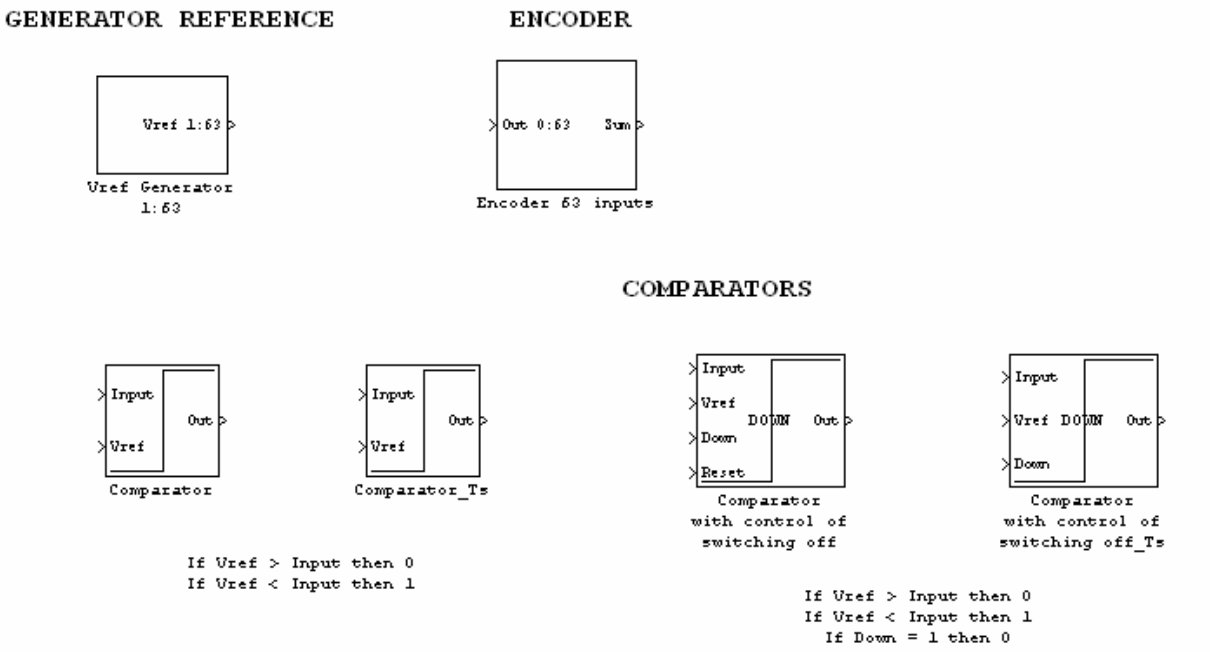


Fig.7.75. Bloques ideales Flash.

4.1.1. Generador de referencia

Este bloque viene modelado por:

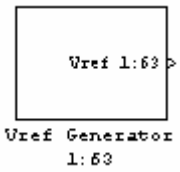


Fig.7.76. Bloques ideales Flash. Gen_ref.

Este bloque dará la entrada de la tensión de referencia a un banco de 63 comparadores. Como estamos en el caso ideal, el valor de la variable offset es nulo. Una visión parcial de este bloque lo vemos en la siguiente figura:

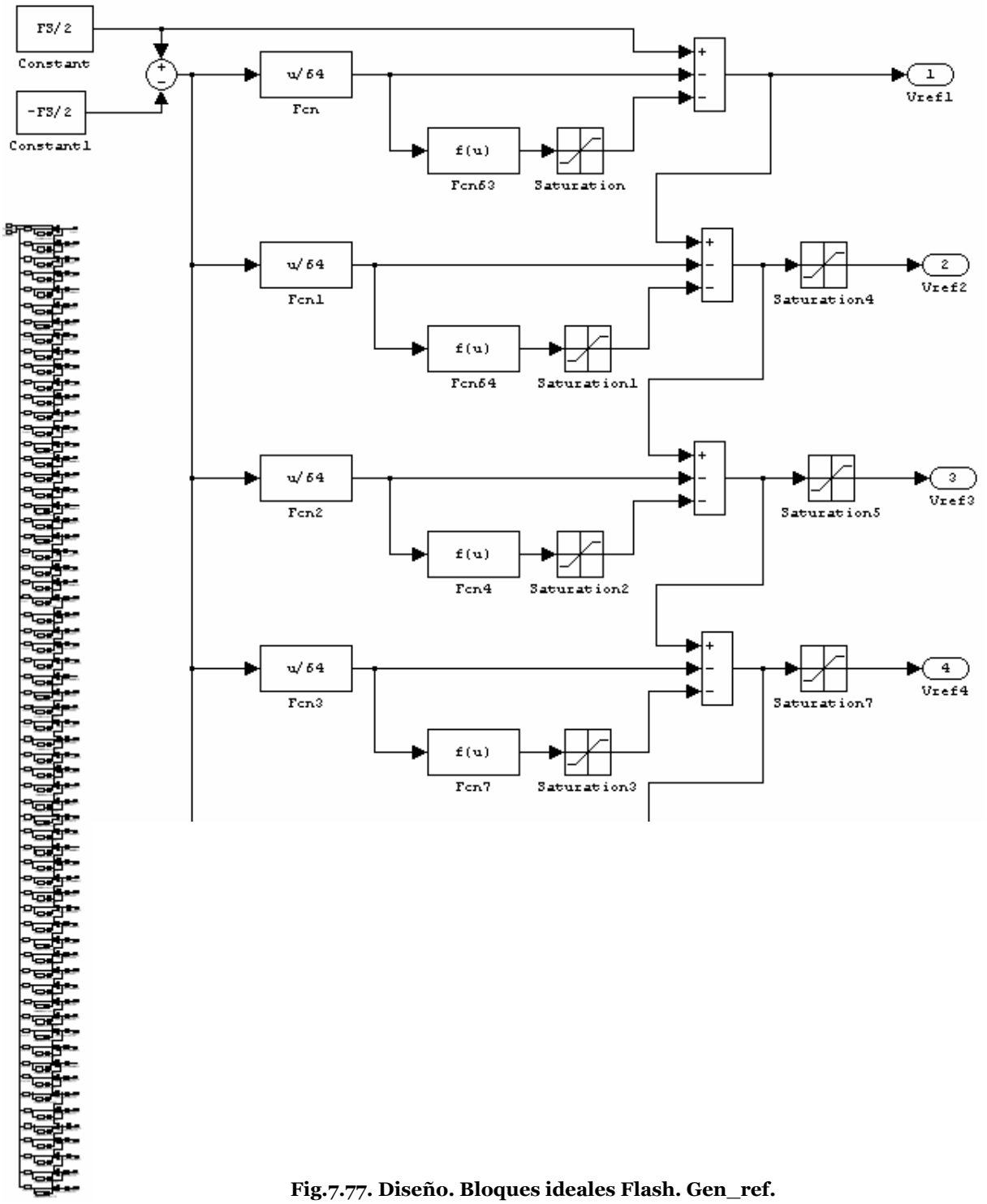


Fig.7.77. Diseño. Bloques ideales Flash. Gen_ref.

4.1.2. Comparadores

En este apartado incluido varios tipos de comparadores. Los presentamos a continuación:

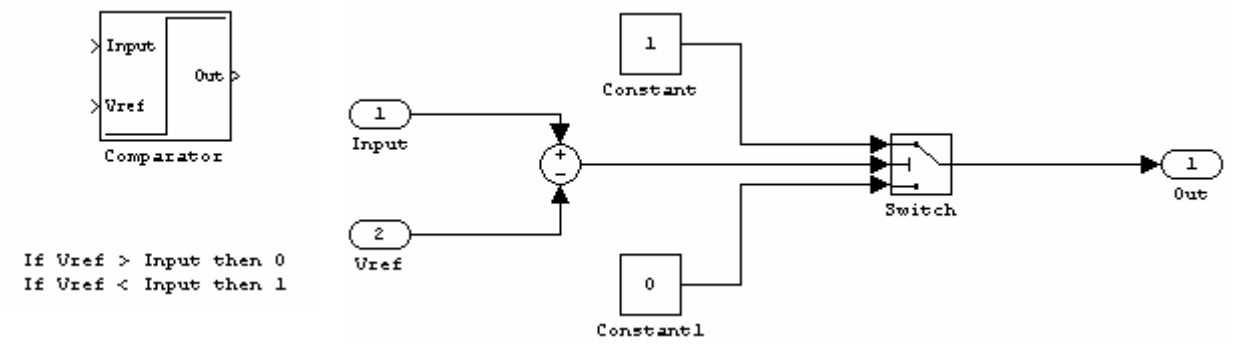


Fig.7.78. Diseño. Bloques ideales Flash. Comp.

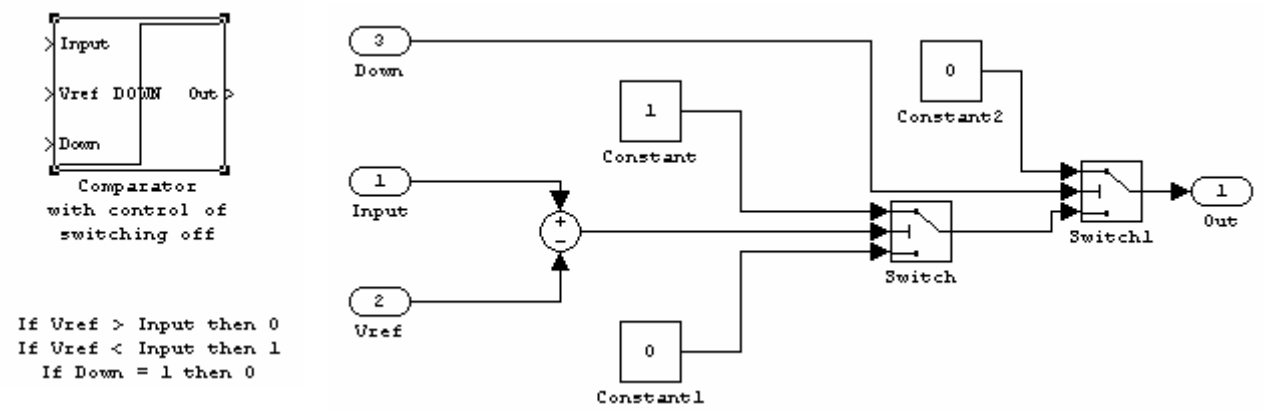


Fig.7.79. Diseño. Bloques ideales Flash. Comp_down.

La diferencia entre el comparador de la Fig.7.78 y el de la Fig.7.79, es que en el segundo caso se incluye al comparador una señal de power down para apagar el comparador en caso de que no queramos que funcione. Esto puede servirnos de utilidad en métodos de calibración.

Los otros modelos que presentamos, son análogos a éstos con la salvedad que la señal de entrada incluye un sample and hold. Estos dos nuesvos modelos de comparadores los incluimos en las Fig. 7.80 y 7.81.

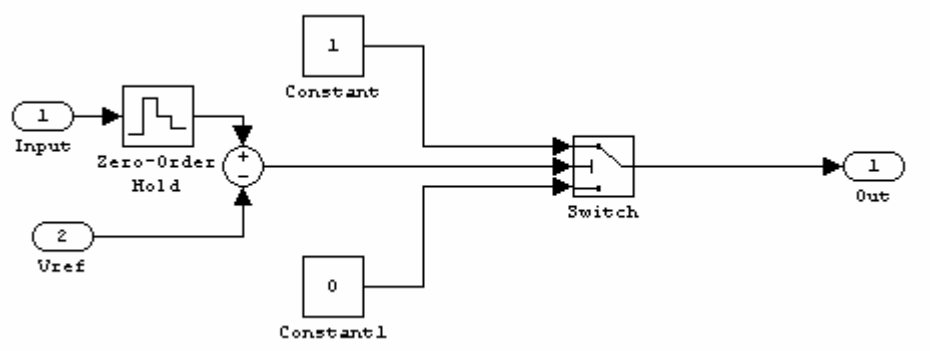


Fig.7.80. Diseño. Bloques ideales Flash. Comp con Ts.

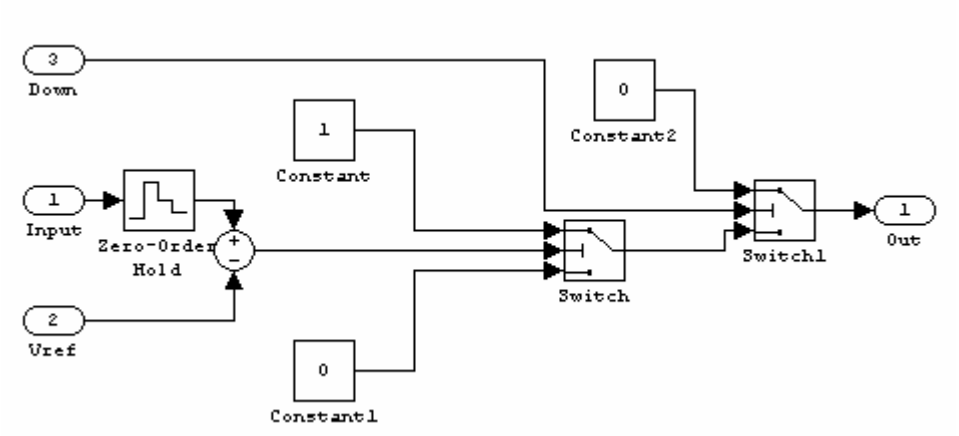


Fig.7.81. Diseño. Bloques ideales Flash. Comp_down con Ts.

4.1.3. Encoder

Se encarga de tomar todas las salidas del banco de comparadores y sumarlas. El modelo con el que realizaremos esta función:

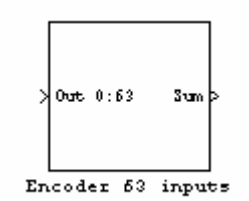


Fig.7.81. Bloques ideales Flash. Encoder.

Veamos a continuación el diseño para en caso de un encoger que tome 63 entradas.

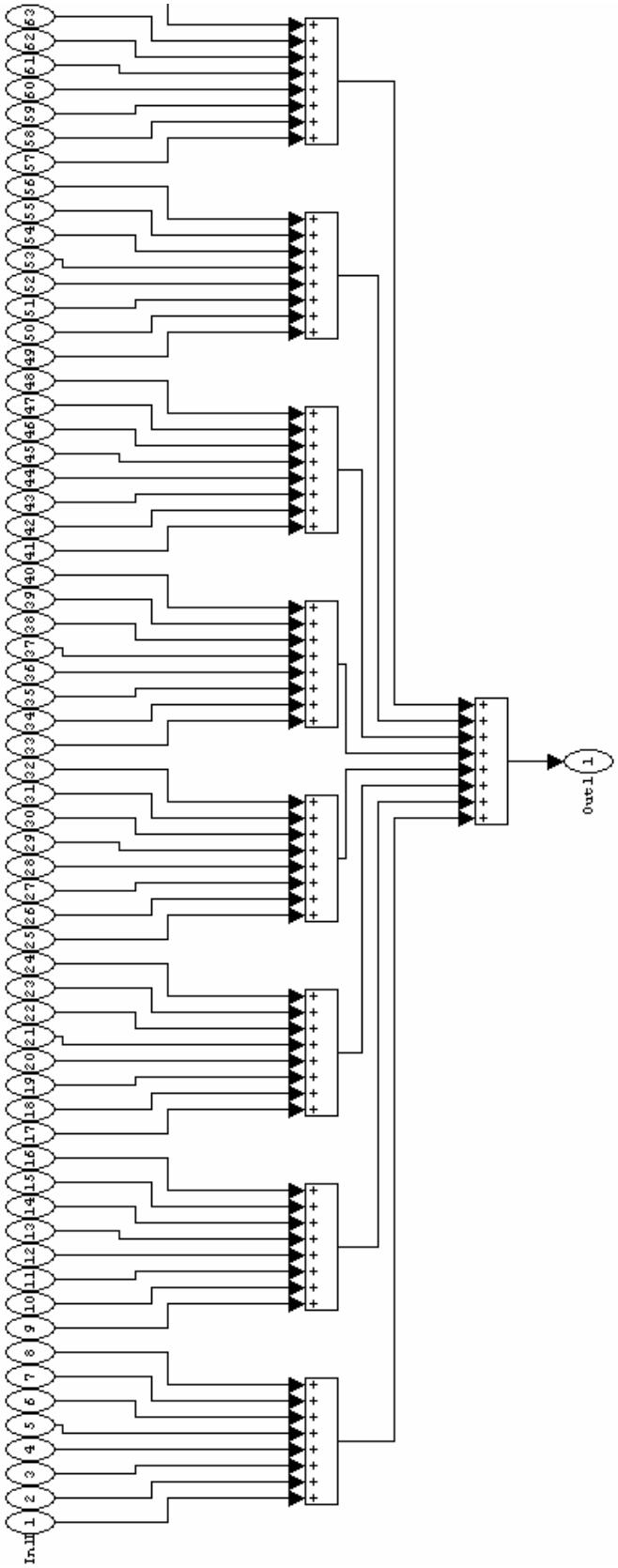


Fig.7.82. Diseño. Bloques ideales Flash. Encoder.

4.2. Bloques no ideales

En este apartado mostraremos los diferentes diseños de bloques para convertidores analógico-digitales flash con no idealidades. Presentaremos el bloque y el diagrama de bloques que implementa su funcionalidad.

4.2.1. Generador de referencia

Este modelo, representado en la Fig.7.83. tiene el mismo diseño que el comentado en el apartado 4.1.1. La única diferencia es que debemos dar un valor al offset, lo que modelaría la tolerancia a la que está sometida la cadena de resistencias en una estructura flash .

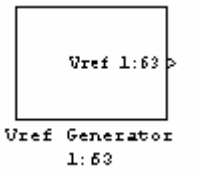


Fig.7.83. Bloque no ideal Flash. Gen_ref_offset.

4.2.1. Comparadores

En este caso vamos a presenta tres modelos: uno con offset, otro con offset y relay, y por último, uno con offset, relay y efecto del polo. Cada uno de estos modelos los representamos en las Fig. 7.84, 7.85 y 7.86 respectivamente.

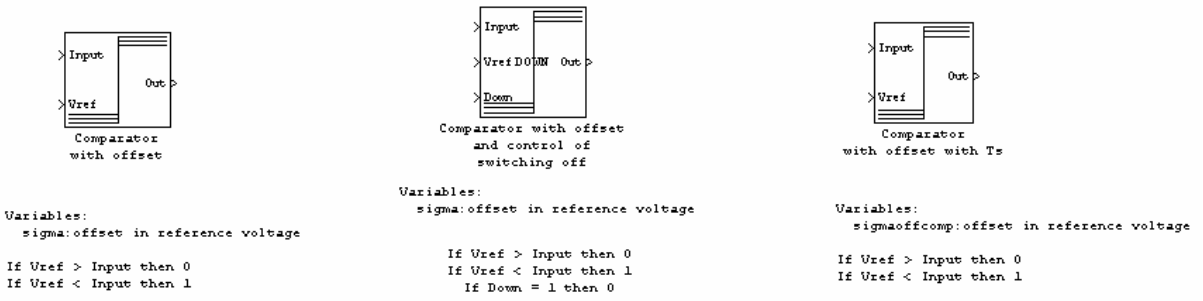


Fig.7.84. Bloques no ideales Flash. Com_offset.

Su diseño ya fue presentado en el apartado 4.1.2.

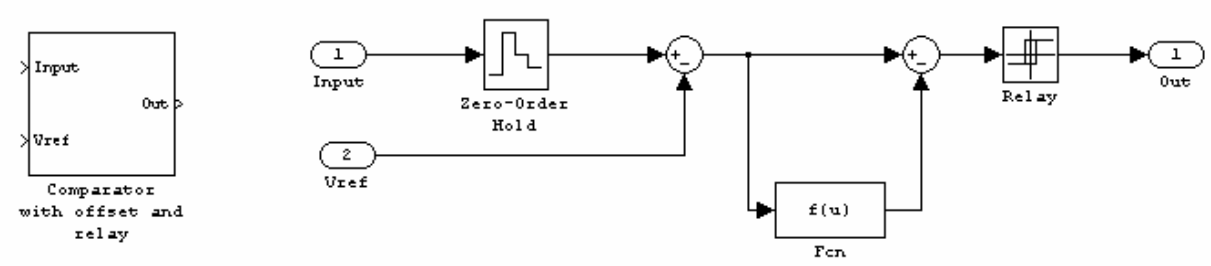


Fig.7.85. Bloque no ideales Flash. Com_offset_relay.

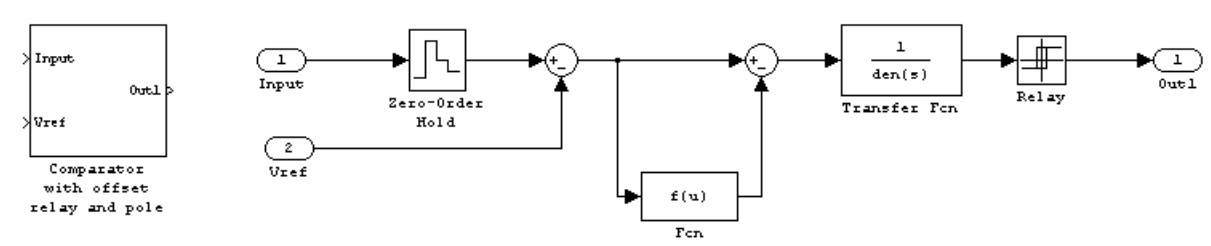


Fig.7.86. Bloque no ideales Flash. Com_offset_relay_polo.

4.3. Ejemplos

En este apartado vamos a presentar un ejemplo completo de un flash.

En este modelo se representa un convertidor flash de 6 bits, considerándose las siguientes no idealidades: offset en las tensiones de referencia, offset en los comparadores, histéresis en los comparadores. Contiene los bloques GenVref6bits, Comparator Ts Offset Relay Bank 6bits, DigToThermo, Binary Logic Converter, y ConvBitstoNumber. Estos bloques se han diseñado en exclusiva para este ejemplo.

El diagrama de bloques diseñado se muestra en la siguiente figura:

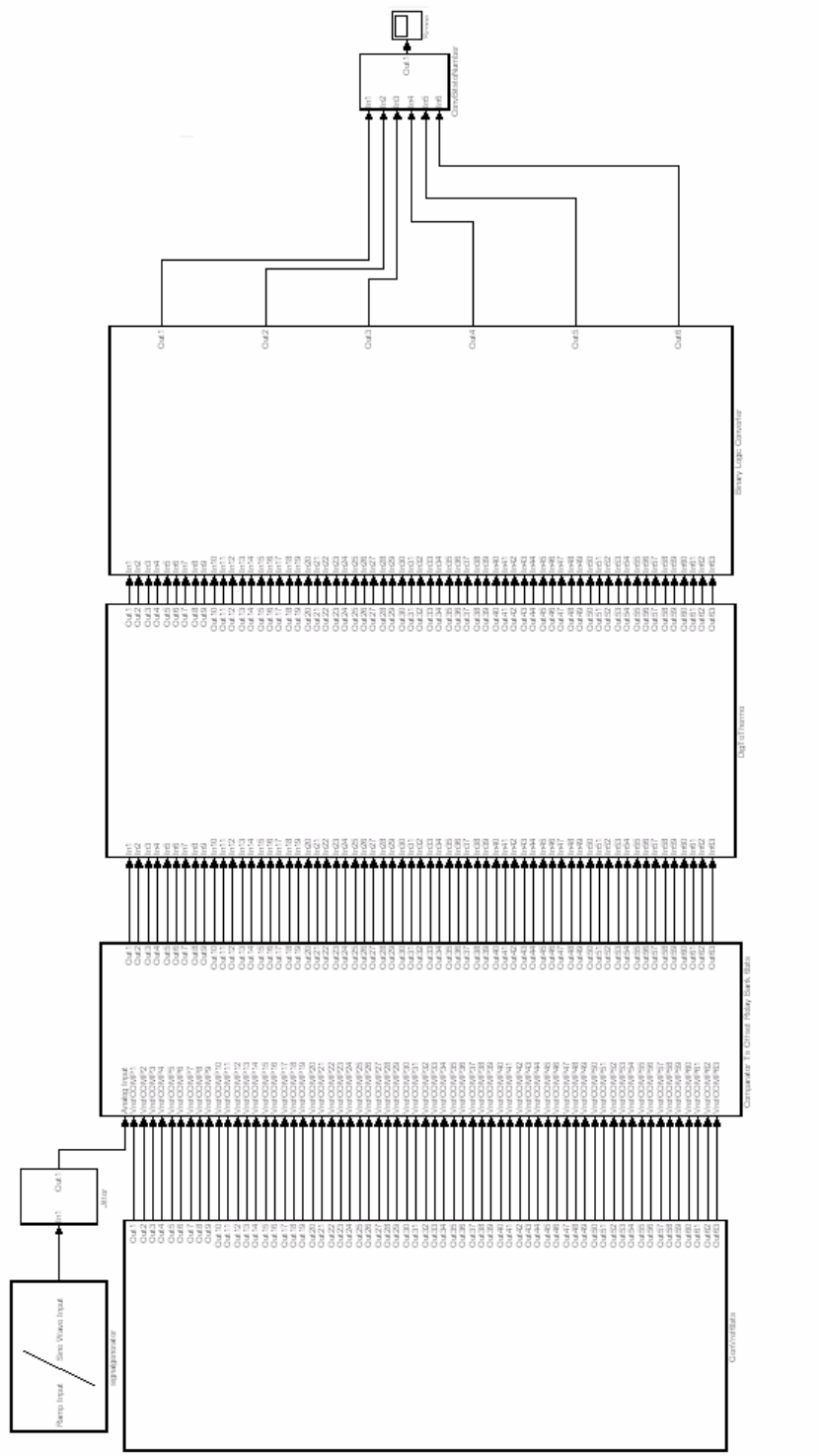


Fig.7.87. Ejemplo Flash de 6 bits.

CAPÍTULO 8

Simulaciones

Índice:

1. Motivaciones

En este capítulo vamos a mostrar la versatilidad del programa diseñado simconverter así como la utilidad de la librería. Para ello vamos a simular un ejemplo para cada tipo de convertidor. Utilizaremos modelos concretos y veremos para cada caso un estudio detallado como el cálculo del SNDR, estudio del efecto de las no idealidades, espectro de la señal...

Tras finalizar este capítulo tendremos más claro la potencia de esta herramienta de simulación así como la utilidad en los desarrollos de los modelos que podemos dar a la misma.

2. ADC Sigma Delta

En este apartado nos vamos a centrar en la simulación de un convertidor analógico digital sigma delta de segundo orden, paso de baja y con una estructura simple.

Realizaremos dos estudios. Primeramente un estudio ideal para ver las características obtenidas dadas unas especificaciones, y posteriormente, un estudio del efecto de las no idealidades, cuantificando los valores máximos y mínimos de las variables del modelo para que el estudio de las características con iguales especificaciones, permanezca dentro del rango respecto al caso ideal.

2.1. Estudio ideal

El modelo ideal en la libraría nos lo encontramos en:

1. CONVERTERS: TD Sigma Delta
2. BLOCKS: Examples
3. COMPONENTS: -
4. TYPES: TD Sigma Delta_LP_2order_ideal_simple

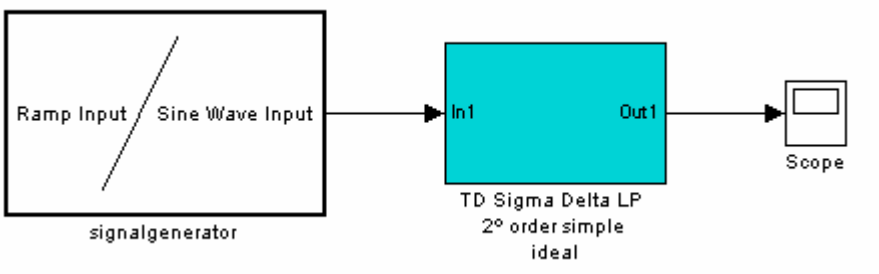


Fig. 8.1. Ejemplo: Sigma Delta LP 2orden ideal simple.

El esquema de nuestro modelo es el siguiente:

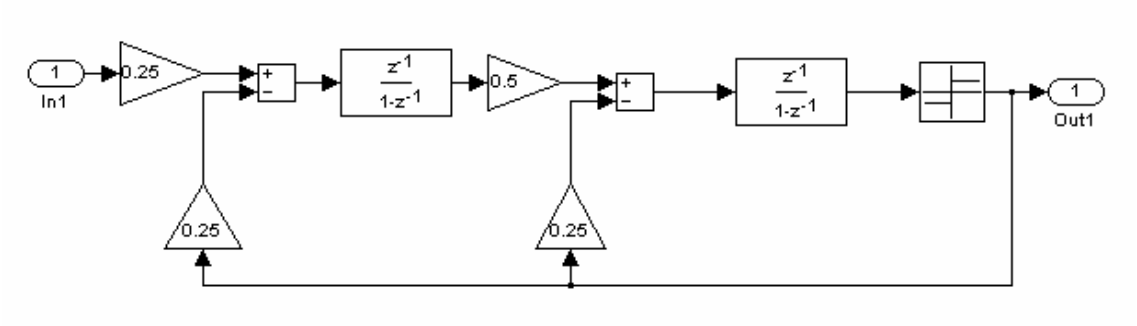


Fig. 8.2. Esquema del convertidor Sigma Delta ideal.

Todos los elementos presentados son ideales. Los valores de los coeficientes son tomados para una aplicación médica, concretamente para el estudio de un electrocardiograma.

En todos nuestros ejemplos se cargan automáticamente los valores de entrada en la zona de entradas del simulador simconverter. Para este caso en concreto, tomaremos los siguientes valores:

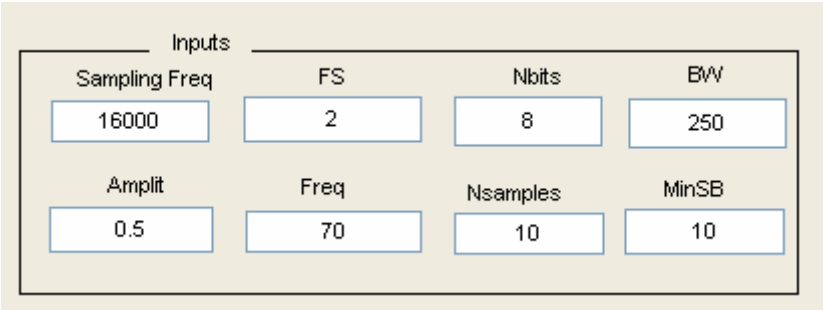


Fig. x.3. Valores de entrada.

Como estamos en el caso ideal, todos las variables de la zona de no idealidades las cargaremos con valor cero.

Introducimos los siguientes datos para la simulación:

- Tipo: Simulación simple[15]
- Entrada: Senoidal[16]
- Effort: 3.

[15] Podemos realizar simulaciones simple, dados unos valores únicos de entrada obtener la salida, o simulaciones con barridos de una o dos variables, tomar un rango de valores para las entradas y obtener para cada una el valor a la salida.
[16] Recordar que existen dos tipos de entradas, rampa o senoidal.

El aspecto completo de la interfaz tras la simulación sería el siguiente:

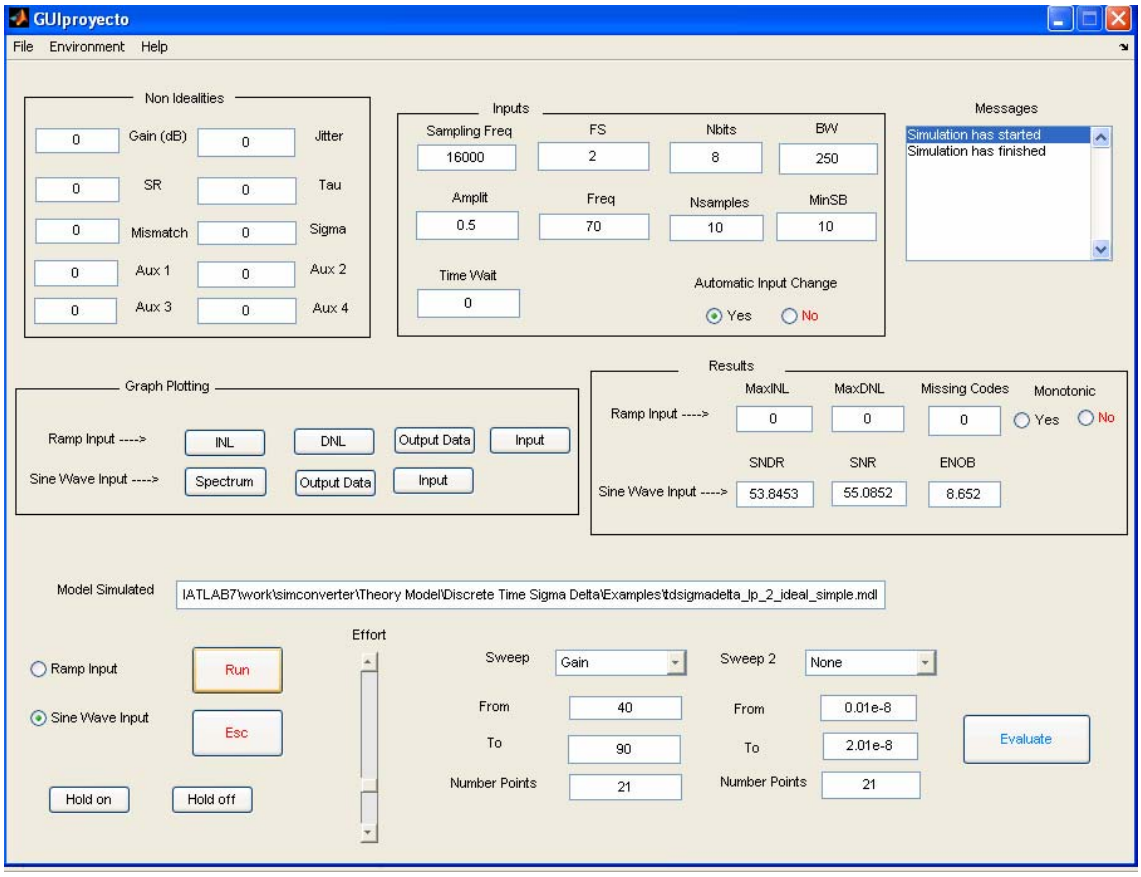


Fig. 8.4. Interfaz simulación ideal Sigma Delta.

Los valores de SNDR y SNR obtenidos son para el caso ideal:

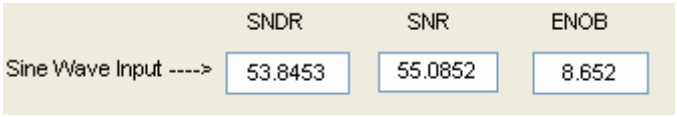


Fig. 8.5. Valores simulación ideal Sigma Delta.

Este valor de SNDR, relación señal a ruido y distorsión, lo vamos a tomar de referencia para el estudio de las no idealidades en el segundo caso, de forma que consideramos que a nuestra señal no caiga más de 2 dB, es decir, que tras añadir todas las no idealidades se mantenga entre 51-52 dB de SNDR.

Y el espectro de salida lo muestro en la siguiente figura 8.6:

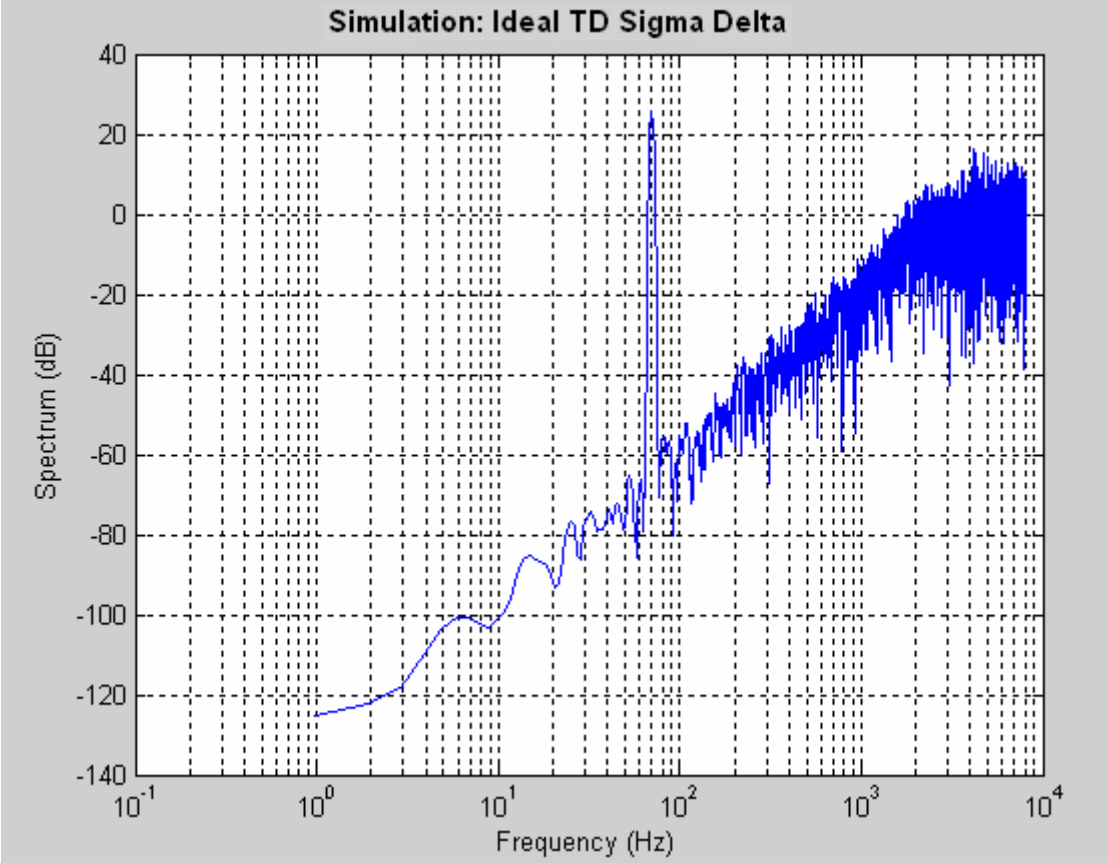


Fig. 8.6. Valores simulación ideal Sigma Delta.

Observamos como la potencia de la señal está centrada en la frecuencia de 70 Hz (frecuencia del seno de entrada) obteniendo en este valor su máximo de potencia.

2.2. Estudio no ideal

En este apartado vamos hacer un estudio exhaustivo del efecto de las no idealidades en el modelo. Para ello vamos a estudiar una a una cada no idealidad de forma que el SNDR tras añadir todas las no idealidades al modelo se mantenga entre 51-52 dB, caiga entre 1 y 2 dB se su valor ideal.

Vamos a presentar cada modelo, especificando la no idealidad en estudio y los resultados obtenidos tras su simulación. Haremos barridos de las variables en estudio, para concretar los valores límites de forma que se cumplan las especificaciones mencionadas.

2.2.1. Ruido del amplificador operacional

El modelo de la librería que simularemos para este caso esta formado por el modelo ideal sustituyendo el integrador por el modelo del integrador no ideal que incluye el ruido térmico y el ruido intrínseco de los amplificadores operacionales.

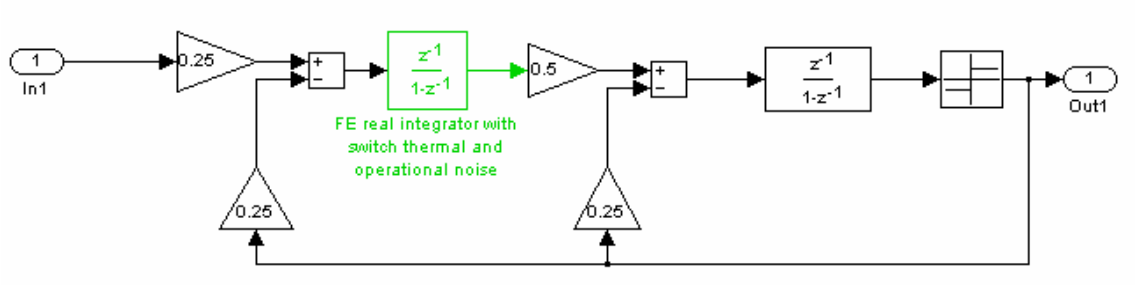


Fig. 8.7. Esquema simulación no-ideal. Ruido op y ktc.

