Universidad Zaragoza
1542

# Trabajo Fin de Grado

## Medida de corriente y tensión en etapas de potencia para inducción doméstica. Análisis de las alternativas de digitalización

# ANEXOS

Autor

Félix Gómez Aguaviva

Universidad de Zaragoza / Escuela de Ingeniería y Arquitectura
2015

# ANEXO A: CONVERSORES A/D

En este anexo se pretende explicar de forma breve en qué consiste un conversor analógico digital, así como el funcionamiento de varios tipos de estos conversores. Su lectura puede ayudar a la comprensión de ciertos aspectos de la memoria de este Trabajo Fin de Grado.

## 1. CONVERSORES QUE NO UTILIZAN SOBREMUESTREO

Visto de forma general, un conversor analógico digital es un dispositivo electrónico que consta de una entrada analógica, una entrada que establece su referencia de tensión, y una salida digital. Su misión es convertir la señal analógica de entrada en un valor digital que represente el valor de esa señal en el instante de muestreo en comparación con el voltaje de la entrada de referencia [8].

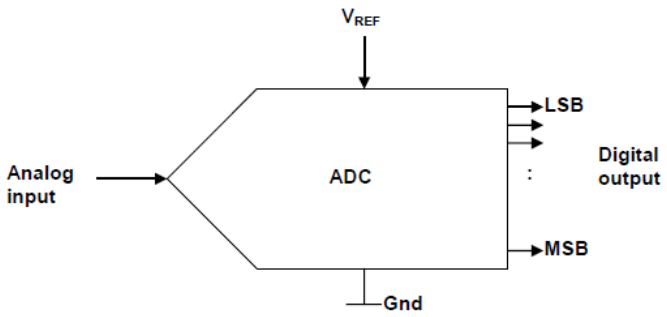


*Figura 43. Símbolo genérico de un ADC*

La tensión de referencia del conversor determina el fondo de escala que se puede digitalizar. Este rango analógico está dividido en un determinado número de sub-rangos, tantos como valores digitales puede tomar la salida ($2^n$, siendo n el número de bits del conversor). Cada uno de estos sub-rangos tendrá el valor de un LSB, que viene determinado por la siguiente expresión:

$$V_{LSB} = \frac{V_{ref}}{2^n - 1} = V/bit$$

Donde $V_{ref}$ es la tensión de referencia, y n el número de bits del conversor.

El dato de salida de un ADC depende de su función de transferencia (si está desplazada o no), y de la codificación del dato de salida (binario sin signo, o en complemento a 2).

A modo de ejemplo sencillo, con un conversor de 8 bits (FdT no desplazada, dato de salida sin signo) que tenga un rango de conversión de 0 a 3.3 V, para un valor analógico de 2.7 V en su entrada se obtendrá como resultado un valor digital en su salida de:

$$nº\ bin = \frac{V_{IN}}{V_{LSB}} = \frac{2.7\ V}{3.3\ V/_{2^8 - 1}} = 208.63 \approx 209\ (11010001)$$

El error cometido al redondear al entero más cercano se denomina error de cuantización. Este error se comete siempre en los ADC, ya que con un número de bits dado es imposible representar exactamente todos los valores que puede tomar una señal continua de entrada.

Existen distintas arquitecturas. A continuación se nombran algunas de las más típicas [9].

### Aproximaciones sucesivas

Es la arquitectura más usada en la mayoría de los ADC comerciales. Consta de un comparador que se encarga de comparar la entrada analógica con el valor de salida de un DAC de un número de bits igual que el del ADC.

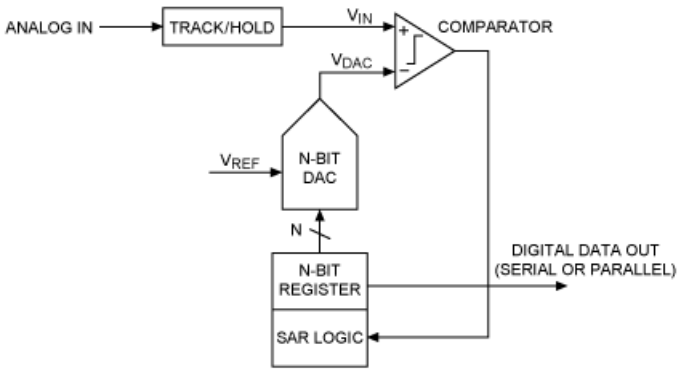


*Figura 44. Bloques que componen la arquitectura de aproximaciones sucesivas de un ADC*

Tiene una lógica que implementa un algoritmo de búsqueda binaria encargado de generar el dato de entrada de este DAC en función del resultado de la comparación.

Tras n comparaciones, siendo n el número de bits del ADC, es posible encontrar el dato de salida que corresponde al valor analógico de entrada.

### Flash

Este tipo de arquitectura es mucho más cara que la anterior, y no puede tener una precisión de muchos bits debido al elevado número de componentes que requiere. Por otro lado, es sin duda la que realiza la conversión de forma más rápida.

Lo que hace es comparar simultáneamente la tensión de entrada con cada una de las tensiones que se pueden representar con el número de bits de salida disponibles.

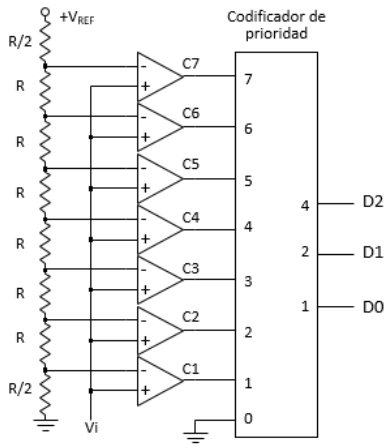


*Figura 45. Componentes de la arquitectura flash de un ADC*

## Two Step

Realiza la conversión en dos pasos, utilizando también una arquitectura flash. Primero se hace una división gruesa de la tensión de entrada para después hacer ya una división fina y obtener el resultado final.

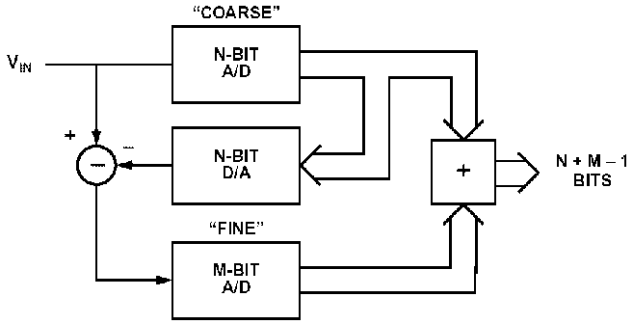


*Figura 46. Bloques que componen la arquitectura Two Step de un ADC*

## Pipeline

Esta arquitectura utiliza de nuevo una arquitectura flash, pero en este caso la conversión se realiza en n pasos, en lugar de en dos.

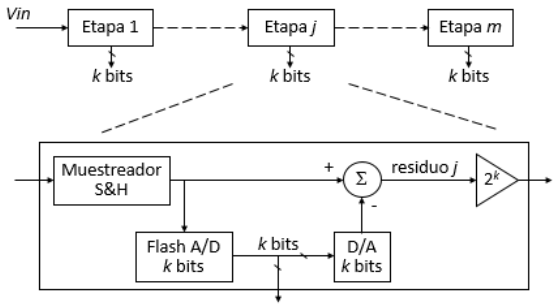


*Figura 47. Bloques que componen la arquitectura Pipeline de un ADC*

## 2. CONVERSORES QUE UTILIZAN SOBREMUESTREO

Este otro tipo de conversores se basan en un principio completamente diferente. Tienen muy poca resolución (1 bit), pero trabajan a una frecuencia muy alta.

Están formados básicamente por un modulador Sigma-Delta (Σ-Δ) seguido de un filtro digital [8]. A continuación se va a explicar más detalladamente cómo funcionan cada uno de estos bloques.

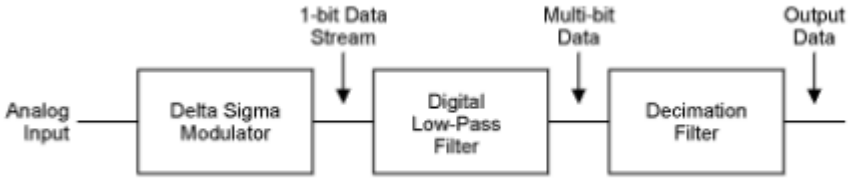


*Figura 48. Bloques que componen un conversor Σ-Δ*

A la hora de convertir una señal analógica continua en una señal digital, que puede tomar una serie de valores discretos, se produce una pérdida de información y una distorsión de la señal original. Este efecto se conoce como ruido de cuantización, y se traduce en un ruido aleatorio que se distribuye por todo el espectro frecuencial de la señal.

El modulador de un conversor Σ-Δ está compuesto por un restador, un integrador y un comparador con un lazo de realimentación que contiene un ADC de un bit, tal y como se puede ver en esta figura:

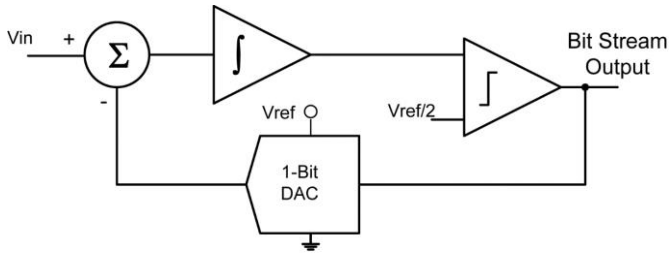


*Figura 49. Bloques que componen el modulador de un conversor Σ-Δ de primer orden*

El objetivo del lazo de realimentación es mantener el valor medio de la salida del integrador cercano al nivel de referencia del comparador.

El modulador se encarga de digitalizar la señal analógica de entrada, y además realiza una función denominada "*noise shaping*", que se encarga de desplazar el ruido de cuantización de baja frecuencia a frecuencias más altas. Concretamente, es el integrador el elemento que produce este efecto.

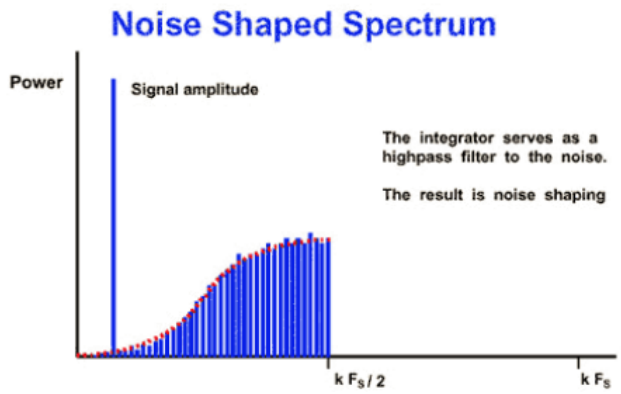


*Figura 50. Efecto del "noise shaping" en el espectro frecuencial de la señal*

En la salida del comparador se obtiene un tren de pulsos de anchura variable, con una densidad de pulsos en alto proporcional a la señal de entrada. Es decir, se genera un mayor número de pulsos en alto cuando la señal de entrada está creciendo, y viceversa cuando está decreciendo. Este tren de pulsos se almacena en un biestable que trabaja a la frecuencia de muestreo, y de ahí pasa directamente al filtro digital. Para una mejor comprensión del proceso, al final de esta sección (figura 53) se pueden ver las formas de onda en la salida de cada elemento que forma el modulador del conversor Σ-Δ.

El filtro digital de un conversor Σ-Δ se encarga de extraer la información de este tren de pulsos. Se trata de un filtro de paso bajo que realiza una media de los datos de un bit que contiene el tren de pulsos, reconstruyendo la señal original de entrada y eliminando el ruido de cuantización que está fuera de la banda frecuencial de interés.

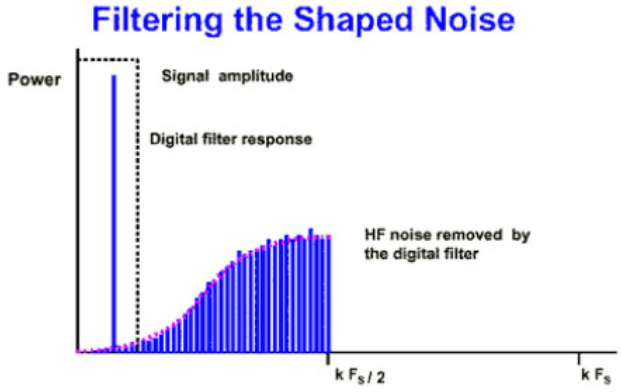


*Figura 51. Espectro frecuencial de la señal de salida tras el filtrado (Línea discontinua)*

Una manera de reducir el ruido de cuantización de la banda frecuencial de interés es colocando en el modulador dos o más integradores en lugar de uno, es decir, aumentando el orden del modulador [10]. Los moduladores de un orden mayor desplazan el ruido de cuantización de baja frecuencia a frecuencias aún más altas, consiguiendo reducir el ruido que permanece en el espectro frecuencial de interés tras el filtrado.

