_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Desarrollo de un Modelo de Verificación en Verilog-AMS de un Convertidor DC-DC Integrado

Jaume Sanjuan Campos

Tutor: Vicente Herrero Bosch

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2015-16

Valencia, 13 de Septiembre de 2016

Resumen

Este trabajo consiste en la realización de un modelo de verificación en Verilog-A de un modulador Sigma Delta en distintas configuraciones a partir de la descripción individual de sus componentes. Para ello se ha partido de un modelo en Simulink del mismo en una configuración de primer orden. Las variaciones testeadas incluyen aumento de orden del modulador, variación de la frecuencia de oversampling y variación de la resolución del convertidor ADC interno.

Resum

Aquest treball consistix en la realització d'un model de verificació en Verilog-A d'un modulador Sigma Delta en distintes configuracions a partir de la descripció individual dels seus components. Per a això s'ha partit d'un model en Simulink del mateix en una configuració de primer orde. Les variacions testeadas inclouen augment d'orde del modulador, variació de la freqüència d'oversampling i variació de la resolució del convertidor ADC intern.

Abstract

This work consist of designing a verification model in Verilog-A of a Sigma Delta modulator in different configurations from the individual description of its components. For this matter the starting point is a Simulink model of a first order modulator. The tested variations include increasing the modulator order and changing both the oversampling frequency and the resolution of its internal quantizer.

Índice

Capítulo 1. Introducción 3
1.1 Motivación 3
1.2 Conversión ADC 3
1.2.1 Convertidores Flash 3
1.2.2 Convertidores Flash en dos pasos 4
1.2.3 Convertidores por etapas 4
1.2.4 Convertidores de rampa 4
1.2.5 Convertidores por aproximaciones sucesivas 5
1.2.7 Ruido en los ADC 6
1.3 Sigma Delta ADC 7
1.3.1 *Oversampling* 7
1.3.2 El modulador Sigma Delta de primer orden 7
1.3.3 El modulador Sigma Delta de segundo orden 9
1.3.4 Variaciones adicionales 11
1.3.5 Modulación en cascada MASH 11
1.3.6 Filtrado digital 11
1.4 Herramientas empleadas 11
1.4.1 MATLAB 11
1.4.2 Verilog-A 12
1.4.3 Virtuoso 12
Capítulo 2. Desarrollo de los modelos 12
2.1 Esquema completo 13
2.2 Signal Buffer 13
2.2.1 Modelo en Verilog-A 14
2.3 FrontEnd Integrator 17
2.3.1 Modelo en Verilog-A 18
2.4 Noisy Sample & Hold 21
2.4.1 Modelo en Verilog-A 21
2.5 Quantizer 24
2.5.1 Modelo en Verilog-A 24
2.6 DAC 27
2.6.1 Modelo en Verilog-A 27
2.7 Bloques adicionales 28

Capítulo 3. Funcionamiento 29
3.1 Modulador Sigma Delta de primer orden 30
3.2 Modulador Sigma Delta de segundo orden 33
3.3 Modulador Sigma Delta de segundo orden con cuantizador de tres y cuatro niveles . 36
3.4 Simulaciones en Virtuoso 38
Capítulo 4. Conclusiones y propuesta de mejora 39
Bibliografía 40

Capítulo 1. Introducción

En este capítulo procederemos a presentar las motivaciones detrás de este proyecto así como a realizar una exposición teórica de los conceptos y herramientas empleados en el trabajo.

1.1 Motivación

El objetivo de este proyecto es realizar un modelo en Verilog-A de un módulo ADC que forma parte de un diseño microchip para poder realizar su simulación con Spectre partiendo de un modelo realizado previamente en Simulink. Esto permitirá observar su funcionamiento dentro del sistema completo. Además se realizarán distintas configuraciones y se compararán los resultados proporcionados por cada una de ellas. El convertidor debe poder operar con frecuencias de hasta 1 MHz.

1.2 Conversión ADC

En este apartado se describirán distintas arquitecturas ADC comentando sus ventajas e inconvenientes hasta llegar a la empleada en el diseño.