El modelo para este caso del integrador podríamos verlo en el capítulo 7 donde vemos detalladamente cada uno. A modo de ejemplo para este caso pinchando sobre el integrador obtenemos el siguiente cuadro que nos muestra la relación entre el nombre físico y el de las variables del simulador:

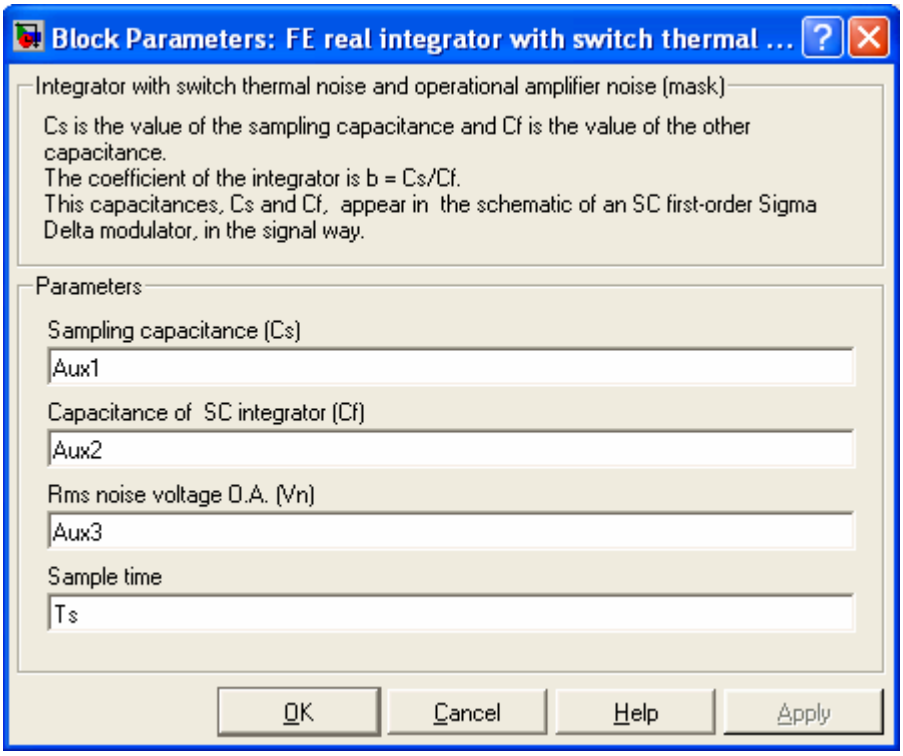
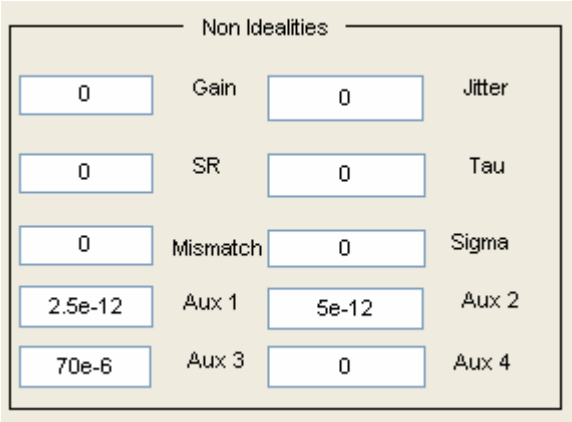


Fig. 8.8. Variables simulación del modelo del integrador no-ideal.

Tomamos unos valores de orientación para las variables y obtener una aproximación de a variación del SNDR.



Variable simulador	Variable física
Aux1	Cs
Aux2	Cf
Aux3	Vn

Fig. 8.9. Variables no ideales.

Los valores tras esta simulación de aproximación son:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Aux1, Aux2, Aux3.
- Effort: 3.

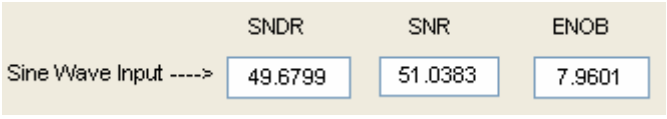


Fig. 8.10. Valores simulación no-ideal Sigma Delta. Ruido op y ktc.

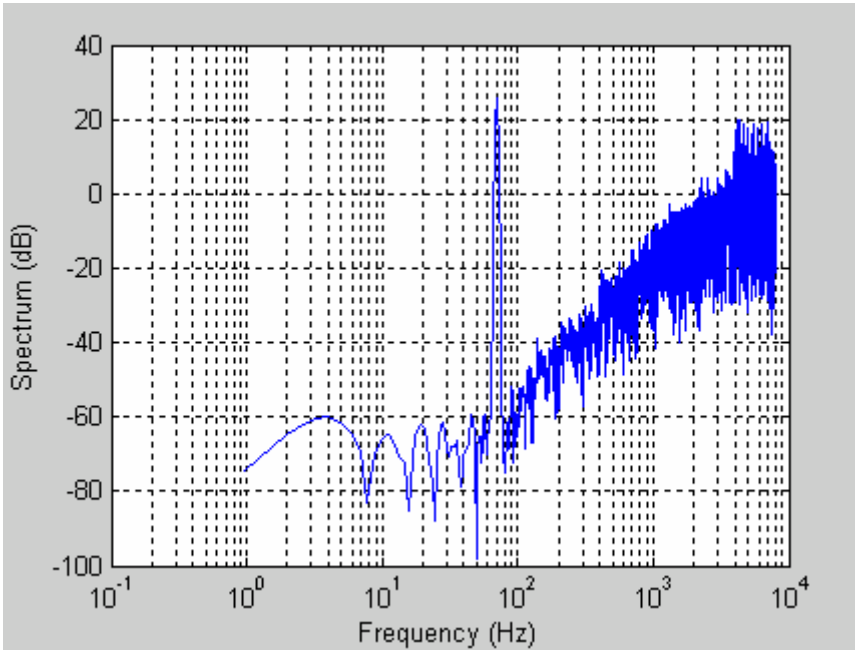


Fig. 8.11. Espectro simulación no-ideal Sigma Delta. Ruido op y ktc.

Tras esta simulación inicial, vamos a realizar un barrido doble de las variables Aux1 (Cs) y Aux2 (Cf), para encontrar un valor más apropiado de éstas de forma que se cumplan las especificaciones de SNDR.

- Tipo: Simulación barrido doble de Aux1 y Aux2
- Entrada: Senoidal
- Variables no ideales: Aux1, Aux2, Aux3.
- Effort: 3.

Los valores que hemos tomados para el barrido de ambas variables son:

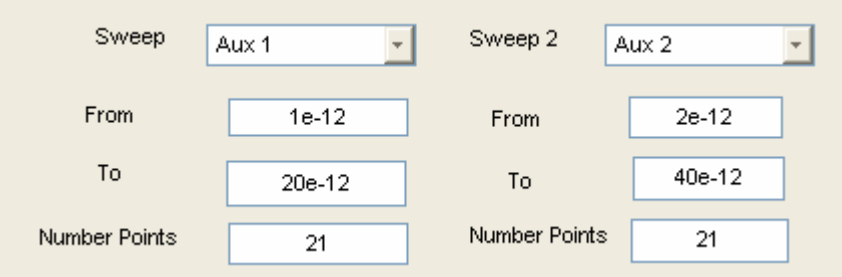


Fig. 8.12. Variables de barrido doble. Ruido op y ktc.

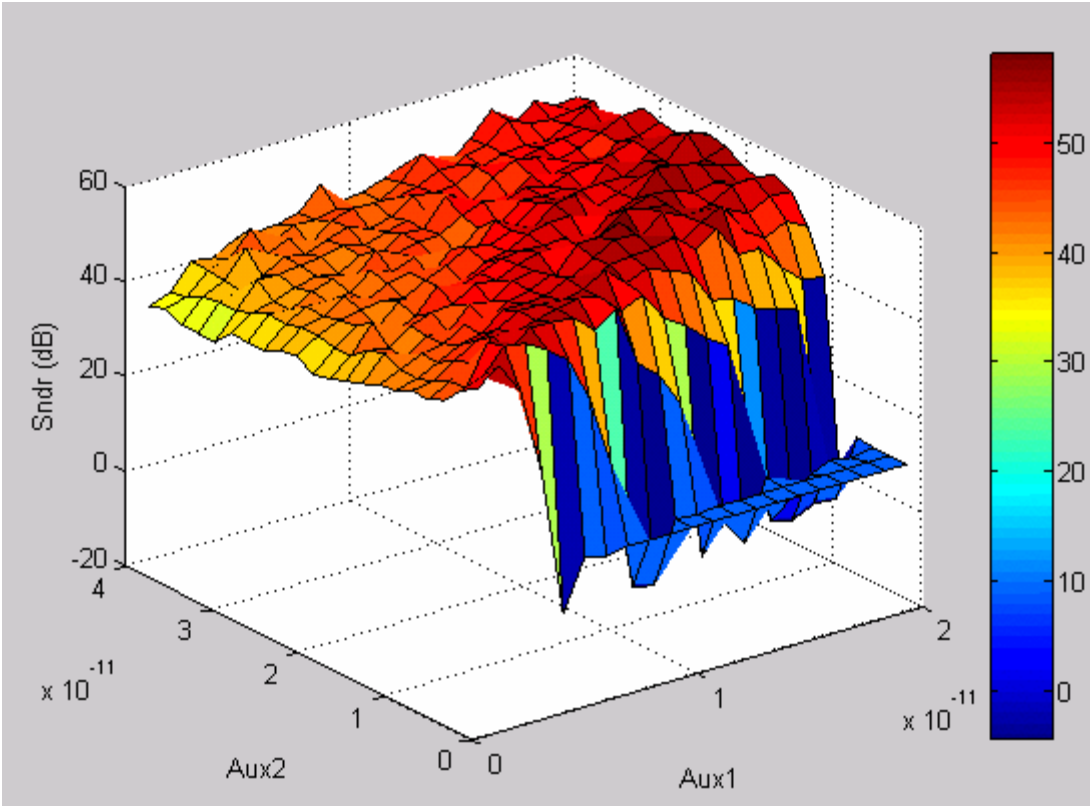


Fig. 8.13. Resultado barrido doble. Ruido op y ktc.

Según los resultados obtenidos en la gráfica de barrido doble, los valores de las variables Aux1 y Aux2 para cumplir las especificaciones deben encontrarse entre el rango:

- Aux 1 (Cs) > 1 pF
- Aux2 (Cf) > 2 pF
- Aux3 (Vn) = 70 uV.

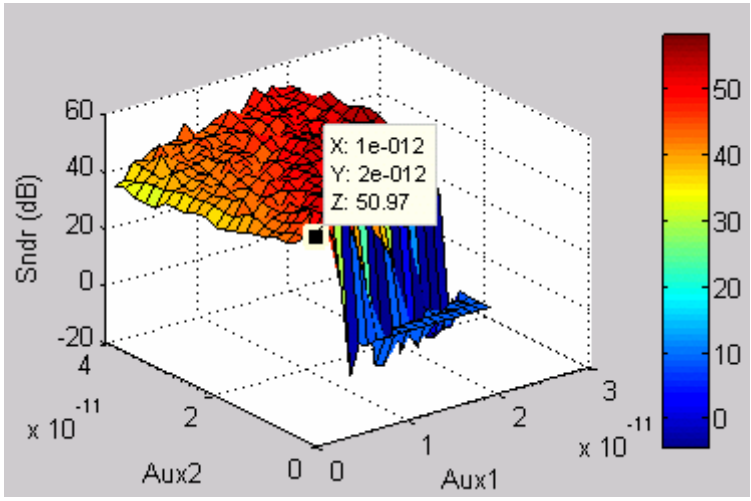


Fig. 8.14. Resultado barrido doble. Ruido op y ktc. Detalle.

Realizando ahora un barrido simple para Aux3 (Vn):

- Tipo: Simulación barrido simple Aux3
- Entrada: Senoidal
- Variables no ideales: Aux1, Aux2, Aux3.
- Effort: 3.

Los valores tomados para el barrido y el resultado obtenido los visualizamos en las figuras:

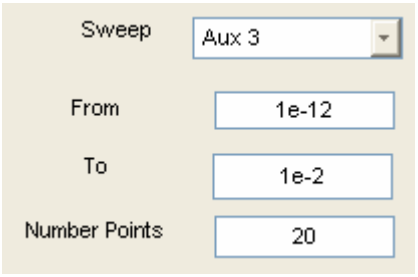


Fig. 8.15. Variables de barrido simple. Ruido op y ktc.

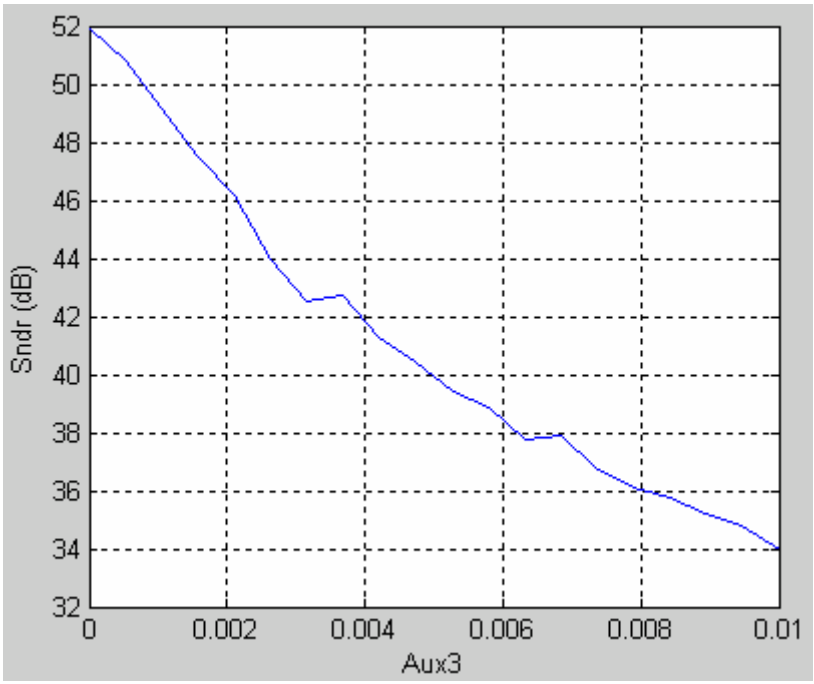


Fig. 8.16. Resultado barrido simple. Ruido op y ktc.

A la vista de la gráfica concluimos que:

- Vn > 0.1 mV_{rms}

Realicemos para finalizar una simulación con estos valores para cacular el nuevo valor del SNDR. Para ello tomamos el valor más limitante de cada variable:

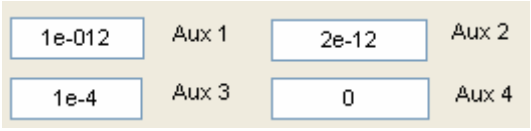


Fig. 8.17. Variables ruido op y ktc.

El valor de SNDR obtenido:

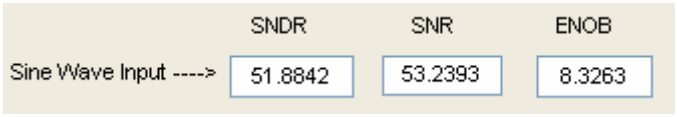


Fig. 8.18. Valores simulación ruido op y ktc.

2.2.2. Ganancia finita

Para la simulación de la ganancia finita utilizamos el diseño del siguiente modelo:

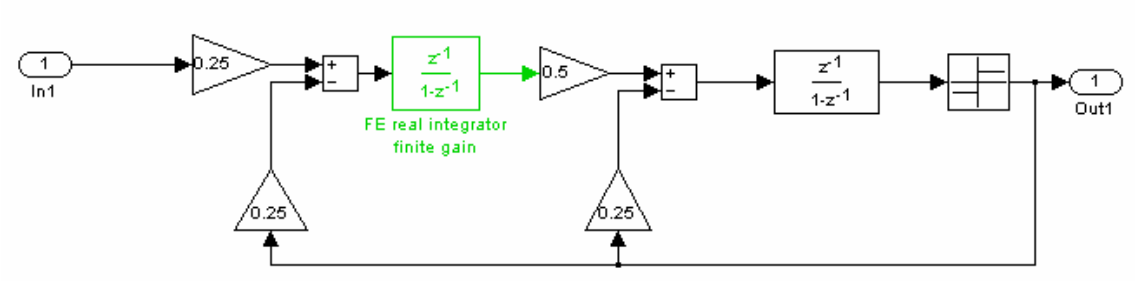
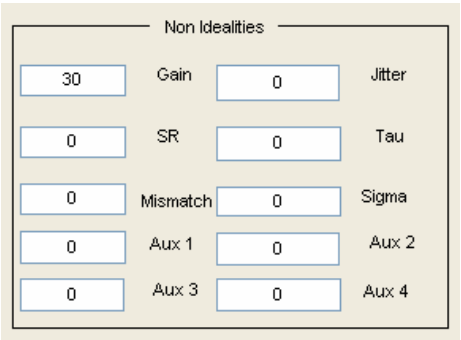


Fig. 8.19. Esquema simulación no-ideal. Ganancia Finita.

Como orientación inicial vamos a tomar el valor



Variable simulador	Variable física
Gain	AodB

Fig. 8.20. Variables no ideales.

Los valores tras esta simulación de aproximación son:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Gain.
- Effort: 3.

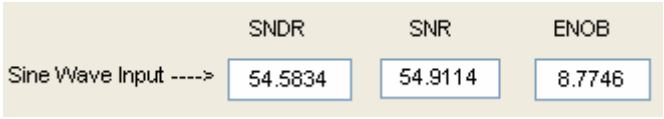
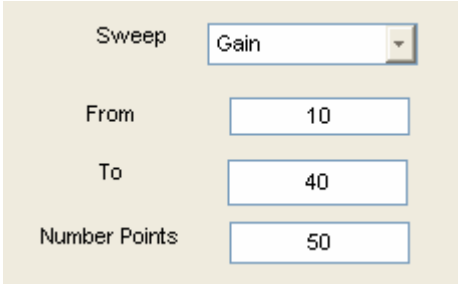


Fig. 8.21. Valores simulación no-ideal Sigma Delta. Ganancia Finita.

Ahora realicemos un barrido simple de la variable Gain, para obtener el valor de ganancia mínima que podemos utilizar para cumplir las especificaciones de SNDR.

- Tipo: Simulación barrido simple Gain
- Entrada: Senoidal
- Variables no ideales: Gain
- Effort: 3

Los valores que hemos tomados para ambas variables son:



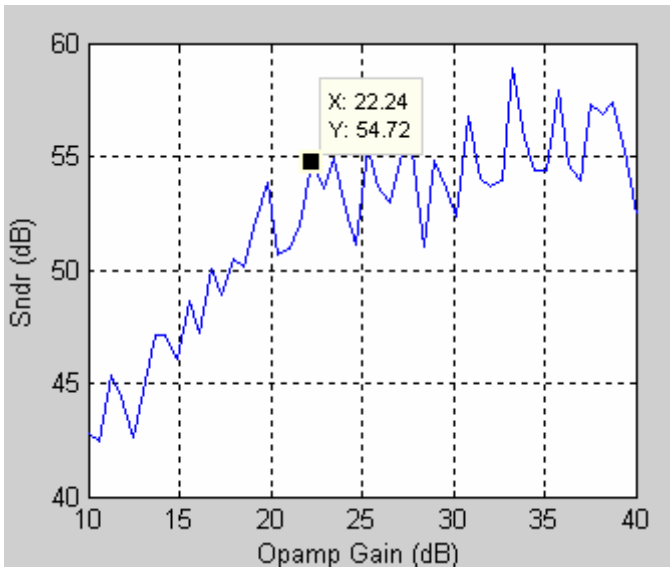


Fig. 8.22. Variables de barrido simple. Resultado. Ganancia Finita.

Concluimos que el valor mínimo de la ganancia debe ser:

- Gain > 22 dB

Para este valor realizamos una simulación obteniendo los siguientes valores y el siguiente espectro:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Gain
- Effort: 3

	SNDR	SNR	ENOB
Sine Wave Input ---->	52.257	52.3938	8.3882

Fig. 8.23. Valores simulación no-ideal Sigma Delta. Ganancia Finita.

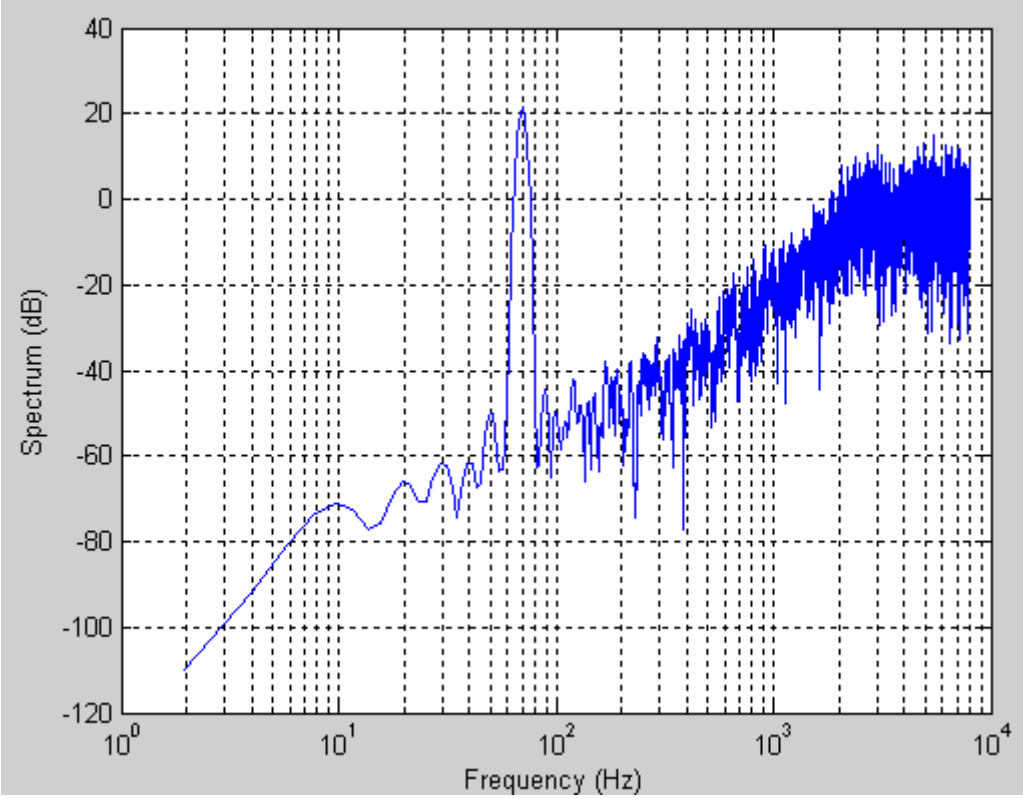


Fig. 8x.24. Espectro simulación no-ideal Sigma Delta. Ganancia Finita.

2.2.3. Slew-rate y GBW

En este apartado nos vamos a encargar de simular las no idealidades del slew-rate y del producto ganancia ancho de banda, todo ello en un mismo integrador que hemos diseñado. Lo presentamos en el siguiente esquema:

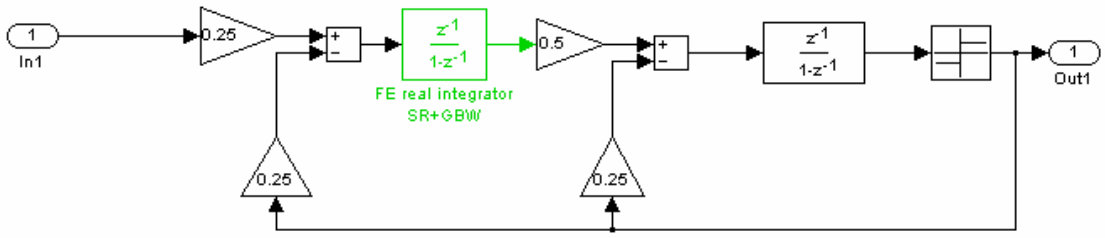
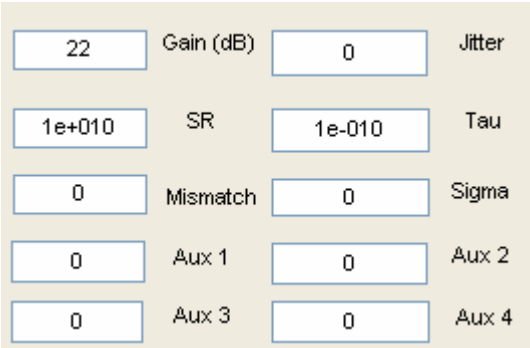


Fig. 8.25. Esquema simulación no-ideal. SR y GBW.

Inicialmente vamos a tomar unos valores orientativos[17] para las variables que vamos a utilizar:



Variable simulador	Variable física
Gain	AodB
SR	SR
Tau	Tau

Fig. 8.26. Variables no ideales.

Los valores tras esta simulación de aproximación son:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Gain.
- Effort: 3.

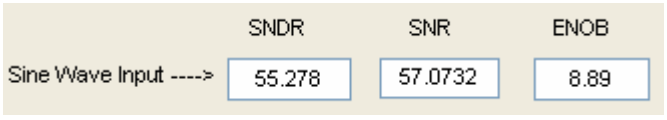


Fig. 8.27. Valores simulación no-ideal Sigma Delta. SR y GBW.

A continuación, tras una primera aproximación, realizaremos un barrido doble con ambas variables, SR y Tau, para obtener unos valores óptimos de las mismas cumpliendo las especificaciones de SNDR.

[17] El valor de la ganancia, variable que necesitamos para la simulación, tomaremos 22dB ya que es el valor al que llegamos en el apartado 2.2.2.

- Tipo: Simulación barrido doble SR y Tau
- Entrada: Senoidal
- Variables no ideales: Gain, SR y Tau
- Effort: 3

Los valores que hemos tomados para ambas variables son:

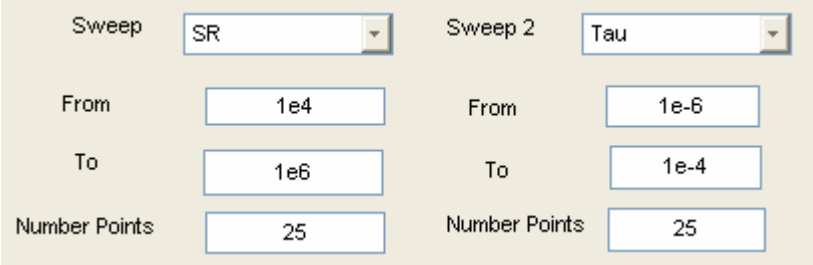


Fig. 8.28. Variables de barrido doble. SR y GBW.

Y la gráfica obtenida para estos valores:

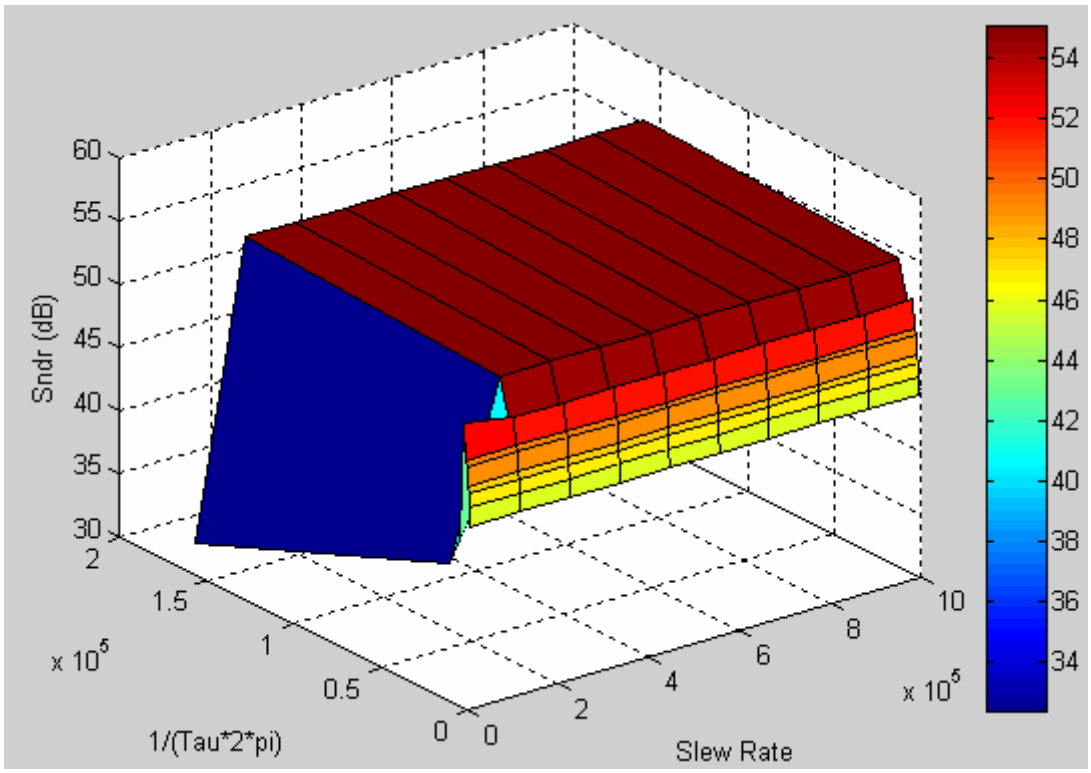


Fig. 8.29. Resultado de barrido doble. SR y GBW.

A la vista de los resultados, los valores limites que tomaremos son:

- SR > 0.1 V/us
- Tau < 10 us

Para estos valores, junto con el de la ganancia, obtenemos los siguientes resultados de SNDR:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Gain, SR y Tau
- Effort: 3

	SNDR	SNR	ENOB
Sine Wave Input ---->	54.0712	55.088	8.6896

Fig. 8.30. Valores simulación no-ideal Sigma Delta. SR y GBW.

Y el espectro obtenido lo mostramos en la siguiente figura:

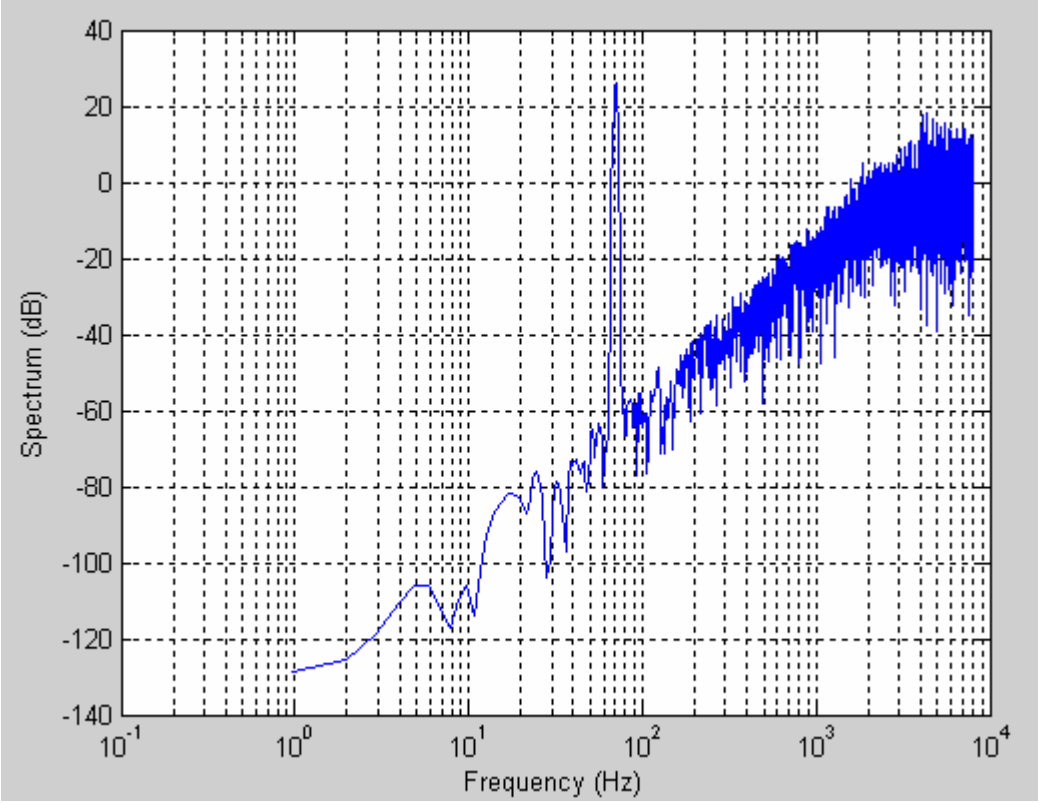


Fig. 8.31. Espectro simulación no-ideal Sigma Delta. SR y GBW.

2.2.4. Saturación en tensión

Veamos otra de las no idealidades que afectan a los integradores, y es la saturación en tensión. Idealmente la salida está entre FS/2 y –FS/2, siendo FS el fondo de escala. Estudiemos como varía el SNDR si la tensión de salida toma valores menores a FS/2 para el límite superior, y mayores a –FS/2 para el límite inferior.

El modelo que hemos diseñado para este caso:

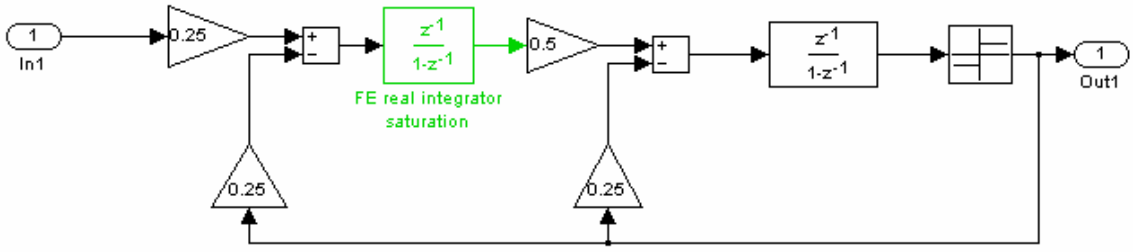
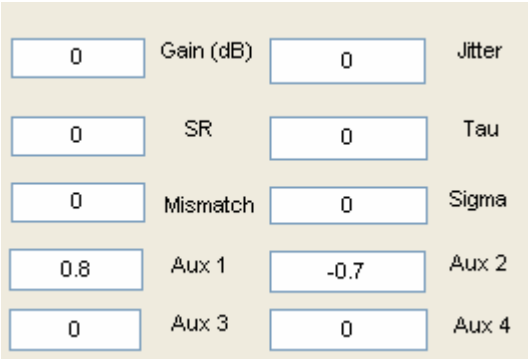


Fig. 8.32. Esquema simulación no-ideal. Saturación.

Inicialmente, vamos a dar unos valores para las variables Aux1 y Aux2, que modelan la saturación superior e inferior respectivamente. Estos valores los podemos simular de forma que no tiene porque ser Aux2 el negado del valor absoluto de Aux1.



Variable simulador	Variable física
Aux1	Vmax
Aux2	Vmin

Fig. 8.33. Variables no-ideales.

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Aux1 y Aux2
- Effort: 3

Obtenemos como resultado de SNDR:

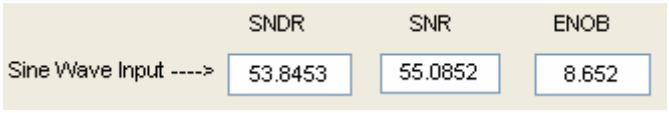


Fig. 8.34. Valores simulación no-ideal Sigma Delta. Saturación.

Veamos ahora si hacemos un barrido de ambas variables:

- Tipo: Simulación barrido doble Aux1 y Aux2
- Entrada: Senoidal
- Variables no ideales: Aux1 y Aux2
- Effort: 3

Tomando los valores para las variables:

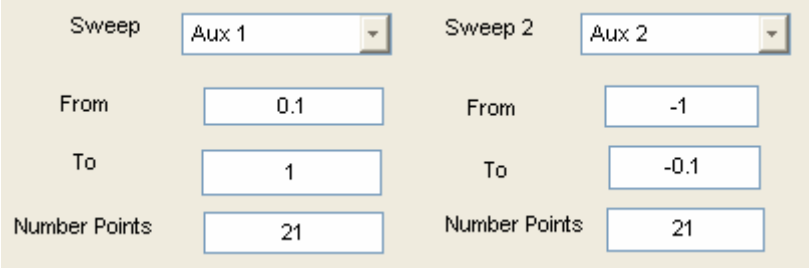


Fig. 8.35. Valores simulación barrido doble. Saturación.

Obtenemos las siguiente gráfica que nos muestra como el SNDR aumenta a medida que los valores de Vmax y Vmin se acercan a los valores FS/2 y –FS/2. Para estas simulaciones hemos considerado FS = 2, la cual era una de las especificaciones de entrada de nuestro convertidor.

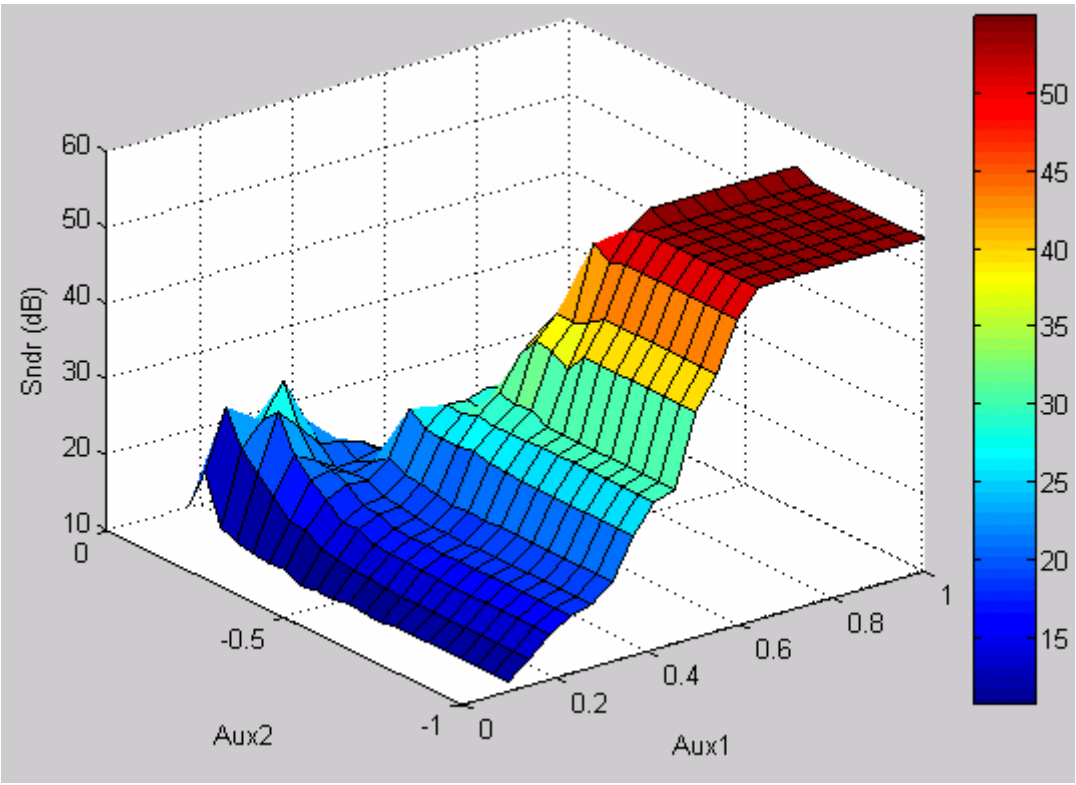
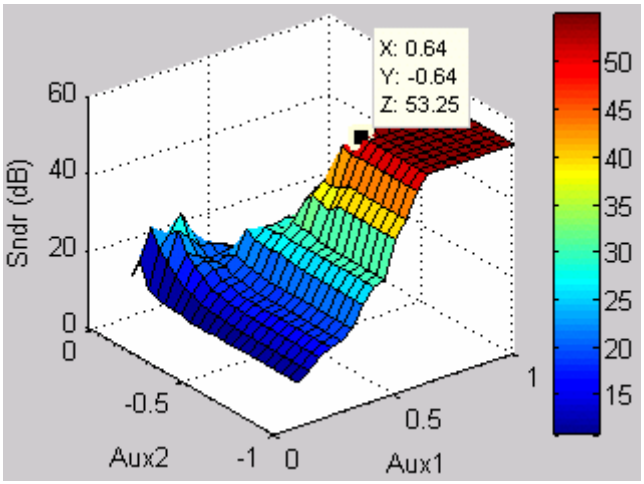


Fig. 8.36. Resultado simulación barrido doble. Saturación.

Tomaremos como valores límite para cumplir las especificaciones:

- Aux1 (Vmax) > 0.65 V
- Aux2 (Vmin) < -0.65 V



Con estos valores visualizamos el SNDR y el espectro de la señal.

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Aux1 y Aux2
- Effort: 3

	SNDR	SNR	ENOB
Sine Wave Input ---->	54.8138	55.8433	8.8129

Fig. 8.37. Resultado simulación simple. Saturación.