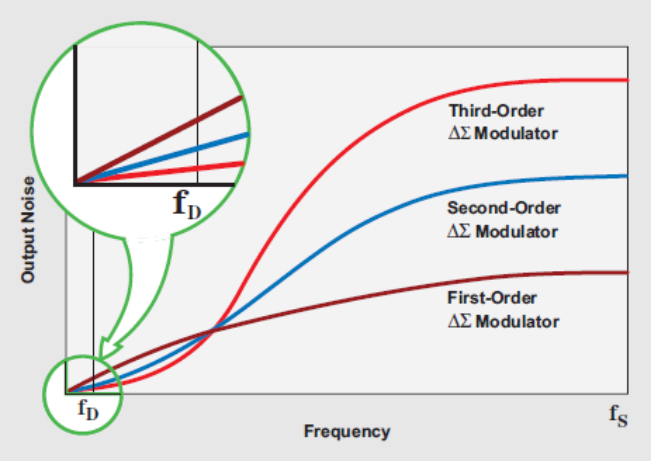


*Figura 52. Ruido de cuantización en función del orden del modulador de un conversor Σ-Δ*

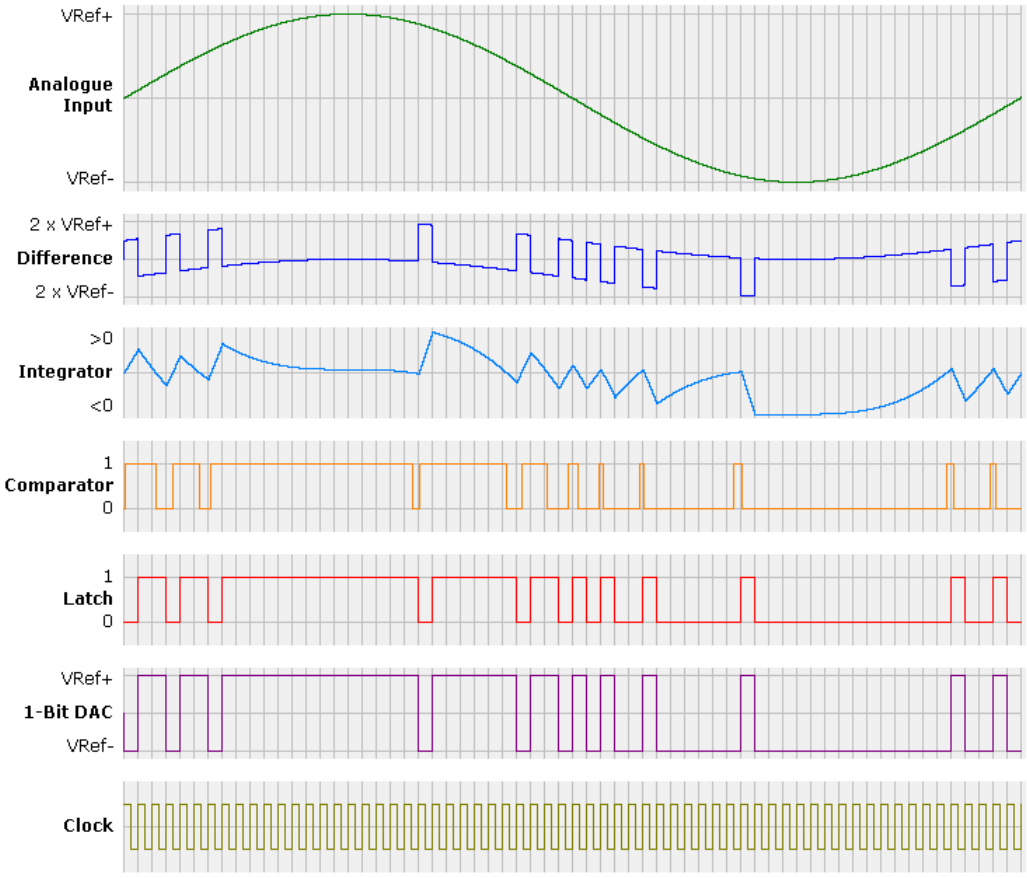


*Figura 53. Formas de onda en la salida de cada elemento que forma el conversor Σ-Δ*

# ANEXO B: CÁLCULOS UTILIZADOS EN DISEÑO Y SIMULACIÓN

## 1. FRECUENCIA DE RESONANCIA DE LA ETAPA

Los valores de los componentes que forman la etapa de potencia son los siguientes:

L= 31 µH
C= 720 nF (Dos en paralelo) → $C_T$ = 1440 nF

Por tanto, según la siguiente expresión, la frecuencia de resonancia de la etapa será:

$$f_0 = \frac{1}{2\pi\sqrt{LC}} = \frac{1}{2\pi\sqrt{31 \cdot 10^{-6} \cdot 1440 \cdot 10^{-9}}} = 23.82\ kHz$$

## 2. GANANCIAS DE LAS ETAPAS DE LOS CONVERSORES Σ-Δ

La señal de entrada al modulador del conversor Σ-Δ de corriente sufre una conversión a tensión y una atenuación para ser adaptada a las características de este.

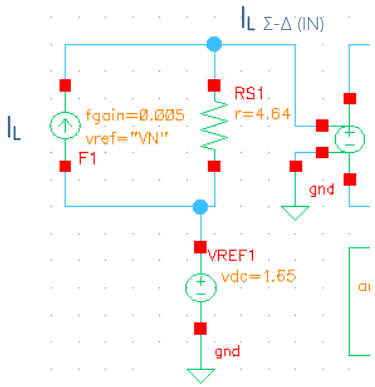


*Figura 54. Etapa de captación de la señal de entrada del conversor Σ-Δ de corriente*

Teniendo en cuenta los elementos que componen la etapa de captación de $I_L$, la señal de entrada al modulador es:

$$I_{L\,\Sigma-\Delta\,(\mathrm{IN})}(V) = I_L \cdot 0.005 \cdot 4.64\,\Omega + 1.65\,V = I_L \cdot 0.0232 + 1.65\,V$$

Esta señal de tensión podrá tomar valores comprendidos entre 0 y 3.3 V, que es la tensión de referencia del conversor.

La salida del filtro sin embargo, por ser el resultado del filtrado de un tren de pulsos digitales tomará valores comprendidos entre 0 y Vdd V. Teniendo en cuenta también este efecto, la ganancia total a la que se ve sometida la señal de entrada será de:

$$I_{L\,\Sigma-\Delta\,(\mathrm{OUT})}\,(V) = I_L \cdot \frac{0.0232 + 1.65\,V}{3.3\,V}$$

La señal de entrada del conversor Σ-Δ de tensión también es necesario atenuarla. En este caso, la atenuación de la señal antes de entrar al modulador se produce únicamente por la ganancia que incorpora la fuente de tensión dependiente con la que se capta la señal, que es de 0.008100512. Teniendo en cuenta las mismas consideraciones que en el caso anterior la expresión de la tensión de salida del conversor queda de la siguiente manera:

$$V_{BUS\,\Sigma-\Delta\,(\mathrm{OUT})} = V_{BUS} \cdot \frac{0.008100512}{3.3}$$

Estas ganancias se tienen en cuenta en simulación, una vez obtenidos los datos de salida de los conversores, para poder comparar las formas de onda resultantes con las de las señales originales.

## 3. GANANCIAS DE LAS ETAPAS PREVIAS DE LOS ADC NYQUIST

Las señales de entrada de los ADC Nyquist hay que adaptarlas de la misma manera que se ha hecho para los conversores Σ-Δ. Es necesario elegir los valores máximos que pueden tomar estas señales, y en base a ellos ajustar el fondo de escala de los ADC para conseguir la mejor resolución que nos permita cada uno.

Ya que la carga del sistema varía en función del recipiente, y por tanto también lo hace la corriente que circula por ella, se han elegido unos valores máximos de $I_L$ y $V_{BUS}$ de 90 A y 340 V, respectivamente.

La forma de conocer la ganancia que es necesario aplicar a cada señal para atenuarla es la siguiente:

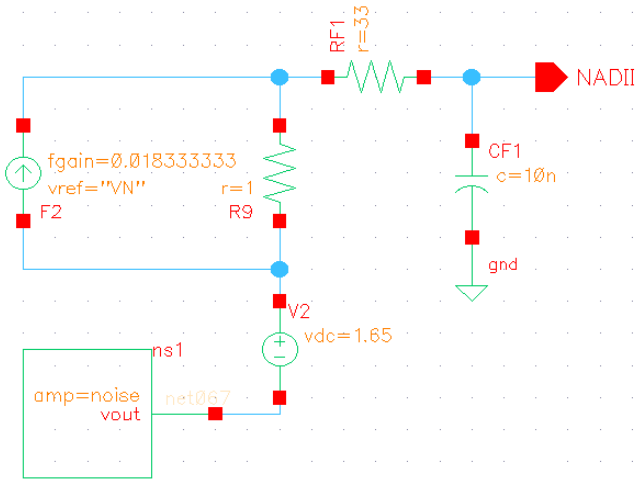


*Figura 55. Etapa de atenuación y filtrado de la señal de entrada del ADC de corriente*

La señal de corriente es una señal de corriente alterna, por tanto a la hora de muestrearla es necesario atenuarla, convertirla en tensión, y posteriormente sumarle un offset para que tome siempre valores positivos. Este offset será de 1.65 V, es decir, la mitad del rango de tensión que es capaz de convertir el ADC.

Según este razonamiento, cuando la corriente tome su valor máximo, se deberán obtener 3.3 V en la entrada del ADC, o lo que es lo mismo, 1.65 V antes de aplicar el offset. La forma de conseguir los 1.65V es haciendo circular 1.65 A a través de una resistencia de 1 Ω. La ganancia necesaria para conseguir 1.65 A a partir del valor máximo que puede tomar la señal de entrada es:

$$I_{L\ max} \cdot G = 1.65\ A \rightarrow G = \frac{1.65\ A}{90\ A} = 0.018333333$$

Mediante esta etapa hemos pasado de una señal en el rango [-90, 90 A] a una señal en el rango [0, 3.3 V].

Con la señal de tensión se usa la misma idea, pero en este caso el cálculo de la ganancia se hace de forma más directa, ya que tan solo hay que pasar una tensión en el rango [0, 340 V] al rango [0, 3.3 V].

$$V_{BUS\ max} \cdot G = 3.3\ V \rightarrow G = \frac{3.3\ V}{340\ V} = 0.009705882$$

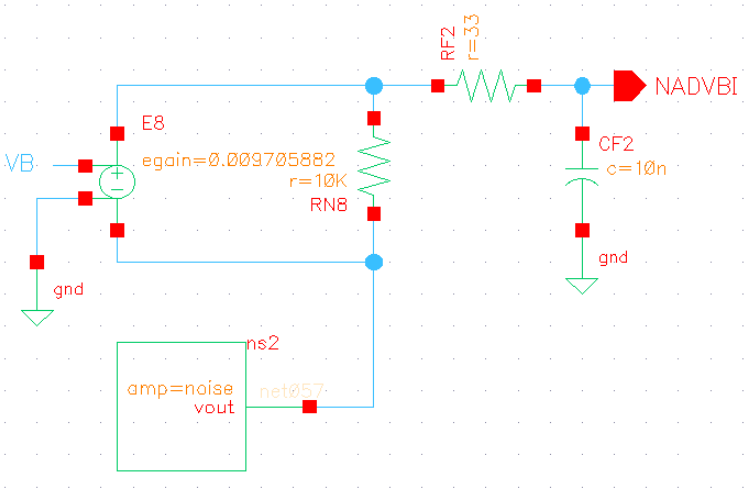


*Figura 56. Etapa de atenuación y filtrado de la señal de entrada del ADC de tensión*

Igual que ocurre con las ganancias utilizadas para atenuar la señal de entrada de los conversores Σ-Δ, estas ganancias se tienen en cuenta en simulación para poder comparar las formas de onda de las señales originales con las obtenidas mediante los ADC Nyquist.

## 4. FILTRO RC

El filtro elegido fue finalmente un filtro RC. Los valores de los componentes se eligieron de manera que la $f_c$ del filtro fuera de 500 kHz.

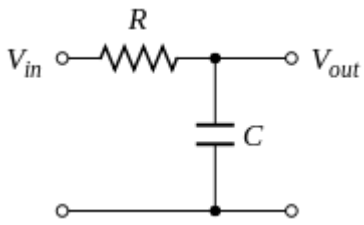


*Figura 57. Filtro RC*

A continuación se desarrolla la expresión que permite relacionar la $f_c$ con los valores de los componentes utilizados.

$$\frac{V_{out}}{V_{in}} = H(j\omega) = \frac{1/j\omega C}{R + 1/j\omega C} = \frac{1}{1 + j\omega RC} \rightarrow |H(j\omega)| = \sqrt{\frac{1^2}{1^2 + \omega^2 R^2 C^2}}$$

$$|H(j\omega_0)| = \sqrt{\frac{1}{1 + {\omega_0}^2 R^2 C^2}} = \frac{1}{\sqrt{2}} \rightarrow \sqrt{1 + {\omega_0}^2 R^2 C^2} = \sqrt{2} \rightarrow {\omega_0}^2 R^2 C^2 = 1 \Rightarrow$$

$$\Rightarrow \omega_0 = \frac{1}{RC} = 2\pi f_c$$

Según esta expresión, si elegimos un valor para el condensador de 10 nF, podemos calcular el valor de R que necesitaremos para conseguir una $f_c$ de 500 kHz.

$$R = \frac{1}{2\pi f_c \cdot C} = \frac{1}{2\pi \cdot 500 \cdot 10^3\ Hz \cdot 10 \cdot 10^{-9}\ F} = 31.83\ \Omega$$

En nuestro diseño utilizaremos un valor de 33 Ω para R, obteniendo así un filtro con una $f_c$ de:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 33\ \Omega \cdot 10 \cdot 10^{-9}\ F} \cong 482.3\ kHz$$

Esta frecuencia de corte es prácticamente la que pretendíamos conseguir inicialmente.

# ANEXO C: PROCEDIMIENTO DE CÁLCULO DE LA INCERTIDUMBRE DEL ADC

En este anexo se explica el procedimiento aplicado para evaluar la exactitud de las medidas obtenidas como resultado de las simulaciones llevadas a cabo durante el análisis realizado.

En primer lugar, incertidumbre (*u*) se define como el margen de error que delimita un intervalo de valores que contiene el verdadero valor de la medida con una determinada confianza. Este intervalo se denomina intervalo de confianza (IC) [2].

$$x_{real} \in [\, x_{med} - u, x_{med} + u \,]$$

El valor de incertidumbre obtenido como resultado del procedimiento depende de parámetros como el número de muestras tenidas en cuenta, o la confianza del intervalo.

Asumiendo que las medidas obtenidas siguen una distribución normal, y que el número de muestras son infinitas, el parámetro *k* que determina la confianza del intervalo puede tomar un valor de 2 para una confianza del 95%, o un valor de 3 para una confianza del 99%.

En el presente análisis, se ha considerado un intervalo de confianza del 95% (*k*=2), y se han tomado 50 muestras (*N*) para cada una de las situaciones a analizar. Este número de muestras es lo suficientemente alto para dar por válida la suposición de que se toman un número de muestras infinitas.

Este procedimiento se aplica a varios puntos de operación, que en este caso se tratan de los diferentes niveles de ruido presentes en el sistema. Se han elegido tres puntos de operación, coincidentes con niveles de ruido del 10%, 5% y 2%, y el procedimiento es el que sigue a continuación.

El primer paso es calcular la variación de la media de las muestras tomadas $(\bar{x}_c)$, respecto del valor real de referencia $(x_{ref,c})$, en cada punto de operación *c*.

$$\Delta x_c = \bar{x}_c - x_{ref,c}$$

La desviación estándar de las medidas en cada punto de operación se calcula con la siguiente expresión.

$$\sigma_c = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_{ci} - \bar{x}_c)^2}$$

Donde $x_{ci}$ es la muestra *i* en cada punto de operación.

La varianza en cada punto de operación se puede obtener por medio de la siguiente expresión.

$$\sigma^2{}_{x_c} = \frac{\sigma^2{}_c}{n} + \frac{\sigma^2{}_c}{N}$$

*n* es el número de muestras a tener en cuenta para obtener una medida en el sistema de medida implementado finalmente. Es nuestro caso *n* será 1.

Con la varianza ya se puede calcular el valor de incertidumbre para cada punto de operación haciendo uso de esta expresión:

$$u_c = \sqrt{k^2 \cdot {\sigma^2}_{x_c} \cdot \left(\frac{1}{N} + \frac{1}{n}\right) + {\Delta x^2}_c}$$

Una vez que tengamos calculados todos los valores de incertidumbre de cada uno de los puntos de operación, el valor de incertidumbre para todo el rango de trabajo será el mayor de ellos.

$$u = \max(u_c)$$

Por tanto, se puede asegurar que los valores reales de las medidas tomadas por el ADC analizado con este procedimiento estarán dentro del intervalo $[\, x_{med} - u,\ x_{med} + u \,]$ con una confianza del 95%.