1.2.1 Convertidores Flash

Esta arquitectura consiste en la comparación simultánea de la señal de entrada con diversas tensiones que se obtienen aplicando un divisor resistivo a una tensión de referencia *Vref* de manera que a la salida de cada comparador obtengamos un cero o un uno, posteriormente con un decodificador obtenemos el valor final de la conversión en digital. A continuación se muestra un ejemplo de convertidor Flash de 2 bits.

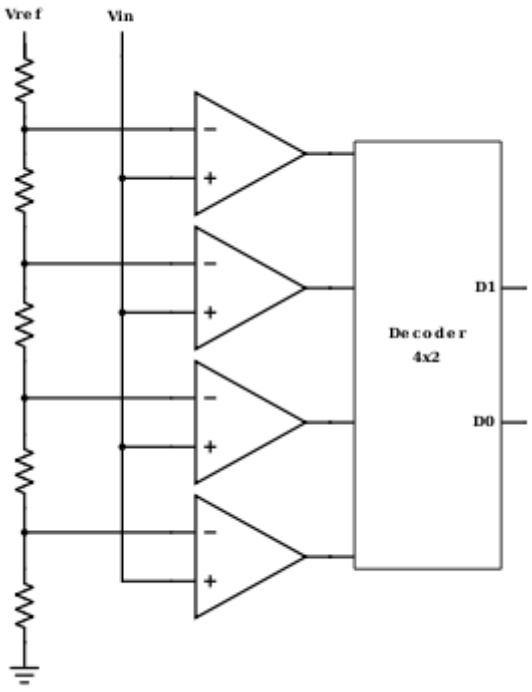

Figura 1: Convertidor Flash de 2 bits

Esta arquitectura es rápida pues permite realizar una conversión completa en un único ciclo de muestreo, y la limitación temporal viene dada por la velocidad de los comparadores y el decodificador. El principal problema de este convertidor es que a medida que aumentamos la resolución el número de comparadores se duplica con cada bit añadido, requiriéndose un total de $2^N - 1$ comparadores siendo *N* el número de bits. Por ejemplo, para un convertidor de 8 bits, el número de comparadores ascendería hasta 255, esto supone unos costes de área de transistores y de consumo que hace impracticable el diseño de estructuras de mayor resolución.

1.2.2 Convertidores Flash en dos pasos

Una forma de obtener mayor resolución empleando estructuras Flash distintas de manera que la conversión se realice en dos pasos, el primer ADC proporciona una primera conversión cuyo resultado se corresponderá con los bits más significativos, este resultado es reconvertido a analógico y restado al valor de entrada, el residuo resultante es amplificado por $2^{N/2}$ y pasa por el segundo ADC que proporciona los bits menos significativos, con esto conseguimos duplicar la resolución de los convertidores empleados, empleando una cantidad de comparadores mucho menor, pasando de $2^N - 1$ a $2(2^{N/2} - 1)$. Este aumento de resolución, sin embargo, se produce obviamente a costa de aumentar el tiempo necesario para la conversión, que ahora vendrá limitado por el amplificador y el restador. En cuanto a la precisión de esta arquitectura el mayor peso recae sobre el primero de los convertidores pues si hay un error en dicha conversión los bits correspondientes a la segunda serán todos erróneos. Otro problema con este método es que el amplificador que empleamos tras obtener el residuo debe comportarse con la mayor linealidad posible y la ganancia debe estar ajustada con la mayor precisión posible.

1.2.3 Convertidores por etapas

Siguiendo la filosofía de diseño detrás del convertidor Flash de dos etapas, este método podría extenderse a un mayor número de etapas. Supongamos ahora un ADC de N etapas, en el que empleamos convertidores de un tamaño de bits muy reducido, por ejemplo un único bit, en este caso el amplificador debería tener únicamente una ganancia de 2 por lo que nos encontraríamos con una estructura por etapa bastante sencilla. El problema con este modelo sería que el tiempo de conversión sería el tiempo necesario de operación acumulado de todas las etapas, por ello el proceso no ocurre simultáneamente en todas las etapas, sino que cada etapa retrasa un ciclo de reloj la señal antes de propagarla a la siguiente, aun así esto no afecta a la frecuencia de muestreo pues cada etapa sigue funcionando de forma independiente. Por tanto tenemos un convertidor que permite una elevada frecuencia de operación mientras presenta una latencia igual a N ciclos de reloj. Dependiendo de la aplicación este retraso en recibir la señal en digital puede ser o no un problema. De igual manera que ocurría con el convertidor de dos pasos el mayor peso de la conversión recae sobre aquellas etapas que se corresponden con los bits más significativos.