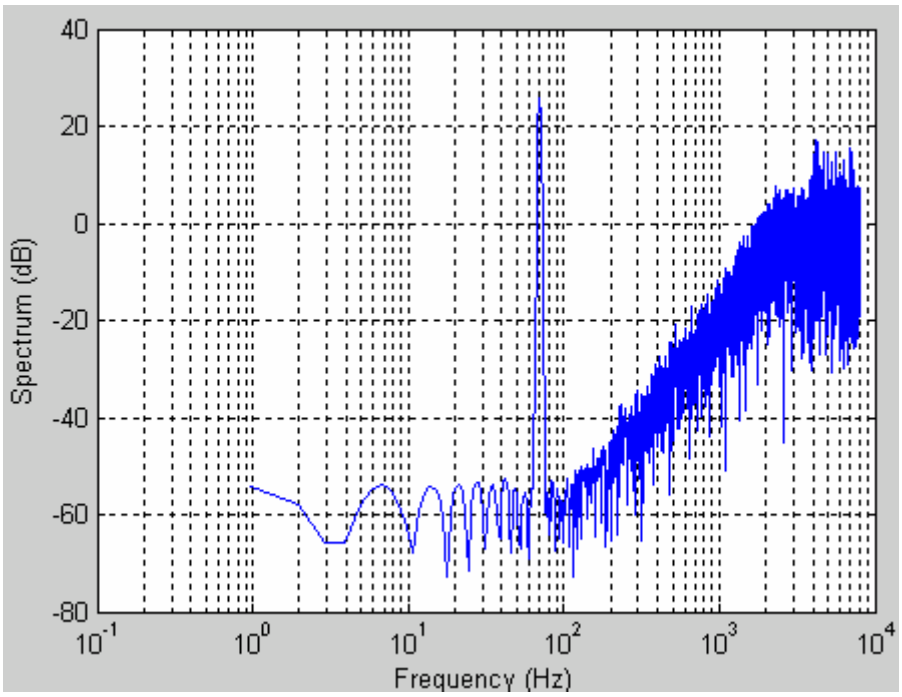


Fig. 8.38. Espectro simulación simple. Saturación.

2.2.5. Ruido Jitter

Por último, vamos a realizar un estudio del ruido jitter. Para ello introducimos el modelo diseñado del jitter a la entrada, utilizando el resto de componentes ideales.

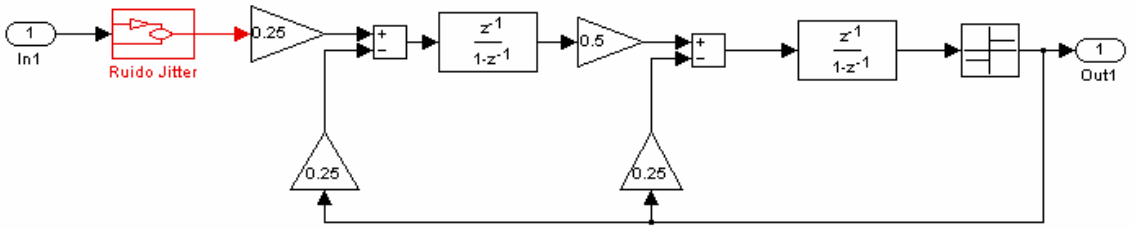


Fig. 8.39. Esquema simulación no-ideal. Jitter.

Tomamos un valor inicial de la variable Jitter, que modela la desviación del ideal, en concreto 1 ns.

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Jitter
- Effort: 3

Para este valor obtenemos:

	SNDR	SNR	ENOB
Sine Wave Input ---->	53.4834	54.7101	8.5919

Fig. 8.40. Resultado simulación simple. Jitter.

Hagamos ahora un barrido para esta variable, lo que nos ofrecerá la información de hasta qué valor es posible considerar el ruido jitter de forma que no afecte negativamente a nuestras especificaciones de SNDR.

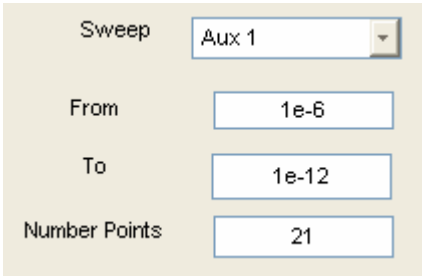


Fig. 8.41. Valores simulación barrido simple. Jitter.

Veremos como a partir que la desviación se hace mayor, el SNDR disminuirá. Tomaremos el valor aproximado a partir del cual cumpla las especificaciones.

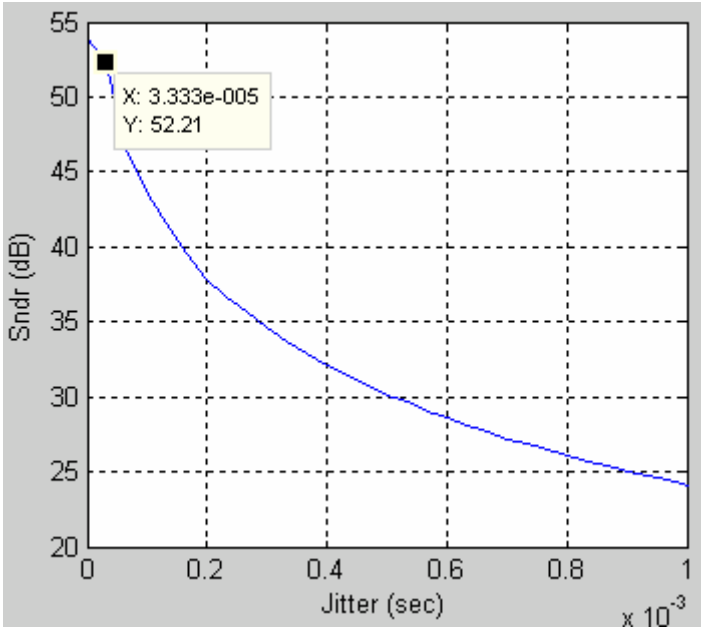


Fig. 8.42. Resultado simulación barrido simple. Jitter.

Tomaremos como límite de la especificación:

- Jitter < 30 us

Para este valor, realizamos una nueva simulación para el valor del SNDR y el espectro de la señal de salida:

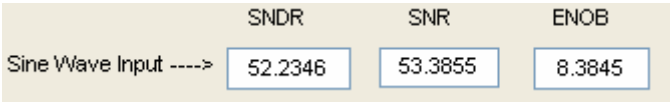


Fig. 8.43. Resultado simulación simple. Jitter.

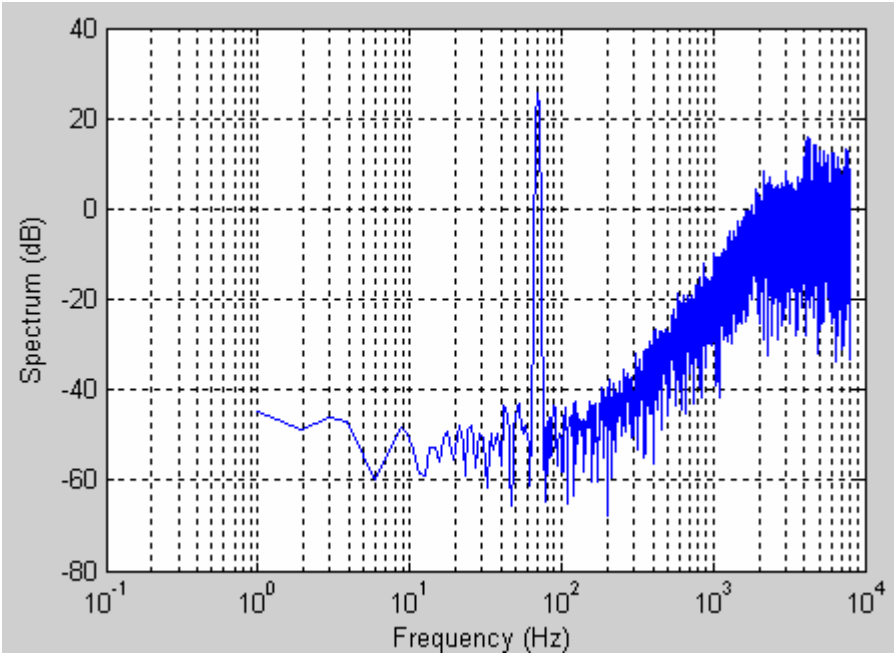


Fig. 8.44. Espectro simulación simple. Jitter.

2.2.6. Integrador con todas las no idealidades

Tras realzar un estudio exhaustivo de las no idealidades, y llegar a unos valores para cada una de las variables, vamos a realizar una simulación con todas ellas, para ver el resultado final del SNDR.

El modelo tomado es aquel con el bloque que simla el jitter a la entrada y el modelo del integrador tomado de la librería con todas las no idealidades:

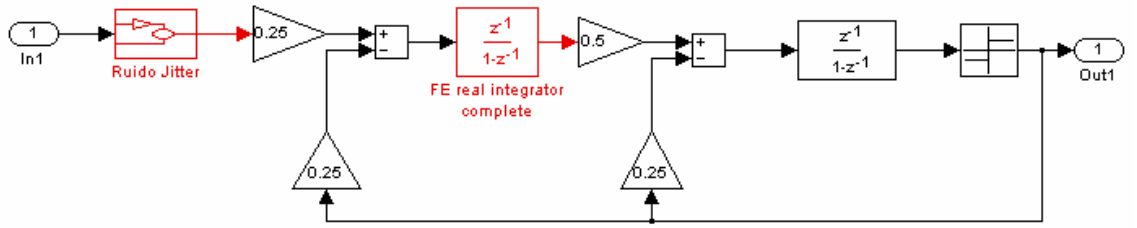


Fig. x.85. Esquema simulación no-ideal completo.

Los valores de las no idealidades son:

22	Gain (dB)	3e-005	Jitter
100000	SR	1e-005	Tau
0	Mismatch	0	Sigma
1e-012	Aux 1	0.0001	Aux 2
0.65	Aux 3	0	Aux 4

Tras realizar la simulación:

- Tipo: Simulación simple
- Entrada: Senoidal
- Variables no ideales: Gain, Jitter, SR, Tau, Aux1 (Cs, Cf/2), Aux2 (Vn), Aux3 (Vmax/Vmin).
- Effort: 3

	SNDR	SNR	ENOB
Sine Wave Input ---->	50.1843	51.1328	8.0439

Fig. 8.46. Valores simulación no-ideal completo.

Tomando todas las no idealidades a su valor límite obtenemos que el SNDR cae 3 dB de su valor ideal, es decir, en su caso más desfavorable. Este valor de SNDR podría aumentarse relajando los valores de alguna no idealidad.

El espectro de la señal quedaría representado en la siguiente figura:

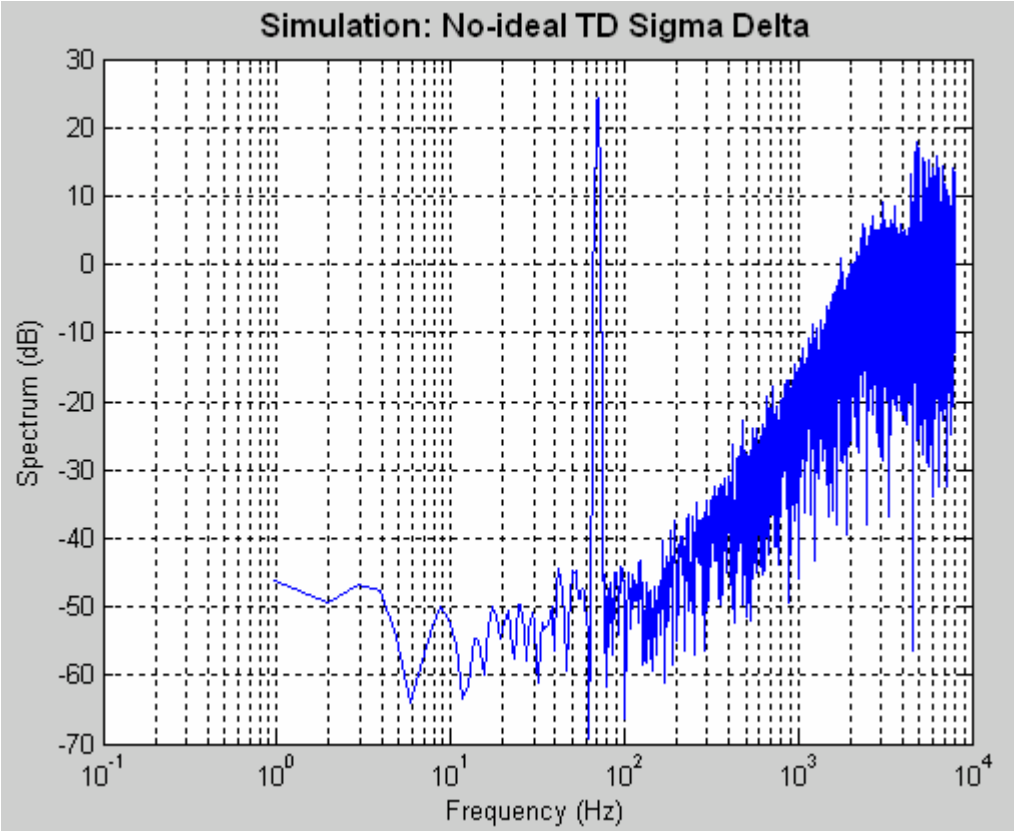


Fig. 8.47. Espectro simulación no-ideal completo.

3. ADC Pipeline

A continuación, pasaremos a detallar el estudio de un convertidor analógico digital pipeline de 10 bits. En primer lugar, mostraremos los resultados obtenidos tras un análisis ideal, y posteriormente introduciremos no idealidades al modelo para ver las variaciones del INL y DNL.

3.1. Estudio ideal

Veamos los resultados de algunas simulaciones realizadas para el caso de bloques ideales. Tomaremos como ejemplo el convertidor pipeline de 10 bits.

- Tipo: Simulación simple
- Entrada: Rampa
- Modelo: pipeline_10b_ideal.mdl

El aspecto de la GUI puede verse en la figura 8.48., en la que pueden apreciarse los valores de las variables de entrada (zona de Inputs) y de las no idealidades (zona de Non Idealities), y los resultados siguientes (zona de Results):

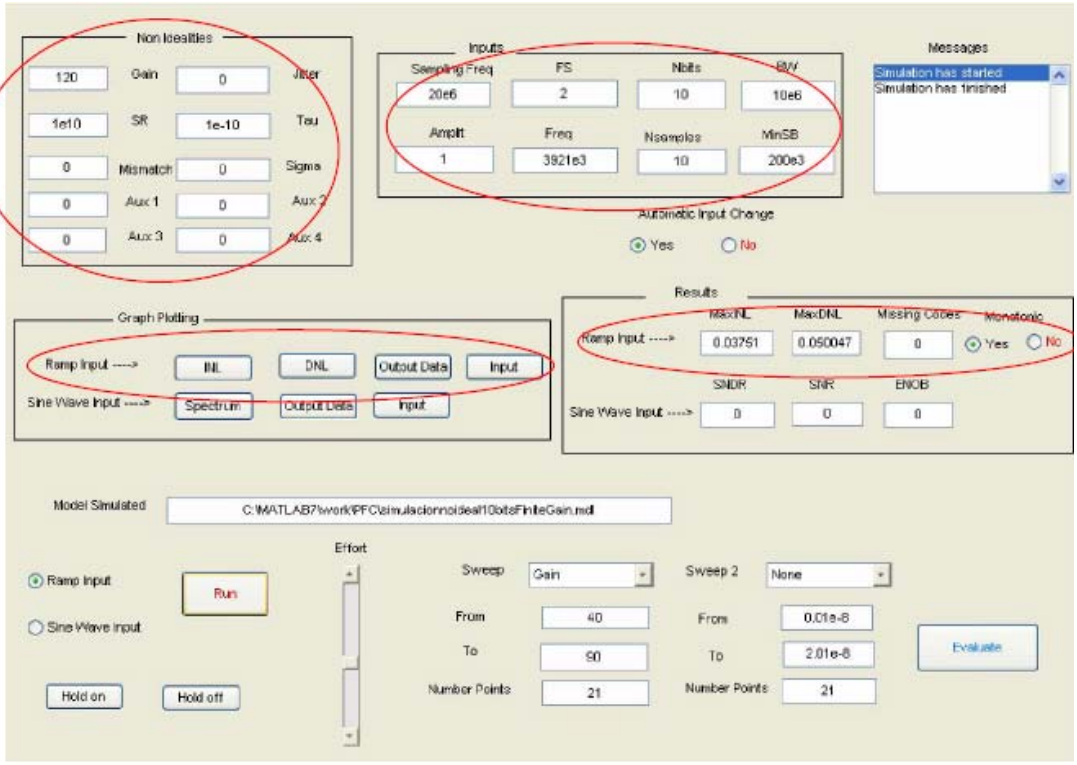


Fig. 8.48. GUI tras la simulación del pipeline ideal.

Los valores obtenidos son:

MaxINL=0.03751
MaxDNL=0.050047
Missing Codes=0

Las gráficas de INL, DNL, Output Data e Input pueden verse en las figuras 8.49(a), (b), (c) y (d) respectivamente. Puede apreciarse que el valor máximo del valor absoluto de la gráfica de INL coincidirá con el resultado MaxINL (MaxINL tiene en cuenta los códigos perdidos pero, al salir los códigos perdidos 0, coincide el máximo del valor absoluto de la gráfica con MaxINL).

De la misma forma, coincide el máximo del valor absoluto de la gráfica de DNL con MaxDNL (en MaxDNL no se tienen en cuenta los códigos perdidos, luego el máximo de la gráfica de DNL siempre coincidirá con MaxDNL).

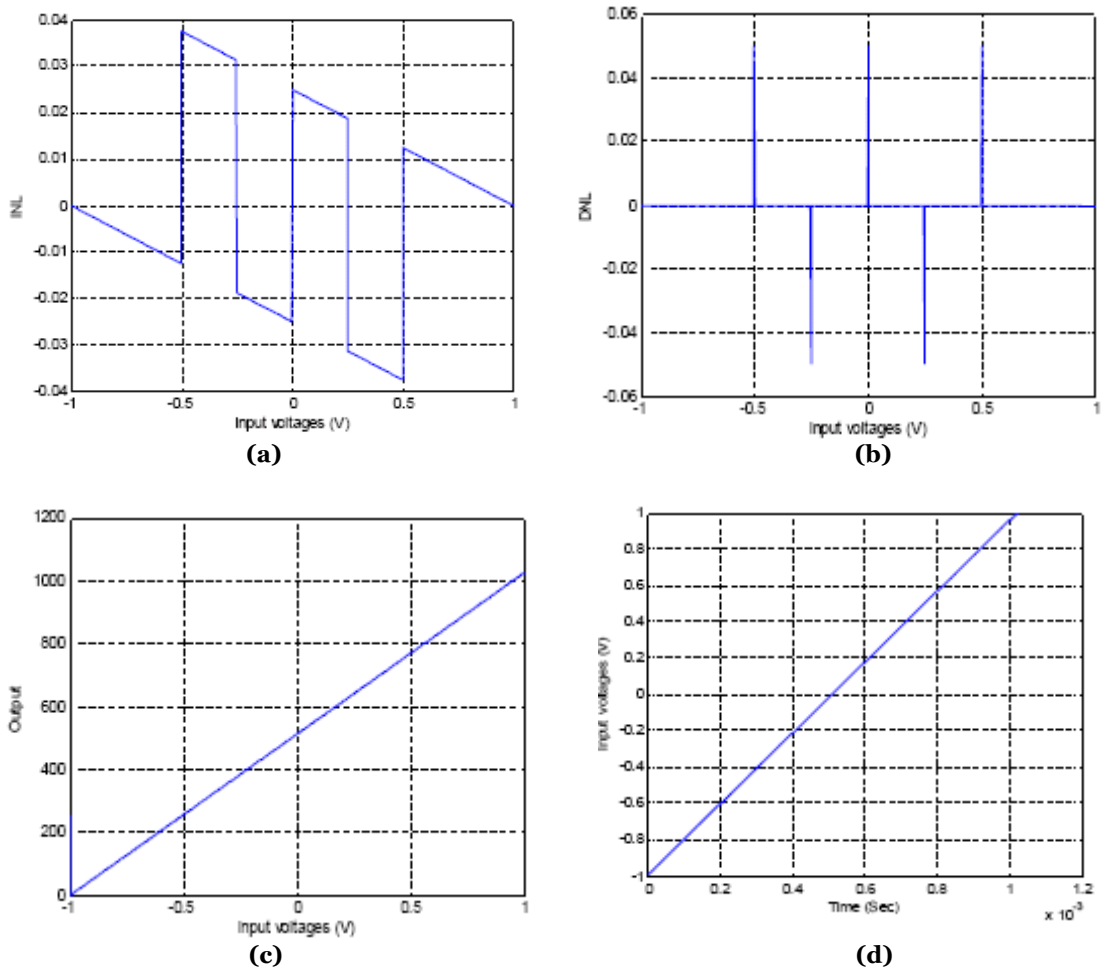


Fig. 8.49. (a) INL (b) DNL (c) Output (d) Input

3.2. Estudio no ideal

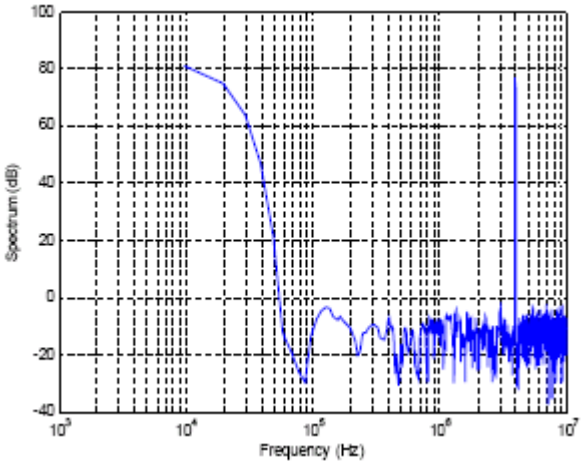
Veamos ahora unas cuantas simulaciones en el caso de que introduzcamos no idealidades.

3.2.1. Finite Gain

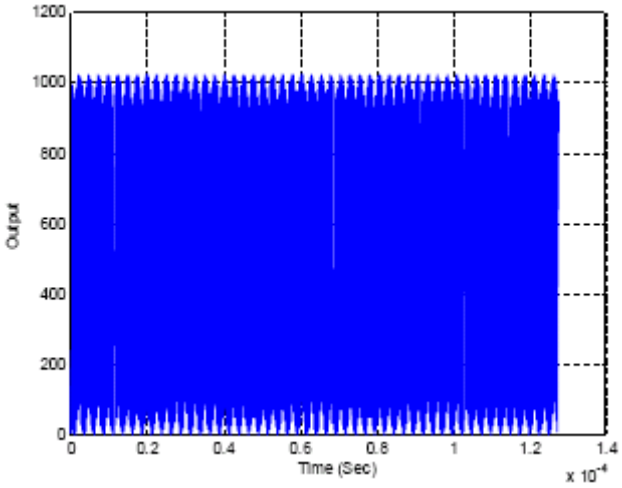
- Tipo: Simulación ideal
- Entrada: Seno
- Modelo: pipeline_10b_finitegain.mdl

Los resultados obtenidos: SNDR=61.9508, SNR=62.0351, ENOB=9.9985.

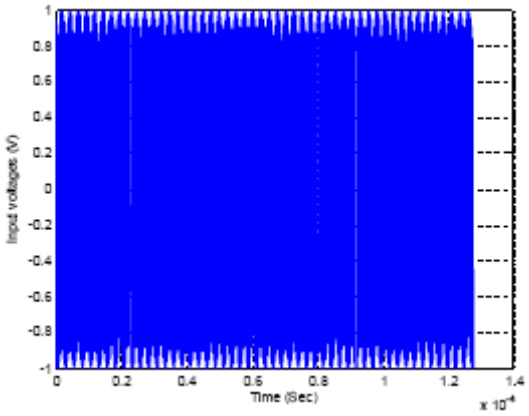
Las gráficas del espectro y de la entrada y salida son:



(a)



(b)



(c)

Fig. 8.50. (a) Espectro (b) Ouput (c) Input

3.2.2. Mismatch

Para este caso tomamos los siguientes parámetros de simulación:

- Tipo: Barrido simple de Desapareamiento entre Capacidades
- Entrada: Rampa
- Modelo: pipeline_10b_mismatch.mdl
- Effort: 5
- Start: -0.2
- Stop: 0.2
- Points: 41

La gráfica resultante puede verse en la figura 8.51., y una ampliación en la zona de mayor interés en la figura 8.52. De esta última se extrae que, para asegurar que INL sea menor de 0.25, el desapareamiento de las capacidades de

la primera etapa debe ser inferior al 0.02%.

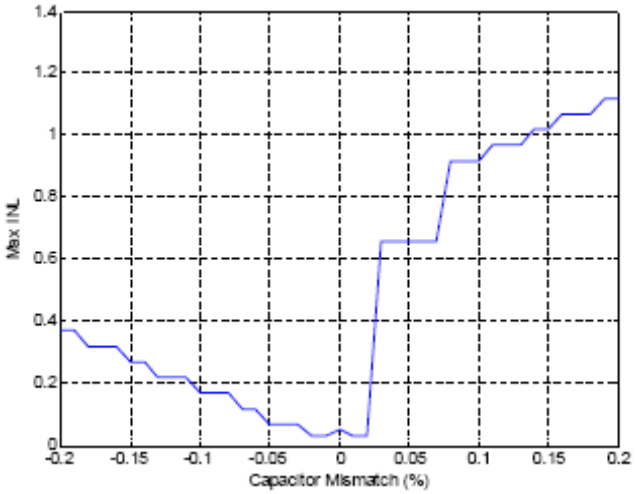


Fig. 8.51. Gráfica de MaxINL frente al desapareamiento de las capacidades de la primera etapa

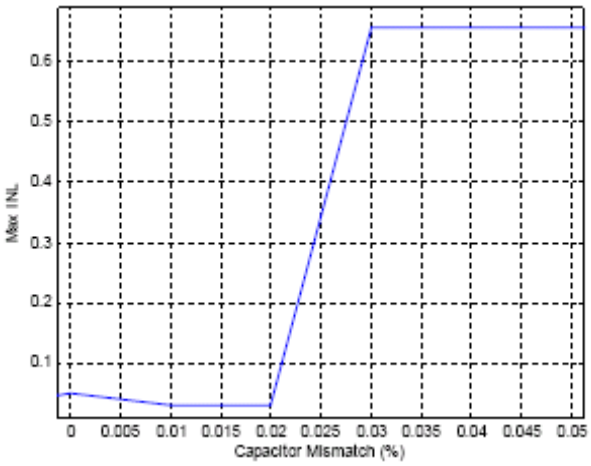


Fig. 8.52. Ampliación de la figura 8.51 en la zona de mayor interés

Si realizamos una unión entre los dos ejemplos anteriores:

- Tipo: Barrido doble de Ganancia y Desapareamiento entre Capacidades
- Entrada: Seno
- Modelo: pipeline_10b_finitegain.mdl y pipeline_10b_mismatch.mdl
- Effort: 7
- Start: 40
- Stop: 90
- Points: 21
- Start2: -0.2
- Stop2: 0.2
- Points2: 21

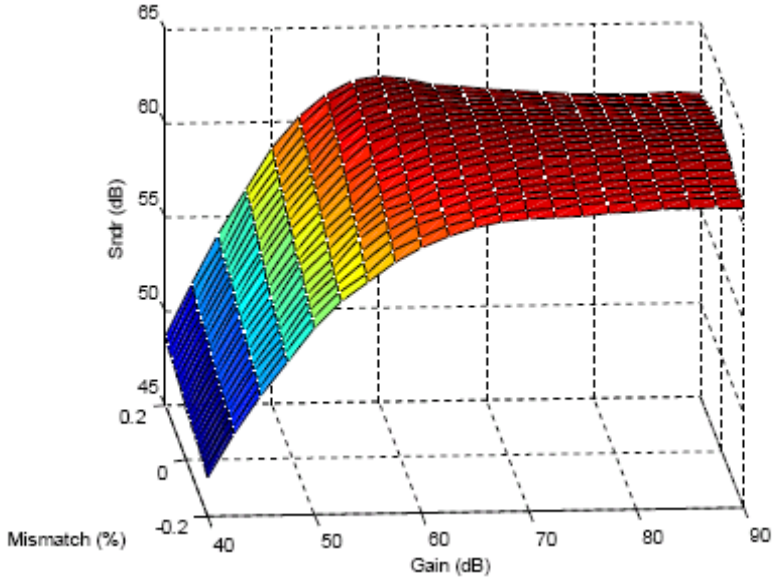


Fig. 8.53. Gráfica de SNDR frente a la Ganancia y al Desapareamiento entre Capacidades.

Y si simulamos la ganancia finita junto al jitter con los datos de simulación Effort: 7, Start: 40, Stop: 90, Points: 21, Start2: 0.1, Stop2: 1 y Points2: 21.

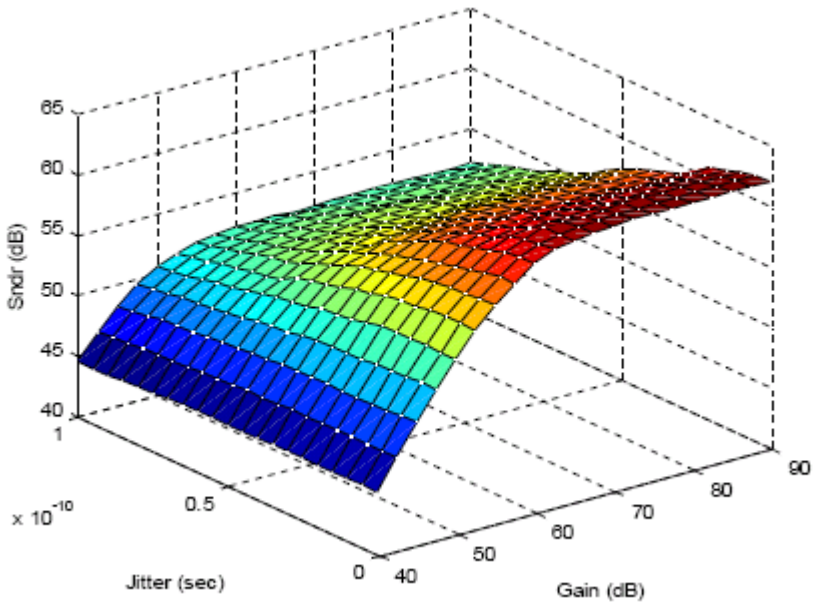


Fig. 8.54. Gráfica de SNDR frente a la Ganancia y al Jitter

4. ADC Flash

Y para completar este capítulo, vamos a realizar una simulación de un ejemplo del convertidor analógico digital Flash.

4.1. Estudio ideal

Comenzaremos con las simulaciones ideales de este convertidor, después pasaremos a los barridos simples y por último expondremos los barridos dobles:

- Tipo: Simulación ideal
- Entrada: Rampa
- Modelo: flash_offset_relay.mdl

El aspecto de la GUI puede verse en la figura 8.55., en la que pueden apreciarse los valores de las variables de entrada (zona de Inputs) y de las no idealidades (zona de Non Idealities), y los resultados siguientes (zona de Results):

MaxINL=0
MaxDNL=0
Missing Codes=0

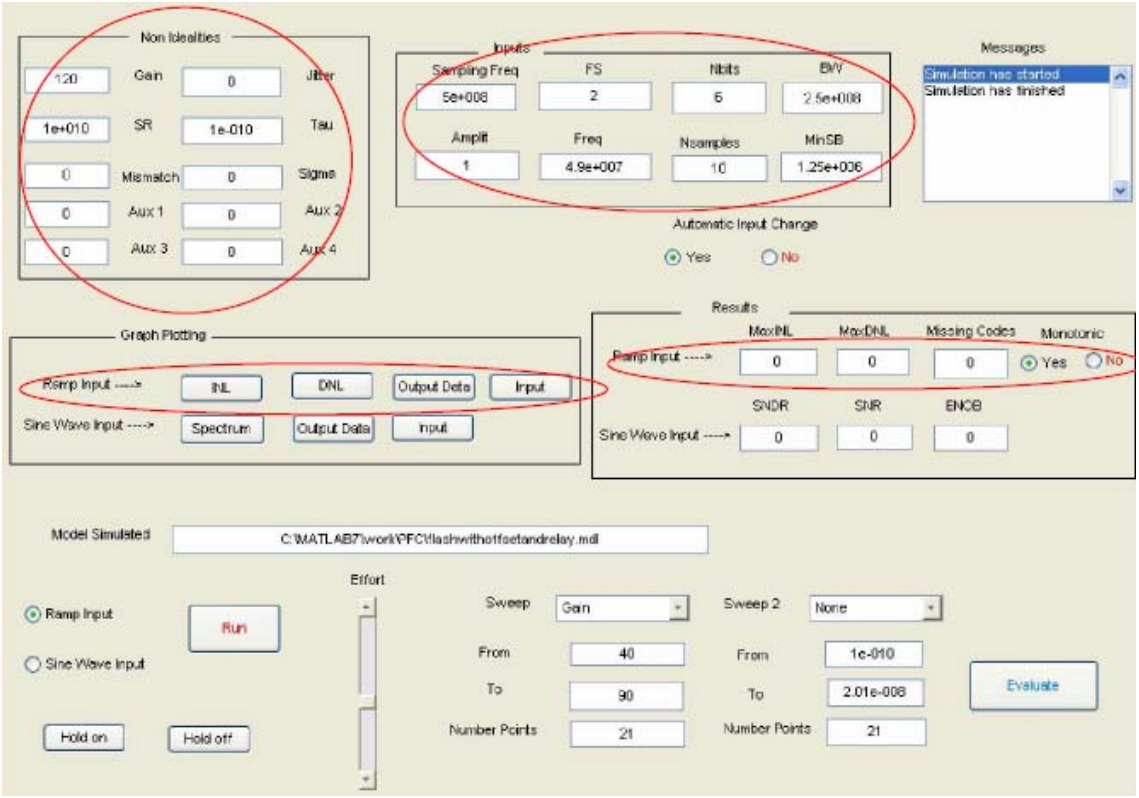


Fig. 8.55.: GUI tras Flash

Las gráficas de INL, DNL, Output Data e Input pueden verse en las figuras 8.56, 8.57, 8.58 y 8.59 respectivamente. Puede apreciarse que el valor máximo del valor absoluto de la gráfica de INL coincidirá con el resultado MaxINL (MaxINL tiene en cuenta los códigos perdidos pero, al salir los códigos perdidos 0, coincide el máximo del valor absoluto de la gráfica con MaxINL).

De la misma forma, coincide el máximo del valor absoluto de la gráfica de DNL con MaxDNL (en MaxDNL no se tienen en cuenta los códigos perdidos, luego el máximo de la gráfica de DNL siempre coincidirá con MaxDNL).

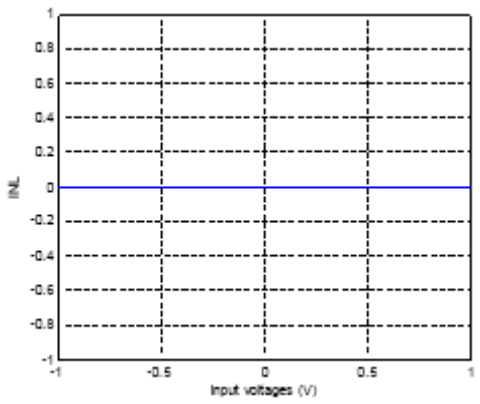


Fig. 8.56.: INL

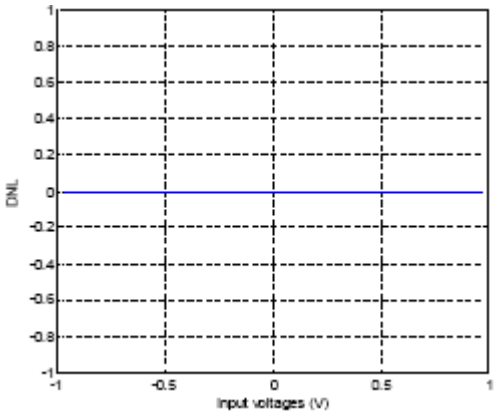


Figura 8.57.: DNL

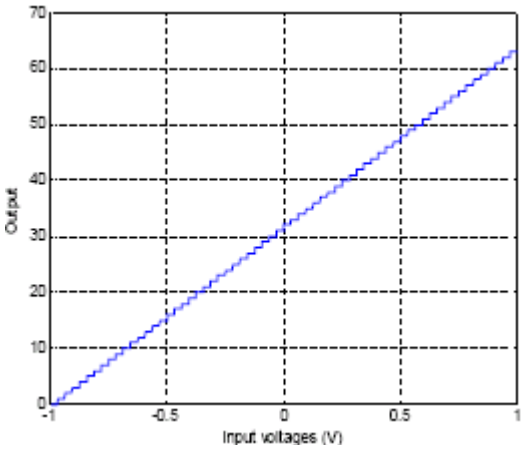


Fig. 8.58.: Output

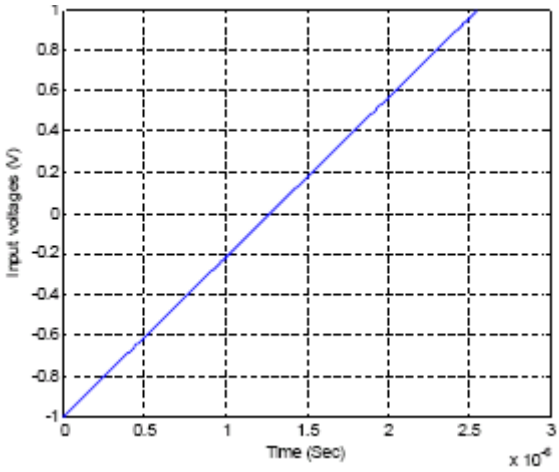


Figura 8.59.: Input

- Tipo: Simulación simple
- Entrada: Seno
- Modelo: flash_offset_relay.mdl

El aspecto de la GUI es similar al de la figura 8.55., ya que se han elegido los mismos valores de las variables de entrada y de las no idealidades, sólo hemos cambiado el tipo de entrada, por lo que ahora aparecerán los valores en la zona de Results en los cuadros de texto correspondientes a entrada seno. Se obtienen los resultados siguientes:

SNDR=37.6702
SNR=38.0628
ENOB=5.9651

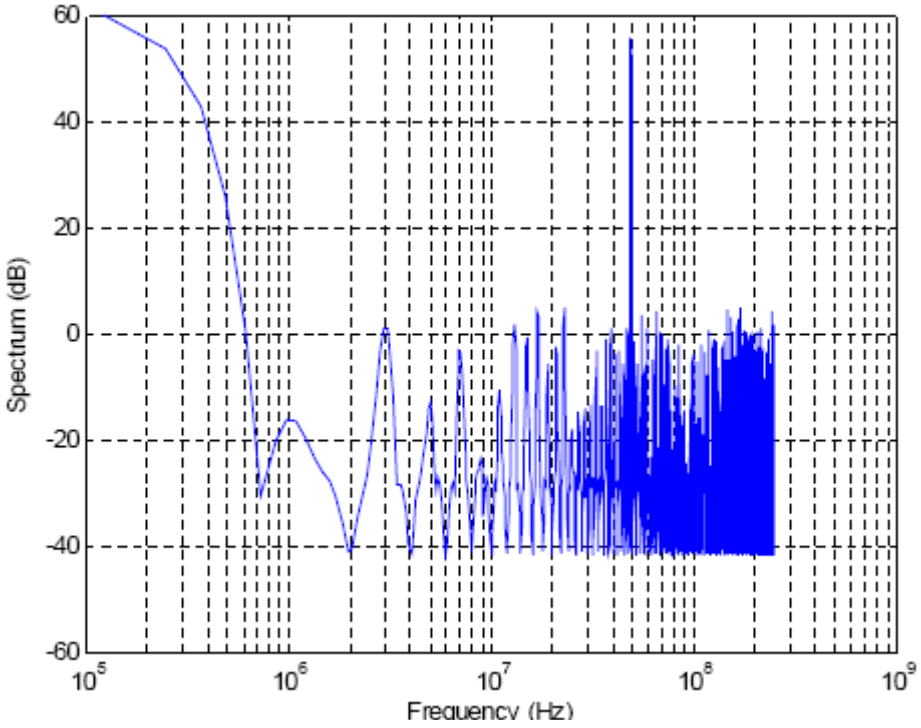


Fig.8.60 Espectro: Ejemplo flash.

4.2. Estudio no ideal

Veamos para el mismo ejemplo que tenemos en estudio, la incorporación de no idealidades.

- Tipo: Barrido simple de offset en las tensiones de referencia (Aux1)
- Entrada: Seno
- Modelo: flash_offset_relay.mdl
- Effort: 5

- Start: 1e-2
- Stop: 1e-3
- Points: 11

La gráfica resultante puede verse en la figura 8.61, en la que puede apreciarse cómo, a medida que aumentamos la desviación típica del offset en las tensiones de referencia, la SNDR tiende a disminuir, aunque a veces puede aumentar debido al carácter aleatorio de la variable, pero como hemos dicho en general tiende a disminuir.