# ANEXO D: MANUAL DE UTILIZACIÓN DE LA HERRAMIENTA DE SIMULACIÓN

# MANUAL DE UTILIZACIÓN DE LA HERRAMIENTA DE SIMULACIÓN VIRTUOSO® DE cadence®

## INTRODUCCIÓN

Este manual trata sobre la herramienta de diseño y simulación de circuitos electrónicos Virtuoso® Design Environment de Cadence®. En concreto, lo que se pretende con este manual es guiar al lector a lo largo del proceso de diseño de un esquema electrónico desde cero, y de la posterior creación del entorno de simulación que permita verificar su comportamiento.

Las posibilidades que ofrece esta herramienta van mucho más allá de las mencionadas en este manual, ya que está principalmente pensada para el diseño de circuitos integrados. Por este motivo, sólo nos vamos a centrar en los aspectos comentados, con el objetivo de poder realizar una simulación mixta.

Al ser una herramienta muy compleja, cuenta con un gran número de opciones y comandos accesibles desde los menús de la ventana de trabajo, pero tan sólo se explicarán aquellos que sean útiles para la tarea que se esté desarrollando en cada punto del manual. En cualquier caso, se anima al lector a que investigue por su cuenta y consulte la ayuda disponible para descubrir todas las acciones que se pueden llevar a cabo en la herramienta.

## OBJETIVOS

- Utilización de librerías y *cellviews*. Library Manager.
- Creación del esquema electrónico de un diseño.
- Importación de bloques digitales a un diseño.
- Configuración del entorno de simulación. Simulación mixta.
- Obtención de resultados de simulación. Uso de la calculadora.

## LIBRARY MANAGER

Cuando arrancamos el entorno, lo primero que vemos es la ventana CIW (Command Interpreter Window). Es una ventana en la que podemos escribir comandos directamente, y además muestra información de diagnóstico. Cada vez que se ejecuta un comando aparece aquí el resultado de su ejecución, así como los posibles errores y avisos (warnings) que puedan aparecer.

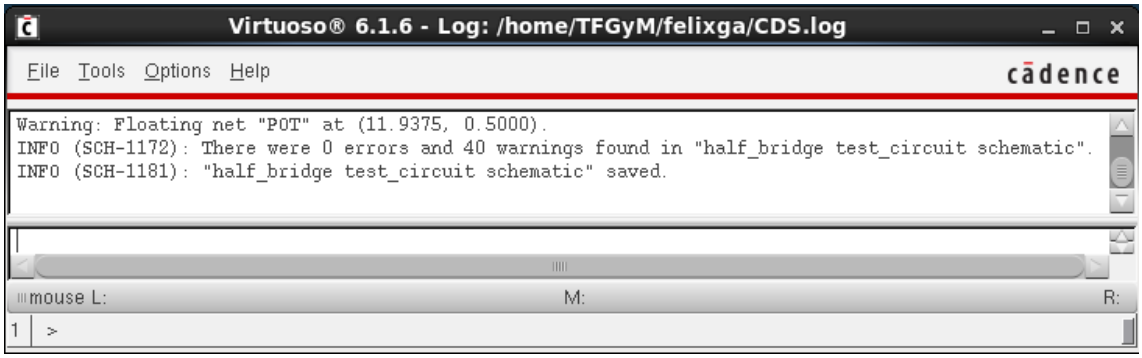


*Figura 58. Command Interpreter Window (CIW)*

Para empezar a trabajar lo primero que debemos comprender es como el programa distribuye los archivos que componen los diferentes diseños y componentes. Desde la pestaña Tools podemos abrir el *Library Manager*. Aparecerá una ventana como esta:

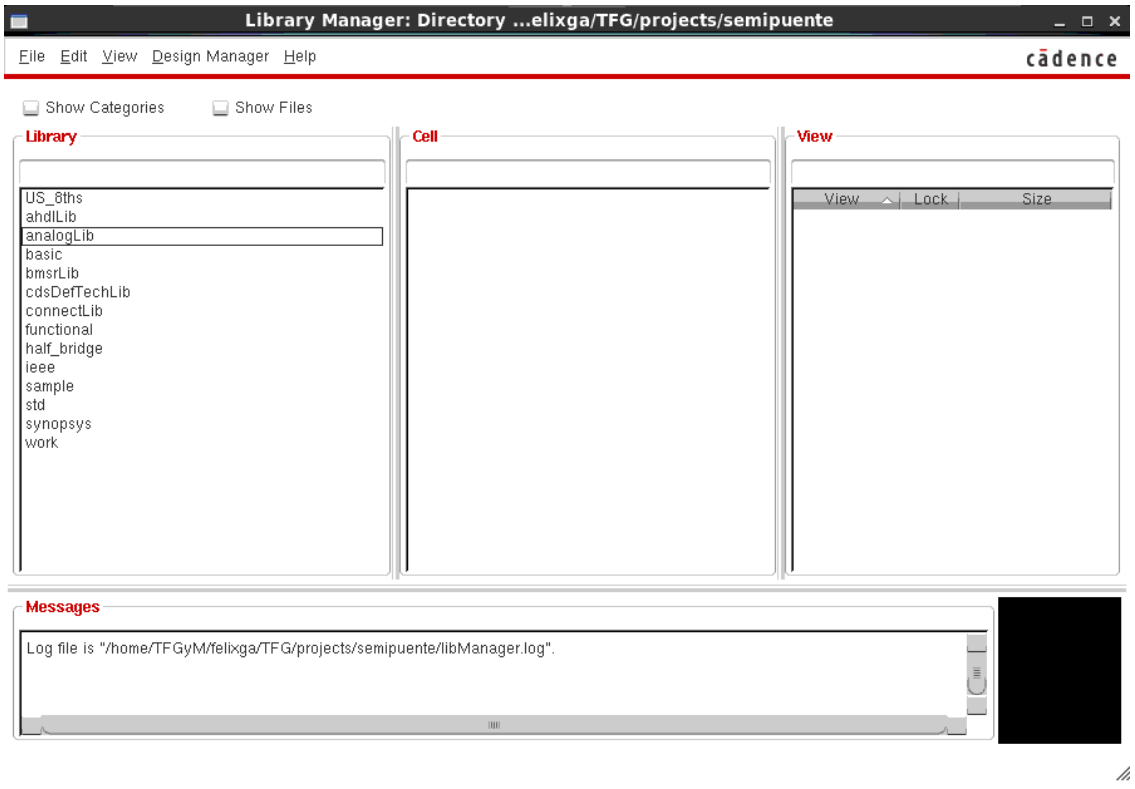


*Figura 59. Library Manager*

Como se puede ver en la imagen, los diseños se distribuyen en librerías (Library), que a su vez contienen celdas (Cells), cada una de las cuales puede tener una o varias representaciones (Views).

Una librería se puede ver como una biblioteca en la que podemos agrupar varios diseños que hayamos creado junto con otros componentes, como bloques importados que previamente habían sido creados en otro entorno. Las que vienen por defecto están distribuidas por categorías y en ellas podemos encontrar todo tipo de componentes tanto analógicos como digitales que utilizaremos en nuestros diseños.

Los diseños que vamos creando dentro de una misma librería se denominan celdas. Estos pueden tener representaciones en forma de esquemático, en forma de símbolo con unos pines de entrada y salida, en forma comportamental utilizando lenguajes de descripción de hardware como VHDL o Verilog, etc. Este tipo de representaciones son lo que se denomina como *views*. Una misma celda puede tener varias representaciones al mismo tiempo.

En las siguientes secciones se irá viendo cómo crear una librería, en la cual añadiremos celdas con las diferentes partes de nuestro diseño, algunas de ellas con varias representaciones.

## CREACIÓN DE UN ESQUEMÁTICO

Como forma de ir viendo el método de introducción de un esquema electrónico en la herramienta y su posterior simulación, vamos a crear el modelo de un conversor Σ-Δ, ya que este tipo de ADC combina una parte analógica con otra digital. De esta manera podremos llevar a cabo una simulación mixta y ver cómo configurar el simulador AMS de la herramienta que estamos utilizando.

El primer paso es crear una nueva librería en la que iremos guardando las diferentes partes de nuestro diseño. Para ello, desde la ventana del *Library Manager* hacemos lo siguiente:

File → New → Library

Elegimos el directorio en el que la queremos guardar, y le asignamos un nombre. En nuestro caso se llamará "*my_design*".

Antes de seguir debemos seleccionar una tecnología para nuestro diseño. En nuestro caso no importa cual elijamos ya que no vamos a realizar ningún *layout*. Marcamos la opción *Attach to an existing technology library* y a continuación seleccionamos la librería *analogLib*.

Una vez creada la librería, creamos una nueva celda de la misma manera:

File → New → Cell view

En la ventana que aparece, debemos indicar la librería a la que queremos que pertenezca, asignarle un nombre, y seleccionar el tipo de representación que queremos que tenga. Vamos a crear la parte analógica del conversor Σ-Δ, por ello le daremos el nombre "*modulador*" a la celda, y el tipo de representación será *schematic*.

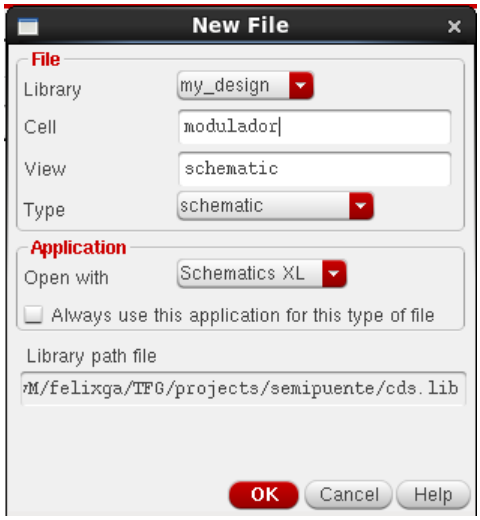


*Figura 60. Creación de una nueva celda*

Al pulsar *OK* se abre la ventana *Schematic XL Editing*. Es en esta ventana donde vamos a añadir e interconectar todos los componentes que forman nuestro diseño.

Nuestro diseño consta de 3 resistencias, un condensador, una fuente de tensión continua, y un comparador ideal. En la siguiente tabla se muestran las librerías en las que se encuentran cada uno de esos componentes, y el nombre de su celda correspondiente.

| Componente | Librería | Celda | View | Parámetros |
|---|---|---|---|---|
| Resistencia | analogLib | res | symbol | R1 = R2 = 4.75 kΩ |
| Condensador | analogLib | cap | symbol | C = 100 nF |
| Fuente DC | analogLib | vdc | symbol | $V_{DC}$ = 1.65 V |
| Comparador | functional | comparator | symbol | $V_{HIGH}$ = 3.3 V ; $V_{LOW}$ = 0 V |
| GND | analogLib | gnd | symbol | ---- |

*Tabla 16. Elementos necesarios para nuestro diseño*

Para comenzar a añadir componentes, hacemos lo siguiente:

Create → Instance, o bien pulsando  en la barra de herramientas superior.

En la ventana que aparece hacemos clic en *Browse*, y desde el Library Manager seleccionamos el componente que queremos utilizar. Una vez lo hemos encontrado, ya podemos hacer clic en *Close* y rellenar los campos correspondientes a los parámetros del componente. Por último, colocamos el componente en la parte del diseño que queramos.

No se va a utilizar en nuestro diseño, pero también existe la posibilidad de introducir ruido en él por medio de una celda de la librería "*ahdlLib*" llamada "*noise_src*". Se trata de una fuente de ruido blanco, que genera un ruido aleatorio de una amplitud máxima en voltios igual al valor introducido como parámetro.

Una vez introducidos todos los componentes necesarios en el esquemático, los podemos colocar organizadamente y utilizar la herramienta de cableado para interconectarlos entre sí.

Create → Wire (narrow), o bien pulsando  en la barra de herramientas superior.

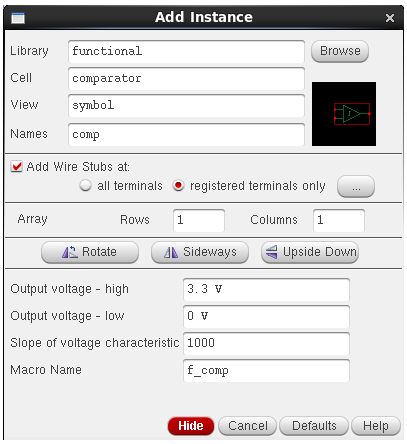


*Figura 61. Añadir un componente a un esquemático*

Para finalizar el esquemático solo tenemos que añadir los pines de entrada y salida:

Create → Pin, o bien pulsando  en la barra de herramientas superior.

| Nombre | Dirección |
|---|---|
| Vin | input |
| SDVO | output |
| SDVI | inputOutput |

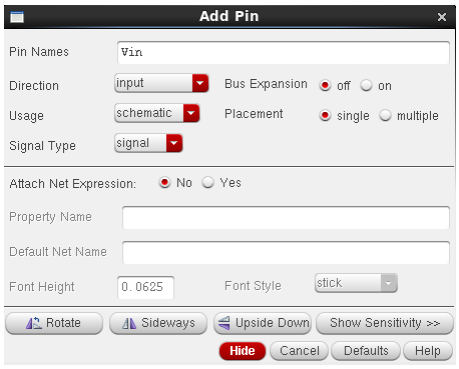


*Figura 62. Añadir un pin a un esquemático*

Una vez terminado el proceso, guardamos el diseño y comprobamos que no haya ningún error en la ventana CIW:

File → Check and Save, o bien pulsando  en la barra de herramientas superior.