1.2.4 Convertidores de rampa

Los convertidores de rampa parten de un planteamiento completamente distinto, sacrificando en gran medida la frecuencia de conversión para conseguir resoluciones elevadas. Integrando la señal a la entrada y midiendo con una unidad de control digital los tiempos de integración podemos obtener valores de conversión bastante precisos, existen dos tipos de convertidores de este tipo, el de rampa simple y el de doble rampa, a continuación explicaremos únicamente el funcionamiento del convertidor de rampa simple pues el de doble rampa es bastante similar.

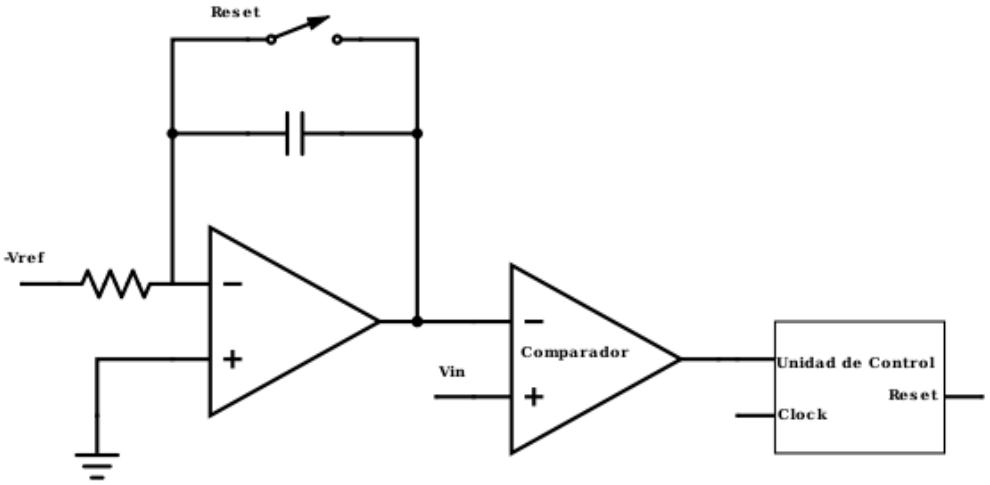

Figura 2: Convertidor de rampa simple

El convertidor de rampa simple consta de lo siguiente, un integrador, un comparador y una unidad de control, la unidad de control consiste en un contador de N bits siendo N la resolución en bits de nuestro convertidor. El proceso de conversión es el siguiente, simultáneamente tomamos el valor de Vin con un circuito de Sample and Hold y se cierra el integrador permitiendo que realice la integral del valor constante $-Vref$ obteniendo así la rampa que da nombre a esta arquitectura, mientras tanto el valor del contador va incrementando hasta que la integral alcanza el valor de la tensión de entrada, en ese momento se registra el estado de cuenta de la unidad de control ya que se corresponde con el valor de conversión digital. El uso de este método tiene las siguientes consecuencias que comparte con el convertidor de doble rampa:

- El tiempo de muestreo depende del valor de la muestra pues la medida termina cuando el integrador alcanza el valor de la tensión de entrada, es decir, para un valor menor de Vin podremos obtener la muestra en un periodo más corto que si este valor fuese más alto.
- La resolución del ADC depende directamente de la relación entre la frecuencia del reloj y el ancho de banda del convertidor. Cada bit adicional en la resolución supone aumentar al doble la frecuencia de reloj.

El principal inconveniente de estos ADC es que el proceso es bastante lento lo que hace que solo sean útiles con señales de bajas frecuencias, aun así el coste es bastante bajo para la resolución que ofrecen.