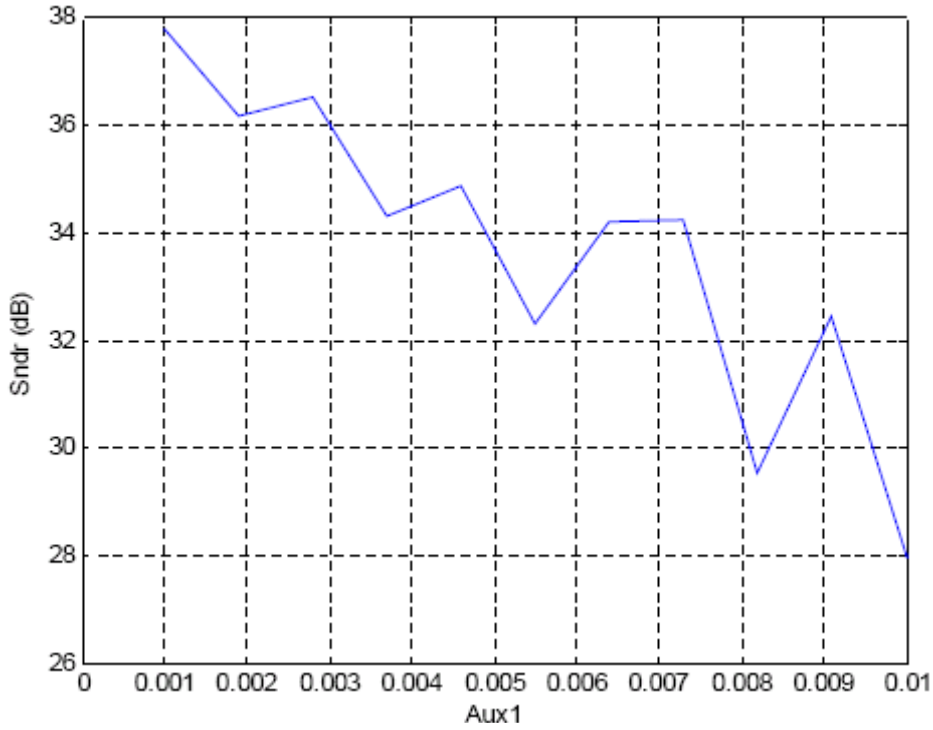


Fig. 8.61.: Gráfica SNDR frente a la desviación típica del offset en las tensiones de referencia.

Simulemos otro tipo de no idealidad, como puede ser el offset y la histéresis de los comparadores:

- Tipo: Barrido simple de offset en los comparadores (Aux2)
- Entrada: Seno
- Modelo: flash_offset_realy.mdl
- Effort: 5
- Start: 1e-2
- Stop: 1e-3
- Points: 11

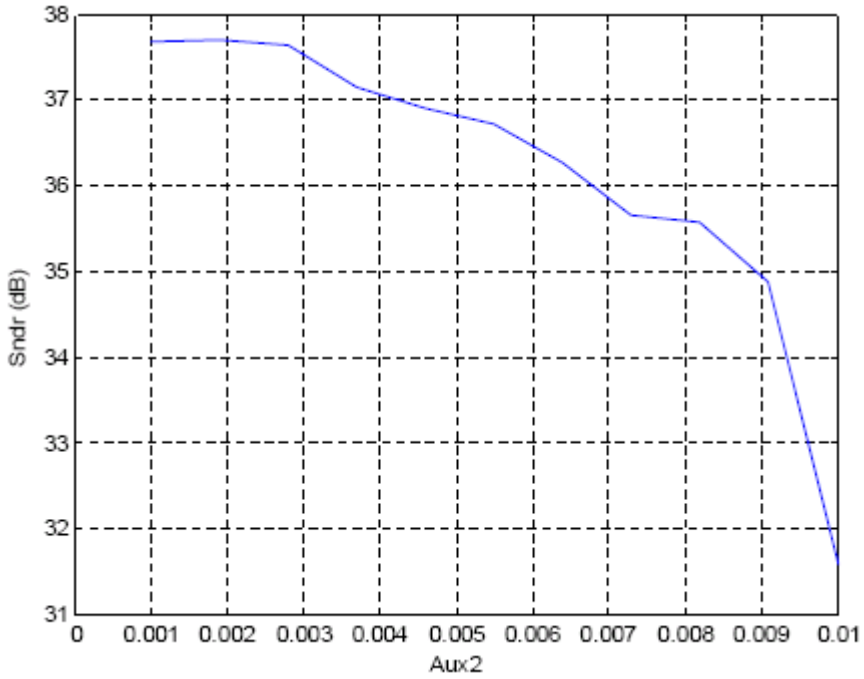


Fig 8.62: Gráfica SNDR frente a la desviación típica del offset en los comparadores

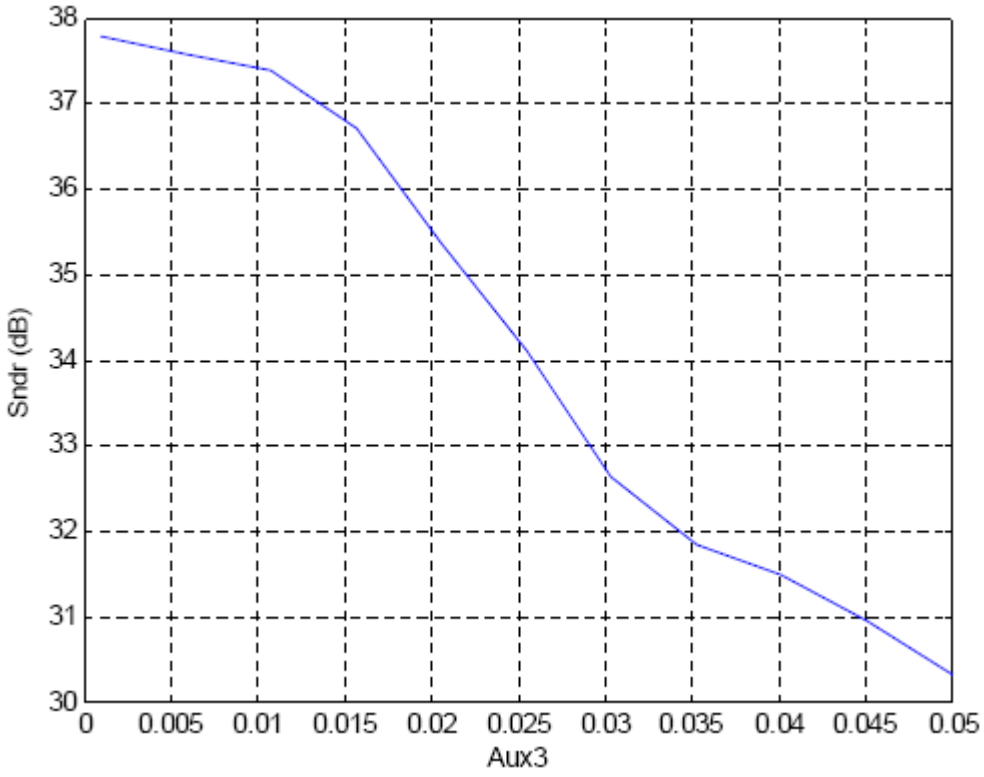


Fig 8.63.: Gráfica SNDR frente a la histéresis en los comparadores

Si realizamos una simulación realizando un barrido doble entre anbas variables obtenemos la gráfica mostrada en la Fig.8.64:

- Tipo: Barrido doble de offset en las tensiones de referencia (Aux1) e histéresis en los comparadores (Aux3)
- Entrada: Seno
- Modelo: flash_offset_relay.mdl
- Effort: 5
- Start: 1e-2
- Stop: 1e-3
- Points: 7
- Start2: 5e-2
- Stop2: 1e-3
- Points2: 7

La gráfica resultante puede verse en la figura 8.64., donde puede apreciarse que la SNDR va subiendo en la dirección en la que disminuyen la histéresis en los comparadores y la desviación típica del offset en las tensiones de referencia, aunque a veces puede disminuir debido al carácter aleatorio del offset, pero como hemos dicho en general tiende a subir en la dirección indicada.

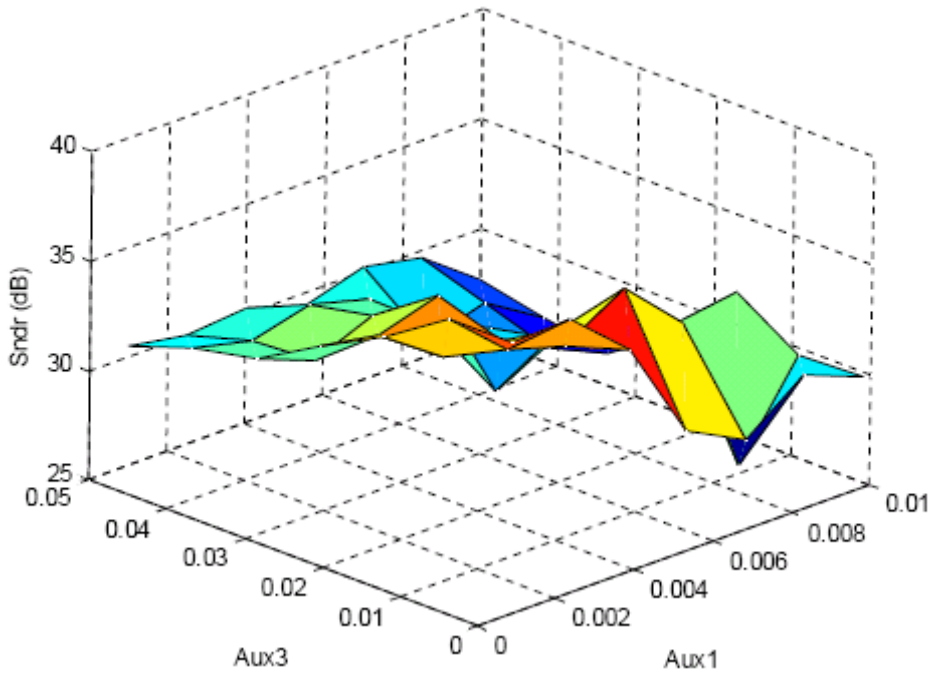


Figura 8.64.: Gráfica SNDR frente a la desviación típica del offset en las tensiones de referencia y a la histéresis en los comparadores

CAPÍTULO 9

Conclusiones

Índice:

1. Aportaciones

Mediante la realización de este proyecto se ha dotado a la herramienta de simulación de convertidores analógico-digitales *simconverter* una libraría compuesta por modelo ideales, no ideales y ejemplos de convertidores Sigma Delta, Pipeline y Flash. Para la visualización de estos modelos desde la GUI, se ha desarrollado:

- Interfaz que permite de forma dinámica la búsqueda del modelo deseado.
- Interconexión y comunicación entre las dos interfaces, simconverter y library, de manera transparente al usuario.

Por otra parte, gracias a la base de modelos, se podrán utilizar para diversas aplicaciones electrónicas. Por esta razón se han tomado casos reales que actualmente están en estudio o en desarrollo, para demostrar como esta herramienta puede ser útil en el proceso.

Un aspecto muy interesante de la librería es que permite simular cualquier tipo de convertidor con cualquier resolución y con un alto número de no idealidades que no tienen por qué exactamente las descritas por el programa, sólo necesitaríamos construir el modelo de ese convertidor y llamar a las no idealidades que queramos incluir con los mismos nombres que se les ha dado en el programa (para éste será transparente, es decir, si a un modelo con offset por ejemplo llamamos al offset ganancia podriamos hacer simulaciones asumiendo que los valores que pongamos en ganancia corresponderán al offset). Esto nos permite, en la práctica, simular infinidad de convertidores diferentes.

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.

Como conclusión final podríamos decir que la versatilidad y la variedad de posibilidades que nos permite esta herramienta, junto a la potencia de Matlab, hace que el programa implementado pueda suponer un elemento muy interesante y de gran ayuda para los diseñadores de convertidores.

2. Futuras líneas

Este proyecto, en cuanto a la librería, podría continuarse:

- Añadiendo más modelos e incluyendo otros tipos de convertidores folding, interleaving...
- Aumentando el número de ejemplo o aplicaciones.

CAPÍTULO 10

Bibliografía

Índice:

1. Libros

- M. Gustavsson, J.J. Wikner, N. Tan: **"CMOS Data Converters forCommunications"**. Kluwer Academic Publishers, 2000.
- David Johns, Ken Martin, **"Analog Integrated Circuit Design"**, John Wiley and Sons, Inc., 1997.
- The MathWorks, Inc., **"Creating Graphical User Interfaces: Versión 7"**, 2004.
- D. Falcón, **"Simulación de convertidores pipeline"**. Proyecto Fin de Carrera, Ingeniería de Telecomunicación, Universidad de Sevilla, 2005.

2. Artículos

- S. Brigati, F. Francesconi, P. Malcovati, D. Tonietto, A. Baschirotto and F. Maloberti: **"Modeling Sigma-Delta Modulator Non-Idealities in Simulink"**, *Proc. of the 1999 IEEE Int. Symposium on Circuits And Systems (ISCAS)*, pp. II.384-387.
- P. Malcovati, S. Brigati, F. Francesconi, D. Tonietto, F. Maloberti, F. Maloberti and A. Baschirotto: **"Behavioral modeling of Switched-Capacitor Sigma-Delta Modulators"**, *Proc. of the 2003 IEEE Int. Transaction on Circuits And Systems*, Vol 50, Nº3, Marzo 2003, pp. II.352-364.
- Andrew M. Abo, Paul R. Gray, **"A 1.5-V, 10-bit, 14.3MS/s CMOS Pipeline Analog-to-Digital Converter"**, IEEE Journal of Solid-State Circuits, Vol. 34, Nº 5, Mayo 1999.
- Dr. Mahmoud Fawzy Wagdy, Qiong Xie, **"Comparative ADC Performance Evaluation Using a New Emulation Model for Flash ADC Architectures"**, Department of Electrical Engineering, IEEE 1995.

ANEXO 1

Manual de Usuario

Índice:

1. Introducción

Este apéndice pretende ser una completa guía para los usuarios de la aplicación. Dicha aplicación es un simulador de convertidores analógicos-digitales, que nos permitirá el análisis sistemático y rápido de dichos convertidores: errores, no idealidades, barridos, etc.

Para instalar esta herramienta de simulación, hemos de copiar en una subcarpeta (creada previamente) de Matlab7\work el contenido de la subcarpeta simconverter de la raíz del CD suministrado (es decir, si por ejemplo tenemos el CD introducido en la unidad D: y Matlab7 instalado en la raíz de C:, habría que copiar todo el contenido de D:\simconverter en C:\Matlab7\work\PFC).

Una vez hayamos instalado la herramienta, es decir, hayamos copiado el contenido del CD, y tras abrir Matlab, situarnos en el directorio en el que hayamos copiado la aplicación, y colocar el cursor en la ventana de comandos. Tras hacer todo esto ya podemos usar la orden *simconverter* para arrancar el simulador. El resultado es la aparición de la ventana que se muestra en la Fig. A1.1, que describiremos con detalle en los siguientes apartados.

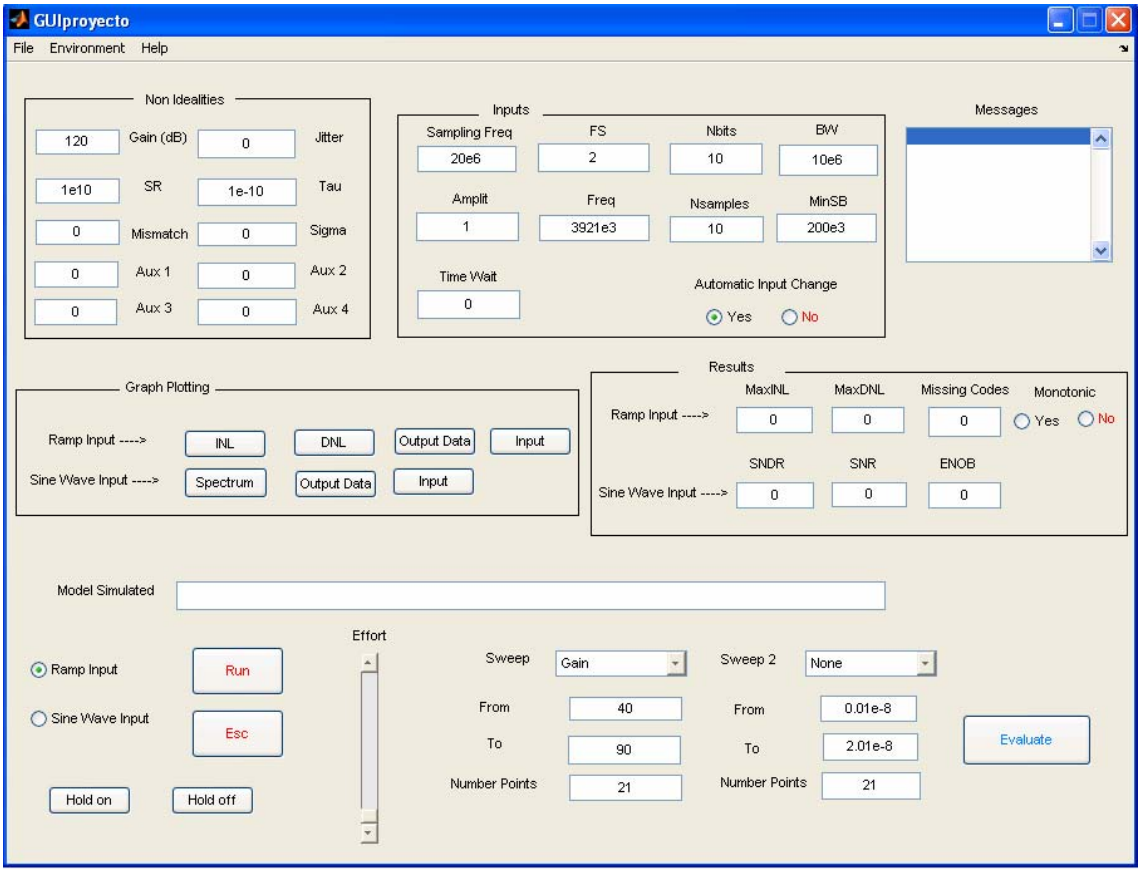


Fig A1.1.: GUI del simulador

2. Estructura de la GUI simconverter

En este apartado, vamos a explicar con detalle la estructura de la GUI de la aplicación implementada.

Como puede apreciarse en la Fig. A1.1., a grandes rasgos podemos distinguir 7 zonas o partes principales en la GUI, que mencionaremos aquí y describiremos con más detalle en los siguientes sub-apartados:

- **Zona de Menús**: Situada en la parte superior de la ventana, contiene 3 menús: File, Environment y Help.
- **Zona de Non Idealities**: Debajo de los menús a la izquierda, contiene los valores de las no idealidades para los estudios simples.
- **Zona de Inputs**: Debajo de los menús en el centro, contiene los valores de las entradas, son generales para todas las simulaciones. También presenta la opción de cambio de valores automáticos.
- **Zona de Messages**: Debajo de los menús a la derecha, contiene los mensajes de progreso en las simulaciones que genera la aplicación.
- **Zona de Graph Plotting**: Debajo de la Zona de Non Idealities, contiene las posibles gráficas que podemos pedirle al programa que dibuje.
- **Zona de Results**: Debajo de la Zona de Inputs, contiene los valores de salida que genera el programa.
- **Zona de Simulación**: Situada en la parte inferior de la ventana, contiene todas las opciones de simulación, tanto los estudios simples como los barridos.

2.1. Zona de Menús

La Zona de Menús, como puede verse en la figura A1.1., está compuesta por 3 menús diferentes: File, Environment y Help. Veremos con detalle cada uno de estos menús en los siguientes sub-apartados.

2.1.1. Menú File

El menú File, como su nombre indica, permitirá el tratamiento de archivos por parte del programa. Podemos apreciar las opciones que nos proporciona el menú File en la figura A1.2:

- **New:** Crea un archivo nuevo.
- **Open:** Abre un archivo existente.

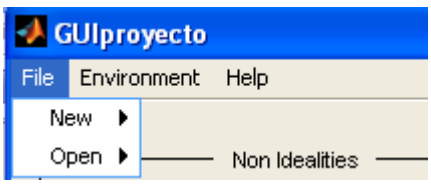


Fig A1.2.: Menu File

Vemos estas opciones con más detalle a continuación.

2.1.1.1. Menú File Submenú New

El contenido del submenú New puede verse en la figura A1.3.

Fig. A1.3.: Menú File Submenú New

La única opción de este submenú es Model. Al seleccionarla, se producen 2 efectos:

- Se abre el archivo **current.mdl** del directorio de la aplicación. Dicho archivo es el molde que usaremos para crear nuestros modelos de simulación, porque sólo contiene un generador de señal y un Scope para visualizar, quedando el resto vacío para que el usuario introduzca ahí el modelo que desea simular. El aspecto de este archivo puede verse en la figura A1.4.

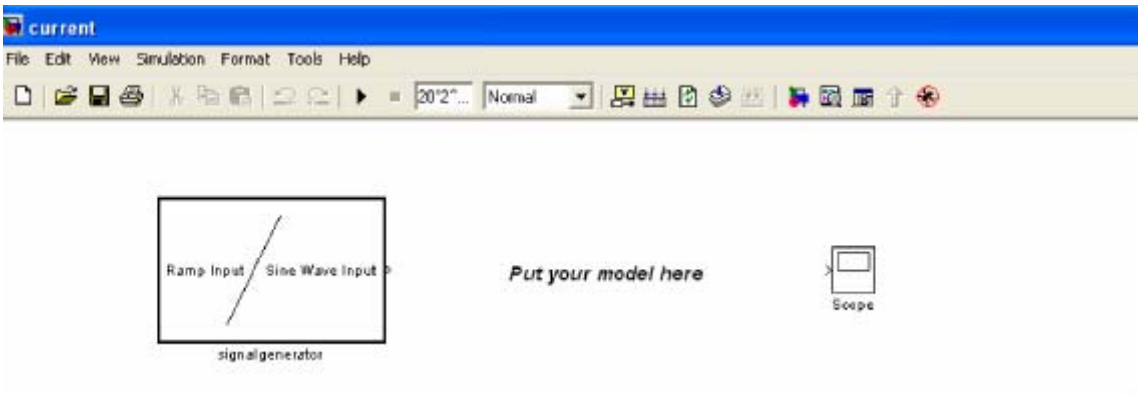


Fig. A1.4.: Archivo current.mdl

- Aparece en el cuadro de texto Model Simulated de la Zona de Simulación la ruta completa del modelo, como puede apreciarse en la figura A1.5.

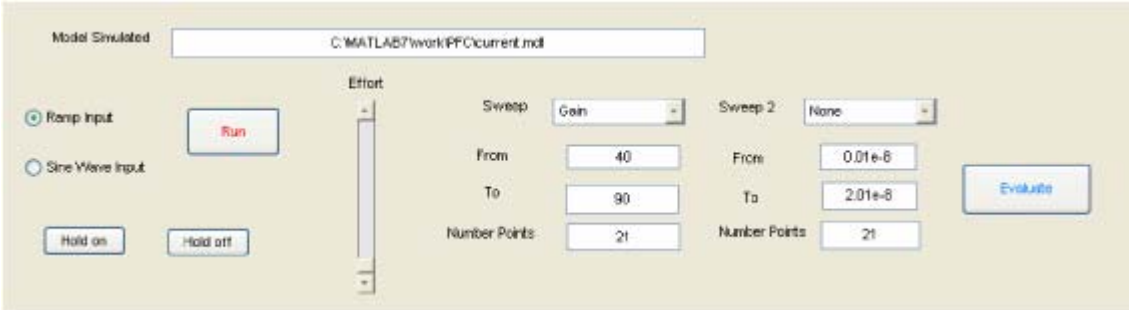


Fig. A1.5.: Zona de Simulación tras elegir File->New->Model

2.1.1.2. Menú File Submenú Open

El contenido del submenú Open puede verse en la figura A1.6.

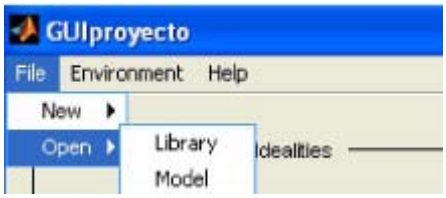


Fig. A1.6.: Menú File Submenú Open

Este submenú presenta 2 opciones distintas:

- **Library**: Al elegir esta opción, nos aparece la interfaz de la figura A1.7., en ella podremos seleccionar el tipo de bloque o ejemplo que queremos simular, ver una descripción del mismo y abrirlo. Esta interfaz la explicamos en el apartado 3, Estructura de la GUI library.

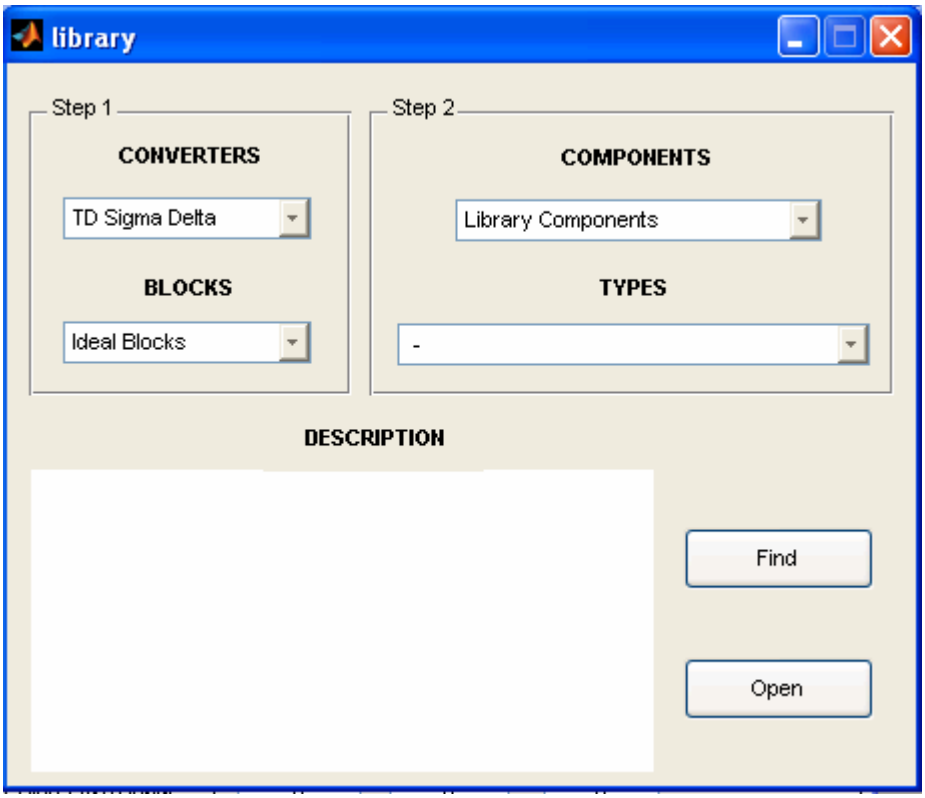


Fig. A1.7.: Interfaz library al elegir File->Open->Library

- **Model**: Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.8., en el que habrá que seleccionar el modelo que queremos abrir para simularlo. En este cuadro de diálogo, se muestran todos los archivos .mdl del directorio de trabajo. Cuando se elige uno de estos modelos, se abre y además su ruta completa aparece en el cuadro de texto Model Simulated de la Zona de

Simulación.

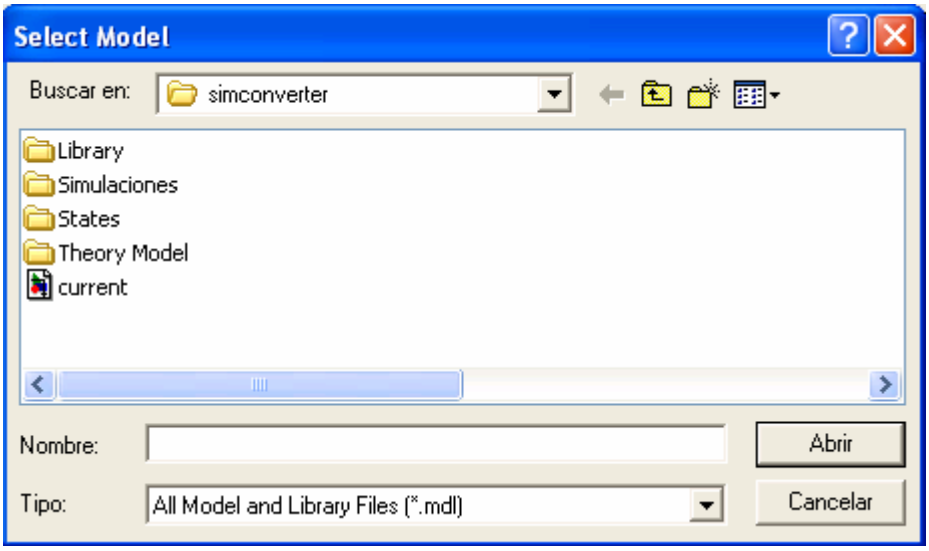


Fig. A1.8.: Cuadro de diálogo al elegir File->Open->Model

2.1.2. Menú Environment

El menú Environment permitirá al programa cargar y guardar estados (valores de las variables, barridos, etc.) del mismo, como veremos a continuación. Estos estados se guardarán en archivos .m, porque la carga del estado consistirá en la ejecución del script (archivo .m) que lo almacena.

Podemos apreciar las opciones que nos proporciona este menú en la figura A1.9.:

Fig. A1.9.: Menú Environment

Vemos estas opciones con más detalle a continuación:

- **Load**: Carga un estado existente. Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.10., en el que habrá que seleccionar el estado que queremos cargar. En este cuadro de diálogo, se muestran todos los archivos .m del subdirectorio States del directorio de trabajo.

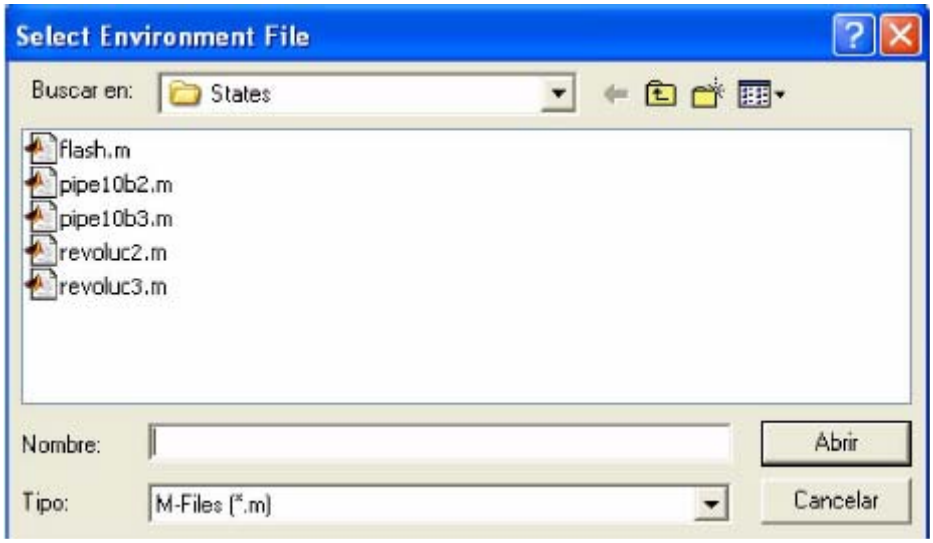


Fig. A1.10.: Cuadro de diálogo al elegir Environment->Load

Esta carga del estado sólo será necesario si cargas un modelo a partir de la opción de menú Model. Si cargas un ejemplo desde la interfaz library, se cargan los valores automáticamente, siendo estos pasos de carga y descarga innecesarios.

- **Save**: Guarda el estado actual. Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.11., en el que habrá que seleccionar el nombre del archivo (podemos elegir uno ya existente o especificar un nombre nuevo) en el que queremos almacenar el estado actual. En este cuadro de diálogo, se muestran todos los archivos .m del subdirectorio States del directorio de trabajo.

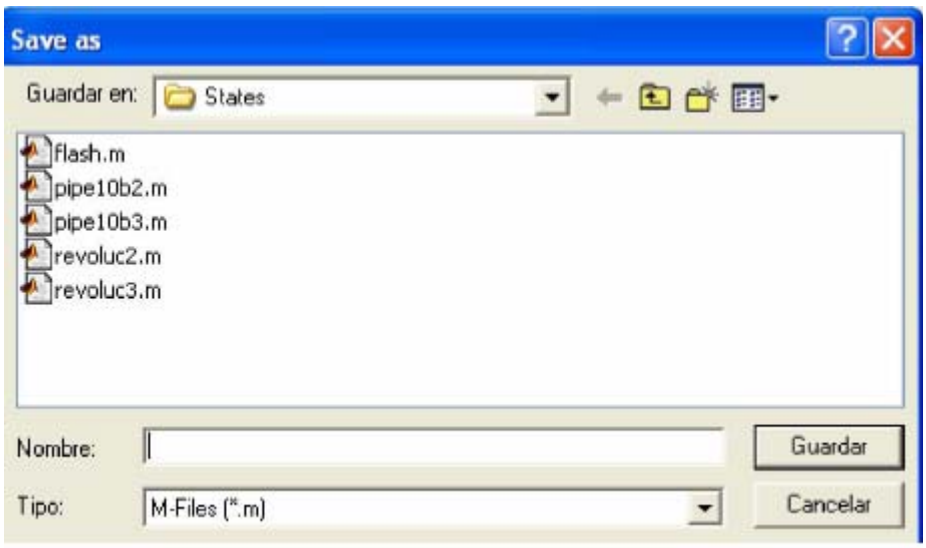


Fig. A1.11.: Cuadro de diálogo al elegir Environment->Save

2.1.3. Menú Help

El menú Help nos permitirá hacer 3 cosas distintas, que pueden apreciarse en la figura A1.12.: por un lado, la opción **"User Guide"** nos abre un pdf con el contenido de este manual para que el usuario reciba una ayuda on-line de forma rápida y sencilla; por otro lado, **"User Guide Advance"** nos abre un pdf con el contenido de un manual que presentaremos en el Anexo 2 para que un usuario avanzado con conocimientos de programación en Matlab pueda introducir nuevos modelos a la interfaz; y por otro lado, la opción **"About"** nos presenta información de la versión del simulador ante la que se encuentra el usuario.

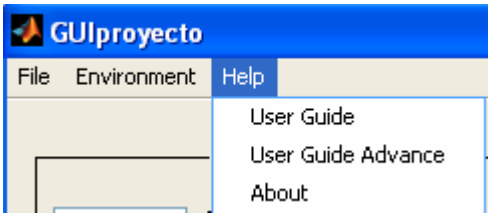


Fig. A1.12.: Menú Help

2.2. Zona de Non Idealities

La Zona de Non Idealities está compuesta por una serie de cuadros de texto en los que aparecen los valores de las no idealidades que pueden ser consideradas en el modelo a simular, y cuyo valor puede ser modificado editando dichos cuadros de texto. La estructura de esta zona puede verse en la Fig. A1.13.

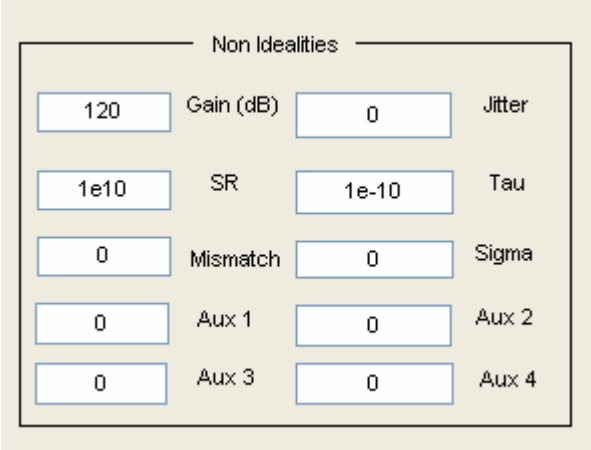


Fig. A1.13.: Zona de Non Idealities

Veamos a qué variable (no idealidad) representa cada cuadro de texto:

- **Gain**: Representa la ganancia finita del amplificador operacional en dB. La variable correspondiente es AodB. A partir de ella se genera Ao, que es la ganacia en unidades naturales.

- **Jitter**: Representa el jitter en el tiempo de muestreo. La variable

correspondiente es sigmajitter.

- **SR**: Representa el slew rate del amplificador operacional. La variable correspondiente es SR.

- **Tau**: Representa la constante de tiempo en segundos. La variable correspondiente es Tau.

- **Mismatch**: Representa el desapareamiento entre las capacidades en %. La variable correspondiente es Eps.

- **Sigma**: Representa la desviación típica del offset en los DACs. La variable correspondiente es sigma, de la cual se calculan (como números aleatorios de media 0 y desviación típica sigma) dos variables más: el offset para la tensión de referencia positiva y negativa (VoffsetVrefpn), y el offset para la tensión de referencia nula (Voffset0).

- **Aux 1, Aux 2, Aux 3, Aux 4**: Variables genéricas que representan otras no idealidades distintas de las ya reseñadas que podríamos incluir en el modelo a simular. Así, en el modelo del sigma delta de segundo orden, paso de baja y lazo simple, se utiliza Aux1 para la capacidad de muestreo, Aux2 pata la tensión rms de los amplificadores operacionales y Aux3 para la saturación de tensión.

2.3. Zona de Inputs

La Zona de Inputs está compuesta por una serie de cuadros de texto, en los que aparecen los valores de las variables de entrada y cuyo valor puede ser modificado editando dichos cuadros de texto, y dos radiobuttons que permiten elegir si se desea que al cambiar ciertos valores puedan modificarse otros de forma automática. La estructura de esta zona puede verse en la Fig. A1.14.

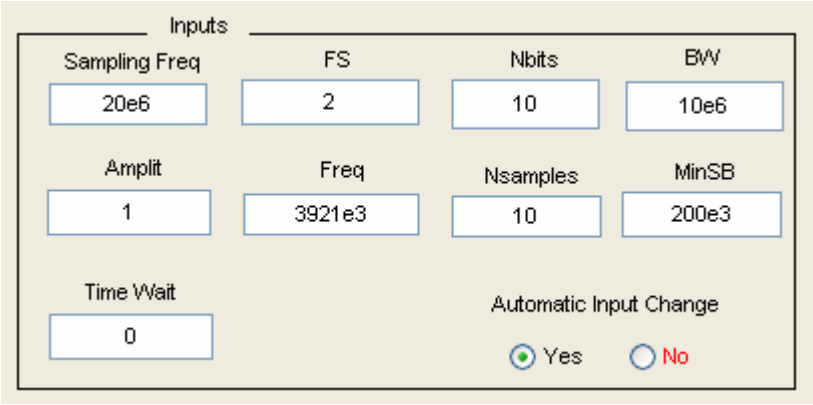


Fig. A1.14.: Zona de Inputs

Veamos a qué variable de entrada representa cada cuadro de texto:

- **Sampling Freq**: Representa la frecuencia de muestreo en hercios. La variable correspondiente será Ts, que representará al tiempo de muestreo (por tanto, será la inversa de la frecuencia de muestreo). Al modificarlo, si está activada la opción de cambio automático de variables, se modifican los cuadros de texto BW, Freq y MinSB (y sus variables correspondientes), con las siguientes ecuaciones:

$$BW = \frac{1}{2Ts}$$

$$Freq = \frac{1}{5.1Ts}$$

$$MinSB = \frac{1}{400Ts}$$

Por último, señalar que, al modificar el tiempo de muestreo, estaremos cambiando también el tiempo de simulación en caso de entrada rampa, ya que éste es de $4 \cdot Effort \cdot 2^N \cdot Ts \cdot \left(1 - \frac{2^{-N}}{4}\right)$ siendo N el número de bits esperados, y Effort una variable que representa el "esfuerzo" en la simulación (influye en el tiempo de la simulación, y por tanto en la rapidez de realización de los cálculos por parte del programa).

- **FS**: Representa el fondo de escala en voltios de los ADCs y DACs de cada etapa. También va a influir en el rango de la señal de entrada rampa, que va de (-FS/2) a FS/2. La variable correspondiente será FS. Al modificar FS, si está activo el cambio automático, se modifica el cuadro de texto Amplit (y la variable correspondiente) con la siguiente ecuación:

$$Amplit = \frac{FS}{2}$$

- **Nbits**: Representa el número de bits esperados del convertidor. La variable correspondiente será N. Al modificar esta variable, cambiaremos el tiempo de simulación $4 \cdot Effort \cdot 2^N \cdot Ts \cdot \left(1 - \frac{2^{-N}}{4}\right)$ en caso de entrada rampa.

- **BW:** Representa el ancho de banda en hercios que tomaremos para la señal en caso de entrada seno. La variable correspondiente será bandwidth.

- **Amplit**: Representa la amplitud en voltios de la señal de entrada seno. La variable correspondiente será A.

- **Freq**: Representa la frecuencia en hercios de la señal de entrada seno. Su variable será fsin. Al modificar la frecuencia de la señal de entrada seno,

estaremos cambiando también el tiempo de simulación en caso de entrada seno, ya que éste es de $\frac{10 \cdot (Effort^2 + 1)}{Freq}$.