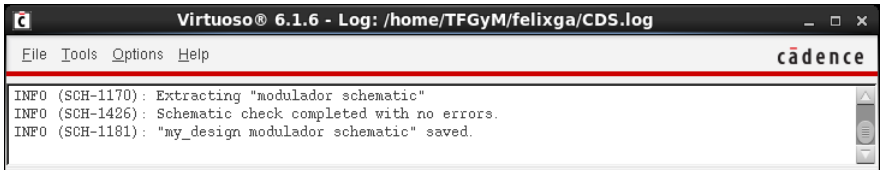


*Figura 63. Información mostrada en la ventana CIW*

Como resultado final tenemos que obtener un circuito similar al de la siguiente imagen:

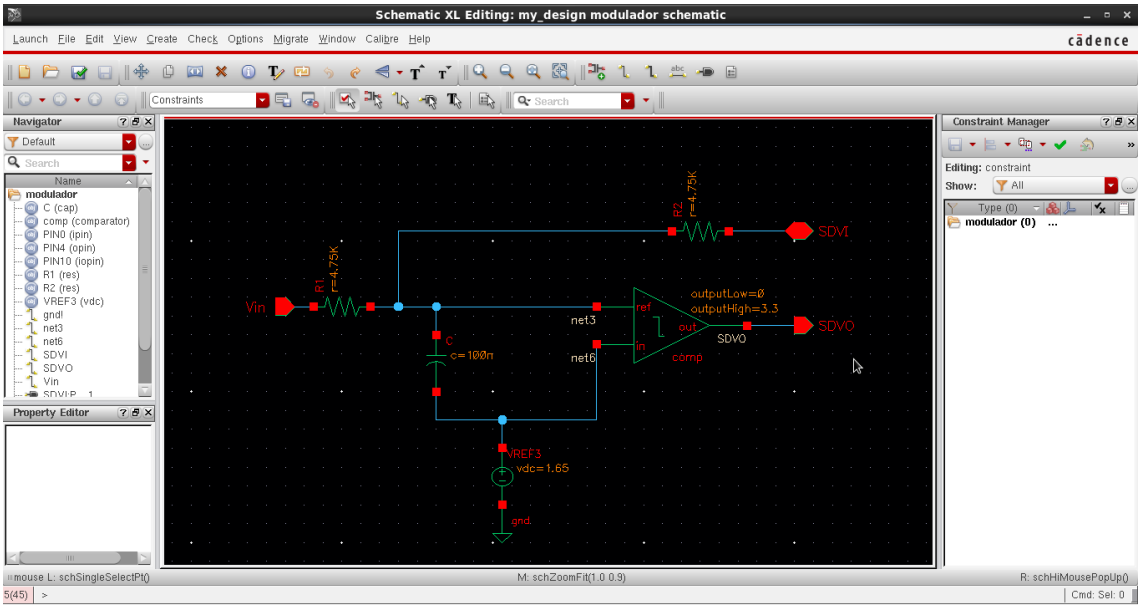


*Figura 64. Esquemático del modulador de un conversor Σ-Δ*

## GENERACIÓN DEL SÍMBOLO DE UNA CELDA

Para poder insertar el esquemático que acabamos de crear en un diseño jerárquico, primero hay que generar la representación *symbol* de este. Se puede generar desde la misma ventana *Schematic XL Editing* en la que estábamos trabajando, haciendo lo siguiente:

Create → Cellview → From Cellview

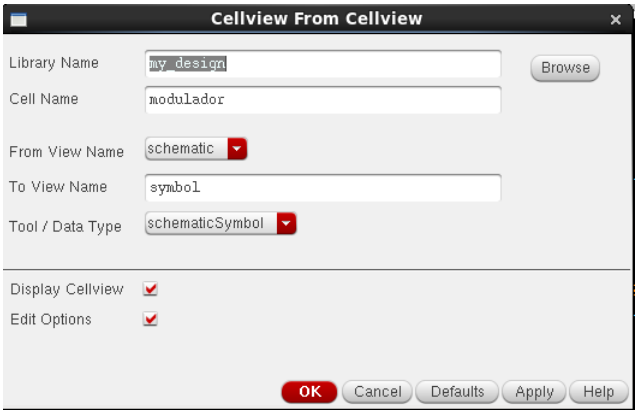


*Figura 65. Creación de una nueva representación de una celda*

Rellenamos los campos que aparecen, indicando la librería y el nombre de la celda donde queremos que se guarde la nueva representación, y elegimos crear una representación *symbol* a partir de la representación *schematic*.

En la siguiente ventana que aparece podemos ordenar los pines de entrada y salida, y seleccionar en qué parte del símbolo se encontrarán.

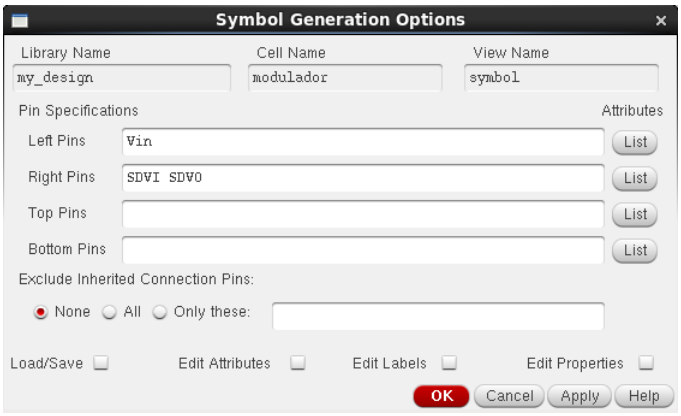


*Figura 66. Orden y posición de los pines de un símbolo*

Una vez realizado este paso se abre la ventana *Symbol L Editing*, en la cual podemos ver el símbolo que se ha generado.

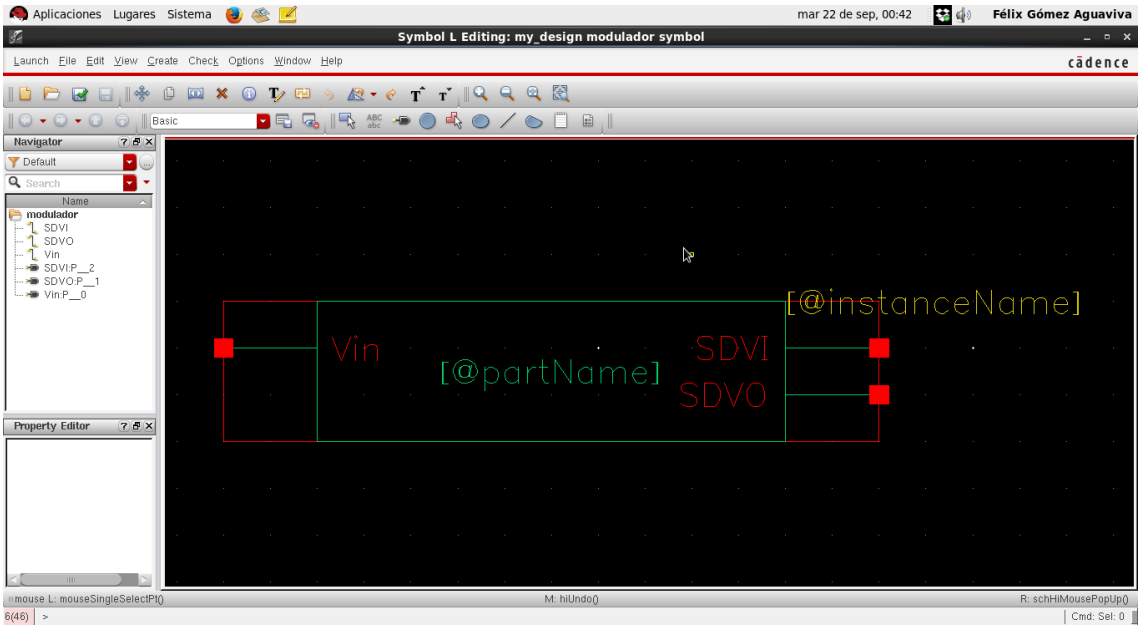


*Figura 67. Representación symbol del modulador de un conversor Σ-Δ*

Desde esta ventana es posible modificar por completo el aspecto del símbolo.

Si el orden de los pines no es el deseado, la manera más sencilla de reordenarlos es volviendo a crear la representación *symbol* a partir del esquemático y reescribir la ya existente.

Finalmente, volviendo al *Library Manager*, si nos fijamos en las representaciones que tenemos de la celda *modulador*, habrá dos tipos de *view*, como esquemático y como símbolo.

## GENERACIÓN DEL *TEST-BENCH*

Para poder simular el sistema al completo, es necesario crear un nuevo esquemático en el cual podamos juntar la parte analógica y digital de nuestro diseño, e incluir estímulos de entrada.

Para ello creamos una nueva celda con representación *schematic*, de la misma manera que se ha hecho anteriormente. En este caso la llamaremos "*test_conv*". En ella deberemos incluir el símbolo de la celda *modulador* que hemos creado, la parte digital del conversor (Se comentará en la siguiente sección como crearla) y una fuente AC con las siguientes características:

| Componente | Librería | Celda | View | Parámetros |
|---|---|---|---|---|
| Fuente AC | analogLib | res | symbol | Amplitude = 1.65 V<br>Frequency = *fr*<br>*Offset Voltage* = 1.65 V |

*Tabla 17. Características de la señal de entrada del test-bench*

A los campos de los componentes es posible asignar variables que posteriormente se utilizarán en el entorno de simulación. La variable *fr* se utilizará para poder realizar simultáneamente simulaciones a diferentes frecuencias de la señal de entrada.

Para la inclusión de todos estos componentes se seguirá el procedimiento general descrito anteriormente.

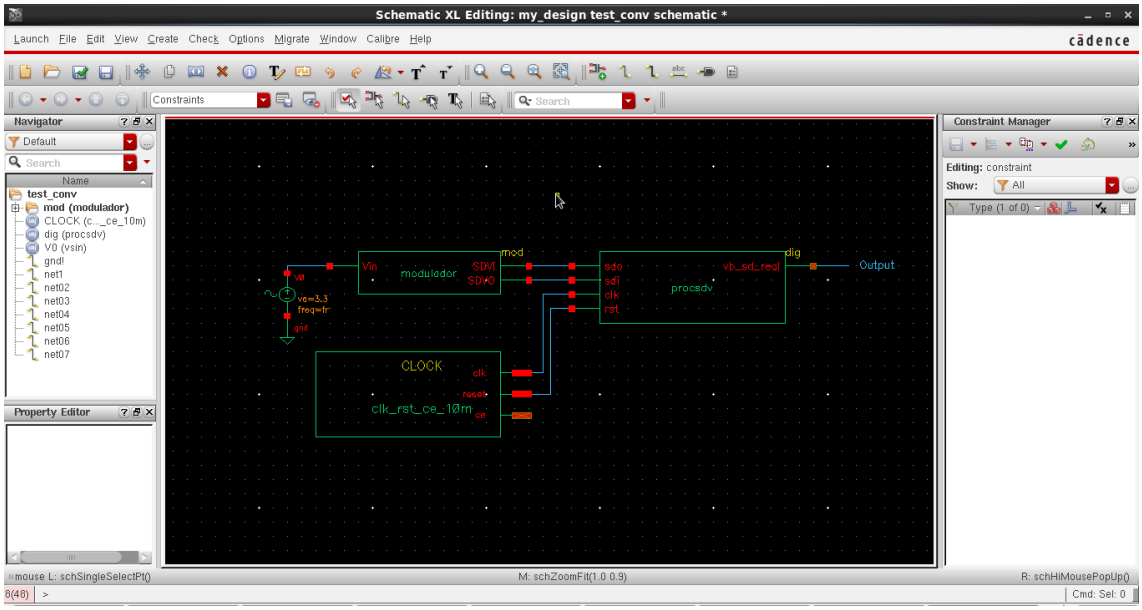


*Figura 68. Test-bench de nuestro diseño*

## IMPORTACIÓN DE BLOQUES EN VHDL

Nuestro diseño consta de una parte digital formada por un biestable, que almacena el tren de pulsos de salida del modulador, y un filtro digital de paso bajo. Ese tren de pulsos se realimenta al modulador y además se hace pasar por el filtro para reconstruir la señal original de entrada del conversor. Todo este bloque digital trabaja a una frecuencia de 10 MHz, que se obtiene mediante otro bloque digital que genera el reloj del sistema.

Partiendo de que tenemos estos bloques modelados en un lenguaje de descripción de hardware como VHDL (Se puede ver el código de ambos bloques en el anexo 2 del manual), lo que tenemos que hacer es importar el correspondiente archivo con el modelo de cada bloque, y crear una nueva celda a partir de él. No se pretende entrar en cómo se han modelado cada uno de los bloques, sino simplemente explicar de forma genérica cómo importar un archivo VHDL.

A la hora de utilizar bloques modelados en lenguaje VHDL con esta herramienta es importante tener en cuenta algunas consideraciones, ya que existen ciertas restricciones a la hora de trabajar con este lenguaje en concreto. Esto es así porque la herramienta Virtuoso® Design Environment de Cadence® está pensada para trabajar con Verilog, otro lenguaje de descripción de hardware muy similar a VHDL y muy utilizado, el cual pertenece a la compañía. Las dos principales restricciones encontradas son las siguientes:

- No es posible utilizar entradas y salidas de los bloques digitales de tipo *integer*, ya que es un tipo de dato que la herramienta no puede interpretar.
- No se pueden utilizar genéricos como parámetros de los bloques importados en VHDL, ya que la herramienta no los reconoce.
- Si el bloque tiene entradas o salidas de tipo real, no se generará su representación *symbol* automáticamente a la hora de importarlo, y habrá que generarla posteriormente de forma manual.

A continuación se indica el procedimiento para importar un archivo VHDL como una nueva celda de una librería.

Paso 1. Desde la ventana CIW hacemos File → Import → VHDL, y se abre la ventana *VHDL Import*

En ella debemos seleccionar la librería destino en la que se importará el bloque, y el archivo que queremos importar. En cuanto al tipo de arquitectura elegimos *netlist*. El resto de opciones que no aparecen en la imagen las dejamos como están, salvo el campo que indica el nombre de la librería WORK, que en nuestro caso la hemos llamado de la misma manera, "*work*". Esta carpeta debe crearse previamente en el mismo directorio desde el cual arranquemos la herramienta, y será donde se guarden los archivos VHDL compilados. Además debemos marcar el tic del campo *v93 Option* para que la herramienta sea capaz de utilizar código VHDL compatible con la versión VHDL 93.

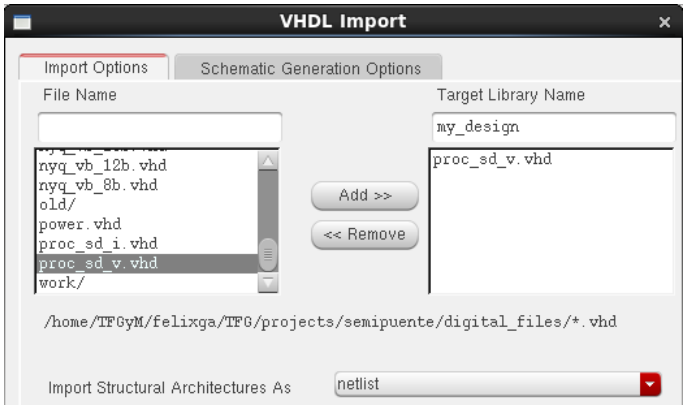


*Figura 69. Ventana VHDL Import*

Finalmente, hacemos clic en *Ok* y la herramienta nos preguntará si queremos ver un archivo llamado *log file*, en el cual podemos comprobar si la importación se ha llevado a cabo correctamente.

Paso 2. Como se ha comentado anteriormente, si el bloque importado tiene entradas o salidas de tipo real será necesario generar manualmente su representación *symbol*. Es necesario disponer de este tipo de representación para poder incluirlo en nuestro diseño. La manera de generarla es muy similar a la utilizada anteriormente para generar el símbolo de una celda con representación *schematic*.

Desde cualquier ventana de edición, como puede ser la ventana *Schematic XL Editing*, hacemos Create → Cellview → From Cellview.

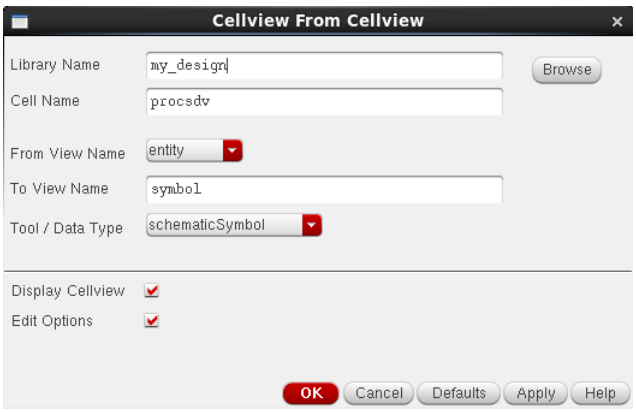


*Figura 70. Generación del símbolo de un archivo importado*

En este caso tenemos que modificar las opciones que salen por defecto, y seleccionar la celda en la cual queremos generar la representación *symbol*. En el campo *From View Name* debemos seleccionar *entity*. Hacemos clic en *Ok* y aparece una ventana desde la cual podemos organizar el orden y la posición de los pines de entrada y de salida.