1.2.5 Convertidor por aproximaciones sucesivas

Otra arquitectura bastante común es la SAR (*Successive approximation ADC*) cuyo funcionamiento consiste en la implementación de un algoritmo iterativo en el que se generan valores y se comparan de manera que el resultado de la comparación es reutilizado para aproximar a cada iteración. El proceso en un SAR de N bits es el siguiente:

- Se genera un valor con el bit N a 1 y el resto de bits a 0.
- Este valor es convertido a analógico empleando un DAC.
- Comparamos dicha tensión con la tensión de la señal de entrada.
- El resultado de la comparación es devuelto y el bit N pasa a valer lo que vale dicho resultado, es decir si la señal de entrada es mayor vale 1 si es menor vale 0.

- Se repite el proceso con cada bit hasta llegar al menos significativo.

Como podemos ver nos encontramos de nuevo ante un convertidor que sacrifica la frecuencia de muestreo para obtener mejor resolución, sin embargo el SAR puede funcionar a frecuencias mucho mayores que los convertidores de rampa. Mientras que en los convertidores de rampa la frecuencia de reloj debía superar a la de muestreo en 2^N en los SAR solo debe incrementarse por $2N$. Esta arquitectura en concreto es bastante popular en la actualidad debido a sus características, pues permite obtener alta resolución a frecuencias razonablemente altas con un diseño bastante simple.

1.2.6 Ruido en los ADC

Toda conversión analógico digital conlleva inevitablemente un deterioro de la señal a causa de la cuantización, esto es debido al paso de una señal que puede tomar infinitos valores dentro del rango del convertidor a una señal que únicamente puede tomar valores discretos. Llamamos a esta diferencia entre los valores de la señal y los de su conversión error de cuantización.

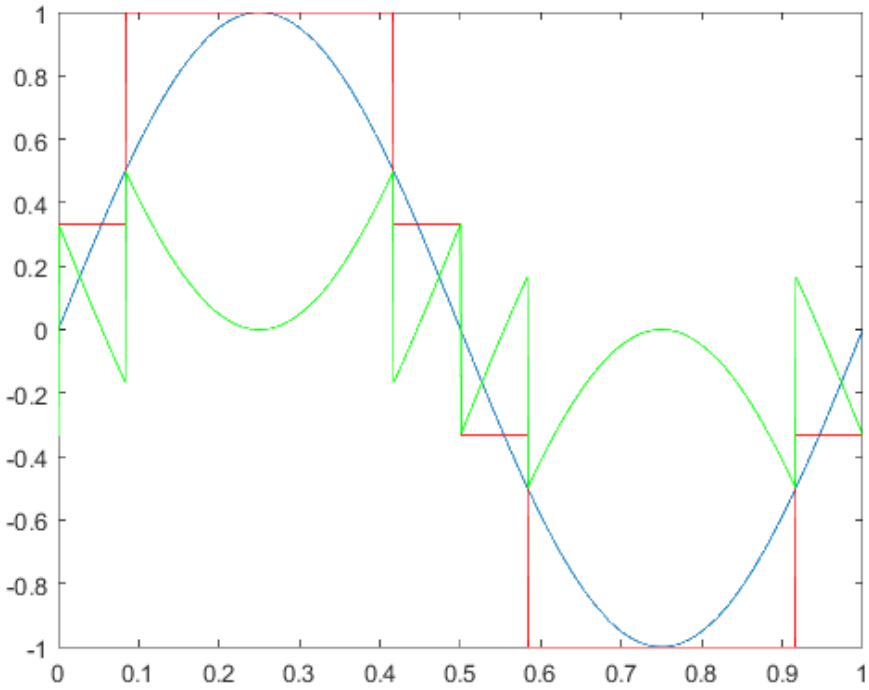

Figura 3: Error de cuantización en un ADC de 2 bits

Suponiendo un convertidor ADC ideal, donde el ruido no afecte a la señal en todo el proceso de conversión, nos encontramos con un error de cuantización que se distribuye entre $-1/2$ y $1/2$ del valor del bit menos significativo. Podemos observar este comportamiento en la Figura 3 en la que se muestra el error de cuantización producido al realizar una conversión de 2 bits sobre una señal sinusoidal. Este error de conversión se traduce finalmente en lo que se conoce como ruido de cuantización, que se reparte de manera plana a lo largo del espectro de frecuencias.