- **Nsamples**: Representa el número de muestras alrededor de la frecuencia de la señal que tomaremos para calcular la potencia de dicha señal al hacer cálculos de SNR y SNDR. A mayor número de muestras menor será la relación SNR, ya que mayor es al cálculo del ruido. La variable correspondiente será nsamples.

- **MinSB**: Representa la banda mínima de señal en hercios, es decir, el valor inferior de frecuencia a partir del cual calcularemos el ruido (hay que coger un valor inferior para que no aparezca ruido flicker). La variable correspondiente será minsignalband.

- **Timewait**: Representa el tiempo de espera antes de comenzar la simulación junto con la señal de entrada. Se utiliza en modelos donde hay que realizar por ejemplo una calibración inicial y la simulación debe tomar la señal de entrada pasado un tiempo inicial. Está variable afecta al tiempo de simulación de la siguiente forma:

$$\text{Entrada rampa: } wait + 4 \cdot 2^N \cdot Ts \cdot \left(1 - \frac{2^{-N}}{4}\right)$$

$$\text{Entrada seno: } wait + 10 \cdot \left(\frac{Effort^2 + 1}{f \sin}\right)$$

- **Automatic Input Change Yes/No**: Son 2 radiobuttons que indican, al estar uno de ellos encendido, si se efectuarán modificaciones automáticas en el valor de ciertas variables de entrada al cambiar el valor de Sampling Freq y de FS. Aparece por defecto seleccionado el valor Yes. La variable correspondiente será Auto.

2.4. Zona de Messages

La Zona de Messages está compuesta por una list box en la cual aparecen los mensajes que genera la aplicación (simulación iniciada, simulación concluida, barrido iniciado, barrido en proceso, barrido finalizado). La estructura de esta zona puede verse en la Fig. A1.15. (esta zona aparece vacía al arrancar el programa, al ejecutar simulaciones simples o barridos nos irán apareciendo los mensajes mencionados en cada caso).

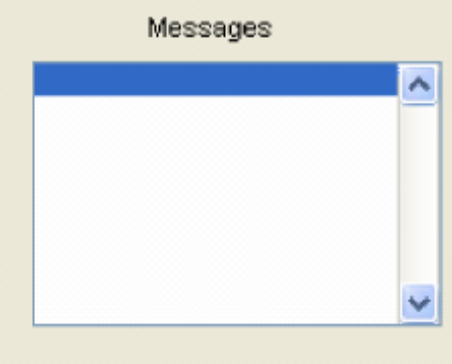


Fig. A1.15.: Zona de Messages

2.5. Zona de Graph Plotting

La Zona de Graph Plotting está compuesta por una serie de botones que nos servirán para controlar la gráficas que se quiere visualizar. La estructura de esta zona puede verse en la Fig. A1.16.

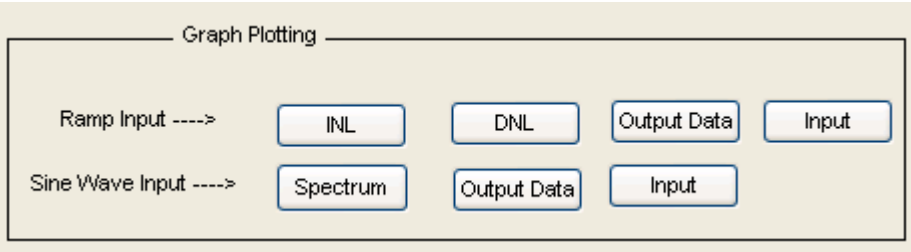


Fig. A1.16.: Zona de Graph Plotting

La orden que realiza cada uno de los botones es la siguiente:

- **INL**: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada rampa. En ese caso, mostrará una gráfica de INL (error de no linealidad integral, del inglés "Integral Nonlinearity") frente a la señal de entrada en voltios. Cabe destacar que la gráfica de INL que se muestra no tiene en cuenta los códigos perdidos (sí se tiene en cuenta en el cuadro MaxINL de la Zona de Results).

- **DNL**: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada rampa. En ese caso, mostrará una gráfica de DNL (error de no linealidad diferencial, del inglés "Differential Nonlinearity") frente a la señal de entrada en voltios.

- **Output Data**: Aparecen 2 botones denominados Output Data, tendremos que usar el correspondiente a la entrada elegida. En el caso de entrada rampa, mostrará una gráfica de la señal de salida del convertidor (magnitud adimensional que va de 0 a 2^N -1) frente a la señal de entrada en voltios. En el caso de entrada seno, mostrará una gráfica de la señal de salida frente al tiempo en segundos.

- **Input**: Aparecen 2 botones denominados Input, tendremos que usar el correspondiente a la entrada elegida. En ambos casos, se representa la señal de entrada en voltios frente al tiempo en segundos.

- **Spectrum**: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada seno. En ese caso, mostrará una gráfica de la PSD en dB frente a la frecuencia en Hz en escala logarítmica.

2.6. Zona de Results

La Zona de Results tendrá utilidad en las simulaciones simples. Está compuesta por una serie de cuadros de texto y 2 radiobuttons, en los que aparecen diversos parámetros de interés para analizar cómo de bueno es el convertidor que estamos simulando y cómo le afectan las no idealidades que estemos considerando. La estructura de esta zona puede verse en la Fig. A1.17.

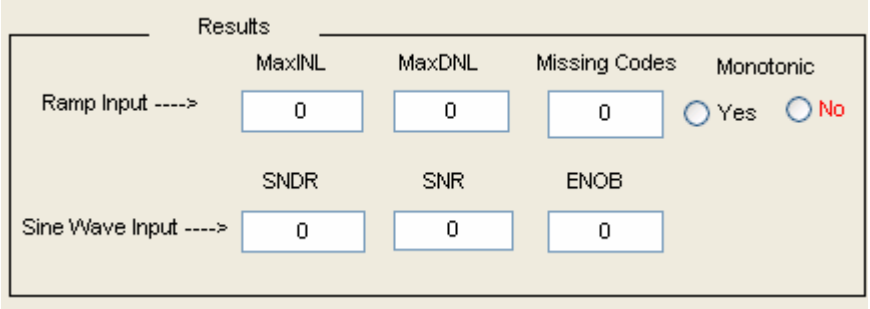


Fig. A1.17.: Zona de Results

Vemos a continuación los parámetros a los que hacíamos referencia anteriormente:

- **MaxINL**: Sólo tiene sentido en una simulación simple con entrada rampa. En este caso, representa la INL máxima del convertidor (incluidos los códigos perdidos).

- **MaxDNL**: Sólo tiene sentido con entrada rampa, y representa la DNL máxima del convertidor.

- **Missing Codes**: Sólo tiene sentido con entrada rampa, y representa los códigos perdidos del convertidor.

- **Monotonic Yes/No**: Sólo tienen sentido con entrada rampa. Son 2 radiobuttons que indican, al estar uno de ellos encendido, si la salida es monotónica o no (aparecen por defecto apagados al arrancar el programa, pero al realizar una simulación simple con entrada rampa ya se enciende uno de los 2).

- **SNDR**: Sólo tiene sentido en una simulación simple con entrada seno. En este

caso, representa la relación señal a ruido y distorsión de la señal de salida en dB.

- **SNR**: Sólo tiene sentido con entrada seno, y representa la relación señal a ruido de la señal de salida en dB.

- **ENOB**: Sólo tiene sentido con entrada seno, y representa el número efectivo de bits del convertidor.

2.7. Zona de Simulations

La Zona de Simulación es el corazón del programa, porque desde aquí van a darse las órdenes generales de simulación para realizar los distintos análisis. La estructura de esta zona puede verse en la Fig A1.18. A grandes rasgos (describiremos cada elemento con más detalle a continuación).

Está compuesta por cuadros de texto (modelo a simular, y parámetros de los barridos), radiobuttons (selección de la entrada), pushbuttons (hold, simulación simple, escape y barridos), pop-up menus (selección de barridos), y una slider o barra deslizante (selección de "esfuerzo" en los cálculos de la simulación).

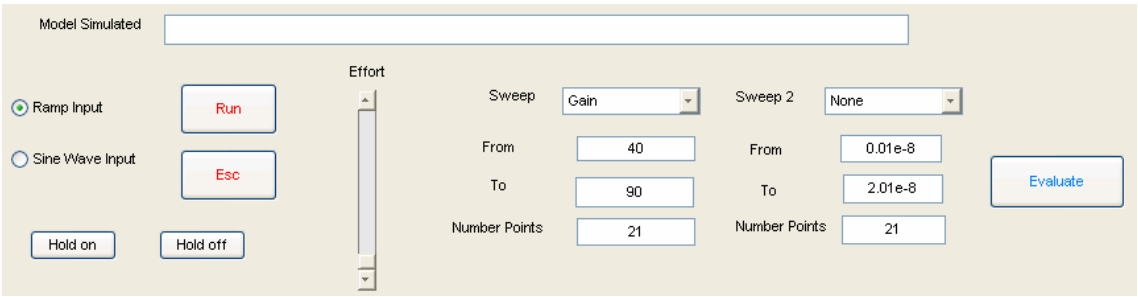


Figura A1.18.: Zona de Simulations

Los elementos de la zona de simulación son:

- **Model Simulated**: Cuadro de texto en el que aparece el nombre del modelo que se está simulando actualmente (al arrancar el programa, aparece en blanco, lo cual quiere decir que el usuario lo primero que deberá hacer será cargar un modelo usando el menú File->Open->Model). La variable en la que se almacenará el nombre del modelo simulado será modelsimulated.

- **Ramp Input / Sine Wave Input**: Son 2 radiobuttons que permiten la selección de la señal de entrada (al arrancar el programa, aparece seleccionada entrada rampa). La variable correspondiente será Select.

- **Run**: Botón que ordena la realización de un análisis simple del modelo indicado en Model Simulated, con el tipo de entrada seleccionado por los radiobuttons Ramp Input y Sine Wave Input.

- **Esc**: Botón que ordena la detención de la simulación de un análisis del modelo indicado por Model Simulated.

- **Hold on**: Botón que, al pulsarse, hace que se dibujen todas las gráficas en los mismos ejes hasta que se pulse Hold off (al arrancar el programa, cada gráfica se dibujará en ejes diferentes).

- **Hold off**: Botón que, al pulsarse, hace que se dibuje cada gráfica en un eje diferente hasta que se pulse Hold on.

- **Effort**: Barra deslizante que controla el "esfuerzo" en los cálculos, esto es, el tiempo de simulación, lo cual nos va a repercutir en la rapidez con la que efectuaremos nuestras simulaciones (a mayor tiempo de simulación, más lentas serán nuestras simulaciones).

- **Sweep**: Menú desplegable que se utiliza para realizar barridos. Si en el menú desplegable Sweep 2 se elige None, se efectuará un barrido simple, de lo contrario se efectúa un barrido doble (hemos de tener en cuenta que no debemos elegir en Sweep 2 el mismo valor que en Sweep, porque no tiene sentido realizar un barrido doble de la misma variable). Las opciones de Sweep se muestran en la Fig. A1.19 (al arrancar el programa, aparece por defecto seleccionado Gain).

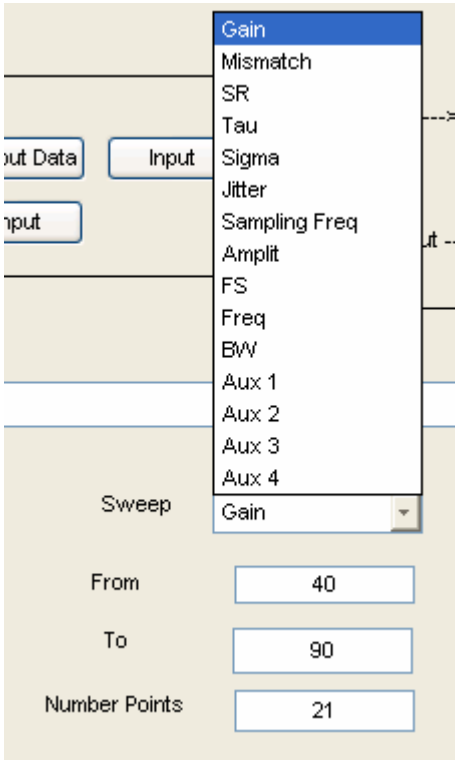


Figura A1.19.: Opciones de Sweep

Cabe destacar también que, según el barrido elegido, cambian automáticamente los parámetros del barrido a unos valores estándar para cada caso.

La variable correspondiente será val.

- **Sweep 2**: Menú desplegable que nos ofrece la posibilidad de realizar barridos dobles (en caso de elegir None se realizaría un barrido simple). Las opciones de Sweep 2 se muestran en la Fig. A1.20. (al arrancar el programa, aparece por defecto seleccionado None). Cabe destacar también que, según el barrido elegido, cambian automáticamente los parámetros del barrido a unos valores estándar para cada caso. La variable correspondiente será val2.

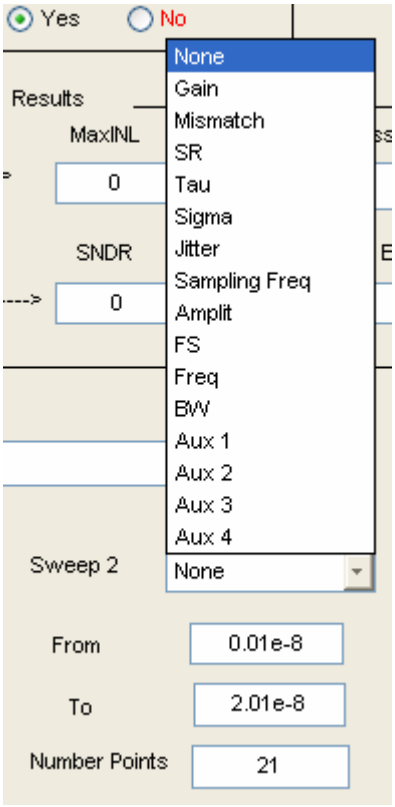


Fig. A1.20.: Opciones de Sweep 2

- **From**: Cuadro de texto que aparece con Sweep (variable Start) y Sweep 2 (variable Start2). En cada caso, indica el valor inicial de la variable para el barrido.

- **To**: Cuadro de texto que aparece con Sweep (variable Stop) y Sweep 2 (variable Stop2) . En cada caso, indica el valor final de la variable para el barrido.

- **Number Points**: Cuadro de texto que aparece con Sweep (variable Points) y Sweep 2 (variable Points2). En cada caso, indica el número de puntos (valores distintos de la variable) para el barrido.

- **Evaluate**: Botón que ordena la realización de un barrido (del tipo seleccionado por Sweep y Sweep 2) del modelo indicado en Model Simulated, con el tipo de entrada seleccionado por los radiobuttons Ramp Input y Sine Wave Input. Al concluir el barrido, dependiendo de la entrada y de si es

barrido simple o doble, aparece una determinada gráfica: con entrada rampa, si es barrido simple la gráfica es de INL máxima (incluyendo los códigos perdidos) frente a la variable que varía, pero si es barrido doble la gráfica será de INL máxima frente a las 2 variables que varían (en 3 dimensiones); con entrada seno, las gráficas son análogas pero con SNDR en dB. Se han implementado todos los barridos simples, y la relación de barridos dobles implementados puede verse en las tablas de las Fig. A1.21. y A1.22. (en las filas se han representado los valores de Sweep, en las columnas los valores de Sweep 2):

	Gain	Mis match	SR	Tau	Sigma	Jitter	Ts	Amplit
Gain	NO	*SI*	NO	NO	*SI*	*SI*	*SI*	*SI*
Mis match	NO	NO	NO	NO	*SI*	*SI*	*SI*	*SI*
SR	NO	NO	NO	*SI*	*SI*	*SI*	NO	NO
Tau	NO	NO	NO	NO	*SI*	*SI*	NO	NO
Sigma	NO	NO	NO	NO	NO	*SI*	NO	NO
Jitter	NO	NO	NO	NO	NO	NO	NO	NO
Sam pling Freq	NO	NO	NO	NO	NO	NO	NO	NO
Amplit	NO	NO	NO	NO	NO	NO	NO	NO
FS	NO	NO	NO	NO	NO	NO	NO	NO
Freq	NO	NO	NO	NO	NO	NO	NO	NO
BW	NO	NO	NO	NO	NO	NO	NO	NO
Aux 1	NO	NO	NO	NO	NO	NO	NO	NO
Aux 2	NO	NO	NO	NO	NO	NO	NO	NO
Aux 3	NO	NO	NO	NO	NO	NO	NO	NO
Aux 4	NO	NO	NO	NO	NO	NO	NO	NO

Fig. A1.21.: Barridos dobles (I)

	Sam pling Freq	Freq	BW	Aux 1	Aux 2	Aux 3	Aux 4
Gain	*SI*	*SI*	NO	NO	NO	NO	NO
Mis match	*SI*	*SI*	NO	NO	NO	NO	NO
SR	NO	NO	NO	NO	NO	NO	NO
Tau	NO	NO	NO	NO	NO	NO	NO
Sigma	NO	NO	NO	NO	NO	NO	NO
Jitter	NO	NO	NO	NO	NO	NO	NO
Sam pling Freq	NO	NO	NO	NO	NO	NO	NO
Amplit	NO	NO	NO	NO	NO	NO	NO
FS	NO	NO	NO	NO	NO	NO	NO
Freq	NO	NO	NO	NO	NO	NO	NO
BW	NO	NO	NO	NO	NO	NO	NO
Aux 1	NO	NO	NO	NO	*SI*	*SI*	*SI*
Aux 2	NO	NO	NO	NO	NO	*SI*	*SI*
Aux 3	NO	NO	NO	NO	NO	NO	*SI*
Aux 4	NO	NO	NO	NO	NO	NO	NO

Fig. A1.22.: Barridos dobles (II)

3. Estructura de la GUI library

Tras pulsar en la interfaz simconverter File → Open → Library, nos aparece la

interfaz library que mostramos en la Fig.A1.23:

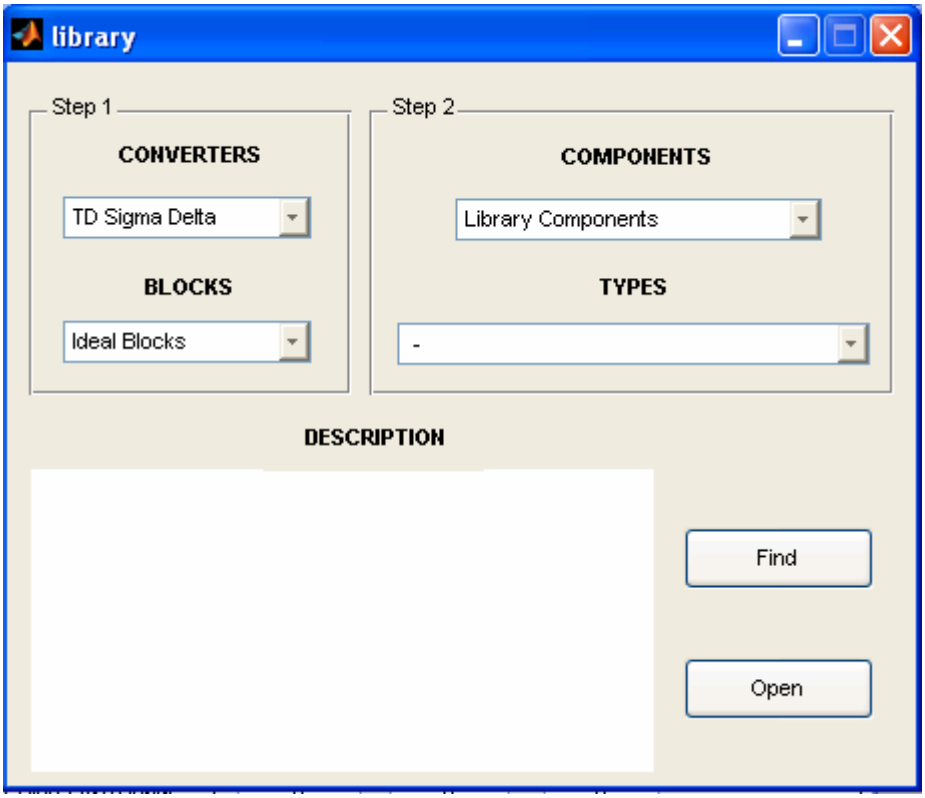


Fig. A1.23.: Interfaz library

Esta interfaz nos va a permitir de forma dinámica tener acceso a todos los bloques y ejemplos de los que está compuesta la librería. Su apariencia es simple, lo que la hace más eficaz para la búsqueda. De forma resumida su utilización se basa en: la elección del modelo por medio de cuatro pasos correspondientes a cada uno de los pop-menu (Converters, Blocks, Components y Types). Posteriormente podremos visualizar una descripción del modelo elegido por medio del text- box Description y del push bottom Find, y por último, para abrir el fichero correspondiente de Simulink, pulsaremos el push-bottom Open.

3.1. Elección del modelo

Para la elección del modelo nos centramos en la parte superior de la interfaz. Describamos cada uno de los pop-menu para ver la funcionalidad que desempeñan en la búsqueda del modelo:

3.1.1. Converters

Tras desplegar el menú podemos elegir entre los diferentes tipos de

convertidores que dispone la librería. En este caso los visualizamos en la Fig. A1.24:

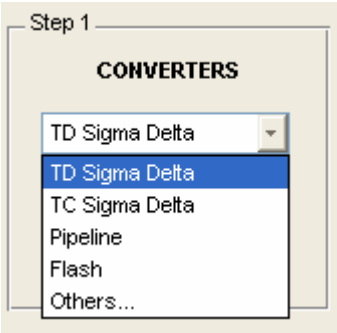


Fig. A1.24.: Converters

Por defecto aparece la elección de convertidores Sigma Delta de Tiempo Discreto. La variable en el simulador correspondiente es m1. Dependiendo del valor de este pop-menu, variará las posibles elecciones en los pop-menu Components y Types de forma dinámica.

3.1.2. Blocks

Una vez seleccionado el tipo de convertidor, podemos elegir entre la características del modelo que nos interesaría atendiendo a su idealidad o en caso de querer simular un ejemplo poder elegirlo.

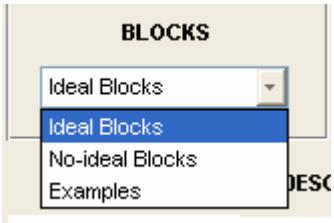


Fig. A1.24.: Blocks

Por defecto aparece la elección de convertidores Bloques Ideales. La variable en el simulador correspondiente es m2. Este pop-menu es el mismo para todo tipo de convertidores, ya que lo hemos organizado los bloques de los convertidores atendiendo a esta clasificación: bloques ideales, bloques no ideales y ejemplos, la cual es general para cada clase de convertidor.

Dependiendo de la elección de este menú, y junto con la elección de menú de convertidores, se modificarán dinámicamente las elecciones de los siguientes menús.

3.1.3. Components

Este menú posee una multitud de opciones, ya que como venimos explicando, sus valores dependerán de las elecciones anteriores. La variable correspondiente es m4. Mostremos en las figuras siguientes algunos ejemplos de elección:

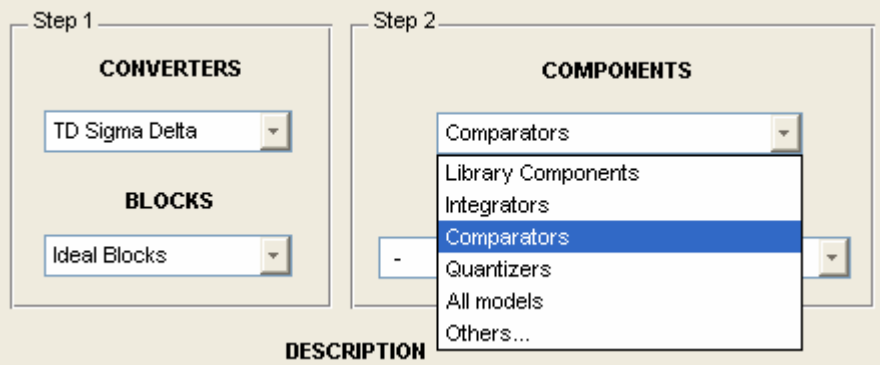


Fig. A1.25.: TD Sigma Delta → Ideal Blocks → Comparators

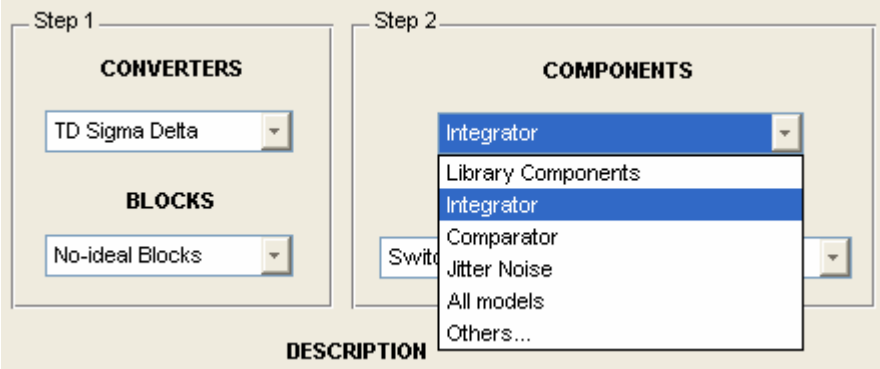


Fig. A1.26.: TD Sigma Delta → No-ideal Blocks → Integrator

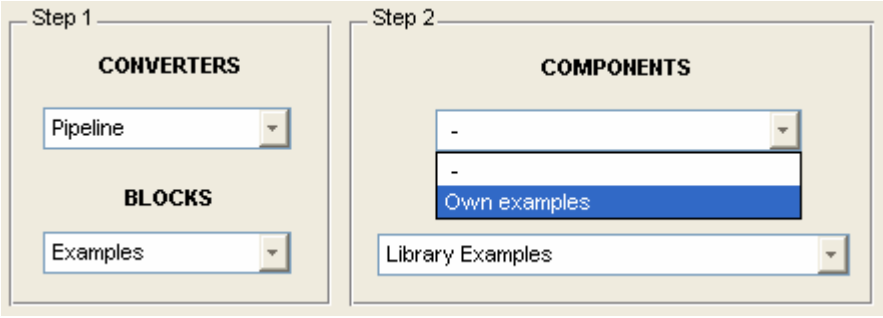


Fig. A1.27.: Pipeline → Examples → Own examples

Para cada tipo de convertidor y tipo de bloque, este menú nos ofrece una amplia variedad de todos los componentes de los que está compuesto su estructura. Según esto, un convertidor sigma delta está formado por integradores, comparadores, cuantizadores...

Este menú, recordamos que es diferente para cada tipo de elección, pero posee unas opciones que son comunes a todos.

- **Library components**, en la cual tenemos acceso al subdirectorio con todos los archivos para la elección tomada.

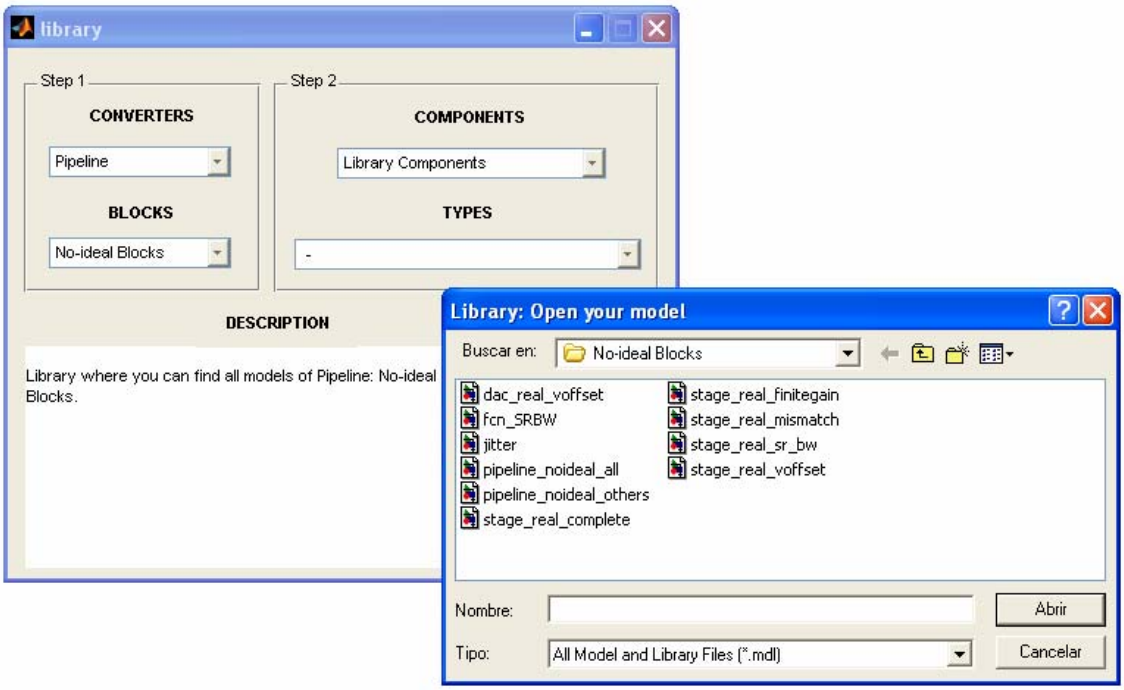


Fig. A1.28.: Pipeline → No-ideal Blocks → Library components

- **All models**, fichero Simulink donde se encuentran todos los bloques para la opción de Convertidor y Bloque tomada.

- **Others...,** fichero Simulink donde cualquier usuario puede introducir otros bloques que haya diseñado.

- **-,** aparece en el caso de seleccionar Examples en el menú de Boques. Indica que lo que va a abrir el usuario es un ejemplo del tipo de convertidor indicado en el menú Converters. Para ello debe seleccionarlo en el menú Types que explicaremos en el siguiente apartado.

- **Own examples,** sólo a parece en el caso de haber seleccionado Examples en el menú de Bloques. En este caso se abre la posibilidad de que un usuario avanzado pueda añadir ejemplos a la librería tendiendo las mismas condiciones que los ya incluidos, es decir, poder abrirlos y cargarlos en la otra interfaz, cargar directamente los valores de las variables... La explicación de cómo hacerlo, se deja para otro manual más avanzado dónde se exponga los pasos que hay que hacer para modificar el código del programa.

3.1.4. Types

Este es el menú más dinámico, ya que varía dependiendo de las elecciones de las elecciones de los tres menús anteriores. Este menú se encarga de seleccionar el

bloque concreto de la elección o del ejemplo[18]. La variable correspondiente a este modelo en el código es m4. Veamos algunas opciones a modo de ejemplo:

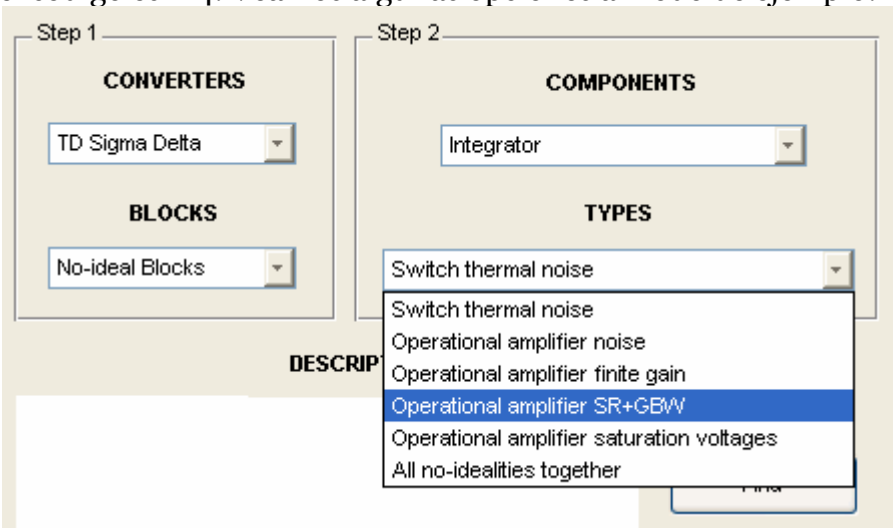


Fig. A1.29.: TD Sigma Delta → No-ideal Blocks → Integrator → Operacional amplifier SR+GBW

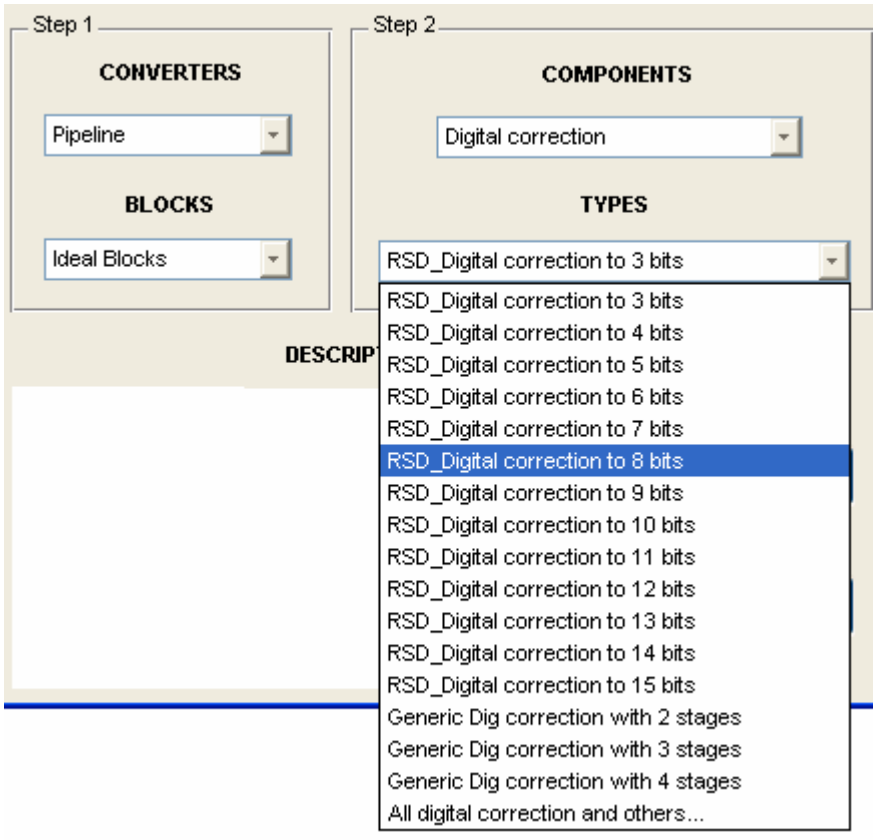


Fig. A1.30.: Pipeline → Ideal Blocks → Digital correction → RSD_Digital correction to 8 bits

[18] En caso de encontrarnos - significa que para ese modelo ha terminado su búsqueda en el paso anterior, es decir, que para el componente elegido tan sólo hay un archivo para abrir.

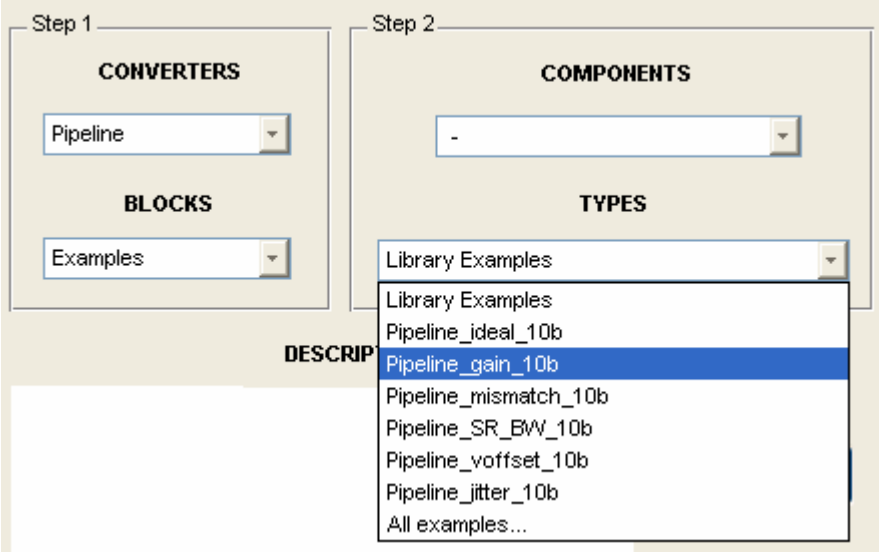


Fig. A1.31.: Pipeline → Examples → - → Pipeline_gain_10b

3.2. Descripción del modelo

Una vez elegido nuestro modelo, podemos optar antes de abrirlo, por leer una breve descripción del mismo y comprobar si este es el modelo que verdaderamente buscamos. Para ello tan sólo hay que pulsar el botón Find y nos aparecerá en Description un mensaje relacionado al modelo. La variable correspondiente es find.

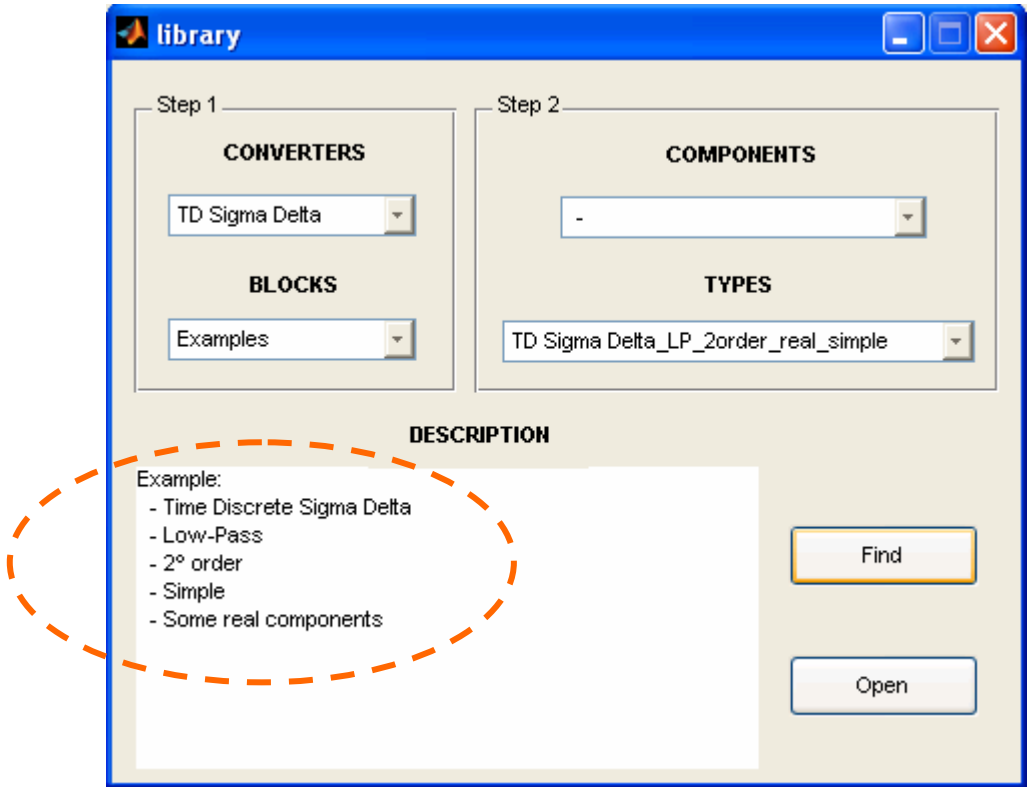


Fig. A1.32.: Description: TD Sigma Delta → Examples → - → LP_2order_real_simple

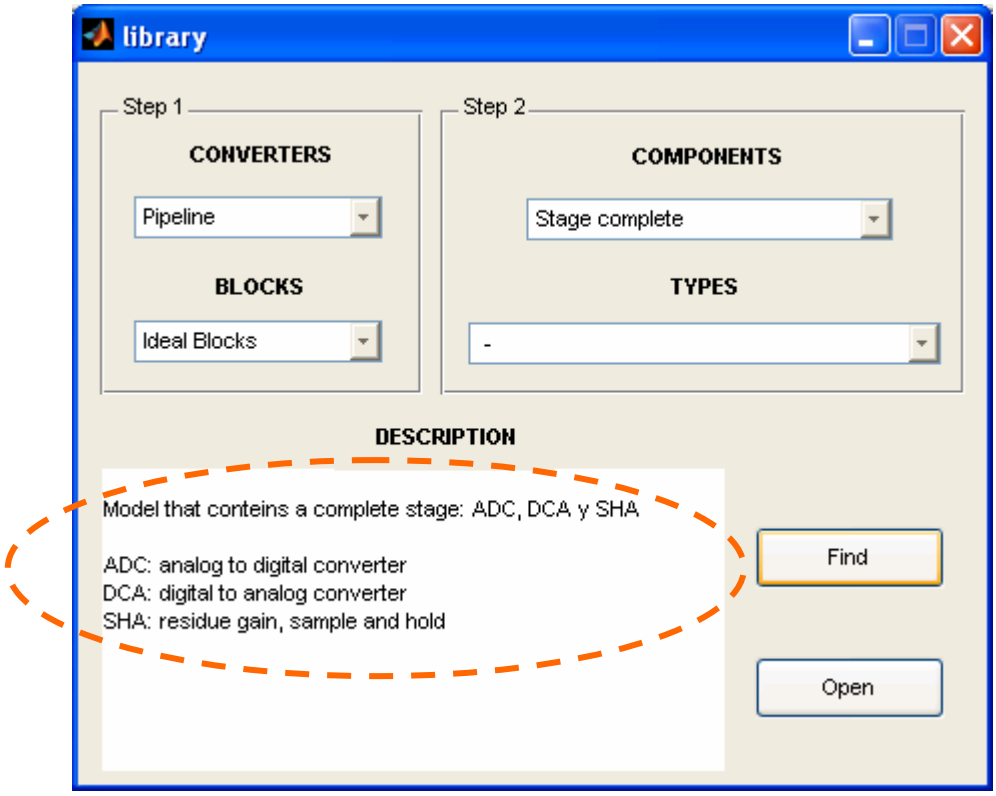


Fig. A1.32.: Description: Pipeline →Ideal Blocks → Stage complete → -

3.3. Apertura del modelo

Por último nos queda abrir nuestro modelo. Para ello basta con pulsar la tecla Open. Dependiendo de si lo que hemos abierto es bloque o ejemplo pasaran 2 cosas:

- **Bloque**: aparecerá una ventana de Simulink con el bloque elegido. Este bloque pondremos pulsar sobre el y obtendremos más información sobre él, como una descripción más especifica, las variables que necesitamos, el modelo utilizado para su diseño... Con este bloque ya podemos trabajar, por ejemplo para añadirlo a nuestro modelo en una nueva ventana.