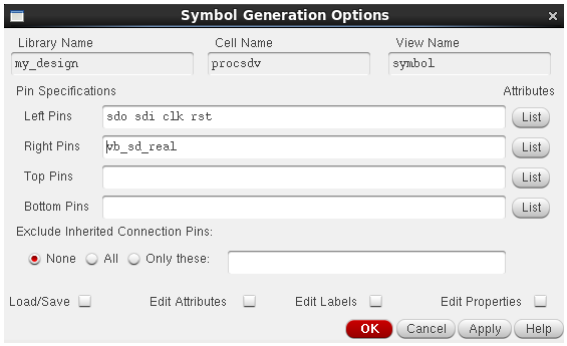


*Figura 71. Orden y posición de los pines de entrada y salida*

Una vez completados estos pasos, ya disponemos de una nueva celda en nuestra librería, con varias representaciones o *views*, una de las cuales será de tipo *symbol*. Ahora ya podemos incluirla en nuestro diseño como un componente más.

De la misma manera se procedería para importar el bloque encargado de generar la frecuencia de reloj, sólo que en este caso bastaría con completar el Paso 1.

Con el test-bench completado, el diseño obtenido debería ser similar al de la Figura 68, y el siguiente paso sería pasar a configurar el entorno de simulación para realizar un análisis de su comportamiento. En ocasiones, a la hora de simular aparece un error relativo a un archivo corrupto de la base de datos. De ser así, la forma de solucionarlo es abrir desde el *Library Manager* los archivos VHDL tanto de entidad como de arquitectura de nuestros modelos, escribir un carácter para modificarlos (que no afecte a su funcionamiento), y guardarlos. De esta manera se regenera el archivo relativo a la base de datos que causaba el error, y la simulación se lleva a cabo correctamente.

## GENERACIÓN DE LA REPRESENTACIÓN *CONFIG* PARA SIMULACIÓN MIXTA

Para poder llevar a cabo una simulación mixta, primero es necesario crear un nuevo tipo de representación para la celda de nuestro test-bench, en este caso de tipo *config*. La creamos desde el *Library Manager* de la misma manera que se han creado anteriormente las celdas de ambos esquemáticos, pero en este caso simplemente crearemos una nueva representación de la celda "*test_conv*".


*Figura 72. Creación del tipo de representación config*

En la ventana que aparece, en el campo *View* de la sección *Top Cell* debemos seleccionar *schematic*. Para rellenar el resto de campos de la sección *Global Bindings*, hacemos clic en *Use Template*, y seleccionamos en el desplegable *AMS*.

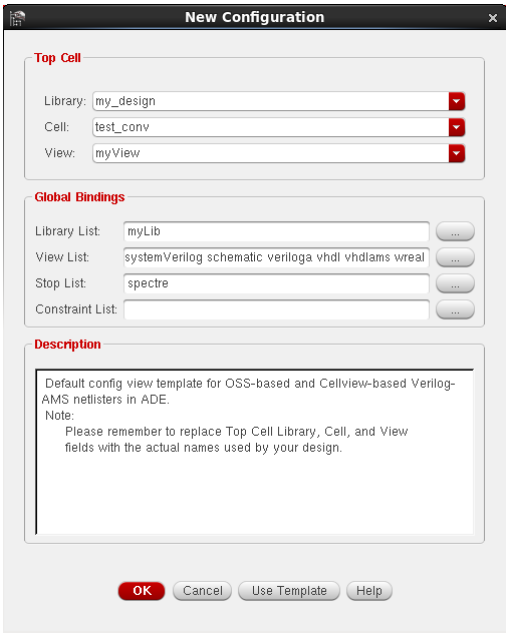


*Figura 73. Ventana de configuración de la view de tipo config*

A continuación aparece la ventana *Hierarchy Editor*, desde la cual ya podemos guardar la configuración asignada. En ella vemos todos los componentes utilizados en nuestro diseño, con el tipo de representación que se va a utilizar para cada uno de ellos en simulación. Si alguna de estas representaciones no es la adecuada, podemos escribir en la columna *View To Use* la representación que queremos que se utilice.

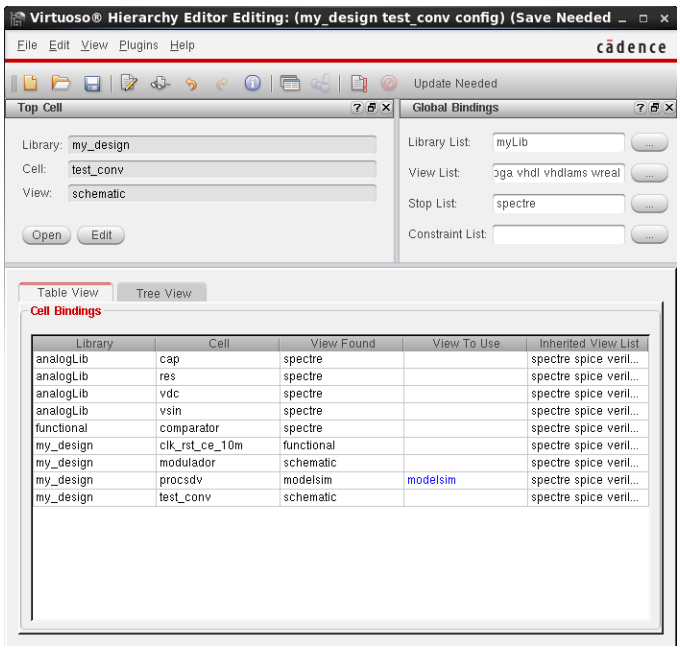


*Figura 74. Hierarchy Editor*

## PERSONALIZACIÓN DEL DISEÑO

Existe la posibilidad de utilizar algunos recursos que ofrece la herramienta para hacer que el diseño quede más organizado, más limpio y sea más visual. Los más destacados son los que se nombran a continuación:

**Etiquetado de las diferentes señales.** Es posible ahorrar cables de interconexión utilizando etiquetas para las señales de diferentes puntos de un esquema. De tal manera, si queremos unir dos pines de dos componentes alejados entre sí dentro de un mismo esquema, basta con unir un pequeño trozo de cable a cada uno, y etiquetarlos a ambos con el mismo nombre.

La forma de asignar una etiqueta es:

Create → Wire Name, o bien pulsando  en la barra de herramientas superior.

**Añadir una nota de texto.** Podemos añadir notas de texto en nuestros diseños para indicar información útil sobre este.

Create → Note → Text

**Creación de marco con cajetín.** Es posible añadir un marco alrededor de nuestro diseño, con un cajetín en el que añadir su título y autor, por ejemplo.

Edit → Sheet Size → (Selección del tamaño del marco)

## CONFIGURACIÓN DEL ENTORNO DE SIMULACIÓN. CREACIÓN DE UN NUEVO TEST

Para poder simular y comprobar el comportamiento del diseño que hemos creado, primero debemos configurar un nuevo Test.

Si la ventana *Schematic XL Editing* con nuestro test-bench no está abierta, podemos abrirla desde el *Library Manager* haciendo doble clic sobre la celda correspondiente, y desde ahí ya podemos arrancar *ADE XL*, el entorno que utilizaremos para realizar la simulación.

Launch → ADE XL → Create New View

Aparece una ventana en la cual tenemos que indicar la celda que queremos simular. Al haber arrancado el entorno desde el esquemático de nuestro test-bench, aparece esa celda por defecto. Una vez completado este paso se abre en una nueva pestaña la ventana de bienvenida de *ADE XL*.

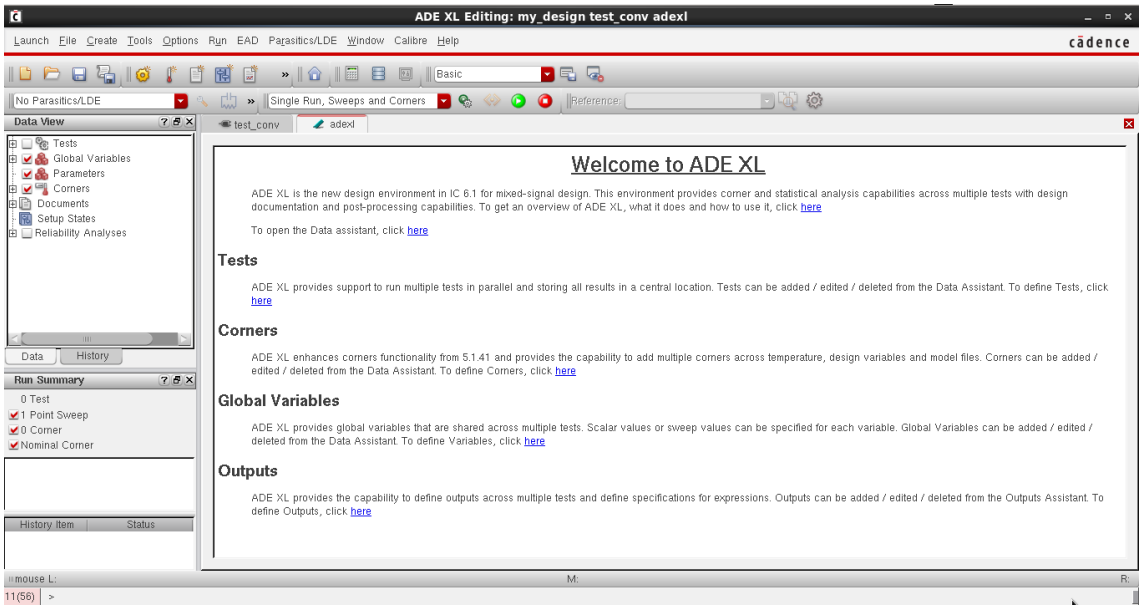


*Figura 75. Ventana de bienvenida de ADE XL*

En el panel *Data View* abrimos el desplegable *Tests*, y hacemos clic para generar un nuevo test. Seleccionamos la celda que queremos simular con representación de tipo *config*, y se abre el editor de tests.

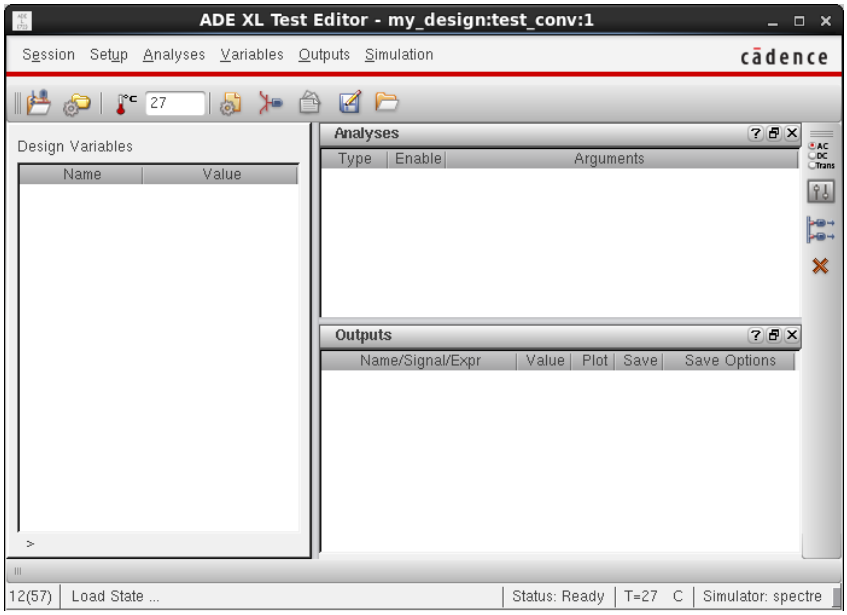


*Figura 76. Editor de tests*

Desde esta ventana tenemos que seleccionar el tipo de análisis que se va a realizar y las señales que se quieren visualizar. También podemos dar valores a las variables de nuestro diseño en caso de tenerlas.

Lo primero de todo es comprobar que el simulador que se va a utilizar es el correcto.

Setup → Simulator

En nuestro caso como vamos a simular un diseño mixto, con partes analógicas y digitales, utilizaremos el simulador *ams*.

A continuación comenzamos con la configuración del test, y seleccionamos un análisis transitorio de 1 ms.

Analyses → Choose, o bien pulsando 

en la barra de herramientas lateral.

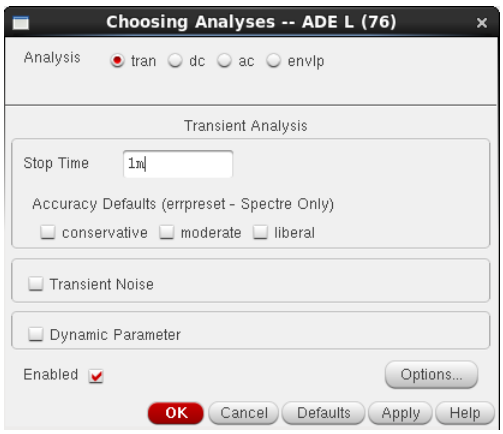


*Figura 77. Selección del tipo de análisis y del tiempo de simulación.*

Ahora tenemos que seleccionar las señales que queremos representar frente al tiempo. Las podemos seleccionar directamente haciendo clic sobre los cables correspondientes si se tratan de señales de tensión. Si por el contrario queremos ver la corriente que circula por un nodo, tendremos que pinchar sobre él en lugar de sobre el cable. En nuestro caso vamos a visualizar la señal de entrada y la de salida del conversor, para poder compararlas entre sí.

Outputs → Setup, o bien pulsando  en la barra de herramientas lateral.

En la ventana que aparece, hacemos clic en *From Schematic*, y seleccionamos directamente sobre el esquemático de nuestro test-bench las señales que queremos visualizar.