Como es de suponer el error de cuantización disminuye a medida que aumentamos la resolución por tanto nos encontramos con una relación señal a ruido SNDR que depende del número de bits de la siguiente manera:

$$SNDR = 1.761 + 6.02N\ dB \quad \textbf{(1.1)}$$

A continuación se muestra una tabla con el máximo SNDR obtenible teóricamente para distintas resoluciones:

Resolución	SNDR
4 bits	25.84 dB
8 bits	49.92 dB
12 bits	74 dB
16 bits	98.08 dB

Tabla 1: SNDR ideal según número de bits

Hay que tener en cuenta que los valores que encontramos en convertidores reales están por debajo de los mostrados por propias imperfecciones del convertidor.

1.3 Sigma-Delta ADC

En el apartado anterior se han presentado varias arquitecturas de ADC, todas ellas tienen en común que trabajan a la frecuencia de Nyquist, es decir, que muestrean a $f = 2f_s$ donde f_s se corresponde con el ancho de banda de la señal. En cambio una de las principales características del convertidor Sigma-Delta es que la frecuencia de muestreo es mucho más alta, este proceso se conoce como *oversampling.*

1.3.1 *Oversampling*

El proceso de *oversampling* consiste básicamente en la toma de muestras a frecuencias muy superiores al ancho de banda y después mediante procesado digital y diezmado obtener muestras a la frecuencia deseada. En cuanto al diseño del convertidor el uso de oversampling simplifica el diseño bastante en la parte analógica, ya que al realizar el procesado digitalmente se necesita menor precisión que en los convertidores a frecuencia de Nyquist. Otra ventaja de trabajar a frecuencias por encima de la de Nyquist es que se simplifica el diseño de los filtros anti-aliasing previos a la conversión, pudiendo hasta prescindir de ellos según el diseño.

1.3.2 El modulador Sigma-Delta de primer orden

En cuanto al convertidor Sigma-Delta presenta la siguiente arquitectura:

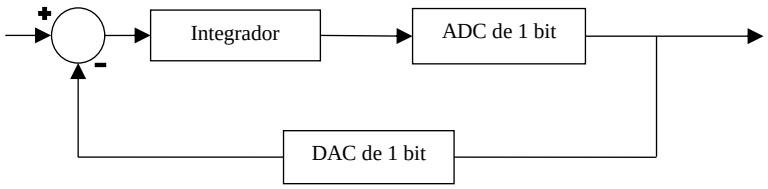

Figura 4: Esquema del modulador Sigma-Delta

Para explicar el funcionamiento del modulador nombraremos primero las señales correspondientes a cada nodo:

- $x(t)$ se corresponde con la señal de entrada.
- $\Sigma(t)$ es la salida del integrador.
- $d(t)$ es la salida del DAC de 1 bit.

- $y(t)$ es el resultado de la modulación.

La señal a la salida del integrador, que es la previa a la conversión ADC es la siguiente:

$$\Sigma(t) = x(t-1) - d(t-1) + \Sigma(t-1) \quad \textbf{(1.2)}$$

Si volvemos a la definición de error de cuantización del apartado anterior podemos describir su comportamiento temporal con la siguiente ecuación:

$$\varepsilon(t) = y(t) - \Sigma(t) \quad \textbf{(1.3)}$$

Juntando ambas ecuaciones obtenemos la señal de salida como:

$$y(t) = \varepsilon(t) + x(t-1) - d(t-1) + \Sigma(t-1) \quad \textbf{(1.4)}$$

Si asumimos que el DAC se comporta de manera ideal el valor a su salida es igual al de la salida del modulador, por tanto partiendo de las dos ecuaciones anteriores se cumple que:

$$y(t) = x(t-1) + \varepsilon(t) - \varepsilon(t-1) \quad \textbf{(1.5)}$$

Como podemos ver desaparecen los términos del integrador y el DAC y nos encontramos con la señal de la entrada retrasada una muestra y una resta entre términos de ruido, lo que significa que se traduce en una cancelación del mismo. Esta cancelación de ruido es lo que hace valiosa la modulación sigma delta.