- **Ejemplo**: aparecerá en una ventana Simulink dispuesto a ser simulado con la entrada y salida ya conectada. En este momento, y para estos casos, devolvemos el control[19] a la interfaz simconverter, donde ya aparecerá el Model Simulated la dirección y nombre de nuestro modelo, y en la Zona de Inputs y Non Idealities los valores necesarios para su simulación.

Veamos un ejemplo de lo que aparecería en caso de abrir un bloque y de abrir un ejemplo:

[19] Este paso del control de una interfaz a otra en transparente al usuario. Para ello se ha utilizado una estructura virtual de comunicación. En cualquier momento el usuario puede de nuevo acceder a la interfaz de librería abriéndola desde el menú con File → Open → Library.

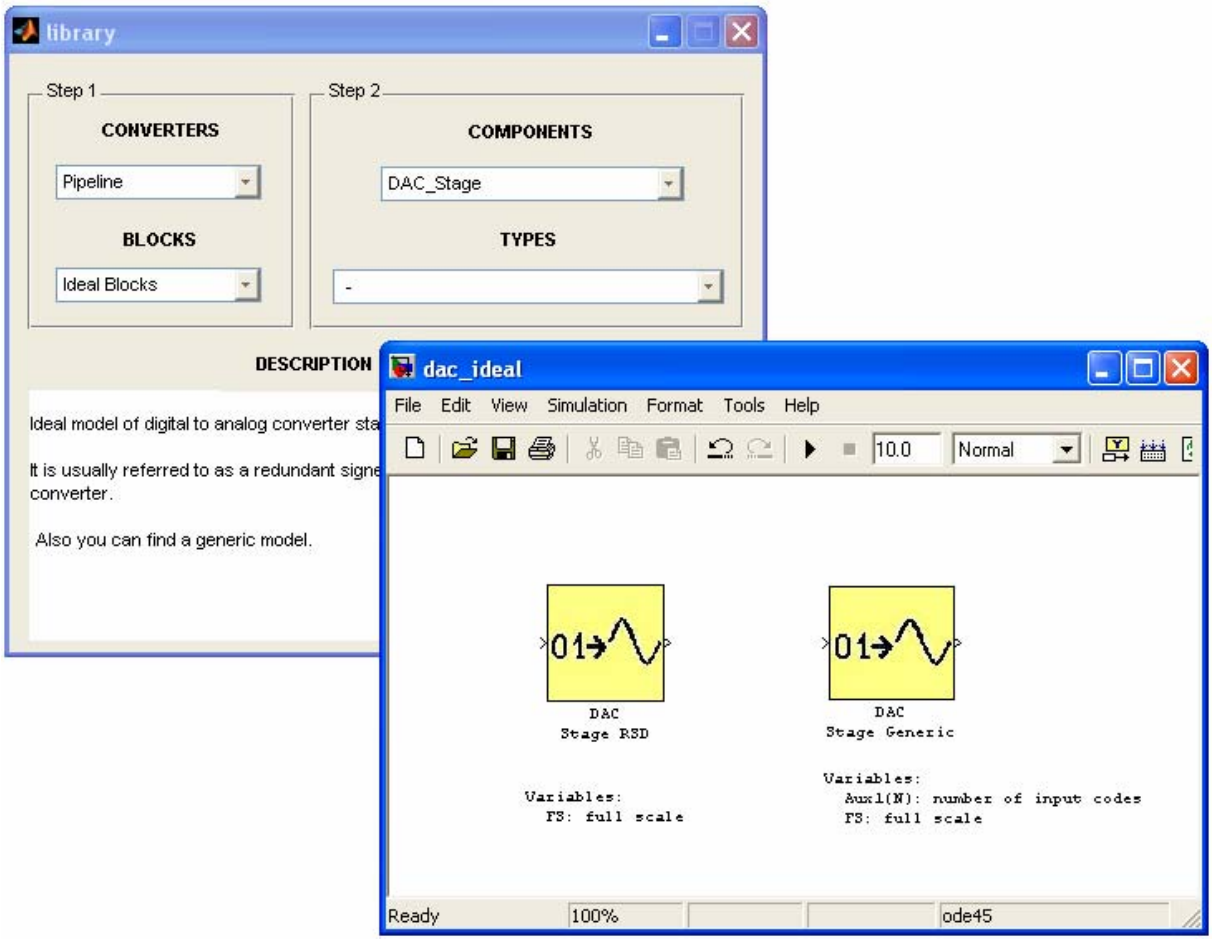


Fig. A1.33.: Open Bloque: Pipeline → Ideal Blocks → DAC_Stage → -

Y en caso de un ejemplo:

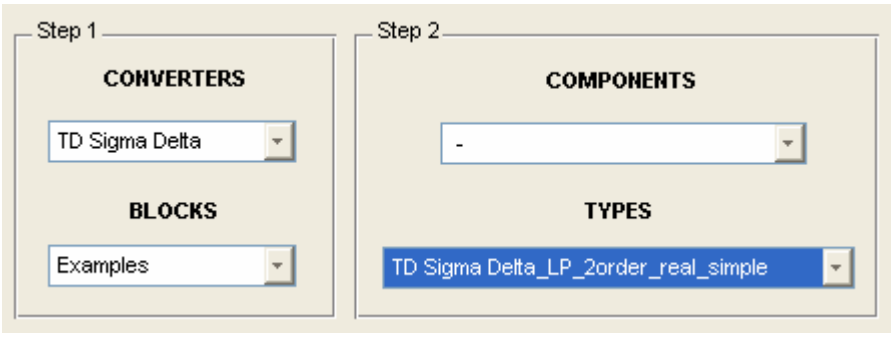


Fig. A1.34.: Open Example: TD Sigma Delta → Examples → - → LP_2order_real_simple

Al darle a Open, aparecerá el ejemplo junto con la interfaz simconverter como hemos explicado anteriormente:

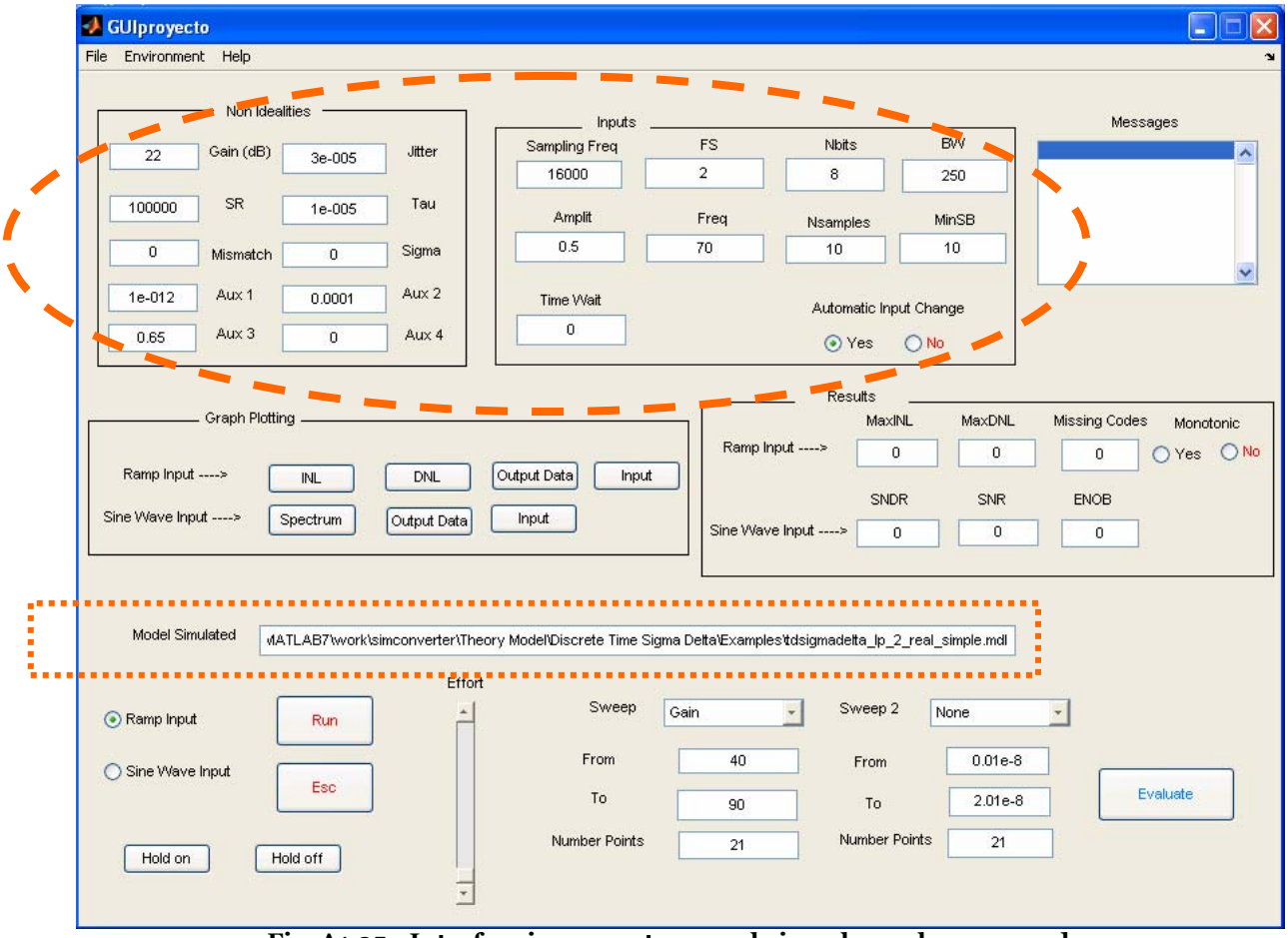


Fig. A1.35.: Interfaz simconverter con el ejemplo y valores cargados

4. Ejemplos

Para hacer unas demostraciones de los pasos seguidos en este manual, se han desarrollado unas demos que pueden accederse a ellas a partir del directorio del CD de instalación:

D:\simconverter\work\Demos\

O a través del directorio de trabajo una vez instalado el programa:

C:\Matlab7\work\simconverter\Demos\

ANEXO 2

Manual de Usuario Avanzado

El script principal library contiene el código[20] del todo el programa, el cual además contiene otros que se reparten las diversas funcionalidades de nuestra aplicación.

- **library**: función principal desde dónde se programa la funcionalidad de todos los objetos de la interfaz. Dicho archivo, denominado library.m, inicializa la GUI, contiene todas las callbacks (comandos que se ejecutan cuando un usuario hace clic sobre un componente de la GUI) de la GUI, y realiza todas las llamadas a funciones y scripts auxiliares necesarias para la implementación deseada en cada caso (entraremos en detalle en las funciones auxiliares a las que llama este script a continuación). Otra de sus funciones principales es dotar a la librería de la actualización dinámica de los menús.

- **var_global**: se encarga de la declaración de las variables globales del programa y de su inicialización.

- **description**: base de datos con todas las descripciones de los modelos. Se ha creado una estructura de forma que el acceso es directo. Cada bloque o ejemplo presenta una breve descripción de su funcionalidad y características.

- **archive**: función para distinguir entre la apertura de una ventana de librería, o la apertura de un modelo.

- **arch_library_comp**: apertura de la librería de componentes seleccionados. Programada para que abra el modelo una vez que lo selecciones y pulses la tecla Aceptar, o te devuelva el control a la interfaz library en caso de pulsar la tecla de Cancel.

- **arch_model_comp**: apertura del modelo seleccionado. Para ello se mueve en el árbol de directorios y abre el fichero que contiene el bloque deseado.

Pasos para introducir un bloque

1. En library.m introducir los diferentes tipos de valores que toman los menús.
2. Introducir la descripción en descrition.m
3. Distinguir tipo de apertura en archive.m
4. Colocar el nombre y directorio en arch_model_comp.m
5. Si es un ejemplo:
    a. colocar el valor de las variables en state.m
    b. actualizar el callback de Open

[20] El código desarrollado se puede consultar en el Anexo 4 del proyecto.

ANEXO 3

Código Matlab interfaz *simconverter*

Este apéndice contiene el listado del código Matlab comentado del programa, se separarán los trozos de código por ficheros:

- **GUIproyecto.m**

```
% Script that has got the code that controls the behaviour of the GUI

function varargout = GUIproyecto(varargin)
% GUIPROYECTO M-file for GUIproyecto.fig
%      GUIPROYECTO, by itself, creates a new GUIPROYECTO or raises the existing
%      singleton*.
%
%      H = GUIPROYECTO returns the handle to a new GUIPROYECTO or the handle to
%      the existing singleton*.
%
%      GUIPROYECTO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUIPROYECTO.M with the given input arguments.
%
%      GUIPROYECTO('Property','Value',...) creates a new GUIPROYECTO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUIproyecto_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUIproyecto_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 GUIproyecto

% Last Modified by GUIDE v2.5 05-Dec-2005 14:38:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUIproyecto_OpeningFcn, ...
                   'gui_OutputFcn',  @GUIproyecto_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 GUIproyecto is made visible.
function GUIproyecto_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 GUIproyecto (see VARARGIN)

fig = handles.Current;                  % Handle to the figure

GUIinit(handles);                       % Iniatilize values of global variables

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

% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = GUIproyecto_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;

% --- Initialize values of global variables
function GUIinit(handles)

global Ts;                          % Global variables to be initialized
global FS;
global A;
global fsin;
global N;
global sigma;
global sigmajitter;
global bandwidth;
global minsignalband;
global val;
global val2;
global Select;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global nsamples;
global Effort;
global Eps;
global SR;
global Tau;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global Ao;
global AodB;
global Auto;
global wait;

% Initializing global variables from text boxes, etc.
```

```
Ts = 1/(str2double(get(handles.SamplingFrequency,'String')));
FS = str2double(get(handles.FullScale,'String'));
A = str2double(get(handles.Amplitude,'String'));
fsin = str2double(get(handles.Frequency,'String'));
N = str2double(get(handles.NumberBits,'String'));
sigma = str2double(get(handles.Sigma,'String'));                % Sigma is the typical deviation of
the offset distribution
bandwidth = str2double(get(handles.Bandwidth,'String'));
minsignalband = str2double(get(handles.Minsignalband,'String'));
sigmajitter = str2double(get(handles.Jitter,'String'));
modelsimulated = get(handles.ModelSimulated,'String');
set(handles.ModelSimulated,'String',modelsimulated);
wait = str2double(get(handles.Timewait,'String'));
val = get(handles.Sweep_Popupmenu,'Value');
val2 = get(handles.Sweep2_Popupmenu,'Value');

slider_step(1)=1/9;
slider_step(2)=1/9;
set(handles.Effort_Slider,'sliderstep',slider_step,'Max',10.0,'Min',1.0,'Value',1.0);
Effort = get(handles.Effort_Slider,'Value');


nsamples = str2double(get(handles.Nsamples,'String'));
VoffsetVrefpn = sigma * randn(1);                    % Offsets are random numbers with typical
deviation sigma
Voffset0 = sigma * randn(1);
Select = 1;                                 % Ramp Input by default


AodB = str2double(get(handles.Gain,'String'));
Eps = str2double(get(handles.Mismatch,'String'));
SR = str2double(get(handles.SR,'String'));
Tau = str2double(get(handles.Tau,'String'));
Aux1 = str2double(get(handles.Aux1,'String'));
Aux2 = str2double(get(handles.Aux2,'String'));
Aux3 = str2double(get(handles.Aux3,'String'));
Aux4 = str2double(get(handles.Aux4,'String'));
A0 = 10^(AodB/20);
Auto=1;                                     % Automatic inputs change by default



% --- Executes during object creation, after setting all properties.
function SamplingFrequency_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SamplingFrequency (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 when editing Sampling Freq text box
function SamplingFrequency_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to SamplingFrequency (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 SamplingFrequency as text
%        str2double(get(hObject,'String')) returns contents of SamplingFrequency as a double
global Ts;
global bandwidth;
global fsin;
global minsignalband;
global Auto;
Ts = 1/(str2double(get(hObject,'String')))

% If automatic inputs change is selected, some values change
if(Auto==1)
   bandwidth = 1/(2*Ts);
   set(handles.Bandwidth,'String',bandwidth);
   fsin = 1/(5.1*Ts);
   set(handles.Frequency,'String',fsin);
   minsignalband = 1/(400*Ts);
   set(handles.Minsignalband,'String',minsignalband);
end

% --- Executes during object creation, after setting all properties.
function FullScale_CreateFcn(hObject, eventdata, handles)
% hObject    handle to FullScale (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 when editing FS text box
function FullScale_Callback(hObject, eventdata, handles)
% hObject    handle to FullScale (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 FullScale as text
%        str2double(get(hObject,'String')) returns contents of FullScale as a double
global FS;
global A;
global Auto;
FS = str2double(get(hObject,'String'))

% If automatic inputs change is selected, some values change
if(Auto==1)
   A = FS/2;
   set(handles.Amplitude,'String',A);
end

% --- Executes on button press in DNL_pushbutton.
function DNL_pushbutton_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to DNL_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global DNLCOD;
global axisxlabelVolt;
global axisylabelDNL;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','DNL','Visible','on',...
          'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(DNLCOD(1,:), DNLCOD(2,:),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','DNL')
xlabel(axisxlabelVolt)
ylabel(axisylabelDNL)
grid(handles.PlotAxes,'on')


% --- Executes on button press in Spectrum_pushbutton.
function Spectrum_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Spectrum_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global F;
global potencia1;
global axisxlabelFrequency;
global axisylabelSpectrum;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','Spectrum','Visible','on',...
          'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure on logarithmic axes, giving name, xlabel, ylabel and grid
semilogx(F,potencia1,'Parent',handles.PlotAxes);
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Spectrum')
xlabel(axisxlabelFrequency)
ylabel(axisylabelSpectrum)
grid(handles.PlotAxes,'on')


% --- Executes on button press in INL_pushbutton.
function INL_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to INL_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global INLCOD;
global axisxlabelVolt;
global axisylabelINL;
```

```
% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','INL','Visible','on',...
        'NumberTitle','off','HandleVisibility','on','IntegerHandle','off','Position',[10     40     500
400]);
handles.PlotAxes = axes('Parent',handles.PlotFigure);
  guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(INLCOD(1,:), INLCOD(2,:),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','INL')
xlabel(axisxlabelVolt)
ylabel(axisylabelINL)
grid(handles.PlotAxes,'on')


% --- Executes on button press in Output_Data_SWI_pushbutton.
function Output_Data_SWI_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Output_Data_SWI_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global vector;
global axisxlabelTime;
global axisylabelOutput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','Output','Visible','on',...
        'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(vector(:,1),vector(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Output')
xlabel(axisxlabelTime)
ylabel(axisylabelOutput)
grid(handles.PlotAxes,'on')

% --- Executes during object creation, after setting all properties.
function MaxINL_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MaxINL (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
```

```
% Not used (output result)
function MaxINL_Callback(hObject, eventdata, handles)
% hObject    handle to MaxINL (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 MaxINL as text
%        str2double(get(hObject,'String')) returns contents of MaxINL as a double


% --- Executes during object creation, after setting all properties.
function MaxDNL_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MaxDNL (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


% Not used (output result)
function MaxDNL_Callback(hObject, eventdata, handles)
% hObject    handle to MaxDNL (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 MaxDNL as text
%        str2double(get(hObject,'String')) returns contents of MaxDNL as a double


% --- Executes during object creation, after setting all properties.
function SNDR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SNDR (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


% Not used (output result)
function SNDR_Callback(hObject, eventdata, handles)
% hObject    handle to SNDR (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 SNDR as text
%        str2double(get(hObject,'String')) returns contents of SNDR as a double
```

```
% --- Executes during object creation, after setting all properties.
function SNR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SNR (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


% Not used (output result)
function SNR_Callback(hObject, eventdata, handles)
% hObject    handle to SNR (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 SNR as text
%        str2double(get(hObject,'String')) returns contents of SNR as a double


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

% Hint: listbox 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


% Not used (output messages)
function Messages_Listbox_Callback(hObject, eventdata, handles)
% hObject    handle to Messages_Listbox (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 Messages_Listbox contents as cell array
%        contents{get(hObject,'Value')} returns selected item from Messages_Listbox


% --- Executes on button press in Evaluate_pushbutton.
function Evaluate_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Evaluate_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Start;
global Stop;
global Points;
```

```
global Start2;
global Stop2;
global Points2;
global val;
global val2;

% Take sweep parameters from text boxes
Start = str2double(get(handles.Start,'String'))
Stop = str2double(get(handles.Stop,'String'))
Points = str2double(get(handles.Points,'String'))
Start2 = str2double(get(handles.Start2,'String'))
Stop2 = str2double(get(handles.Stop2,'String'))
Points2 = str2double(get(handles.Points2,'String'))

% Sweep select (if val2 equals 1 => Single Sweep; Others => Double sweep)
switch val2
  case 1              % Sweep 2 = None => Single Sweep
    switch val
      case 1          % Gain sweep
        EstudioGananciaParamGlobal(Start,Stop,Points,handles,hObject);
      case 2          % Mismatch sweep
        EstudioMismatchEtapa1ParamGlobal(Start,Stop,Points,handles,hObject);
      case 3          % Slew Rate sweep
        EstudioSREtapa1ParamGlobal(Start,Stop,Points,handles,hObject);
      case 4          % Tau sweep
        EstudioTauEtapa1ParamGlobal(Start,Stop,Points,handles,hObject);
      case 5          % Sigma sweep
        EstudioSigmaParamGlobal(Start,Stop,Points,handles,hObject);
      case 6          % Jitter sweep
        EstudioJitterParamGlobal(Start,Stop,Points,handles,hObject);
      case 7          % Sampling Frequency sweep
        EstudioSamplingFreqParamGlobal(Start,Stop,Points,handles,hObject);
      case 8          % Input amplitude sweep
        EstudioAmplitParamGlobal(Start,Stop,Points,handles,hObject);
      case 9          % Full Scale sweep
        EstudioFSParamGlobal(Start,Stop,Points,handles,hObject);
      case 10          % Input frequency sweep
        EstudioFreqParamGlobal(Start,Stop,Points,handles,hObject);
      case 11         % Bandwidth sweep
        EstudioBWParamGlobal(Start,Stop,Points,handles,hObject);
      case 12          % Aux 1 sweep
        EstudioAux1ParamGlobal(Start,Stop,Points,handles,hObject);
      case 13         % Aux 2 sweep
        EstudioAux2ParamGlobal(Start,Stop,Points,handles,hObject);
      case 14         % Aux 3 sweep
        EstudioAux3ParamGlobal(Start,Stop,Points,handles,hObject);
      case 15         % Aux 4 sweep
        EstudioAux4ParamGlobal(Start,Stop,Points,handles,hObject);
    end
  case 3              % Sweep 2 = Mismatch
    switch val
      case 1          % Gain and Mismatch sweep

EstudioGananciaMismatchParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObjec
t);
    end
  case 5              % Sweep 2 = Tau
    switch val
      case 3          % Slew Rate and Tau sweep
```

```
EstudioSRTauEtapa1ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 6                % Sweep 2 = Sigma
      switch val
         case 1          % Gain and Sigma sweep

EstudioGananciaSigmaParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 2          % Mismatch and Sigma sweep

EstudioMismatchSigmaParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 3          % Slew Rate and Sigma sweep

EstudioSRSigmaParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 4          % Tau and Sigma sweep

EstudioTauSigmaParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 7                % Sweep 2 = Jitter
      switch val
         case 1          % Gain and Jitter sweep

EstudioGananciaJitterParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 2          % Mismatch and Jitter sweep

EstudioMismatchJitterParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 3          % Slew Rate and Jitter sweep
            EstudioSRJitterParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 4          % Tau and Jitter sweep

EstudioTauJitterParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 5          % Sigma and Jitter sweep

EstudioSigmaJitterParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 8                % Sweep 2 = Sampling Freq
      switch val
         case 1          % Gain and Sampling Frequency sweep

EstudioGananciaSamplingFreqParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hO
bject);
         case 2          % Mismatch and Sampling Frequency sweep

EstudioMismatchSamplingFreqParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,h
Object);
      end
   case 9                % Sweep 2 = Amplit
      switch val
         case 1          % Gain and input amplitude sweep

EstudioGananciaAmplitParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
         case 2          % Mismatch and input amplitude sweep

EstudioMismatchAmplitParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 10               % Sweep 2 = FS
      switch val
         case 1          % Gain and Full Scale sweep
```

```
EstudioGananciaFSParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
        case 2          % Mismatch and Full Scale sweep

EstudioMismatchFSParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 11              % Sweep 2 = Freq
      switch val
        case 1          % Gain and input frequency sweep

EstudioGananciaFreqParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
        case 2          % Mismatch and input frequency sweep

EstudioMismatchFreqParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 14              % Sweep 2 = Aux 2
      switch val
        case 12         % Aux 1 and Aux 2 sweep

EstudioAux1Aux2ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 15              % Sweep 2 = Aux 3
      switch val
        case 12         % Aux 1 and Aux 3 sweep

EstudioAux1Aux3ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
        case 13         % Aux 2 and Aux 3 sweep

EstudioAux2Aux3ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
   case 16              % Sweep 2 = Aux 4
      switch val
        case 12         % Aux 1 and Aux 4 sweep

EstudioAux1Aux4ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
        case 13         % Aux 2 and Aux 4 sweep

EstudioAux2Aux4ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
        case 14         % Aux 3 and Aux 4 sweep

EstudioAux3Aux4ParamGlobal(Start,Stop,Points,Start2,Stop2,Points2,handles,hObject);
      end
end


% --- Executes during object creation, after setting all properties.
function Gain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Gain (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 when editing Gain text box
function Gain_Callback(hObject, eventdata, handles)
% hObject    handle to Gain (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 Gain as text
%        str2double(get(hObject,'String')) returns contents of Gain as a double
global AodB;
global Ao;
AodB = str2double(get(hObject,'String'))
Ao = 10^(AodB/20)

% --- Executes during object creation, after setting all properties.
function SR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SR (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 when editing SR text box
function SR_Callback(hObject, eventdata, handles)
% hObject    handle to SR (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 SR as text
%        str2double(get(hObject,'String')) returns contents of SR as a double
global SR;
SR = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Mismatch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Mismatch (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 when editing Mismatch text box
function Mismatch_Callback(hObject, eventdata, handles)
% hObject    handle to Mismatch (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 Mismatch as text
%        str2double(get(hObject,'String')) returns contents of Mismatch as a double
global Mismatch;
Mismatch = str2double(get(hObject,'String'))


% --- Executes on button press in Run_pushbutton.
function Run_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Run_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Gain;
global Mismatch;
global SlewRate;
global TimeConstant;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global wait;
global sigmajitter;
global Select;
global INLMAX;
global DNLMAX;
global misingcode;
global Sndr;
global Snr;
global Enob;
global axisxlabel;
global axisylabel;
global Timewait;
global go;


% Take single simulation parameters from text boxes
Gain = str2double(get(handles.Gain,'String'));
Mismatch = str2double(get(handles.Mismatch,'String'));
SlewRate = str2double(get(handles.SR,'String'));
TimeConstant = str2double(get(handles.Tau,'String'));
Aux1 = str2double(get(handles.Aux1,'String'));
Aux2 = str2double(get(handles.Aux2,'String'));
Aux3 = str2double(get(handles.Aux3,'String'));
Aux4 = str2double(get(handles.Aux4,'String'));
sigmajitter = str2double(get(handles.Jitter,'String'));
sigma = str2double(get(handles.Sigma,'String'));
Timewait = str2double(get(handles.Timewait,'String'));

% Offsets are random numbers with typical deviation sigma
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);

% Calling to the function which controls single simulations


SimulacionNoIdeal(Gain,Mismatch,SlewRate,TimeConstant,sigmajitter,sigma,Aux1,Aux2,Aux3,
Aux4,Timewait,handles,hObject);

% Which results change depends on the input type
```

```
if (Select==1)       % Select==1 equals Ramp Input => Change INL, DNL and MissingCodes
       set(handles.MaxINL,'String',num2str(INLMAX));
       set(handles.MaxDNL,'String',num2str(DNLMAX));
       set(handles.MissingCodes,'String',num2str(misingcode));
else                 % Select==0 equals Sine Wave Input => Change SNDR, SNR and ENOB
       set(handles.SNDR,'String',num2str(Sndr));
       set(handles.SNR,'String',num2str(Snr));
       set(handles.ENOB,'String',num2str(Enob));
end


% --- Executes on button press in Esc.
function Esc_Callback(hObject, eventdata, handles)
% hObject    handle to Esc (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.

set(handles.Messages_Listbox,'String',char('Simulation has paused'));

pause;


function Start_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Start (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 when editing Start text box
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (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 Start as text
%        str2double(get(hObject,'String')) returns contents of Start as a double
global Start;
Start = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Stop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Stop (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 when editing Stop text box
function Stop_Callback(hObject, eventdata, handles)
% hObject    handle to Stop (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 Stop as text
%        str2double(get(hObject,'String')) returns contents of Stop as a double
global Stop;
Stop = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Points_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Points (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 when editing Points text box
function Points_Callback(hObject, eventdata, handles)
% hObject    handle to Points (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 Points as text
%        str2double(get(hObject,'String')) returns contents of Points as a double
global Points;
Points = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Tau_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tau (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 when editing Tau text box
```

```
function Tau_Callback(hObject, eventdata, handles)
% hObject    handle to Tau (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 Tau as text
%        str2double(get(hObject,'String')) returns contents of Tau as a double
global Tau;
Tau = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Start2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Start2 (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 when editing Start2 text box
function Start2_Callback(hObject, eventdata, handles)
% hObject    handle to Start2 (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 Start2 as text
%        str2double(get(hObject,'String')) returns contents of Start2 as a double
global Start2;
Start2 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Stop2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Stop2 (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 when editing Stop2 text box
function Stop2_Callback(hObject, eventdata, handles)
% hObject    handle to Stop2 (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 Stop2 as text
%        str2double(get(hObject,'String')) returns contents of Stop2 as a double
global Stop2;
Stop2 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Points2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Points2 (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 when editing Points2 text box
function Points2_Callback(hObject, eventdata, handles)
% hObject    handle to Points2 (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 Points2 as text
%        str2double(get(hObject,'String')) returns contents of Points2 as a double
global Points2;
Points2 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Amplitude (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 when editing A text box
function Amplitude_Callback(hObject, eventdata, handles)
% hObject    handle to Amplitude (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 Amplitude as text
%        str2double(get(hObject,'String')) returns contents of Amplitude as a double
global A;
A = str2double(get(hObject,'String'))
```

```
% --- Executes during object creation, after setting all properties.
function Frequency_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Frequency (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 when editing Freq text box
function Frequency_Callback(hObject, eventdata, handles)
% hObject    handle to Frequency (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 Frequency as text
%        str2double(get(hObject,'String')) returns contents of Frequency as a double
global fsin;
fsin = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function NumberBits_CreateFcn(hObject, eventdata, handles)
% hObject    handle to NumberBits (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 when editing N text box
function NumberBits_Callback(hObject, eventdata, handles)
% hObject    handle to NumberBits (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 NumberBits as text
%        str2double(get(hObject,'String')) returns contents of NumberBits as a double
global N;
N = str2double(get(hObject,'String'))


% --- Executes on button press in SineWaveInput_Radiobutton.
function SineWaveInput_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to SineWaveInput_Radiobutton (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 SineWaveInput_Radiobutton

% If this button turns on, RampInput will be turned off
off = [handles.RampInput_Radiobutton];
mutual_exclude(off)
global Select;
Select=0;                % Sine Wave Input


% --- Executes on button press in RampInput_Radiobutton.
function RampInput_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to RampInput_Radiobutton (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 RampInput_Radiobutton

% If this button turns on, SineWaveInput will be turned off
off= [handles.SineWaveInput_Radiobutton];
mutual_exclude(off)
global Select;
Select=1;                % Ramp Input


% --- Function that ensures mutual exclusion between a group of radiobuttons
function mutual_exclude(off)
set(off,'Value',0)


% --- Executes during object creation, after setting all properties.
function Jitter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to 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


% --- Executes when editing Jitter text box
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)

% Hints: get(hObject,'String') returns contents of Jitter as text
%        str2double(get(hObject,'String')) returns contents of Jitter as a double
global sigmajitter;
sigmajitter = str2double(get(hObject,'String'))


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

```
function MissingCodes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MissingCodes (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


% Not used (output result)
function MissingCodes_Callback(hObject, eventdata, handles)
% hObject    handle to MissingCodes (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 MissingCodes as text
%        str2double(get(hObject,'String')) returns contents of MissingCodes as a double


% --- Executes during object creation, after setting all properties.
function Sigma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sigma (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 when editing Sigma text box
function Sigma_Callback(hObject, eventdata, handles)
% hObject    handle to Sigma (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 Sigma as text
%        str2double(get(hObject,'String')) returns contents of Sigma as a double
global sigma;
global VoffsetVrefpn;
global Voffset0;
sigma = str2double(get(hObject,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);


% --- Executes on button press in Output_Data_RI_pushbutton.
function Output_Data_RI_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Output_Data_RI_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global vector;
global vectorcode;
global axisxlabelVolt;
global axisylabelOutput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','Output','Visible','on',...
          'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(vectorcode(:,1),vector(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Output')
xlabel(axisxlabelVolt)
ylabel(axisylabelOutput)
grid(handles.PlotAxes,'on')


% --- Executes during object creation, after setting all properties.
function ENOB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ENOB (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


% Not used (output result)
function ENOB_Callback(hObject, eventdata, handles)
% hObject    handle to ENOB (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 ENOB as text
%        str2double(get(hObject,'String')) returns contents of ENOB as a double


% --- Executes during object creation, after setting all properties.
function Bandwidth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bandwidth (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 when editing BW text box
function Bandwidth_Callback(hObject, eventdata, handles)
% hObject    handle to Bandwidth (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 Bandwidth as text
%        str2double(get(hObject,'String')) returns contents of Bandwidth as a double
global bandwidth;
bandwidth = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Minsignalband_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Minsignalband (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 when editing MinSB text box
function Minsignalband_Callback(hObject, eventdata, handles)
% hObject    handle to Minsignalband (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 Minsignalband as text
%        str2double(get(hObject,'String')) returns contents of Minsignalband as a double
global minsignalband;
minsignalband = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function ModelSimulated_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ModelSimulated (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 when editing Model Simulated text box
```

```
function ModelSimulated_Callback(hObject, eventdata, handles)
% hObject    handle to ModelSimulated (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 ModelSimulated as text
%        str2double(get(hObject,'String')) returns contents of ModelSimulated as a double
global modelsimulated;
modelsimulated = get(handles.ModelSimulated,'String')


% --- Executes during object creation, after setting all properties.
function Nsamples_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nsamples (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 when editing Nsamples text box
function Nsamples_Callback(hObject, eventdata, handles)
% hObject    handle to Nsamples (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 Nsamples as text
%        str2double(get(hObject,'String')) returns contents of Nsamples as a double
global nsamples;
nsamples = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Sweep2_Popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sweep2_Popupmenu (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 Sweep2_Popupmenu.
function Sweep2_Popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to Sweep2_Popupmenu (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 Sweep2_Popupmenu contents as cell array
%        contents{get(hObject,'Value')} returns selected item from Sweep2_Popupmenu
global val2;
global Start2;
global Stop2;
global Points2;
val2 = get(hObject,'Value');

% Automatic values on Sweep 2 parameters depends on the option
% selected on Sweep 2
switch val2
   case 2                    % Sweep 2 = Gain
      Start2=40;
      Stop2=90;
      Points2=21;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 3                    % Sweep 2 = Mismatch
      Start2=-0.2;
      Stop2=0.2;
      Points2=41;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 4                    % Sweep 2 = SR
      Start2=14e8;
      Stop2=38e8;
      Points2=25;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 5                    % Sweep 2 = Tau
      Start2=0.01e-8;
      Stop2=2.01e-8;
      Points2=21;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 6                    % Sweep 2 = Sigma
      Start2=1e-3;
      Stop2=1e-2;
      Points2=11;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 7                    % Sweep 2 = Jitter
      Start2=1e-8;
      Stop2=1e-12;
      Points2=41;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
      set(handles.Points2,'String',Points2);
   case 8                    % Sweep 2 = Sampling Freq
      Start2=2e8;
      Stop2=2e6;
      Points2=21;
      set(handles.Start2,'String',Start2);
      set(handles.Stop2,'String',Stop2);
```

```
    set(handles.Points2,'String',Points2);
  case 9                    % Sweep 2 = Amplit
    Start2=0.1;
    Stop2=1;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 10                   % Sweep 2 = FS
    Start2=1;
    Stop2=3;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 11                   % Sweep 2 = Freq
    Start2=0.5e6;
    Stop2=4e6;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 12                   % Sweep 2 = BW
    Start2=5e6;
    Stop2=15e6;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 13                   % Sweep 2 = Aux 1
    Start2=1e-2;
    Stop2=1e-3;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
   case 14                  % Sweep 2 = Aux 2
    Start2=1e-2;
    Stop2=1e-3;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 15                   % Sweep 2 = Aux 3
    Start2=1e-2;
    Stop2=1e-3;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
  case 16                   % Sweep 2 = Aux 4
    Start2=1e-2;
    Stop2=1e-3;
    Points2=21;
    set(handles.Start2,'String',Start2);
    set(handles.Stop2,'String',Stop2);
    set(handles.Points2,'String',Points2);
end
```

```
% --- Executes during object creation, after setting all properties.
function Aux1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux1 (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 when editing Aux 1 text box
function Aux1_Callback(hObject, eventdata, handles)
% hObject    handle to Aux1 (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 Aux1 as text
%        str2double(get(hObject,'String')) returns contents of Aux1 as a double
global Aux1;
Aux1 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Aux3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux3 (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 when editing Aux 3 text box
function Aux3_Callback(hObject, eventdata, handles)
% hObject    handle to Aux3 (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 Aux3 as text
%        str2double(get(hObject,'String')) returns contents of Aux3 as a double
global Aux3;
Aux3 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Aux2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux2 (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 when editing Aux 2 text box
function Aux2_Callback(hObject, eventdata, handles)
% hObject    handle to Aux2 (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 Aux2 as text
%        str2double(get(hObject,'String')) returns contents of Aux2 as a double
global Aux2;
Aux2 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Aux4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux4 (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 when editing Aux 4 text box
function Aux4_Callback(hObject, eventdata, handles)
% hObject    handle to Aux4 (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 Aux4 as text
%        str2double(get(hObject,'String')) returns contents of Aux4 as a double
global Aux4;
Aux4 = str2double(get(hObject,'String'))


% --- Executes on button press in RampInput_pushbutton.
function RampInput_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to RampInput_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global siginput;
global axisxlabelTime;
global axisylabelInput;
```

```
% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','Input','Visible','on',...
        'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(siginput(:,1),siginput(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Input')
xlabel(axisxlabelTime)
ylabel(axisylabelInput)
grid(handles.PlotAxes,'on')


% --- Executes on button press in SineWaveInput_pushbutton.
function SineWaveInput_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to SineWaveInput_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global siginput;
global axisxlabelTime;
global axisylabelInput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
handles.PlotFigure = figure('Name','Input','Visible','on',...
        'NumberTitle','off','HandleVisibility','on','IntegerHandle','off');
handles.PlotAxes = axes('Parent',handles.PlotFigure);
guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(siginput(:,1),siginput(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Input')
xlabel(axisxlabelTime)
ylabel(axisylabelInput)
grid(handles.PlotAxes,'on')


% Not used (output result)
function Yes_Monotonic_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Yes_Monotonic_Radiobutton (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 Yes_Monotonic_Radiobutton