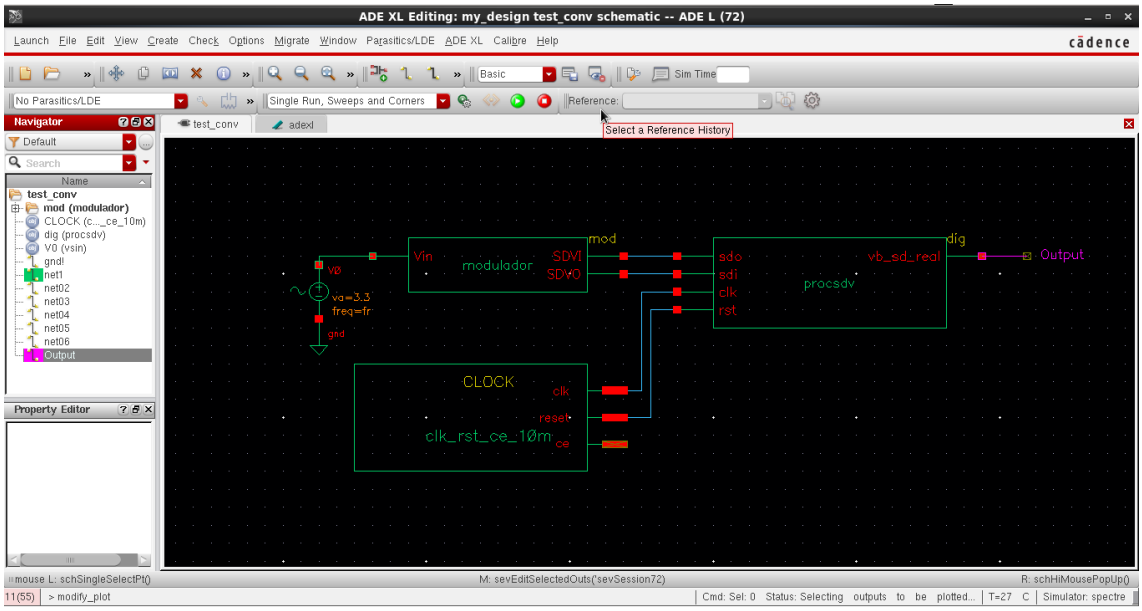


*Figura 78. Selección de señales a visualizar en la simulación*

Una vez lo hayamos hecho, ya habremos acabado esta parte de la configuración y podemos cerrar la ventana del editor de tests.

Volviendo al panel *Data View*, debemos definir como variable global el parámetro *fr* que hemos utilizado como frecuencia de la fuente AC que hace de señal de entrada de nuestro conversor. Podemos asignarle un único valor, o un rango de valores, de manera que se ejecuten diferentes simulaciones para cada uno de estos valores. En nuestro caso vamos a crear un intervalo de 5 frecuencias diferentes que vayan de 200 Hz a 1 kHz.

Abrimos el desplegable de *Global Variables*, y hacemos clic para crear una nueva variable. La llamamos igual que la variable utilizada en el test-bench, y en un primer momento le asignamos un valor fijo cualquiera. Una vez la variable esté creada se visualizará en el panel *Data View*. Para asignarle un rango de valores hacemos clic sobre su valor, y aparecerá un botón al lado. Al pulsarlo aparece la siguiente ventana:

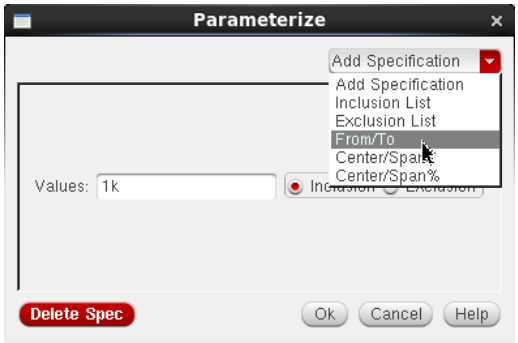


*Figura 79. Crear rango de valores para una variable*

Primero pulsamos *Delete Spec* para borrar el valor fijo que habíamos asignado en un principio. Ahora en el desplegable *Add Specification* ya podemos seleccionar *From/To*, y en los campos que aparecen asignar los valores de inicio y final del rango, así como el número de pasos intermedios.

Con esto ya tendríamos todo configurado para darle a *Run* y comenzar la simulación. Sin embargo, al estar trabajando con un diseño mixto todavía hay que definir unas reglas de conexión, las cuales se explica cómo crear en la siguiente sección.

Adicionalmente, haciendo clic derecho sobre el test creado, podemos modificar muchas más opciones que no se van a explicar en este manual, a excepción de algunas que pueden resultar de especial interés:

- En *Job Setup*, en el campo *Max. Jobs* podemos indicar cuantos procesadores queremos que trabajen a la vez, en caso de disponer de ellos. En nuestro caso utilizaremos 5 procesadores, para que las simulaciones para cada uno de los valores de la variable *fr* se ejecuten en paralelo.
- En *Model Libraries*, podemos añadir archivos que contengan modelos de ciertos componentes que hayamos utilizado en nuestro diseño. Tenemos que añadir el archivo "*allFunc.scs*", que contiene los modelos *spectre* de los componentes de la librería *functional* que hemos utilizado.

## CONFIGURACIÓN DEL ENTORNO DE SIMULACIÓN. DEFINICIÓN DE REGLAS DE CONEXIÓN

Con el fin de relacionar los valores digitales de nuestro diseño con sus correspondientes valores analógicos, y que el simulador pueda llevar a cabo la simulación conjunta de todo el diseño es necesario definir unas reglas de conexión.

Haciendo clic con el botón derecho en el test que hemos creado, seleccionamos *Connect Rules…*

Primero eliminamos las reglas de conexión que vienen por defecto en la parte superior de la ventana que aparece, y en la sección *Built-in and Customized rules* seleccionamos las reglas denominadas *connectLib.ConnRules_3V_basic*.

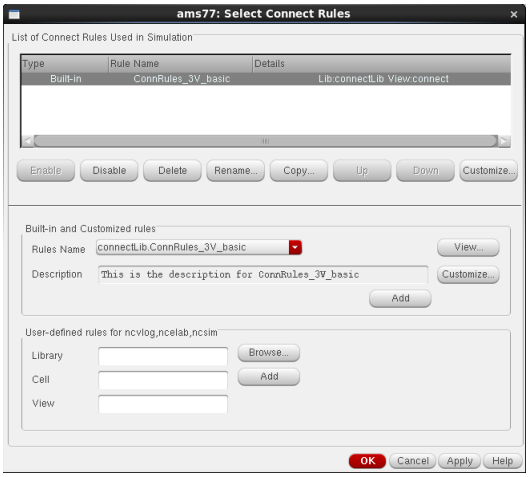


*Figura 80. Definición de las reglas de conexión*

Antes de añadir estas reglas las vamos a modificar para que se adapten a las necesidades de nuestro diseño. Hacemos clic en *Customize*...

Tenemos que modificar los siguientes parámetros en cada uno de los módulos que se indican:

### Módulo *E2L_0* (Electrical to Logic)

vsup (Valor en alto) → 3.3 V
vthi (Umbral a partir del cual se considera un 1 lógico) → 2 V (Suele ser 3/5 del valor en alto)
vtho (Umbral a partir del cual se considera un 0 lógico) → 0.66 V (Suele ser 1/5 del valor en alto)

### Módulo *L2E_0* (Logic to Electrical)

vsup (Valor en alto) → 3.3 V

Cuando hayamos modificado las reglas de conexión, hacemos clic finalmente en *Add*... para añadirlas al diseño.

Otras opciones que es necesario configurar haciendo clic con el botón derecho en el test creado son:

- *Solver*. Seleccionamos *spectre* como el solver que realice la simulación de la parte analógica del diseño.
- *Netlist and Run Options*. Dejamos la configuración tal y como aparece en la siguiente imagen.

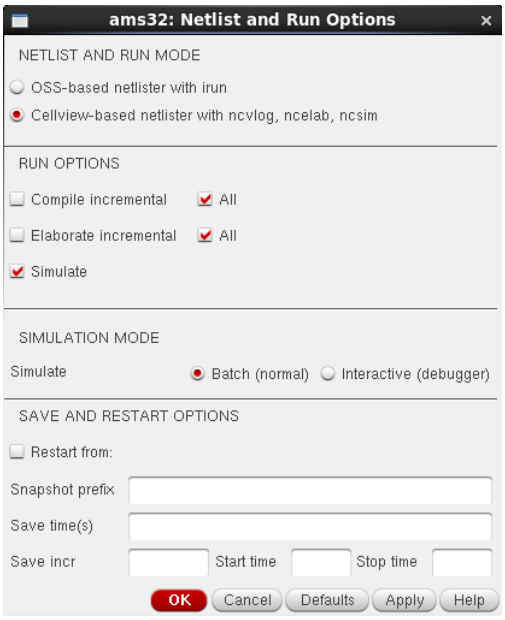


*Figura 81. Configuración de Netlist and Run Options*

- *Options → AMS Simulator*. En la sección *Miscellaneous* marcamos la opción *Enable IE Report*.

Finalmente, ya tenemos el entorno de test correctamente configurado para llevar a cabo una simulación mixta.

## UTILIZACIÓN DEL ENTORNO DE SIMULACIÓN.

Para que comience la simulación debemos hacer clic en

Run → Single Run, Sweeps and Corners, o bien pulsando 

Antes de eso, podemos añadir más señales que queramos visualizar en la pestaña *Outputs Setup*. También es posible añadir expresiones que nos devuelvan directamente ciertos parámetros de las señales que estamos simulando. Vamos a calcular el valor de pico y el valor RMS de la forma de onda que se obtiene en la salida del conversor. La forma de hacerlo es la siguiente:

- Hacemos clic en el icono  de la pestaña *Outputs Setup*.
- En el campo *Name* podemos asignarle un nombre a la salida creada.
- En la columna *Type* seleccionamos *expr*.
- En el campo *Expression/Signal/File* debemos introducir la expresión que queremos evaluar, siguiendo la siguiente estructura: function ( VT ( "signal" ) )

En nuestro caso, la expresión para evaluar el valor de pico de la salida del conversor será ymax ( VT ( "/Output" )), y la expresión para evaluar su valor RMS será rms ( VT ( "/Output" )).

Se pueden utilizar todas las funciones disponibles en la calculadora de la herramienta.

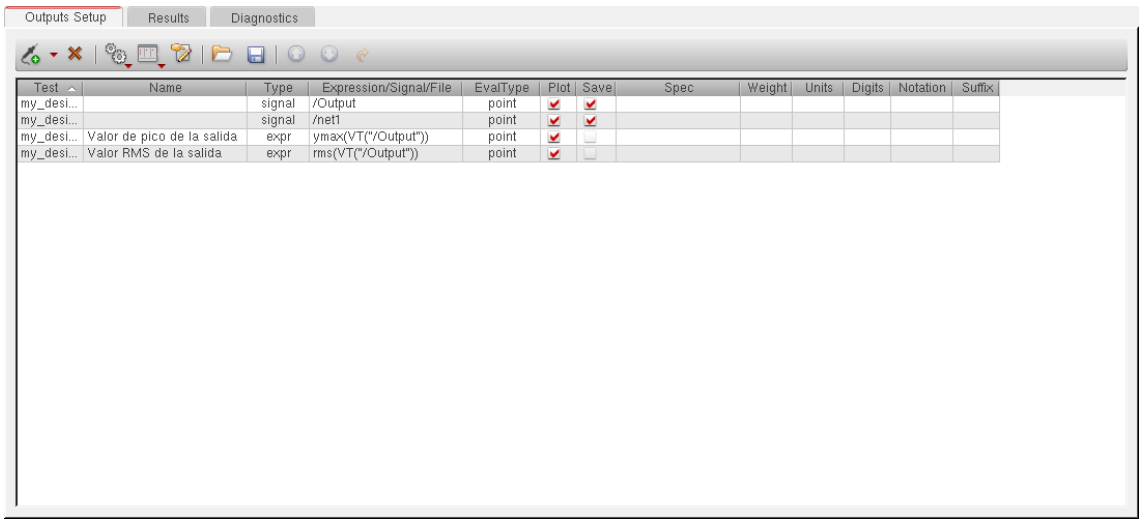


*Figura 82. Salidas que van a ser visualizadas como resultado de la simulación*

Tras la finalización de la simulación, la pestaña *Results* se mostrará de esta manera.

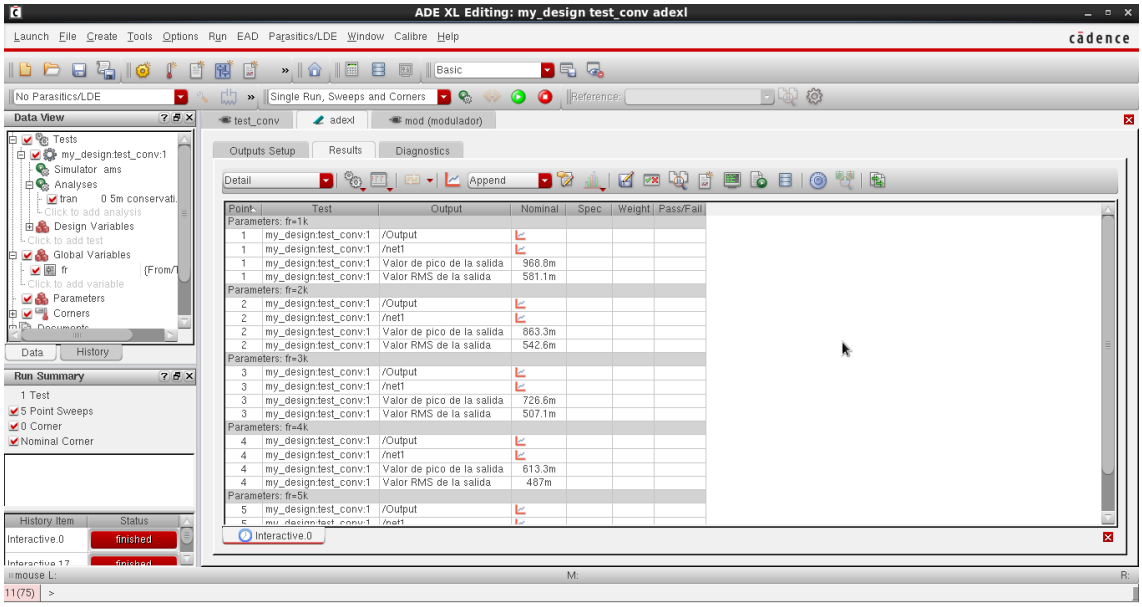


*Figura 83. Resultados de simulación*

En ella podemos ver directamente los resultados de las expresiones que habíamos introducido, y además podemos representar las formas de onda de las salidas.

Para ello tan solo hay que hacer clic en 

En el menú desplegable que hay justo al lado del icono, tenemos varias opciones disponibles para representar las formas de onda:

- *Append*. Si ya tenemos una ventana con formas de onda abierta, las nuevas formas de onda aparecerán sobre la misma gráfica.
- *Replace*. Se eliminan las formas de onda de la ventana de visualización, en caso de haberlas, previamente a la nueva representación de estas.
- *New SubWin*. Se representan las nuevas formas de onda dentro de una nueva sub-ventana de la pestaña que se estaba utilizando en la ventana de visualización.
- *New Win*. Las nuevas formas de onda se representan en una nueva pestaña, sin eliminar las que pudiéramos tener abiertas anteriormente en la ventana de visualización.

Al tener salidas que devuelven el resultado de una expresión que hemos introducido, no tiene sentido representarlas frente al tiempo. Podemos eliminar estas formas de onda haciendo clic con el botón derecho sobre la gráfica correspondiente, y seleccionando *Delete*.

En la gráfica correspondiente a las señales de entrada y salida, vemos que estas señales se han representado para todos los valores de frecuencia que le habíamos asignado a la variable *fr* desde el entorno de test. Podemos separar estas señales una a una en diferentes gráficas arrastrándolas con el ratón, o todas a la vez haciendo clic en el icono  .