% Not used (output result)
function No_Monotonic_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to No_Monotonic_Radiobutton (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 No_Monotonic_Radiobutton
```

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

% If figure exists, set it current and hold on graphics
if isfield(handles,'PlotFigure') & ishandle(handles.PlotFigure),
   set(0,'CurrentFigure',handles.PlotFigure);
   hold on;
end


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

% If figure exists, set it current and hold off graphics
if isfield(handles,'PlotFigure') & ishandle(handles.PlotFigure),
   set(0,'CurrentFigure',handles.PlotFigure);
   hold off;
end


% --- Executes during object creation, after setting all properties.
function Sweep_Popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sweep_Popupmenu (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 Sweep_Popupmenu.
function Sweep_Popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to Sweep_Popupmenu (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 Sweep_Popupmenu contents as cell array
%        contents{get(hObject,'Value')} returns selected item from Sweep_Popupmenu
global val;
global Start;
global Stop;
global Points;
val = get(hObject,'Value');

% Automatic values on Sweep parameters depends on the option
% selected on Sweep
```

```
switch val
   case 1                % Sweep = Gain
      Start=40;
      Stop=90;
      Points=21;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 2                % Sweep = Mismatch
      Start=-0.2;
      Stop=0.2;
      Points=41;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 3                % Sweep = SR
      Start=14e8;
      Stop=38e8;
      Points=25;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 4                % Sweep = Tau
      Start=0.01e-8;
      Stop=2.01e-8;
      Points=21;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 5                % Sweep = Sigma
      Start=1e-3;
      Stop=1e-2;
      Points=11;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 6                % Sweep = Jitter
      Start=1e-8;
      Stop=1e-12;
      Points=41;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 7                % Sweep = Sampling Freq
      Start=2e8;
      Stop=2e6;
      Points=21;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 8                % Sweep = Amplit
      Start=0.1;
      Stop=1;
      Points=21;
      set(handles.Start,'String',Start);
      set(handles.Stop,'String',Stop);
      set(handles.Points,'String',Points);
   case 9                % Sweep = FS
      Start=1;
```

```
    Stop=3;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 10                 % Sweep = Freq
    Start=0.5e6;
    Stop=4e6;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 11                 % Sweep = BW
    Start=5e6;
    Stop=15e6;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 12                 % Sweep = Aux 1
    Start=1e-2;
    Stop=1e-3;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 13                 % Sweep = Aux 2
    Start=1e-2;
    Stop=1e-3;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 14                 % Sweep = Aux 3
    Start=1e-2;
    Stop=1e-3;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
  case 15                 % Sweep = Aux 4
    Start=1e-2;
    Stop=1e-3;
    Points=21;
    set(handles.Start,'String',Start);
    set(handles.Stop,'String',Stop);
    set(handles.Points,'String',Points);
end


% Not used----------------------------------------------------------
function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Not used----------------------------------------------------------
function menu_file_new_Callback(hObject, eventdata, handles)
```

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


% --- Executes on selecting menu File->New->Model
function file_new_model_Callback(hObject, eventdata, handles)
% hObject    handle to file_new_model (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global modelsimulated;

% If is not opened, open file "current.mdl"
if  isempty(find_system('Name','current')),
   open_system('current')
   figure(handles.Current)
end

% Write on Model Simulated text box the complete route of "current.mdl"
modelsimulated = [];
modelsimulated = strcat(pwd,'\current.mdl');

set(handles.ModelSimulated,'String',modelsimulated);


% --- Executes on selecting menu File->Open->Model
function file_open_model_Callback(hObject, eventdata, handles)
% hObject    handle to file_open_model (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global Ts;                      % Global variables to be initialized
global FS;
global A;
global fsin;
global N;
global sigma;
global sigmajitter;
global bandwidth;
global minsignalband;
global val;
global val2;
global Select;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global nsamples;
global Effort;
global Eps;
global SR;
global Tau;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global Ao;
global AodB;
global Auto;
global load_ex;
```

```
global load_example;

load_example = 0;
load_example = load_ex;
if (load_ex == 1)
   load_ex = 0;
   Ts = handles.Ts;
   FS = handles.FS;
   N = handles.N;
   bandwidth = handles.bandwidth;
   A = handles.A;
   fsin = handles.fsin;
   nsamples = handles.nsamples;
   minsignalband = handles.minsignalband;
   AodB = handles.AodB;
   sigmajitter = handles.sigmajitter;
   SR = handles.SR;
   Tau = handles.Tau;
   Eps = handles.Eps;
   sigma = handles.sigma;
   Aux1 = handles.Aux1;
   Aux2 = handles.Aux2;
   Aux3 = handles.Aux3;
   Aux4 = handles.Aux4;
   modelsimulated = handles.model;

   handles = load('Struct_gui');
   guidata(hObject,handles);

   set(handles.ModelSimulated,'String',modelsimulated);
   set(handles.SamplingFrequency,'String',1/Ts);
   set(handles.FullScale,'String',FS);
   set(handles.NumberBits,'String',N);
   set(handles.Bandwidth,'String',bandwidth);
   set(handles.Amplitude,'String',A);
   set(handles.Frequency,'String',fsin);
   set(handles.Nsamples,'String',nsamples);
   set(handles.Minsignalband,'String',minsignalband);
   set(handles.Gain,'String',AodB);
   set(handles.Jitter,'String',sigmajitter);
   set(handles.SR,'String',SR);
   set(handles.Tau,'String',Tau);
   set(handles.Mismatch,'String',Eps);
   set(handles.Sigma,'String',sigma);
   set(handles.Aux1,'String',Aux1);
   set(handles.Aux2,'String',Aux2);
   set(handles.Aux3,'String',Aux3);
   set(handles.Aux4,'String',Aux4);
   close('library')
else
   % Open a dialog box for selecting the file to be opened
   [filename, pathname] = uigetfile( ...
      {'*.mdl', 'All Model and Library Files (*.mdl)'; ...
      '*.*','All Files (*.*)'}, ...
      'Select Model');

   % If "Cancel" is selected then return
   if isequal([filename,pathname],[0,0])
      return
```

```
  % Otherwise construct the fullfilename and Check and load the file.
  else
     File = fullfile(pathname,filename);

     handles.LastFIle = File;
     guidata(hObject,handles)
     open_system(File);

  % Write on Model Simulated text box the complete route of the file
     modelsimulated = [];
     modelsimulated = File;
     set(handles.ModelSimulated,'String',modelsimulated);
     set(handles.Gain,'String',AodB);
     set(handles.SamplingFrequency,'String',1/Ts);

  end
end

% --- Executes on selecting menu File->Open->Library
function file_open_library_Callback(hObject, eventdata, handles)
% hObject    handle to file_open_library (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global handles_gui;
handles_gui = handles;
save Struct_gui  -struct handles_gui;
clear;
library;

% Add Library directory to the path
%diract=pwd;
%cd Library;
%dirlib=pwd;
%path(path,dirlib)

% Open a dialog box for selecting the file to be opened,
% starting in Library directory
%[filename, pathname] = uigetfile( ...
%    {'*.mdl', 'All Model and Library Files (*.mdl)'; ...
%       '*.*','All Files (*.*)'}, ...
%    'Select Library');
%cd ..;
% If "Cancel" is selected then return
%if isequal([filename,pathname],[0,0])
%    return
% Otherwise construct the fullfilename and Check and load the file.
%else
%    File = fullfile(pathname,filename);

%    handles.LastFIle = File;
%    guidata(hObject,handles)
%    open_system(File);
%end


% Not used----------------------------------------------------------
function environment_menu_Callback(hObject, eventdata, handles)
% hObject    handle to environment_menu (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selecting menu Environment->Load
function menu_environment_load_Callback(hObject, eventdata, handles)
% hObject    handle to menu_environment_load (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global modelsimulated;
global Ts;
global FS;
global N;
global bandwidth;
global minsignalband;
global fsin;
global A;
global nsamples;
global AodB;
global Mismatch;
global SR;
global Tau;
global sigma;
global sigmajitter;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global val;
global val2;
global Start;
global Stop;
global Points;
global Start2;
global Stop2;
global Points2;

diract=pwd;
cd States;

% Open a dialog box for selecting the file to be run,
% starting in States directory
[filename, pathname] = uigetfile( ...
   {'*.m','M-Files (*.m)'}, ...
   'Select Environment File');
cd ..;

% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
   return
% Otherwise construct the fullfilename and Check and run the file.
else
   File = fullfile(pathname,filename);

   handles.LastFIle = File;
   guidata(hObject,handles)
   run(File);
end
```

```
% --- Executes on selecting menu Environment->Save
function menu_environment_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_environment_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

diract=pwd;
cd States;

% Open a dialog box for selecting the name of the file in which
% the actual state will be saved, starting in States directory
[filename, pathname] = uiputfile( ...
   {'*.m','M-Files (*.m)'}, ...
   'Save as');
cd ..;
% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
   return
% Otherwise construct the fullfilename and Check and save the file.
else
   File = fullfile(pathname,filename);

   handles.LastFIle = File;
   guidata(hObject,handles)

% Open file with write permission
   fid=fopen(File,'w');

% Write all the code for the m-file that saves the code which
% will be run when "load" option would be selected, and so
% the actual state would be loaded
   fprintf(fid,'Ts=1/');
   fprintf(fid,get((handles.SamplingFrequency),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.SamplingFrequency,''String'',1/Ts)\n');
   fprintf(fid,'FS=');
   fprintf(fid,get((handles.FullScale),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.FullScale,''String'',FS)\n');
   fprintf(fid,'N=');
   fprintf(fid,get((handles.NumberBits),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.NumberBits,''String'',N)\n');
   fprintf(fid,'bandwidth=');
   fprintf(fid,get((handles.Bandwidth),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Bandwidth,''String'',bandwidth)\n');
   fprintf(fid,'A=');
   fprintf(fid,get((handles.Amplitude),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Amplitude,''String'',A)\n');
   fprintf(fid,'fsin=');
   fprintf(fid,get((handles.Frequency),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Frequency,''String'',fsin)\n');
   fprintf(fid,'nsamples=');
   fprintf(fid,get((handles.Nsamples),'String'));
   fprintf(fid,';\n');
```

```
fprintf(fid,'set(handles.Nsamples,"String",nsamples)\n');
fprintf(fid,'minsignalband=');
fprintf(fid,get((handles.Minsignalband),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Minsignalband,"String",minsignalband)\n');
fprintf(fid,'AodB=');
fprintf(fid,get((handles.Gain),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Gain,"String",AodB)\n');
fprintf(fid,'Eps=');
fprintf(fid,get((handles.Mismatch),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Mismatch,"String",Eps)\n');
fprintf(fid,'SR=');
fprintf(fid,get((handles.SR),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.SR,"String",SR)\n');
fprintf(fid,'Tau=');
fprintf(fid,get((handles.Tau),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Tau,"String",Tau)\n');
fprintf(fid,'sigmajitter=');
fprintf(fid,get((handles.Jitter),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Jitter,"String",sigmajitter)\n');
fprintf(fid,'sigma=');
fprintf(fid,get((handles.Sigma),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Sigma,"String",sigma)\n');
fprintf(fid,'Aux1=');
fprintf(fid,get((handles.Aux1),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Aux1,"String",Aux1)\n');
fprintf(fid,'Aux2=');
fprintf(fid,get((handles.Aux2),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Aux2,"String",Aux2)\n');
fprintf(fid,'Aux3=');
fprintf(fid,get((handles.Aux3),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Aux3,"String",Aux3)\n');
fprintf(fid,'Aux4=');
fprintf(fid,get((handles.Aux4),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Aux4,"String",Aux4)\n');
fprintf(fid,'val=');
fprintf(fid,'%d',get((handles.Sweep_Popupmenu),'Value'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Sweep_Popupmenu,"Value",val)\n');
fprintf(fid,'val2=');
fprintf(fid,'%d',get((handles.Sweep2_Popupmenu),'Value'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Sweep2_Popupmenu,"Value",val2)\n');
fprintf(fid,'Start=');
fprintf(fid,get((handles.Start),'String'));
fprintf(fid,';\n');
fprintf(fid,'set(handles.Start,"String",Start)\n');
fprintf(fid,'Stop=');
fprintf(fid,get((handles.Stop),'String'));
```

```
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Stop,''String'',Stop)\n');
   fprintf(fid,'Points=');
   fprintf(fid,get((handles.Points),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Points,''String'',Points)\n');
   fprintf(fid,'Start2=');
   fprintf(fid,get((handles.Start2),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Start2,''String'',Start2)\n');
   fprintf(fid,'Stop2=');
   fprintf(fid,get((handles.Stop2),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Stop2,''String'',Stop2)\n');
   fprintf(fid,'Points2=');
   fprintf(fid,get((handles.Points2),'String'));
   fprintf(fid,';\n');
   fprintf(fid,'set(handles.Points2,''String'',Points2)\n');

% Close file
   fclose(fid);
end


% Not used----------------------------------------------------------
function help_menu_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selecting menu Help->User Guide
function menu_help_userguide_Callback(hObject, eventdata, handles)
% hObject    handle to menu_help_userguide (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Open User Guide (pdf format)
open('ManualdeUsuario.pdf')


% ------- Executes on selecting menu Help->User Guide
function menu_help_userguide2_Callback(hObject, eventdata, handles)
% hObject    handle to menu_help_userguide2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Open User Guide Advance(pdf format)
open('ManualdeUsuarioAvanzado.pdf')


% --- Executes on selecting menu Environment->Save----------------
function menu_help_about_Callback(hObject, eventdata, handles)
% hObject    handle to menu_help_about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Create the character array for the msgbox
aboutmsg={'SimConverter ver 2.0','','Authors:','','Daniel Falcon Medina','Isabel Vacas
```

```
Páez','Fernando Munoz Chavero'};

% Display the msgbox, which will be modal
msgbox(aboutmsg,'About','modal')


% --- Executes during object creation, after setting all properties.
function Effort_Slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Effort_Slider (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 Effort_Slider_Callback(hObject, eventdata, handles)
% hObject    handle to Effort_Slider (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
global Effort;
Effort=get(hObject,'Value')


% --- Executes on button press in Yes_Auto_Radiobutton.
function Yes_Auto_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Yes_Auto_Radiobutton (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 Yes_Auto_Radiobutton

% If this button turns on, No_Auto will be turned off
off= [handles.No_Auto_Radiobutton];
mutual_exclude(off)
global Auto;
Auto=1;                    % Yes_Auto


% --- Executes on button press in No_Auto_Radiobutton.
function No_Auto_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to No_Auto_Radiobutton (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 No_Auto_Radiobutton

% If this button turns on, Yes_Auto will be turned off
off= [handles.Yes_Auto_Radiobutton];
```

```
mutual_exclude(off)
global Auto;
Auto=0;                         % No_Auto

% Not used----------------------------------------------------------
function menu_file_open_Callback(hObject, eventdata, handles)
% hObject    handle to menu_file_open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


function edit32_Callback(hObject, eventdata, handles)
% hObject    handle to edit32 (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 edit32 as text
%        str2double(get(hObject,'String')) returns contents of edit32 as a double


% --- Executes during object creation, after setting all properties.
function edit32_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit32 (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 Timewait_Callback(hObject, eventdata, handles)
% hObject    handle to Timewait (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 Timewait as text
%        str2double(get(hObject,'String')) returns contents of Timewait as a double

global wait;
wait = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Timewait_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Timewait (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
```

ANEXO 4

Código Matlab interfaz library

Veamos las funcione que han sido necesarias para generar la interfaz library y dotarla de funcionalidad. Las funciones son:

- **Library.m**

```
% Script that has got the code that controls the behaviour of the GUI

function [varargout] = library(varargin)
% LIBRARY M-file for library.fig
%      LIBRARY, by itself, creates a new LIBRARY or raises the existing
%      singleton*.
%
%      H = LIBRARY returns the handle to a new LIBRARY or the handle to
%      the existing singleton*.
%
%      LIBRARY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LIBRARY.M with the given input arguments.
%
%      LIBRARY('Property','Value',...) creates a new LIBRARY or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before library_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to library_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

% Copyright 2002-2003 The MathWorks, Inc.

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

% Last Modified by GUIDE v2.5 20-Oct-2005 12:01:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @library_OpeningFcn, ...
                   'gui_OutputFcn',  @library_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(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 library is made visible.
function library_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 library (see VARARGIN)

global m1; % Value menu Converters
global m2; % Value menu Blocks
global m3; % Value menu Components
global m4; % Value menu Types
global menu_comp; % Dinamic menu Components
global menu_type; % Dinamic manu Types
global find; % Activate the description
global open; % Open the selected archive

% Iniatilize some values of global variables
[m1,m2,m3,m4,find,open] = var_global(handles);

handles.m1=m1;
handles.m2=m2;
handles.m3=m3;
handles.m4=m4;
handles.find=find;
handles.open=open;


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

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes library wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = library_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 selection change in menu_conv.
function menu_conv_Callback(hObject, eventdata, handles)
% hObject    handle to menu_conv (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 menu_conv contents as cell array
%        contents{get(hObject,'Value')} returns selected item from menu_conv

global valconv;
```

```
global list_comp;
global list_type;

valconv = get(hObject,'Value');
list_comp = set(handles.menu_comp,'String');
list_type = set(handles.menu_type,'String');
m2 = 1;
m3 = 1;
m4 = 1;
set(handles.menu_bloc,'Value',1);
set(handles.menu_type,'Value',1);
set(handles.menu_comp,'Value',1);

switch valconv
case 1
   m1 = 1; % TD Sigma-Delta Converters
   list_comp = 'Library Components|Integrators|Comparators|Quantizers|All models|Others...';
   list_type = ' -';
   set(handles.menu_comp,'String',list_comp);
   set(handles.menu_type,'String',list_type);
case 2
   m1 = 2; % TC Sigma-Delta Converters
   list_comp = ' -';
   list_type = ' -';
   set(handles.menu_comp,'String',list_comp);
   set(handles.menu_type,'String',list_type);
case 3
   m1 = 3; % Pipeline Converters
   list_comp = 'Library Components|ADC_Stage|DAC_Stage|SHA_Stage|Stage complete|Stage
last|Digital correction|All models|Others...';
   list_type = ' -';
   set(handles.menu_comp,'String',list_comp);
   set(handles.menu_type,'String',list_type);
case 4
   m1 = 4; % Flash Converters
   list_comp = 'Library Components|Reference Generator|Comparators|Encoder|All
models|Others...';
   list_type = ' -';
   set(handles.menu_comp,'String',list_comp);
   set(handles.menu_type,'String',list_type);
end

handles.list_comp = list_comp;
handles.list_type = list_type;
handles.m1 = m1;
handles.m2 = m2;
handles.m3 = m3;
handles.m4 = m4;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function menu_conv_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_conv (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 menu_bloc.
function menu_bloc_Callback(hObject, eventdata, handles)
% hObject    handle to menu_bloc (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 menu_bloc contents as cell array
%       contents{get(hObject,'Value')} returns selected item from menu_bloc

global valbloq;
global list_comp;
global list_type;
global m1;

valbloq = get(hObject,'Value');
m1 = handles.m1;
m3 = 1;
m4 = 1;
set(handles.menu_type,'Value',1);
set(handles.menu_comp,'Value',1);

switch valbloq
case 1
   m2 = 1;
   % Case 1-1: Ideal Blocks
   if (m1 == 1)
      list_comp        =        'Library        Components|Integrators|Comparators|Quantizers|All
models|Others...';
      list_type = ' -';
      set(handles.menu_comp,'String',list_comp);
      set(handles.menu_type,'String',list_type);
   end
   % Case 2-1: Ideal Blocks

   % Case 3-1: Ideal Blocks
   if (m1 == 3)
      list_comp        =        'Library        Components|ADC_Stage|DAC_Stage|SHA_Stage|Stage
complete|Stage last|Digital correction|All models|Others...';
      list_type = ' -';
      set(handles.menu_comp,'String',list_comp);
      set(handles.menu_type,'String',list_type);
   end
   % Case 4-1: Ideal Blocks
   if (m1 == 4)
      list_comp    =    'Library    Components|Reference    Generator|Comparators|Encoder|All
models|Others...';
      list_type = ' -';
      set(handles.menu_comp,'String',list_comp);
      set(handles.menu_type,'String',list_type);
   end
case 2
```

```
   m2 = 2;
   % Case 1-2: No-ideal Blocks
   if (m1 == 1)
      list_comp = 'Library Components|Integrator|Comparator|Jitter Noise|All
models|Others...';
      list_type = ' -';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
   % Case 2-2: No-ideal Blocks

   % Case 3-2: No-ideal Blocks
   if (m1 == 3)
      list_comp = 'Library Components|DAC_Stage|Stage complete|Jitter Noise|All
models|Others...';
      list_type = ' -';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
   % Case 4-2: No-ideal Blocks
   if (m1 == 4)
      list_comp = 'Library Components|Reference Generator|Comparators|All models|Others...';
      list_type = ' -';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
case 3
   m2 = 3;
   % Case 1-3: Examples
   if (m1 == 1)
      list_comp = ' -|Own examples';
      list_type = 'Library Examples|TD Sigma Delta_LP_2order_ideal_simple|TD Sigma
Delta_LP_2order_real_simple|TD Sigma Delta_LP_2order_ideal_cascade|TD Sigma
Delta_LP_2order_real_cascade|All examples...';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
   % Case 2-3: Examples

   % Case 3-3: Examples
   if (m1 == 3)
      list_comp = ' -|Own examples';
      list_type='Library
Examples|Pipeline_ideal_10b|Pipeline_gain_10b|Pipeline_mismatch_10b|Pipeline_SR_BW_
10b|Pipeline_voffset_10b|Pipeline_jitter_10b|All examples...';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
   % Case 4-3: Examples
   if (m1 == 4)
      list_comp = ' -|Own examples';
      list_type = 'Library Examples|Flash_offset_relay|All examples...';
      set(handles.menu_type,'String',list_type);
      set(handles.menu_comp,'String',list_comp);
   end
end

handles.list_comp = list_comp;
handles.list_type = list_type;
```

```
handles.m2 = m2;
handles.m3 = m3;
handles.m4 = m4;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function menu_bloc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_bloc (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 menu_comp.
function menu_comp_Callback(hObject, eventdata, handles)
% hObject    handle to menu_comp (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 menu_comp contents as cell array
%        contents{get(hObject,'Value')} returns selected item from menu_comp

global valcomp;
global m2;
global list_type;

valcomp = get(hObject,'Value');
m1= handles.m1;
m2= handles.m2;
m3 = 1;
m4 = 1;
set(handles.menu_type,'Value',1);

if m2 == 3  % Examples
   switch valcomp
      case 1
         m3 = 1;
         switch m1
            case 1
               % Examples TD Sigma Delta
               list_type = 'Library Examples|TD Sigma Delta_LP_2order_ideal_simple|TD Sigma
Delta_LP_2order_real_simple|TD     Sigma     Delta_LP_2order_ideal_cascade|TD     Sigma
Delta_LP_2order_real_cascade|All examples...';
               set(handles.menu_type,'String',list_type);
            case 2
               % Examples TC Sigma Delta
            case 3
               % Examples Pipeline
               list_type                                    =                                    'Library
Examples|Pipeline_ideal_10b|Pipeline_gain_10b|Pipeline_mismatch_10b|Pipeline_SR_10b|P
```

```
ipeline_voffset_10b|Pipeline_jitter_10b|All examples...';
            set(handles.menu_type,'String',list_type);
          case 4
            % Examples Flash
            list_type = 'Library Examples|Flash_offset_relay|All examples...';
            set(handles.menu_type,'String',list_type);
        end
      case 2
        m3 = 2;
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
  end
else
  switch valcomp
    case 1
      m3 = 1;
      % Case 1-2-1: Library Components
      if (m1 == 1|m1 == 2|m1 == 3|m1 == 4) & (m2 == 1|m2 == 2)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
    case 2
      m3 = 2;
      % Case 1-2-2: TD Sigma Delta & No-ideal Blocks & Integrator
      if (m1 == 1) & (m2 == 2)
        list_type = 'Switch thermal noise|Operational amplifier noise|Operational amplifier
finite gain|Operational amplifier SR+GBW|Operational amplifier saturation voltages|All no-
idealities together';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-1-2: Pipeline & Ideal & ADC
      if (m1 == 3) & (m2 == 1)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-2-2: Pipeline & No-Ideal & DAC
      if (m1 == 3) & (m2 == 2)
        list_type = 'Offset reference voltage';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-2-2: Flash & No-ideal & Reference Generator
      if (m1 == 4) & (m2 == 2)
        list_type = 'Offset reference generator';
        set(handles.menu_type,'String',list_type);
      end
    case 3
      m3 = 3;
      % Case 1-2-3: TD Sigma Delta & No-ideal Blocks & Comparator
      if (m1 == 1) & (m2 == 2)
        list_type = 'Hysteresis|Offset';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-1-3: Pipeline & Ideal & DAC
      if (m1 == 3) & (m2 == 1)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-2-3: Pipeline & No-Ideal & Stage complete
      if (m1 == 3) & (m2 == 2)
```

```
        list_type = 'Stage RSD Finite gain|Stage RSD Mismatch|Stage RSD SR and BW|Stage
RSD Offset|Stage RSD Jitter noise';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-1-3: Flash & Ideal & Comparators
      if (m1 == 4) & (m2 == 1)
        list_type = 'Comparator without Ts|Comparator with Ts';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-2-3: Flash & No-Ideal & Comparators
      if (m1 == 4) & (m2 == 2)
        list_type = 'Offset|Offset Relay|Offset Relay Pole';
        set(handles.menu_type,'String',list_type);
      end
    case 4
      m3 = 4;
      % Case 1-1-4: TD Sigma Delta & No-ideal Blocks & Noise Jitter
      if (m1 == 1) & (m2 == 2)
        list_type = 'Clock Jitter';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-1-4: Pipeline & Ideal & SHA
      if (m1 == 3) & (m2 == 1)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-2-4: Pipeline & No-Ideal & Jitter
      if (m1 == 3) & (m2 == 2)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-1-4: Flash & Ideal & Encoder
      if (m1 == 4) & (m2 == 1)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-2-4: Flash & No-ideal & All
      if (m1 == 4) & (m2 == 2)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
    case 5
      m3 = 5;
      % Case 1-2-5: TD Digma Delta & Ideal and No-ideal Blocks & All models
      if (m1 == 1) & (m2 == 1|m2 == 2)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-1-5: Pipeline & Ideal & Stage complete
      if (m1 == 3) & (m2 == 1)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 3-2-5: Pipeline & No-Ideal & All
      if (m1 == 3) & (m2 == 2)
        list_type = ' -';
        set(handles.menu_type,'String',list_type);
      end
      % Case 4-2-5: Flash & Ideal/No-ideal & All/Others
```

```
        if (m1 == 4) & (m2 == 1|m2 == 2)
          list_type = ' -';
          set(handles.menu_type,'String',list_type);
        end
      case 6
        m3 = 6;
        % Case 1-2-5: TD Digma Delta & Ideal and No-ideal Blocks & Others
        if (m1 == 1) & (m2 == 1|m2 == 2)
          list_type = ' -';
          set(handles.menu_type,'String',list_type);
        end
        % Case 3-2-6: Pipeline & No-Ideal & Others
        if (m1 == 3) & (m2 == 2)
          list_type = ' -';
          set(handles.menu_type,'String',list_type);
        end
        % Case 3-2-6: Flash & No-Ideal & Others
        if (m1 == 4) & (m2 == 2)
          list_type = ' -';
          set(handles.menu_type,'String',list_type);
        end
      case 7
        m3 = 7;
        % Case 3-1-7: Pipeline & All ideal models
        if (m1 == 3) & (m2 == 1)
          list_type = ' -';
                        set(handles.menu_type,'String',list_type);
        end
        % Case 3-1-7: Digital correction
        if (m1 == 3) & (m2 == 1)
          list_type = 'RSD_Digital correction to 3 bits|RSD_Digital correction to 4
bits|RSD_Digital correction to 5 bits|RSD_Digital correction to 6 bits|RSD_Digital correction
to 7 bits|RSD_Digital correction to 8 bits|RSD_Digital correction to 9 bits|RSD_Digital
correction to 10 bits|RSD_Digital correction to 11 bits|RSD_Digital correction to 12
bits|RSD_Digital correction to 13 bits|RSD_Digital correction to 14 bits|RSD_Digital correction
to 15 bits|Generic Dig correction with 2 stages|Generic Dig correction with 3 stages|Generic Dig
correction with 4 stages|All digital correction and others...';
          set(handles.menu_type,'String',list_type);
        end
      case 8
        m3 = 8;
        % Case 3-1-8: Pipeline & All models
        if (m1 == 3) & (m2 == 1)
          list_type = ' -';
                        set(handles.menu_type,'String',list_type);
        end
      case 9
        m3 = 9;
        % Case 3-1-9: Pipeline & Others...
        if (m1 == 3) & (m2 == 1)
          list_type = ' -';
                        set(handles.menu_type,'String',list_type);
        end
    end
end

handles.list_type = list_type;
handles.m4=m4;
handles.m3=m3;
```

```
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function menu_comp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_comp (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 menu_type.
function menu_type_Callback(hObject, eventdata, handles)
% hObject    handle to menu_type (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 menu_type contents as cell array
%        contents{get(hObject,'Value')} returns selected item from menu_type

global valtype;
valtype = get(hObject,'Value');
m2=handles.m2;

switch valtype
   case valtype
      m4 = valtype;
end
handles.m4=m4;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function menu_type_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_type (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 button press in button_find.
function button_find_Callback(hObject, eventdata, handles)
```

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

global find;
find = 0;
m1=handles.m1;
m2=handles.m2;
m3=handles.m3;
m4=handles.m4;
button_state = get(hObject,'Value');

if button_state == get(hObject,'Max')
   % toggle button is pressed
   find = 1;
elseif button_state == get(hObject,'Min')
   % toggle button is not pressed
   find = 0;
end

description(m1,m2,m3,m4,find,handles);

handles.find=find;
guidata(hObject, handles);



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

global open;
global modelsimulated;
global load_ex;

open = 0;
load_ex = 0;
m1=handles.m1;
m2=handles.m2;
m3=handles.m3;
m4=handles.m4;

button_state = get(hObject,'Value');

if button_state == get(hObject,'Max')
   % toggle button is pressed
   open = 1;
elseif button_state == get(hObject,'Min')
   % toggle button is not pressed
   open = 0;
end

handles.open=open;
guidata(hObject, handles);

modelsimulated = archive(m1,m2,m3,m4,open,handles);
```

```
if
((m1==1)&(m2==3)&(m4~=1)&(m4~=5))|((m1==3)&(m2==3)&(m4~=1)&(m4~=8)|((m1==4)
&(m2==3)&(m4~=1)&(m4~=3)))
   load_ex =1;
   handles = state(m1,m3,m4);
   handles.model = modelsimulated;
   handles.load_ex = load_ex;
   handles.open=open;
   guidata(hObject, handles);
   GUIproyecto('file_open_model_Callback',hObject, eventdata, handles);
end
```

- **Description.m**

```
% Show a description of each type

function description(m1,m2,m3,m4,find,handles)

if find == 1
   switch m1
      case 1  % TD Sigma Delta
         switch m2
            case 1
               switch m3
                  case 1
                     switch m4
                        case 1
                           % Case 1-1-1-1:
                           set(handles.description,'String',char('','Library where you can find all
models of TD Sigma Delta: Ideal Blocks.'));
                     end
                  case 2
                     switch m4
                        case 1
                           % Case 1-1-2-1:
                           set(handles.description,'String',char('','Some models of Ideal integrators.'));
                     end
                  case 3
                     switch m4
                        case 1
                           % Case 1-1-3-1:
                           set(handles.description,'String',char('','Some model of Ideal comparators.'));
                     end
                  case 4
                     switch m4
                        case 1
                           % Case 1-1-4-1:
                           set(handles.description,'String',char('','Some model of Ideal quantizers.'));
                     end
                  case 5
                     switch m4
                        case 1
                           % Case 1-1-5-1:
                           set(handles.description,'String',char('','Simulink archive where you can find
all library models.'));
                     end
                  case 6
```

```
                switch m4
                  case 1
                    % Case 1-1-6-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
                end

            end
          case 2
            switch m3
              case 1
                switch m4
                  case 1
                    % Case 1-2-1-1:
                    set(handles.description,'String',char('','Library where you can find all
models of TD Sigma Delta: No-ideal Blocks.'));
                end
              case 2
                switch m4
                  case 1
                    % Case 1-2-2-1:
                    set(handles.description,'String',char('','Model to simulate the switch thermal
noise of SC integrator.'));
                  case 2
                    % Case 1-2-2-2:
                    set(handles.description,'String',char('','Model to simulate the operational
amplifier noise.'));
                  case 3
                    % Case 1-2-2-3:
                    set(handles.description,'String',char('','Model to simulate the operational
finite gain.'));
                  case 4
                    % Case 1-2-2-4:
                    set(handles.description,'String',char('','Model to simulate the operational
slew-rate and gain-bandwith product.'));
                  case 5
                    % Case 1-2-2-5:
                    set(handles.description,'String',char('','Model to simulate the operational
amplifier saturation voltages.'));
                  case 6
                    % Case 1-2-2-6:
                    set(handles.description,'String',char('','Model to simulate all no-idealities in
the same integrator.'));
                end
              case 3
                switch m4
                  case 1
                    % Case 1-2-3-1:
                    set(handles.description,'String',char('','Model to simulate the hysteresis of
comparators.'));
                  case 2
                    % Case 1-2-3-2:
                    set(handles.description,'String',char('','Model to simulate the offset of
comparators.'));
                end
              case 4
                switch m4
                  case 1
                    % Case 1-2-4-1:
```

```
                    set(handles.description,'String',char('','Model to simulate the clock jitter. It
block should be put it at the input.'));
                end
            case 5
                switch m4
                    case 1
                        % Case 1-2-5-1:
                        set(handles.description,'String',char('','Simulink archive where you can find
all library models.'));
                end
            case 6
                switch m4
                    case 1
                        % Case 1-2-6-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
                end
        end
    case 3
        switch m3
            case 1
                switch m4
                    case 1
                        % Case 1-3-1-1:
                        set(handles.description,'String',char('','Library where you can find all
examples of TD Sigma Delta'));
                    case 2
                        % Case 1-3-1-2:
                        set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - Low-Pass','  - 2º order','  - Simple','  - Ideal components'));
                    case 3
                        % Case 1-3-1-3:
                        set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - Low-Pass','  - 2º order','  - Simple','  - Some real components'));
                    case 4
                        % Case 1-3-1-4:
                        set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - ADSL+ application','  - Low-Pass','  - 2º order','  - Cascade','  - Ideal components'));
                    case 5
                        % Case 1-3-1-5:
                        set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - ADSL+ application','  - Low-Pass','  - 2º order','  - Cascade','  - Some real
components'));
                    case 6
                        % Case 1-3-1-6:
                        set(handles.description,'String',char('','Simulink archive where you can find
all examples.'));
                end
            case 2
                switch m4
                    case 1
                        % Case 1-3-2-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other examples.'));
                end
        end
    end
case 2  % TC Sigma Delta
    switch m2
```

```
        case 1
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-1-1-1:
                    end
            end
        case 2
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-2-1-1:
                    end
            end
        case 3
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-3-1-1:
                    end
            end
    end
case 3  % Pipeline
    switch m2
        case 1
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 3-1-1-1:
                            set(handles.description,'String',char('','Library where you can find all models of Pipeline: Ideal Blocks.'));
                    end
                case 2
                    switch m4
                        case 1
                            % Case 3-1-2-1:
                            set(handles.description,'String',char('','Ideal model of analog to digital converter stage.','','It is usually referred to as a redundant signed digit (RSD) converter.','',' Also you can find a generic model.'));
                    end
                case 3
                    switch m4
                        case 1
                            % Case 3-1-3-1:
                            set(handles.description,'String',char('','Ideal model of digital to analog converter stage.','','It is usually referred to as a redundant signed digit (RSD) converter.','',' Also you can find a generic model.'));
                    end
                case 4
                    switch m4
                        case 1
                            % Case 3-1-4-1:
                            set(handles.description,'String',char('','Ideal model to simulate the sample and hold. Also, this block includes the residue gain in the stage.','',' Also you can find a generic model.'));
```

```
                end
            case 5
                switch m4
                    case 1
                        % Case 3-1-5-1:
                        set(handles.description,'String',char('','Model that conteins a complete stage: ADC, DCA y SHA','','ADC: analog to digital converter','DCA: digital to analog converter','SHA: residue gain, sample and hold'));
                end
            case 6
                switch m4
                    case 1
                        % Case 3-1-6-1:
                        set(handles.description,'String',char('','Model that conteins a last stage: only ADC','','ADC: analog to digital converter'));
                end
            case 7
                switch m4
                    case 1
                        % Case 3-1-7-1:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 3 bits','','Valid for RSD stages.'));
                    case 2
                        % Case 3-1-7-2:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 4 bits','','Valid for RSD stages.'));
                    case 3
                        % Case 3-1-7-3:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 5 bits','','Valid for RSD stages.'));
                    case 4
                        % Case 3-1-7-4:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 6 bits','','Valid for RSD stages.'));
                    case 5
                        % Case 3-1-7-5:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 7 bits','','Valid for RSD stages.'));
                    case 6
                        % Case 3-1-7-6:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 8 bits','','Valid for RSD stages.'));
                    case 7
                        % Case 3-1-7-7:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 9 bits','','Valid for RSD stages.'));
                    case 8
                        % Case 3-1-7-8:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 10 bits','','Valid for RSD stages.'));
                    case 9
                        % Case 3-1-7-9:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 11 bits','','Valid for RSD stages.'));
                    case 10
                        % Case 3-1-7-10:
                        set(handles.description,'String',char('','Model to make the digital correction pipeline to 12 bits','','Valid for RSD stages.'));
                    case 11
```

```
                    % Case 3-1-7-11:
                    set(handles.description,'String',char('','Model to make the digital correction
pipeline to 13 bits','','Valid for RSD stages.'));
                case 12
                    % Case 3-1-7-12:
                    set(handles.description,'String',char('','Model to make the digital correction
pipeline to 14 bits','','Valid for RSD stages.'));
                case 13
                    % Case 3-1-7-13:
                    set(handles.description,'String',char('','Model to make the digital correction
pipeline to 15 bits','','Valid for RSD stages.'));
                case 14
                    % Case 3-1-7-14:
                    set(handles.description,'String',char('','Model to make the generic correction
pipeline with 2 stages','','You could get 2*(N-1)+1 bits where N represents the number of bits of
each stage.'));
                case 15
                    % Case 3-1-7-15:
                    set(handles.description,'String',char('','Model to make the generic correction
pipeline with 3 stages','','You could get 3*(N-1)+1 bits where N represents the number of bits of
each stage.','','Also, there is a model with different output bits in each stage.'));
                case 16
                    % Case 3-1-7-16:
                    set(handles.description,'String',char('','Model to make the generic correction
pipeline with 4 stages','','You could get 4*(N-1)+1 bits where N represents the number of bits of
each stage.','','Also, there is a model with different output bits in each stage.'));
                case 17
                    % Case 3-1-7-17:
                    set(handles.description,'String',char('','Models to make the digital correction
pipeline in the range between 3 and 15 bits and others','','Valid for RSD and generic stages.','','In
this simulink archive you can save other models that you create.'));
            end
        case 8
            switch m4
                case 1
                    % Case 3-1-8-1:
                    set(handles.description,'String',char('','Simulink archive where you can find
all models.'));
            end
        case 9
            switch m4
                case 1
                    % Case 3-1-9-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
            end
    end
case 2
    switch m3
        case 1
            switch m4
                case 1
                    % Case 3-2-1-1:
                    set(handles.description,'String',char('','Library where you can find all
models of Pipeline: No-ideal Blocks.'));
            end
        case 2
            switch m4
                case 1
```

```
                    % Case 3-2-2-1:
                    set(handles.description,'String',char('','Model to simulate the ADC with
offset in reference voltage.'));
                end
            case 3
                switch m4
                    case 1
                    % Case 3-2-3-1:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with finite gain.','','Valid to RSD stages.'));
                    case 2
                    % Case 3-2-3-2:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with mismatch between capacitors.','','Valid to RSD stages.'));
                    case 3
                    % Case 3-2-3-3:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with SR and BW.','','Valid to RSD stages.'));
                    case 4
                    % Case 3-2-3-4:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with offset in reference voltage.','','Valid to RSD stages.'));
                    case 5
                    % Case 3-2-3-5:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with jitter noise.','','Put it at the input','','Valid to RSD stages.'));
                end
            case 4
                switch m4
                    case 1
                    % Case 3-2-4-1:
                    set(handles.description,'String',char('','Model to simulate the jitter
noise.','','You have to put this block at the input.'));
                end
            case 5
                switch m4
                    case 1
                    % Case 3-2-5-1:
                    set(handles.description,'String',char('','Simulink archive where you can find
all models.'));
                end
            case 6
                switch m4
                    case 1
                    % Case 3-2-6-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
                end
        end
    case 3
        switch m3
            case 1
                switch m4
                    case 1
                    % Case 1-3-1-1:
                    set(handles.description,'String',char('','Library where you can find all
examples of Pipeline'));
                    case 2
                    % Case 1-3-1-2:
```

```
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Ideal components'));
                case 3
                    % Case 3-3-1-3:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Finite Gain'));
                case 4
                    % Case 3-3-1-4:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Mismatch'));
                case 5
                    % Case 3-3-1-5:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Slew-rate and bandwidth'));
                case 6
                    % Case 3-3-1-6:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Offset in reference voltage'));
                case 7
                    % Case 3-3-1-7:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Jitter noise'));
                case 8
                    % Case 3-3-1-8:
                    set(handles.description,'String',char('','Simulink archive where you can find
all examples'));
            end
        case 2
            switch m4
                case 1
                    % Case 3-3-2-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other examples'));
            end
    end
end
    case 4 % Flash
    switch m2
        case 1
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 3-1-1-1: Library
                            set(handles.description,'String',char('','Library where you can find all
models of Flash: Ideal Blocks.'));
                    end
                case 2
                    switch m4
                        case 1
                            % Case 4-1-2-1:
                            set(handles.description,'String',char('','Ideal model to generate reference
voltages.','','Its output are inputs of comparators'));
                    end
                case 3
                    switch m4
                        case 1
                            % Case 4-1-3-1:
                            set(handles.description,'String',char('','Ideal model of comparator.'));
```

```
                    case 2
                        % Case 4-1-3-2:
                        set(handles.description,'String',char('','Ideal model of comparator with
sample and hold at the input.'));
                end
            case 4
                switch m4
                    case 1
                        % Case 4-1-4-1:
                        set(handles.description,'String',char('','Ideal encoder of 63 inputs.'));
                end
            case 5
                switch m4
                    case 1
                        % Case 4-1-5-1:
                        set(handles.description,'String',char('','Simulink archive where you can find
all ideal models'));
                end
            case 6
                switch m4
                    case 1
                        % Case 4-1-6-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other ideal models'));
                end
        end
    case 2
        switch m3
            case 1
                switch m4
                    case 1
                        % Case 4-2-1-1: Library
                        set(handles.description,'String',char('','Library where you can find all
models of Flash: No-Ideal Blocks.'));
                end
            case 2
                switch m4
                    case 1
                        % Case 4-2-2-1:
                        set(handles.description,'String',char('','Real model to generate reference
voltages. Included offet','','Its output are inputs of comparators'));
                    end
            case 3
                switch m4
                    case 1
                        % Case 4-1-3-1:
                        set(handles.description,'String',char('','Real model of comparator.'));
                    case 2
                        % Case 4-2-3-2:
                        set(handles.description,'String',char('','Real model of comparator with relay
and offset.'));
                    case 3
                        % Case 4-2-3-3:
                        set(handles.description,'String',char('','Real model of comparator with relay,
offset and pole effect.'));
                end
            case 4
                switch m4
                    case 1
```

```
                    % Case 4-2-4-1:
                    set(handles.description,'String',char('','Simulink archive where you can find
all real models'));
                end
            case 5
                switch m4
                    case 1
                    % Case 4-2-5-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other real models'));
                end
            end
        case 3
            switch m3
                case 1
                    switch m4
                        case 1
                        % Case 4-3-1-1:Library
                        set(handles.description,'String',char('','Library where you can find all
models of Flash: Examples.'));
                        case 2
                        % Case 4-3-1-2:
                        set(handles.description,'String',char('','Architecture Flash with:','' -
Generator reference',' - Bank of 63 comparators',' - Converter digital to thermometer',' - Binary
Logic converter'));
                        case 3
                        % Case 4-3-1-3:
                        set(handles.description,'String',char('','Simulink archive where you can find
all examples'));
                end
                case 2
                    switch m4
                        case 1
                        % Case 4-3-2-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other examples'));
                end
            end
    end
  end
else
  % Don't push any find button
end
```