Abriendo el desplegable junto al nombre de cada señal, podemos seleccionar las que queremos visualizar.

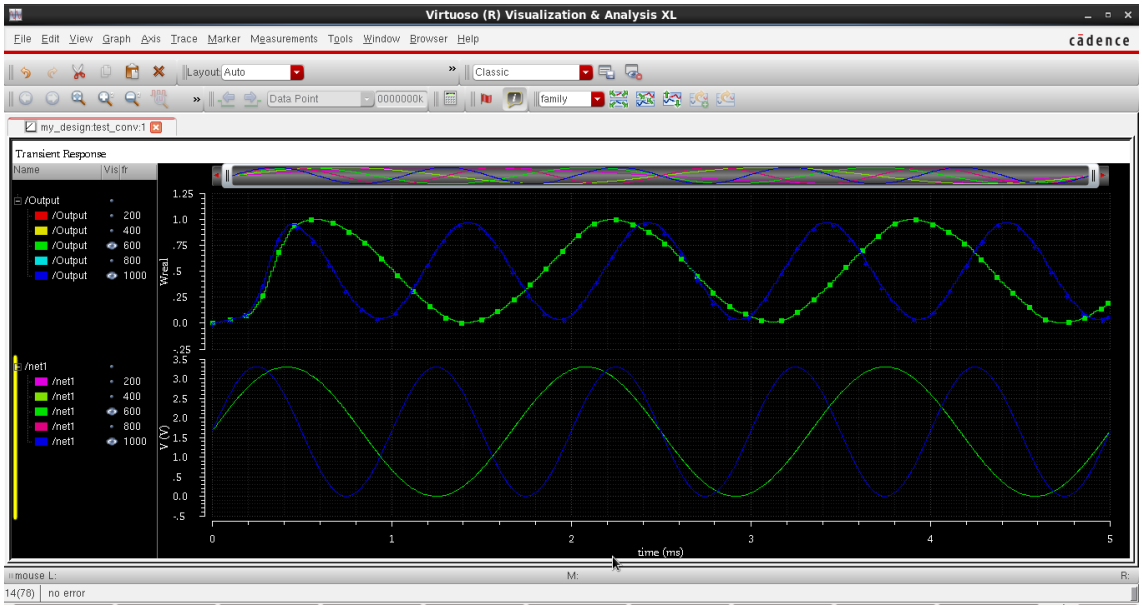


*Figura 84. Formas de onda de las señales de entrada y salida (fr = 600 Hz y 1 kHz)*

El hecho de que la señal de salida esté desfasada respecto a la de entrada se debe al procesado digital que sufre la señal antes de obtener un resultado en la salida del filtro. La amplitud de la señal de salida es menor que la de entrada ya que el rango de tensión de esta va de 0 a 1 V.

Para evaluar una expresión sobre una señal, debemos seleccionarla haciendo clic sobre ella (Se verá que queda más resaltada que el resto), y a continuación hacer clic en el icono de la calculadora de la barra de herramientas superior.

Esta acción hace que se abra la calculadora. En este entorno podemos generar una expresión con las señales que estamos visualizando, y evaluar su resultado.

En la ventana que se abre veremos una expresión en la zona central correspondiente a la señal seleccionada, y simplemente haciendo clic sobre las expresiones del panel *Function Panel*, y rellenando algunos parámetros si es necesario, se irá generando automáticamente la expresión que evalúe la función seleccionada sobre la señal.

Para evaluar la expresión tenemos que hacer clic en  , y el resultado aparecerá sustituyendo a la expresión.

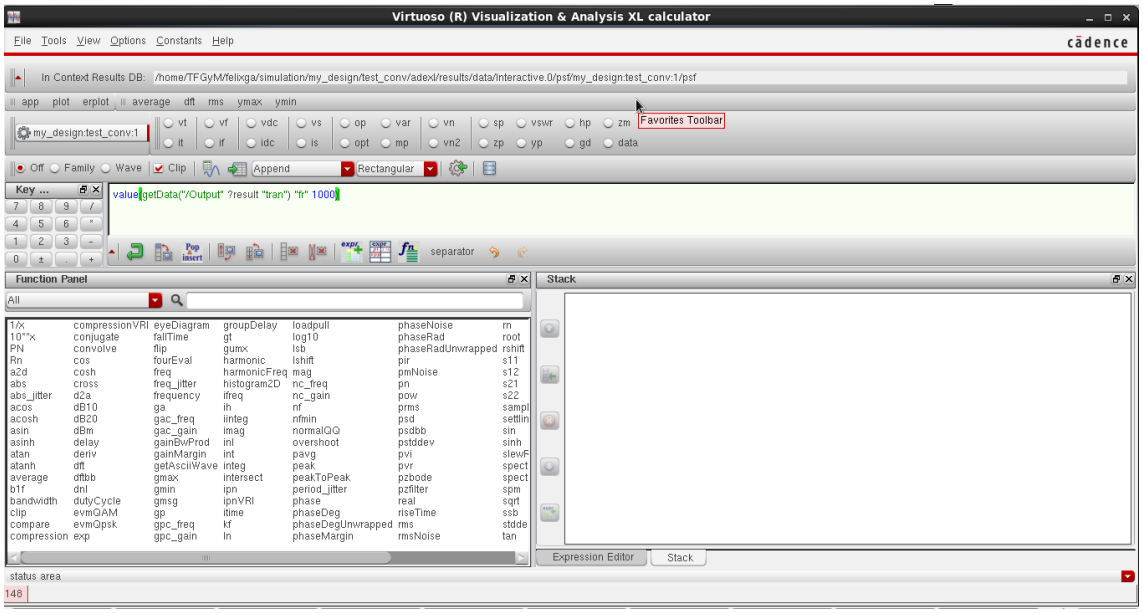


*Figura 85. Calculadora del entorno de simulación*

## ANÁLISIS FRECUENCIAL DE UNA SEÑAL. FFT

La herramienta también nos permite ver la representación en el dominio frecuencial de una señal haciendo una *Fast Fourier Transform* (FFT). Para ver un ejemplo de cómo realizarla vamos a hacer la FFT de la forma de onda obtenida en la salida del conversor cuando la frecuencia de la señal de entrada es de 1 kHz. Desde la ventana de visualización de formas de onda, seleccionamos la señal y hacemos lo siguiente:

Measurements → Spectrum

Aparece una sub-ventana con diferentes opciones.

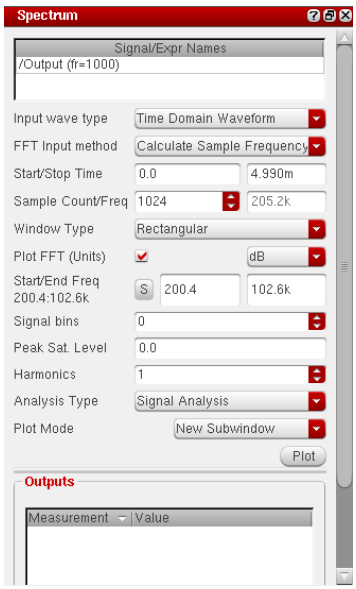


*Figura 86. Ventana de configuración de la FFT*

Las dos principales opciones que podemos modificar son las siguientes:

- *Plot FFT (Units)*. Selección de las unidades en las que se va a mostrar la nueva gráfica
- *Start/End Freq*. Selección del rango de frecuencia para el que queremos que se haga el análisis. Podemos introducir el rango de frecuencia que queramos visualizar, y al pulsar el botón *S* que está justo a la izquierda de los valores introducidos, estos se ajustarán automáticamente. Generalmente con los valores que aparecen por defecto se puede visualizar correctamente todo el espectro de la señal.

Por último haciendo clic en *Plot*, aparece el espectro frecuencial de la señal.

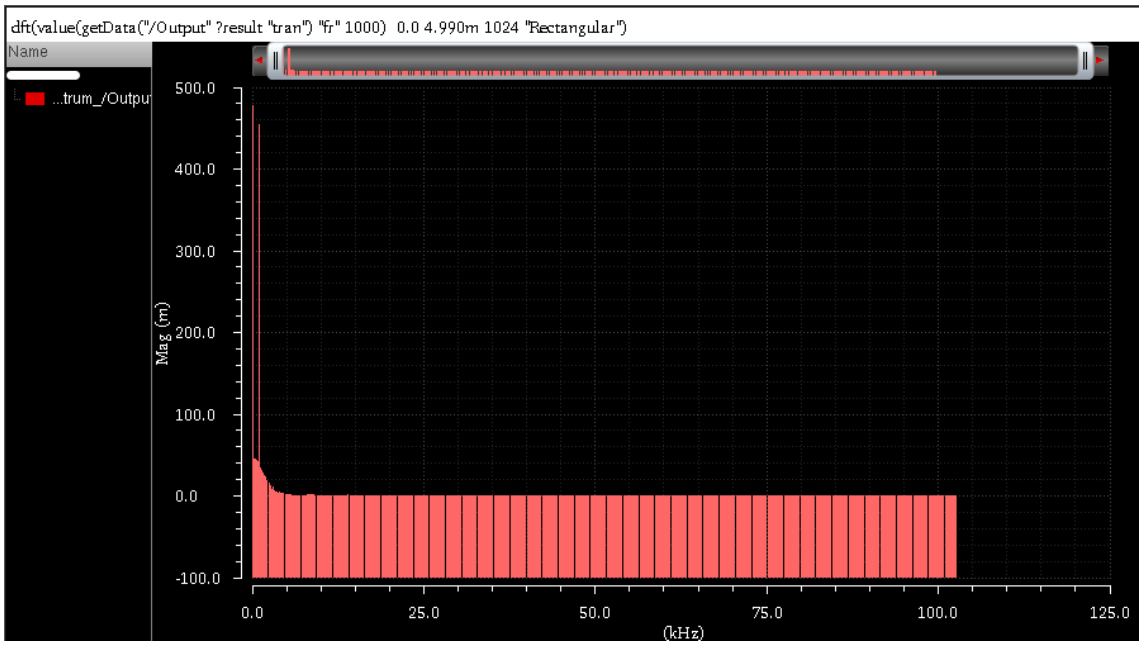


*Figura 87. Espectro frecuencial de la señal de salida del conversor (1 kHz)*

Se aprecia que al tratarse de una señal sinusoidal tan sólo existe un armónico a 1 kHz, valor de frecuencia de la señal. El otro armónico que está situado a una frecuencia de 0 Hz corresponde al valor de tensión continua que tiene la señal.

## HISTORIAL DE SIMULACIONES

Cada vez que realicemos una simulación, esta aparecerá en la pestaña *History* del panel *Data View*. Desde ahí podemos cambiar el nombre de la simulación por uno más representativo que el generado por defecto (Siempre teniendo en cuenta que el nombre no contenga espacios), y bloquearla para evitar eliminarla por error (Clic derecho → *Lock*).

Si queremos volver a cargar en la pestaña *Results* una simulación antigua, tan sólo hay que hacer clic derecho sobre ella y seleccionar *View Results*.

## EXPORTAR RESULTADOS EN FORMATO .CSV

Volviendo a la ventana de resultados, es posible exportar todos los valores obtenidos como resultados en un archivo .CSV. En simulaciones en las que se requiere evaluar un gran número de expresiones, esta puede ser una funcionalidad útil.

Para generar este archivo, debemos hacer clic en  , y seleccionar el directorio de destino.

Existe la problemática de que no se guardan simplemente los resultados de las expresiones, sino que también se guardan otros datos como nombres de señales y valores de variables globales, que pueden resultar inservibles. A continuación se explica una manera de exportar de manera sencilla desde una hoja de cálculo de Microsoft Excel solamente los datos de interés.

Primero debemos abrir una nueva hoja de cálculo, y hacer lo siguiente:

Datos → Importar desde texto

Seleccionamos el directorio donde se encuentra nuestro archivo .CSV, y seguimos los pasos del asistente.

Paso 1. Selección de la fila a partir de la cual empezar a importar datos.

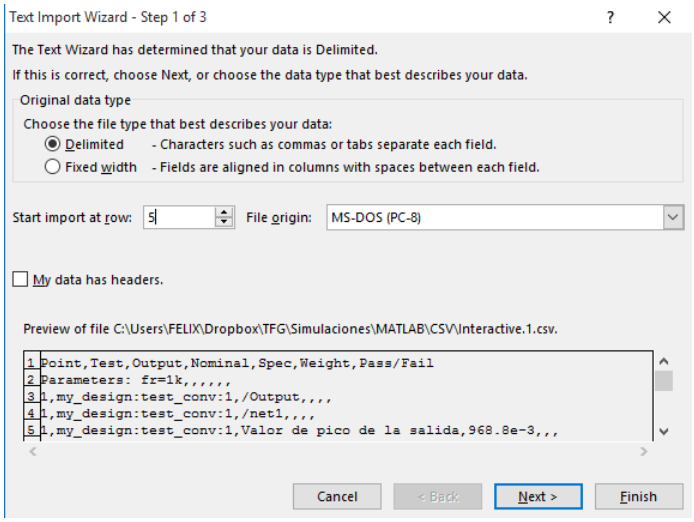


*Figura 88. Asistente de importación de datos a una hoja de cálculo. Paso 1*

**Paso 2.** Selección del carácter delimitador de los datos.

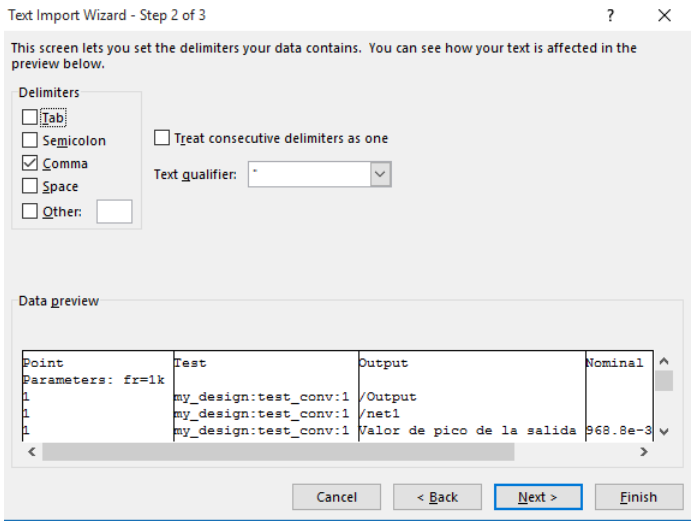


*Figura 89. Asistente de importación de datos a una hoja de cálculo. Paso 2*

**Paso 3.** Selección de las columnas que queremos importar. En nuestro caso tenemos que importar sólo la cuarta columna, en el resto deberemos seleccionar la opción de no importar. En opciones avanzadas, seleccionamos la coma como separador decimal, y el punto como separador de miles.

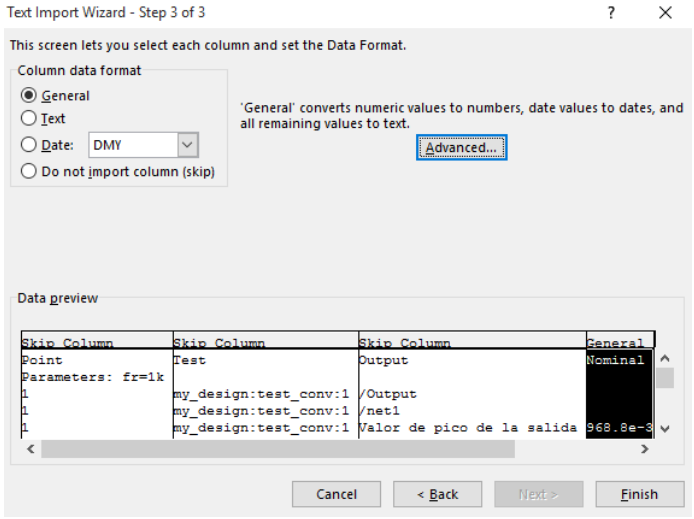


*Figura 90. Asistente de importación de datos a una hoja de cálculo. Paso 3*

Por último seleccionamos la celda sobre la que importar los datos, y de esta manera ya tendremos un archivo que contenga solamente los valores de interés. Este archivo podremos leerlo desde otro software, como por ejemplo Matlab®, que nos permita llevar a cabo un análisis de los datos.

## ANEXO 1: MÉTODOS ABREVIADOS DE TECLADO ÚTILES EN *SCHEMATIC XL EDITING*

| | |
|---|---|
| i | Añadir componente |
| p | Añadir pin |
| w | Añadir cable |
| l | Etiquetar un cable |
| Shift + l | Añadir nota de texto |
| n | Añadir polígono |
| q | Editar propiedades de un componente |
| m | Mover componente |
| c | Copiar |
| r | Girar componente |
| s | Estirar |
| delete | Borrar |
| esc | Cancelar comando actual |
| z | Zoom in sobre la zona seleccionada |
| Shift + z | Zoom out |
| f | Ajustar el tamaño del diseño al de la ventana |
| u | Deshacer ultimo comando |
| Shift + u | Rehacer |
| e | Descender (Modo lectura) |
| Shift + e | Descender (Modo edición) |
| Ctrl + e | Ascender un nivel |
| F3 | Mostrar opciones del comando |

# ANEXO 2: CÓDIGO VHDL DE LOS BLOQUES UTILIZADOS EN EL EJEMPLO

## BIESTABLE + FILTRO DIGITAL DE PASO BAJO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity PROCSDV is
port(         CLK : in std_logic;
              RST : in std_logic;
              SDI : in std_logic;
              SDO : out std_logic;
       VB_SD_real : out real);
end entity PROCSDV;

architecture modelsim of PROCSDV is

   signal SDV1, QV: std_logic;
   signal CONT_CLK: integer;

-- CIC
   type real_vector is array (natural range <>) of real;
   signal CIC1_real, CIC2_real: real_vector (0 to 15) := (others => 0.0);
   signal CIC_IN_real, CIC1_OUT_real, CIC2_OUT_real: real := 0.0;
   signal VB_real: real := 0.0;


begin

   process (RST, CLK)
   begin
       if (RST='1') then
           CONT_CLK <= 0;
       elsif (CLK'event and CLK='1') then
           if (CONT_CLK >= 99) then
               CONT_CLK <= 0;
          else
               CONT_CLK <= CONT_CLK + 1;
          end if;
      end if;
   end process;

   process (RST, CLK)
   begin
      if (RST='1') then
         SDV1 <= '0';
      elsif (CLK'event and CLK='1') then
         if (CONT_CLK = 99) then
            SDV1 <= SDI;
         end if;
      end if;
   end process;

   SDO <= SDV1;
   QV <= not(SDV1);

   CIC_IN_real <= 1.0 when (QV='1') else
                  0.0;
```

```
---Reconstruir la tension------

   process (RST, CLK)   -- Two CIC sections
   begin
      if (RST='1') then
         CIC1_real <= (others => 0.0);
         CIC2_real <= (others => 0.0);
      elsif (CLK'event and CLK='1') then
         if (CONT_CLK = 99) then
             CIC1_real <= CIC_IN_real & CIC1_real(0 to 14);
             CIC2_real <= CIC1_OUT_real & CIC2_real(0 to 14);
         end if;
     end if;
   end process;

   -- Salida CIC1
   process (CIC1_real)
       variable CIC_OUT: real;
   begin
      CIC_OUT := 0.0;
      for i in 0 to 15 loop
         CIC_OUT := CIC_OUT + CIC1_real(i);
      end loop;
      CIC1_OUT_real <= CIC_OUT/16.0;
   end process;

   -- Salida CIC2
   process (CIC2_real)
       variable CIC_OUT: real;
   begin
      CIC_OUT := 0.0;
      for i in 0 to 15 loop
         CIC_OUT := CIC_OUT + CIC2_real(i);
      end loop;
      CIC2_OUT_real<= CIC_OUT/16.0;
   end process;

   VB_SD_real <= CIC2_OUT_real;


end modelsim;
```

SEÑAL DE RELOJ DE 10 MHz

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity CLK_RST_CE_10M is
Port (          CLK : out STD_LOGIC;
              RESET : out STD_LOGIC;
                 CE : out STD_LOGIC);
end CLK_RST_CE_10M;

architecture Functional of CLK_RST_CE_10M is

-- Clock period definitions
constant CLK_period : time := 100 ns; -- Reloj de 10 MHz


begin

process
begin
    CLK <= '1';
    wait for CLK_period/2;
    CLK <= '0';
    wait for CLK_period/2;
end process;


RESET <= '1','0' after 225 ns ;


-- Señal de muestreo : 1 pulso cada CLK (10 MHz)
process
begin
    CE <= '1';
    wait for CLK_period/2;
    CE <= '0';
    wait for CLK_period/2;
end process;


end Functional;
```

# ANEXO E: BLOQUES MODELADOS EN VHDL UTILIZADOS EN EL PROTOTIPO

## PROTOCOLO SPI DE COMUNICACIÓN CON EL AD7356

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.ALL;


entity AD7356 is
 Port ( CLK_SMPL : in  STD_LOGIC;
           RST_N : in  STD_LOGIC;
         SDATA_A : in  STD_LOGIC;
         SDATA_B : in  STD_LOGIC;
       START_ADC : in  STD_LOGIC;
 WAIT_STATES_ADC : in  STD_LOGIC_VECTOR (31 downto 0);
           NBITS : in  STD_LOGIC_VECTOR (31 downto 0);
SCLK_CYCLES_CONV : in  STD_LOGIC_VECTOR (31 downto 0);
             NCS : out STD_LOGIC;
            SCLK : out STD_LOGIC;
          DATA_A : out STD_LOGIC_VECTOR (11 downto 0);
          DATA_B : out STD_LOGIC_VECTOR (11 downto 0);
  Reg_ad7356_spi : out STD_LOGIC_VECTOR(31 downto  0);
         END_ADC : out STD_LOGIC );

end AD7356;

architecture Behavioral of AD7356 is

signal SCLK_s : std_logic ;                                 -- Señal para leer la salida SCLK
signal dataA, dataB : std_logic_vector(11 downto 0) ;   -- Registros para almacenar los datos
convertidos
signal cont : integer range 0 to 135 ;                      -- Contador de ciclos de SCLK
type estado is (INIT, ZERO, DATA, QUIET) ;
signal state : estado ;                                     -- Estado actual

signal nb : std_logic_vector(3 downto 0) ;
signal wait_states_int : integer;
signal sclk_cycles_int : integer;
signal END_ADC_s : std_logic ;
signal DATA_A_s, DATA_B_s : std_logic_vector(11 downto 0) ;

begin


process(CLK_SMPL,RST_N)
begin
        if (RST_N = '0') then
                NCS <= '1' ;
                SCLK_s <= '1' ;
                DATA_A_s <= (others => '0') ;
                DATA_B_s <= (others => '0') ;
                dataA <= (others => '0') ;
                dataB <= (others => '0') ;
                cont <= 0 ;
                state <= INIT ;
                END_ADC_s <= '0' ;

        elsif (CLK_SMPL' event and CLK_SMPL = '1') then

                case state is

                        when INIT => if (START_ADC = '1') then
                                        state <= ZERO ;
                                        NCS <= '0' ;
                                        cont <= sclk_cycles_int ;
                                end if;
                                END_ADC_s <= '0' ;
```

```
                        when ZERO => if (cont <= 12) then
                                        state <= DATA ;
                                     end if;
                                     if (SCLK_s = '0') then
                                        cont <= cont - 1 ;
                                     end if;
                                     SCLK_s <= not SCLK_s ;

                        when DATA => if (cont <= 0) then
                                        state <= QUIET ;
                                     end if;
                                     if (SCLK_s = '0') then
                                        cont <= cont - 1 ;
                                        dataA(cont-1) <= SDATA_A ;
                                        dataB(cont-1) <= SDATA_B ;
                                     end if;
                                     SCLK_s <= not SCLK_s ;

                        when QUIET => if (cont >=wait_states_int ) then
                                        state <= INIT ;
                                        case nb is
                                              when "1000" =>
DATA_A_s  <=  not(dataA(11))&not(dataA(11))&not(dataA(11))&not(dataA(11))&not(dataA(11))&dataA(10
downto 4) ;

DATA_B_s  <=  not(dataB(11))&not(dataB(11))&not(dataB(11))&not(dataB(11))&not(dataB(11))&dataB(10
downto 4) ;
                                              when "1010" =>
DATA_A_s <= not(dataA(11))&not(dataA(11))&not(dataA(11))&dataA(10 downto 2) ;

DATA_B_s <= not(dataB(11))&not(dataB(11))&not(dataB(11))&dataB(10 downto 2) ;

                                               when "1100" =>
DATA_A_s <= not(dataA(11))&dataA(10 downto 0);

DATA_B_S <= not(dataB(11))&dataB(10 downto 0);
                                              when others => DATA_A_s <= (others => '0');

                                                             DATA_B_S <= (others => '0');
                                        end case;
                                        END_ADC_s <= '1' ;
                                     else cont <= cont + 1 ;
                                     end if ;
                                     NCS <= '1' ;
                                     SCLK_s <= '1' ;

                        when others => state <= INIT ;
                                       NCS <= '1' ;
                                       SCLK_s <= '1' ;
                                       DATA_A_s <= (others => '0') ;
                                       DATA_B_s <= (others => '0') ;
                                       dataA <= (others => '0') ;
                                       dataB <= (others => '0') ;
                                       cont <= 0 ;
                                       END_ADC_s <= '0' ;
                end case;
        end if;
end process;


process(CLK_SMPL,RST_N)
begin
        if (RST_N = '0') then
                DATA_A <= (others => '0') ;
                DATA_B <= (others => '0') ;

        elsif (CLK_SMPL' event and CLK_SMPL = '1') then
                if (END_ADC_s <= '1') then
                        DATA_A <= std_logic_vector(DATA_A_s) ;
                        DATA_B <= std_logic_vector(DATA_B_s) ;
                end if;
        end if;
end process;
```

```
process(CLK_SMPL,RST_N)
begin
        if (RST_N = '0') then
                END_ADC <= '0' ;

        elsif (CLK_SMPL' event and CLK_SMPL = '1') then
                END_ADC <= END_ADC_s ;
        end if;
end process;


-- COMBINATIONAL

process (NBITS, WAIT_STATES_ADC, SCLK_s, state) begin
        nb <= NBITS(3 downto 0) ;
        wait_states_int <= to_integer(unsigned(WAIT_STATES_ADC));
        sclk_cycles_int <= to_integer(unsigned(SCLK_CYCLES_CONV));
        SCLK <= SCLK_s ;
        case state is

                when INIT => Reg_ad7356_spi(3 downto 0) <= "0001" ;
                when ZERO => Reg_ad7356_spi(3 downto 0) <= "0010" ;
                when DATA => Reg_ad7356_spi(3 downto 0) <= "0100" ;
                when QUIET => Reg_ad7356_spi(3 downto 0) <= "1000" ;
                when others => Reg_ad7356_spi(3 downto 0) <= "0001" ;

        end case ;
        Reg_ad7356_spi(31 downto 4)<=(others=>'0');
end process;


end Behavioral;
```

## CÁLCULO DEL VALOR DE PICO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.ALL;


entity calc_peak is
Port ( CLK_SMPL : in  STD_LOGIC;
          RST_N : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (11 downto 0);
            INI : in  STD_LOGIC;
        END_ADC : in  STD_LOGIC;
        Ipeak_f : out STD_LOGIC_VECTOR (11 downto 0) );
end calc_peak;

architecture Behavioral of calc_peak is

signal max_data : signed (11 downto 0) ;

begin

process(CLK_SMPL,RST_N)
variable aux_data : signed (11 downto 0) ;
begin
        if (RST_N = '0') then
                aux_data := (others => '0') ;
                max_data <= (others => '0') ;
                Ipeak_f <= (others => '0');

        elsif (CLK_SMPL' event and CLK_SMPL = '1') then

                if(INI = '1') then
                        Ipeak_f <= std_logic_vector(max_data);
                        max_data <= (others => '0') ;

                elsif(END_ADC = '1') then
                        aux_data := signed(DATA) ;
                        if(aux_data > max_data) then
                                max_data <= aux_data;
                        end if;
                end if;

        end if;
end process;

end Behavioral;
```

## CÁLCULO DEL VALOR RMS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.ALL;


entity calc_rms is
Port ( CLK_SMPL : in  STD_LOGIC;
          RST_N : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (11 downto 0);
            INI : in  STD_LOGIC;
        END_ADC : in  STD_LOGIC;
       I2_rms_f : out STD_LOGIC_VECTOR (39 downto 0);
-- Acumula datos de 24b 50k veces (nb = 24b + log2(50k))
  cnt_samples_f : out STD_LOGIC_VECTOR (15 downto 0) );
-- Permite representar hasta 50k muestras (cada 10 ms)
end calc_rms;

architecture Behavioral of calc_rms is

signal num_data : unsigned (15 downto 0) ;
signal sum_rms : unsigned (39 downto 0) ;

begin

process(CLK_SMPL,RST_N)
variable aux_data : unsigned (11 downto 0) ;
variable rms_data : unsigned (23 downto 0) ;
begin
        if (RST_N = '0') then
                aux_data := (others => '0') ;
                rms_data := (others => '0') ;
                 sum_rms <= (others => '0') ;
                num_data <= (others => '0') ;
                I2_rms_f <= (others => '0') ;
                cnt_samples_f <= (others => '0') ;

        elsif (CLK_SMPL' event and CLK_SMPL = '1') then

                if(INI = '1') then
                        I2_rms_f <= std_logic_vector(sum_rms);
                        cnt_samples_f <= std_logic_vector(num_data);
                        sum_rms <= (others => '0') ;
                        num_data <= (others => '0') ;

                elsif(END_ADC = '1') then
                        aux_data := unsigned(DATA);
                        num_data <= num_data + 1 ;        -- 50k samples max cada 10 ms (16 bits)
                        rms_data := aux_data*aux_data ;   -- 12*2 bits
                        sum_rms <= sum_rms + rms_data ;
                end if;

        end if;
end process;
end Behavioral;
```