- **Archive.m**

```
% Function to open the selected archive. Also, if the user doesn't push the
% find button,it will appear the text with the description block.

function modelsimulated = archive(m1,m2,m3,m4,open,handles)

modelsimulated = '-';
if open == 1
   switch m1
      case 1
         switch m2
            case 1
               switch m3
                  case 1
                     switch m4
                        case 1
                           % Case 1-1-1-1:
                           set(handles.description,'String',char('','Library where you can find all
models of TD Sigma Delta: Ideal Blocks'));
                           arch_library_comp(m1,m2);
                     end
                  case 2
                     switch m4
                        case 1
                           % Case 1-1-2-1:
                           set(handles.description,'String',char('','Some models of Ideal integrators'));
                           arch_model_comp(m1,m2,m3,m4);
                     end
                  case 3
                     switch m4
                        case 1
                           % Case 1-1-3-1:
                           set(handles.description,'String',char('','Some model of Ideal comparators'));
                           arch_model_comp(m1,m2,m3,m4);
                     end
                  case 4
                     switch m4
                        case 1
                           % Case 1-1-4-1:
                           set(handles.description,'String',char('','Some model of Ideal quantizers'));
                           arch_model_comp(m1,m2,m3,m4);
                     end
                  case 5
                     switch m4
                        case 1
                           % Case 1-1-5-1:
                           set(handles.description,'String',char('','Simulink archive where you can find
all library models'));
                           arch_model_comp(m1,m2,m3,m4);
                     end
                  case 6
                     switch m4
                        case 1
                           % Case 1-1-6-1:
                           set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
```

```
                    arch_model_comp(m1,m2,m3,m4);
                end
            end
        case 2
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 1-2-1-1:
                            set(handles.description,'String',char('','Library where you can find all
models of TD Sigma Delta: No-ideal Blocks'));
                            arch_library_comp(m1,m2);
                    end
                case 2
                    switch m4
                        case 1
                            % Case 1-2-2-1:
                            set(handles.description,'String',char('','Model to simulate the switch thermal
noise of SC integrator'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 2
                            % Case 1-2-2-2:
                            set(handles.description,'String',char('','Model to simulate the operational
amplifier noise'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 3
                            % Case 1-2-2-3:
                            set(handles.description,'String',char('','Model to simulate the operational
finite gain'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 4
                            % Case 1-2-2-4:
                            set(handles.description,'String',char('','Model to simulate the operational
slew-rate and gain-bandwith product'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 5
                            % Case 1-2-2-5:
                            set(handles.description,'String',char('','Model to simulate the operational
amplifier saturation voltages'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 6
                            % Case 1-2-2-6:
                            set(handles.description,'String',char('','Model to simulate all no-idealities in
the same integrator'));
                            arch_model_comp(m1,m2,m3,m4);
                    end
                case 3
                    switch m4
                        case 1
                            % Case 1-2-3-1:
                            set(handles.description,'String',char('','Model to simulate the hysteresis of
comparators'));
                            arch_model_comp(m1,m2,m3,m4);
                        case 2
                            % Case 1-2-3-2:
                            set(handles.description,'String',char('','Model to simulate the offset of
comparators'));
                            arch_model_comp(m1,m2,m3,m4);
                    end
```

```
            case 4
               switch m4
                  case 1
                     % Case 1-2-4-1:
                     set(handles.description,'String',char('','Model to simulate the clock jitter. It
block should be put it at the input'));
                     arch_model_comp(m1,m2,m3,m4);
               end
            case 5
               switch m4
                  case 1
                     % Case 1-2-5-1:
                     set(handles.description,'String',char('','Simulink archive where you can find
all models'));
                     arch_model_comp(m1,m2,m3,m4);
               end
            case 6
               switch m4
                  case 1
                     % Case 1-2-6-1:
                     set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
                     arch_model_comp(m1,m2,m3,m4);
               end
         end
      case 3
         switch m3
            case 1
               switch m4
                  case 1
                     % Case 1-3-1-1:
                     set(handles.description,'String',char('','Library where you can find all
examples of TD Sigma Delta'));
                     modelsimulated = arch_library_comp(m1,m2);
                  case 2
                     % Case 1-3-1-2:
                     set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - Low-Pass','  - 2º order','  - Simple','  - Ideal components'));
                     modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 3
                     % Case 1-3-1-3:
                     set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - Low-Pass','  - 2º order','  - Simple','  - Some real components'));
                     modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 4
                     % Case 1-3-1-4:
                     set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - ADSL+ application','  - Low-Pass','  - 2º order','  - Cascade','  - Ideal components'));
                     modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 5
                     % Case 1-3-1-5:
                     set(handles.description,'String',char('Example:','  - Time Discrete Sigma
Delta','  - ADSL+ application','  - Low-Pass','  - 2º order','  - Cascade','  - Some real
components'));
                     modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 6
                     % Case 1-3-1-6:
                     set(handles.description,'String',char('','Simulink archive where you can find
all examples'));
```

```
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
            case 2
                switch m4
                    case 1
                        % Case 1-3-2-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other examples'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
        end
    end
case 2  % TC Sigma Delta
    switch m2
        case 1
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-1-1-1:
                    end
            end
        case 2
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-2-1-1:
                    end
            end
        case 3
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 2-3-1-1:
                    end
            end
    end
case 3  % Pipeline
    switch m2
        case 1
            switch m3
                case 1
                    switch m4
                        case 1
                            % Case 3-1-1-1:
                            set(handles.description,'String',char('','Library where you can find all
models of Pipeline: Ideal Blocks.'));
                            arch_library_comp(m1,m2);
                    end
                case 2
                    switch m4
                        case 1
                            % Case 3-1-2-1:
                            set(handles.description,'String',char('','Ideal model of analog to digital
converter stage.','','It is usually referred to as a redundant signed digit (RSD) converter.','' Also
you can find a generic model.'));
                            arch_model_comp(m1,m2,m3,m4);
```

```
            end
        case 3
            switch m4
                case 1
                    % Case 3-1-3-1:
                    set(handles.description,'String',char('','Ideal model of digital to analog converter stage.','','It is usually referred to as a redundant signed digit (RSD) converter.','' Also you can find a generic model.'));
                    arch_model_comp(m1,m2,m3,m4);
            end
        case 4
            switch m4
                case 1
                    % Case 3-1-4-1:
                    set(handles.description,'String',char('','Ideal model to simulate the sample and hold. Also, this block includes the residue gain in the stage.','' Also you can find a generic model.'));
                    arch_model_comp(m1,m2,m3,m4);
            end
        case 5
            switch m4
                case 1
                    % Case 3-1-5-1:
                    set(handles.description,'String',char('','Model that conteins a complete stage: ADC, DCA y SHA','','ADC: analog to digital converter','DCA: digital to analog converter','SHA: residue gain, sample and hold'));
                    arch_model_comp(m1,m2,m3,m4);
            end
        case 6
            switch m4
                case 1
                    % Case 3-1-6-1:
                    set(handles.description,'String',char('','Model that conteins a last stage: only ADC','','ADC: analog to digital converter'));
                    arch_model_comp(m1,m2,m3,m4);
            end
        case 7
            switch m4
                case 1
                    % Case 3-1-7-1:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 3 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                case 2
                    % Case 3-1-7-2:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 4 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                case 3
                    % Case 3-1-7-3:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 5 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                case 4
                    % Case 3-1-7-4:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 6 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                case 5
```

```
                    % Case 3-1-7-5:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 7 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 6
                    % Case 3-1-7-6:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 8 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 7
                    % Case 3-1-7-7:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 9 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 8
                    % Case 3-1-7-8:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 10 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 9
                    % Case 3-1-7-9:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 11 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 10
                    % Case 3-1-7-10:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 12 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 11
                    % Case 3-1-7-11:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 13 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 12
                    % Case 3-1-7-12:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 14 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 13
                    % Case 3-1-7-13:
                    set(handles.description,'String',char('','Model to make the digital correction pipeline to 15 bits','','Valid for RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 14
                    % Case 3-1-7-14:
                    set(handles.description,'String',char('','Model to make the generic correction pipeline with 2 stages','','You could get 2*(N-1)+1 bits where N represents the number of bits of each stage.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 15
                    % Case 3-1-7-15:
                    set(handles.description,'String',char('','Model to make the generic correction pipeline with 3 stages','','You could get 3*(N-1)+1 bits where N represents the number of bits of each stage.','','Also, there is a model with different output bits in each stage.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 16
                    % Case 3-1-7-16:
                    set(handles.description,'String',char('','Model to make the generic correction
```

```
pipeline with 4 stages','','You could get 4*(N-1)+1 bits where N represents the number of bits of each stage.','','Also, there is a model with different output bits in each stage.'));
                        arch_model_comp(m1,m2,m3,m4);
                    case 17
                        % Case 3-1-7-17:
                        set(handles.description,'String',char('','Models to make the digital correction pipeline in the range between 3 and 15 bits and others','','Valid for RSD and generic stages.','','In this simulink archive you can save other models that you create.'));
                        arch_model_comp(m1,m2,m3,m4);
                end
            case 8
                switch m4
                    case 1
                        % Case 3-1-8-1:
                        set(handles.description,'String',char('','Simulink archive where you can find all models.'));
                        arch_model_comp(m1,m2,m3,m4);
                end
            case 9
                switch m4
                    case 1
                        % Case 3-1-9-1:
                        set(handles.description,'String',char('','Simulink archive where you can put other models.'));
                        arch_model_comp(m1,m2,m3,m4);
                end
        end
    case 2
        switch m3
            case 1
                switch m4
                    case 1
                        % Case 3-2-1-1:
                        set(handles.description,'String',char('','Library where you can find all models of Pipeline: No-ideal Blocks.'));
                        arch_library_comp(m1,m2);
                end
            case 2
                switch m4
                    case 1
                        % Case 3-2-2-1:
                        set(handles.description,'String',char('','Model to simulate the ADC with offset in reference voltage.'));
                        arch_model_comp(m1,m2,m3,m4);
                end
            case 3
                switch m4
                    case 1
                        % Case 3-2-3-1:
                        set(handles.description,'String',char('','Model to simulate the complete stage with finite gain.','','Valid to RSD stages.'));
                        arch_model_comp(m1,m2,m3,m4);
                    case 2
                        % Case 3-2-3-2:
                        set(handles.description,'String',char('','Model to simulate the complete stage with mismatch between capacitors.','','Valid to RSD stages.'));
                        arch_model_comp(m1,m2,m3,m4);
                    case 3
                        % Case 3-2-3-3:
```

```
                    set(handles.description,'String',char('','Model to simulate the complete stage
with SR and BW.','','Valid to RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 4
                    % Case 3-2-3-4:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with offset in reference voltage.','','Valid to RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 5
                    % Case 3-2-3-5:
                    set(handles.description,'String',char('','Model to simulate the complete stage
with jitter noise.','','Put it at the input','','Valid to RSD stages.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
              case 4
                switch m4
                  case 1
                    % Case 3-2-4-1:
                    set(handles.description,'String',char('','Model to simulate the jitter
noise.','','You have to put this block at the input.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
              case 5
                switch m4
                  case 1
                    % Case 3-2-5-1:
                    set(handles.description,'String',char('','Simulink archive where you can find
all models.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
              case 6
                switch m4
                  case 1
                    % Case 3-2-6-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other models.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
            end
          case 3
            switch m3
              case 1
                switch m4
                  case 1
                    % Case 1-3-1-1:
                    set(handles.description,'String',char('','Library where you can find all
examples of Pipeline'));
                    modelsimulated = arch_library_comp(m1,m2);
                  case 2
                    % Case 1-3-1-2:
                    set(handles.description,'String',char('Example:',' - Pipeline',' - 10 bits',' -
Ideal components'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 3
                    % Case 3-3-1-3:
                    set(handles.description,'String',char('Example:',' - Pipeline',' - 10 bits',' -
Real components',' - Finite Gain'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 4
```

```
                    % Case 3-3-1-4:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Mismatch'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 5
                    % Case 3-3-1-5:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Slew-rate and bandwidth'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 6
                    % Case 3-3-1-6:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Offset in reference voltage'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 7
                    % Case 3-3-1-7:
                    set(handles.description,'String',char('Example:','  - Pipeline','  - 10 bits','  -
Real components','  - Jitter noise'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                  case 8
                    % Case 3-3-1-8:
                    set(handles.description,'String',char('','Simulink archive where you can find
all examples'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
              case 2
                switch m4
                  case 1
                    % Case 3-3-2-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other examples'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
            end
        end
    case 4 % Flash
      switch m2
        case 1
          switch m3
            case 1
              switch m4
                case 1
                  % Case 3-1-1-1: Library
                  set(handles.description,'String',char('','Library where you can find all
models of Flash: Ideal Blocks.'));
                  arch_library_comp(m1,m2);
              end
            case 2
              switch m4
                case 1
                  % Case 4-1-2-1:
                  set(handles.description,'String',char('','Ideal model to generate reference
voltages.','','Its output are inputs of comparators'));
                  arch_model_comp(m1,m2,m3,m4);
              end
            case 3
              switch m4
                case 1
                  % Case 4-1-3-1:
```

```
                    set(handles.description,'String',char('','Ideal model of comparator.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 2
                    % Case 4-1-3-2:
                    set(handles.description,'String',char('','Ideal model of comparator with
sample and hold at the input.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
              case 4
                switch m4
                  case 1
                    % Case 4-1-4-1:
                    set(handles.description,'String',char('','Ideal encoder of 63 inputs.'));
                    arch_model_comp(m1,m2,m3,m4);
                end
              case 5
                switch m4
                  case 1
                    % Case 4-1-5-1:
                    set(handles.description,'String',char('','Simulink archive where you can find
all ideal models'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
              case 6
                switch m4
                  case 1
                    % Case 4-1-6-1:
                    set(handles.description,'String',char('','Simulink archive where you can put
other ideal models'));
                    modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
            end
          case 2
            switch m3
              case 1
                switch m4
                  case 1
                    % Case 4-2-1-1: Library
                    set(handles.description,'String',char('','Library where you can find all
models of Flash: No-Ideal Blocks.'));
                    arch_library_comp(m1,m2);
                end
              case 2
                switch m4
                  case 1
                    % Case 4-2-2-1:
                    set(handles.description,'String',char('','Real model to generate reference
voltages. Included offet','','Its output are inputs of comparators'));
                    arch_model_comp(m1,m2,m3,m4);
                  end
              case 3
                switch m4
                  case 1
                    % Case 4-1-3-1:
                    set(handles.description,'String',char('','Real model of comparator.'));
                    arch_model_comp(m1,m2,m3,m4);
                  case 2
                    % Case 4-2-3-2:
                    set(handles.description,'String',char('','Real model of comparator with relay
```

```
and offset.'));
                        arch_model_comp(m1,m2,m3,m4);
                    case 3
                        % Case 4-2-3-3:
                        set(handles.description,'String',char('','Real model of comparator with relay,
offset and pole effect.'));
                        arch_model_comp(m1,m2,m3,m4);
                end
            case 4
                switch m4
                    case 1
                        % Case 4-2-4-1:
                        set(handles.description,'String',char('','Simulink archive where you can find
all real models'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
            case 5
                switch m4
                    case 1
                        % Case 4-2-5-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other real models'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
        end
    case 3
        switch m3
            case 1
                switch m4
                    case 1
                        % Case 4-3-1-1:Library
                        set(handles.description,'String',char('','Library where you can find all
models of Flash: Examples.'));
                        arch_library_comp(m1,m2);
                    case 2
                        % Case 4-3-1-2:
                        set(handles.description,'String',char('','Architecture Flash with:','' -
Generator reference','  - Bank of 63 comparators','  - Converter digital to thermometer','  - Binary
Logic converter'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                    case 3
                        % Case 4-3-1-3:
                        set(handles.description,'String',char('','Simulink archive where you can find
all examples'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
            case 2
                switch m4
                    case 1
                        % Case 4-3-2-1:
                        set(handles.description,'String',char('','Simulink archive where you can put
other examples'));
                        modelsimulated = arch_model_comp(m1,m2,m3,m4);
                end
        end
    end
  end
else
  % Don't push any open button
```

end

- **Arch_library_comp.m**

```
% Open a windows with all components.

function modelsimulated = arch_library_comp(m1,m2)

global modelsimulated;

diract=pwd;
switch m1
   case 1
      % Library of Discrete Time Sigma Delta
      switch m2
         case 1
            cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
         case 2
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
         case 3
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
      end
   case 2
      % Library of Continuous Time Sigma Delta
      switch m2
         case 1
            cd 'Theory Model/Continuous Time Sigma Delta/Ideal Blocks';
         case 2
            cd 'Theory Model/Continuous Time Sigma Delta/No-ideal Blocks';
         case 3
            cd 'Theory Model/Continuous Time Sigma Delta/Examples';
      end
   case 3
      % Library of Pipeline
      switch m2
         case 1
            cd 'Theory Model/Pipeline/Ideal Blocks';
         case 2
            cd 'Theory Model/Pipeline/No-ideal Blocks';
         case 3
            cd 'Theory Model/Pipeline/Examples';
      end
   case 4
      % Library of Flash
      switch m2
         case 1
            cd 'Theory Model/Flash/Ideal Blocks';
         case 2
            cd 'Theory Model/Flash/No-ideal Blocks';
         case 3
            cd 'Theory Model/Flash/Examples';
      end
end

dirlib=pwd;
path(path,dirlib);

% Open a dialog box for selecting the file to be opened,
% starting in Library directory
```

```
[filename, pathname] = uigetfile( ...
   {'*.mdl', 'All Model and Library Files (*.mdl)'; ...
      '*.*','All Files (*.*)'}, ...
   'Library: Open your model');
cd(diract);
% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
   return
% Otherwise construct the fullfilename and Check and load the file.
else
   File = fullfile(pathname,filename);
   open_system(File);
end

modelsimulated = File;
```

- **Arch_model_comp.m**

```
% Open a selected model.

function modelsimulated = arch_model_comp(m1,m2,m3,m4)

diract=pwd;

switch m1
   case 1
      switch m2
         case 1
            switch m3
               % TDSD: Ideal Blocks
               case 1
                  switch m4
                     case 1
                        % Case 1-1-1-1: Library
                  end
               case 2
                  switch m4
                     case 1
                        % Case 1-1-2-1:
                        cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
                        filename = 'integrator_ideal';
                  end
               case 3
                  switch m4
                     case 1
                        % Case 1-1-3-1:
                        cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
                        filename = 'comparator_ideal';
                  end
               case 4
```

```
            switch m4
               case 1
                  % Case 1-1-4-1:
                  cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
                  filename = 'quantizer_ideal';
            end
         case 5
            switch m4
               case 1
                  % Case 1-1-5-1:
                  cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
                  filename = 'tdsigmadelta_ideal_all';
            end
         case 6
            switch m4
               case 1
                  % Case 1-1-6-1:
                  cd 'Theory Model/Discrete Time Sigma Delta/Ideal Blocks';
                  filename = 'tdsigmadelta_ideal_others';
            end
      end
   case 2
      switch m3
         % TDSD: No-ideal Blocks
         case 1
            switch m4
               case 1
                  % Case 1-2-1-1: Library
            end
         case 2
            switch m4
               case 1
                  % Case 1-2-2-1:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_ktc';
               case 2
                  % Case 1-2-2-2:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_op';
               case 3
                  % Case 1-2-2-3:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_gainfinite';
               case 4
                  % Case 1-2-2-4:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_sr_gbw';
               case 5
                  % Case 1-2-2-5:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_sat';
               case 6
                  % Case 1-2-2-6:
                  cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
                  filename = 'integr_real_complete';
            end
         case 3
            switch m4
               case 1
```

```
            % Case 1-2-3-1:
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
            filename = 'comp_real_hysteresis';
          case 2
            % Case 1-2-3-2:
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
            filename = 'comp_real_offset';
        end
      case 4
        switch m4
          case 1
            % Case 1-2-4-1:
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
            filename = 'jitter';
        end
      case 5
        switch m4
          case 1
            % Case 1-2-5-1:
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
            filename = 'tdsigmadelta_noideal_all';
        end
      case 6
        switch m4
          case 1
            % Case 1-2-6-1:
            cd 'Theory Model/Discrete Time Sigma Delta/No-ideal Blocks';
            filename = 'tdsigmadelta_noideal_others';
        end
    end
  case 3
    switch m3
      % TDSD: Examples
      case 1
        switch m4
          case 1
            % Case 1-3-1-1: Library
          case 2
            % Case 1-3-1-2:
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
            filename = 'tdsigmadelta_lp_2_ideal_simple.mdl';
          case 3
            % Case 1-3-1-3:
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
            filename = 'tdsigmadelta_lp_2_real_simple.mdl';
          case 4
            % Case 1-3-1-4:
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
            filename = 'tdsigmadelta_lp_2_ideal_cascade.mdl';
          case 5
            % Case 1-3-1-5:
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
            filename = 'tdsigmadelta_lp_2_real_cascade.mdl';
          case 6
            % Case 1-3-1-6:
            cd 'Theory Model/Discrete Time Sigma Delta/Examples';
            filename = 'tdsigmadelta_example_all';
        end
      case 2
```

```
            switch m4
               case 1
                  % Case 1-3-2-1:
                  cd 'Theory Model/Discrete Time Sigma Delta/Examples';
                  filename = 'tdsigmadelta_example_others';
            end
         end
      end
   case 2  % TC Sigma Delta
      switch m2
         case 1
            switch m3
               case 1
                  switch m4
                     case 1
                        % Case 2-1-1-1:
                  end
            end
         case 2
            switch m3
               case 1
                  switch m4
                     case 1
                        % Case 2-2-1-1:
                  end
            end
         case 3
               switch m3
                  case 1
                     switch m4
                        case 1
                           % Case 2-3-1-1:
                     end
               end
      end
   case 3  % Pipeline
         switch m2
            case 1
               switch m3
                  case 1
                     switch m4
                        case 1
                           % Case 4-1-1-1: Library
                     end
                  case 2
                     switch m4
                        case 1
                           % Case 3-1-2-1:
                           cd 'Theory Model/Pipeline/Ideal Blocks';
                           filename = 'ADC_ideal';
                     end
                  case 3
                     switch m4
                        case 1
                           % Case 3-1-3-1:
                           cd 'Theory Model/Pipeline/Ideal Blocks';
                           filename = 'DAC_ideal';
                     end
                  case 4
```

```
    switch m4
      case 1
        % Case 3-1-4-1:
        cd 'Theory Model/Pipeline/Ideal Blocks';
        filename = 'SHA_ideal';
    end
  case 5
    switch m4
      case 1
        % Case 3-1-5-1:
        cd 'Theory Model/Pipeline/Ideal Blocks';
        filename = 'stagecomplete_ideal';
    end
  case 6
    switch m4
      case 1
        % Case 3-1-6-1:
        cd 'Theory Model/Pipeline/Ideal Blocks';
        filename = 'stagelast_ideal';
    end
  case 7
    switch m4
      case 1
        % Case 3-1-7-1:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_3b_rsd';
      case 2
        % Case 3-1-7-2:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_4b_rsd';
      case 3
        % Case 3-1-7-3:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_5b_rsd';
      case 4
        % Case 3-1-7-4:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_6b_rsd';
      case 5
        % Case 3-1-7-5:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_7b_rsd';
      case 6
        % Case 3-1-7-6:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_8b_rsd';
      case 7
        % Case 3-1-7-7:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_9b_rsd';
      case 8
        % Case 3-1-7-8:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_10b_rsd';
      case 9
        % Case 3-1-7-9:
        cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
        filename = 'digcorrection_11b_rsd';
      case 10
```

```
                % Case 3-1-7-10:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_12b_rsd';
            case 11
                % Case 3-1-7-11:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_13b_rsd';
            case 12
                % Case 3-1-7-12:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_14b_rsd';
            case 13
                % Case 3-1-7-13:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_15b_rsd';
            case 14
                % Case 3-1-7-14:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_2stg_gen';
            case 15
                % Case 3-1-7-15:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_3stg_gen';
            case 16
                % Case 3-1-7-16:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_4stg_gen';
            case 17
                % Case 3-1-7-17:
                cd 'Theory Model/Pipeline/Ideal Blocks/Digital Correction';
                filename = 'digcorrection_all';
        end
    case 8
        switch m4
            case 1
                % Case 3-1-8-1:
                cd 'Theory Model/Pipeline/Ideal Blocks';
                filename = 'pipeline_ideal_all';
        end
    case 9
        switch m4
            case 1
                % Case 3-1-9-1:
                cd 'Theory Model/Pipeline/Ideal Blocks';
                filename = 'pipeline_ideal_others';
        end
  end
case 2
  switch m3
    case 1
      switch m4
        case 1
          % Case 3-2-1-1: Library
      end
    case 2
      switch m4
        case 1
          % Case 3-2-2-1:
          cd 'Theory Model/Pipeline/No-ideal Blocks';
```

```
            filename = 'dac_real_voffset';
        end
    case 3
        switch m4
            case 1
                % Case 3-2-3-1:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'stage_real_finitegain';
            case 2
                % Case 3-2-3-2:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'stage_real_mismatch';
            case 3
                % Case 3-2-3-3:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'stage_real_sr_bw';
            case 4
                % Case 3-2-3-4:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'stage_real_voffset';
            case 5
                % Case 3-2-3-5:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'stage_real_complete';
        end
    case 4
        switch m4
            case 1
                % Case 3-2-4-1:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'jitter';
        end
    case 5
        switch m4
            case 1
                % Case 3-2-5-1:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'pipeline_noideal_all';
        end
    case 6
        switch m4
            case 1
                % Case 3-2-6-1:
                cd 'Theory Model/Pipeline/No-ideal Blocks';
                filename = 'pipeline_noideal_others';
        end
  end
case 3
  switch m3
    case 1
      switch m4
        case 1
          % Case 1-3-1-1:Library
        case 2
          % Case 1-3-1-2:
          cd 'Theory Model/Pipeline/Examples';
          filename = 'pipeline_10b_ideal.mdl';
        case 3
          % Case 3-3-1-3:
```

```
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_10b_finitegain.mdl';
              case 4
                % Case 3-3-1-4:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_10b_mismatch.mdl';
              case 5
                % Case 3-3-1-5:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_10b_sr_bw.mdl';
              case 6
                % Case 3-3-1-6:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_10b_voffset.mdl';
              case 7
                % Case 3-3-1-7:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_10b_jitter.mdl';
              case 8
                % Case 3-3-1-8:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_example_all.mdl';
            end
          case 2
            switch m4
              case 1
                % Case 3-3-2-1:
                cd 'Theory Model/Pipeline/Examples';
                filename = 'pipeline_example_others.mdl';
            end
        end
    end
case 4 % Flash
  switch m2
      case 1
        switch m3
          case 1
            switch m4
              case 1
                % Case 3-1-1-1: Library
            end
          case 2
            switch m4
              case 1
                % Case 4-1-2-1:
                cd 'Theory Model/Flash/Ideal Blocks';
                filename = 'ref_gen_63';
            end
          case 3
            switch m4
              case 1
                % Case 4-1-3-1:
                cd 'Theory Model/Flash/Ideal Blocks';
                filename = 'comp';
              case 2
                % Case 4-1-3-2:
                cd 'Theory Model/Flash/Ideal Blocks';
                filename = 'comp_ts';
            end
```

```
        case 4
          switch m4
            case 1
              % Case 4-1-4-1:
              cd 'Theory Model/Flash/Ideal Blocks';
              filename = 'encoder_63';
          end
        case 5
          switch m4
            case 1
              % Case 4-1-5-1:
              cd 'Theory Model/Flash/Ideal Blocks';
              filename = 'flash_ideal_all';
          end
        case 6
          switch m4
            case 1
              % Case 4-1-6-1:
              cd 'Theory Model/Flash/Ideal Blocks';
              filename = 'flash_ideal_others';
          end
      end
    case 2
      switch m3
        case 1
          switch m4
            case 1
              % Case 3-2-1-1: Library
          end
        case 2
          switch m4
            case 1
              % Case 3-2-2-1:
              cd 'Theory Model/Flash/No-ideal Blocks';
              filename = 'ref_gen_63_offset';
            end
        case 3
          switch m4
            case 1
              % Case 4-2-3-1:
              cd 'Theory Model/Flash/No-ideal Blocks';
              filename = 'comp_offset';
            case 2
              % Case 4-2-3-2:
              cd 'Theory Model/Flash/No-ideal Blocks';
              filename = 'comp_offset_relay';
            case 3
              % Case 4-2-3-3:
              cd 'Theory Model/Flash/No-ideal Blocks';
              filename = 'comp_offset_pole';
          end
        case 4
          switch m4
            case 1
              % Case 4-2-4-1:
              cd 'Theory Model/Flash/No-ideal Blocks';
              filename = 'flash_noideal_all';
          end
        case 5
```

```
                switch m4
                   case 1
                      % Case 4-2-5-1:
                      cd 'Theory Model/Flash/No-ideal Blocks';
                      filename = 'flash_noideal_others';
                end
             end
          case 3
             switch m3
                case 1
                   switch m4
                      case 1
                         % Case 4-3-1-1:Library
                      case 2
                         % Case 4-3-1-2:
                         cd 'Theory Model/Flash/Examples';
                         filename = 'flash_offset_relay.mdl';
                      case 3
                         % Case 4-3-1-3:
                         cd 'Theory Model/Pipeline/Examples';
                         filename = 'flash_example_all.mdl';
                   end
                case 2
                   switch m4
                      case 1
                         % Case 4-3-2-1:
                         cd 'Theory Model/Pipeline/Examples';
                         filename = 'flash_example_others.mdl';
                   end
             end
      end
end

dirnew=pwd;

File = fullfile(dirnew,filename);
modelsimulated = File;
open_system(File);

cd(diract);
```

- **State.m**

```
% Function to load the value of variables that we use it to simulate the
% model.

function handles = state(m1,m3,m4)

global handles;

switch m1
  case 1
    switch m3
      % TDSD: Examples
      case 1
        switch m4
          case 1
            % Case 1-3-1-1: Library
          case 2
            % Case 1-3-1-2: tdsigmadelta_lp_2_ideal_simple
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N = 8;
            handles.bandwidth = 250;
            handles.A = 0.5;
            handles.fsin = 70;
            handles.nsamples = 10;
            handles.minsignalband = 10;
              % No-idealities
            handles.AodB = 0;
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
          case 3
            % Case 1-3-1-3: tdsigmadelta_lp_2_real_simple
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N = 8;
            handles.bandwidth = 250;
            handles.A = 0.5;
            handles.fsin = 70;
            handles.nsamples = 10;
            handles.minsignalband = 10;
              % No-idealities
            handles.AodB = 22;
            handles.sigmajitter =30e-6;
            handles.SR = 1e5;
            handles.Tau = 1e-5;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 1e-12; % Cs
```

```
            handles.Aux2 = 0.1e-3; % Vn
            handles.Aux3 = 0.65; % Vmax/Vmin
            handles.Aux4 = 0;
        case 4
            % Case 1-3-1-4: tdsigmadelta_lp_2_ideal_cascade
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N = 8;
            handles.bandwidth = 250;
            handles.A = 0.5;
            handles.fsin = 70;
            handles.nsamples = 10;
            handles.minsignalband = 10;
              % No-idealities
            handles.AodB = 0;
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
        case 5
            % Case 1-3-1-5: tdsigmadelta_lp_2_real_cascade
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N = 8;
            handles.bandwidth = 250;
            handles.A = 0.5;
            handles.fsin = 70;
            handles.nsamples = 10;
            handles.minsignalband = 10;
              % No-idealities
            handles.AodB = 22;
            handles.sigmajitter =30e-6;
            handles.SR = 1e5;
            handles.Tau = 1e-5;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 1e-12; % Cs
            handles.Aux2 = 0.1e-3; % Vn
            handles.Aux3 = 0.65; % Vmax/Vmin
            handles.Aux4 = 0;
        case 6
            % Case 1-3-1-6: All
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N = 8;
            handles.bandwidth = 250;
            handles.A = 0.5;
            handles.fsin = 70;
            handles.nsamples = 10;
            handles.minsignalband = 10;
              % No-idealities
```

```
                handles.AodB = 0;
                handles.sigmajitter =0;
                handles.SR = 0;
                handles.Tau = 0;
                handles.Eps = 0;
                handles.sigma = 0;
                handles.Aux1 = 0;
                handles.Aux2 = 0;
                handles.Aux3 = 0;
                handles.Aux4 = 0;
            end
        case 2
            switch m4
                case 1
                    % Case 1-3-2-1: Others
            end
    end
  case 2  % TC Sigma Delta

  case 3  % Pipeline
    switch m3
      case 1
        switch m4
          case 1
            % Case 1-3-1-1: Library
          case 2
            % Case 1-3-1-2: pipeline_10b_ideal
            handles.Ts  =  1/(2*160*2.2e6);  %  Ts=1/fs=1/M*fb=1/(2*M*BW)  con  fb:frec  de
Nyquist
            handles.FS = 2;
            handles.N=14;
            handles.bandwidth=2.2e6;
            handles.A=0.5;
            handles.fsin=550.0123e3;
            handles.nsamples=10;
            handles.minsignalband=10;
              % No-idealities
            handles.AodB = 0;
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
          case 3
            % Case 1-3-1-3: pipeline_10b_finitegain
            handles.Ts  =  1/(2*160*2.2e6);  %  Ts=1/fs=1/M*fb=1/(2*M*BW)  con  fb:frec  de
Nyquist
            handles.FS = 2;
            handles.N=14;
            handles.bandwidth=2.2e6;
            handles.A=0.5;
            handles.fsin=550.0123e3;
            handles.nsamples=10;
            handles.minsignalband=10;
              % No-idealities
```

```
            handles.AodB = 20*log10(1500);
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
        case 4
            % Case 1-3-1-4: pipeline_10b_mismatch
            handles.Ts = 1/(2*160*2.2e6); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N=14;
            handles.bandwidth=2.2e6;
            handles.A=0.5;
            handles.fsin=550.0123e3;
            handles.nsamples=10;
            handles.minsignalband=10;
              % No-idealities
            handles.AodB = 20*log10(1500);
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
        case 5
            % Case 1-3-1-5: pipeline_10b_sr_bw
            handles.Ts = 1/(2*160*2.2e6); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
            handles.N=14;
            handles.bandwidth=2.2e6;
            handles.A=0.5;
            handles.fsin=550.0123e3;
            handles.nsamples=10;
            handles.minsignalband=10;
              % No-idealities
            handles.AodB = 20*log10(1500);
            handles.sigmajitter =0;
            handles.SR = 0;
            handles.Tau = 0;
            handles.Eps = 0;
            handles.sigma = 0;
            handles.Aux1 = 0;
            handles.Aux2 = 0;
            handles.Aux3 = 0;
            handles.Aux4 = 0;
        case 6
            % Case 1-3-1-6: pipeline_10b_voffset
            handles.Ts = 1/(2*160*2.2e6); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
            handles.FS = 2;
```

```
                handles.N=14;
                handles.bandwidth=2.2e6;
                handles.A=0.5;
                handles.fsin=550.0123e3;
                handles.nsamples=10;
                handles.minsignalband=10;
                  % No-idealities
                handles.AodB = 20*log10(1500);
                handles.sigmajitter =0;
                handles.SR = 0;
                handles.Tau = 0;
                handles.Eps = 0;
                handles.sigma = 0;
                handles.Aux1 = 0;
                handles.Aux2 = 0;
                handles.Aux3 = 0;
                handles.Aux4 = 0;
            case 7
                % Case 1-3-1-7: pipeline_10b_real
                handles.Ts = 1/(2*160*2.2e6); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
                handles.FS = 2;
                handles.N=14;
                handles.bandwidth=2.2e6;
                handles.A=0.5;
                handles.fsin=550.0123e3;
                handles.nsamples=10;
                handles.minsignalband=10;
                  % No-idealities
                handles.AodB = 20*log10(1500);
                handles.sigmajitter =0;
                handles.SR = 0;
                handles.Tau = 0;
                handles.Eps = 0;
                handles.sigma = 0;
                handles.Aux1 = 0;
                handles.Aux2 = 0;
                handles.Aux3 = 0;
                handles.Aux4 = 0;
            case 8
                % Case 1-3-1-8: All
        end
      case 2
        switch m4
          case 1
            % Case 1-3-2-1: Others
        end
    end
  case 4
    switch m3
      % Flash: Examples
      case 1
        switch m4
          case 1
            % Case 4-3-1-1: Library
          case 2
            % Case 4-3-1-2: flash_offset_relay
            handles.Ts = 1/(2*32*250); % Ts=1/fs=1/M*fb=1/(2*M*BW) con fb:frec de
Nyquist
```

```
                handles.FS = 2;
                handles.N = 8;
                handles.bandwidth = 250;
                handles.A = 0.5;
                handles.fsin = 70;
                handles.nsamples = 10;
                handles.minsignalband = 10;
                  % No-idealities
                handles.AodB = 0;
                handles.sigmajitter =0;
                handles.SR = 0;
                handles.Tau = 0;
                handles.Eps = 0;
                handles.sigma = 0;
                handles.Aux1 = 0;
                handles.Aux2 = 0;
                handles.Aux3 = 0;
                handles.Aux4 = 0;
            end
        case 2
            switch m4
                case 1
                    % Case 1-3-2-1: Others
            end
    end
end
```