Es además interesante comprobar el comportamiento en frecuencia de esta modulación, suponiendo una función de transferencia del integrador de $\frac{1}{s}$ tenemos que:

$$y(s) = \varepsilon(s) + \frac{1}{s}[x(s) - y(s)] \quad \textbf{(1.6)}$$

De manera que si despejamos $y(s)$ obtenemos:

$$y(s) = \varepsilon(s)\frac{s}{s+1} + x(s)\frac{1}{s+1} \quad \textbf{(1.7)}$$

Podemos ver que a cada término le corresponde una función de transferencia distinta, esto es un filtro paso-bajo de la señal de entrada y un filtro paso-alto del ruido de cuantización a la frecuencia de muestreo. Si volvemos al apartado 1.2.7 en el que se habla del ruido en los ADC se menciona que presenta una respuesta plana, esto es en los convertidores a frecuencias de Nyquist sin embargo como hemos visto gracias a la modulación sigma delta conseguimos realizar un filtrado paso alto del ruido a la frecuencia de muestreo, es ahí donde cobra importancia el *oversampling* empleado pues nuestra señal se encuentra a una frecuencia muy inferior. A este proceso mediante el cual conseguimos dar forma espectral al ruido se le denomina comúnmente *noise shaping* y es lo que hace tan popular la conversión Sigma-Delta. A continuación se muestra una gráfica con la respuesta en frecuencia de la señal y el ruido de cuantización tras la modulación:

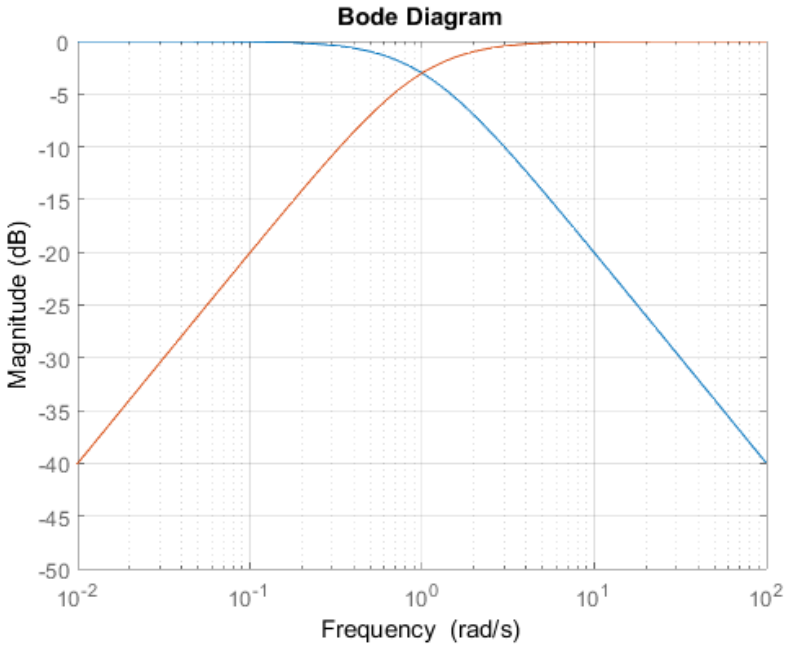

Figura 5: Respuesta en frecuencia de la señal y el ruido tras la modulación en azul y rojo respectivamente

Como podemos ver la figura del ruido presenta una caída de 20 dB por década hacia la frecuencia de muestreo, como era de esperar al presentar una función de transferencia de primer grado.

En los convertidores mostrados previamente cuyo proceso estaba basado en iteración o empleo de distintas etapas encadenadas era necesario que las primeras conversiones tuviesen extrema precisión ya que un error en un bit hacía inservible el valor de todos los bits que eran menos significativos. En el caso del convertidor Sigma-Delta cuyo funcionamiento se basa en el empleo de gran cantidad de muestras por muestra final no presenta dicho problema ya que al haber realimentación negativa un fallo en una muestra se compensa con los demás. Esto hace que la precisión de los elementos analógicos no sea necesariamente tan elevada como en el resto de convertidores, lo que se traduce en un diseño analógico bastante sencillo en comparación.

1.3.3 El modulador Sigma Delta de segundo orden

En la práctica el uso de Sigma-Delta de primer orden es poco común, ya que para conseguir una buena reducción de ruido es necesario aumentar demasiado la frecuencia de muestreo, por ello se emplean convertidores Sigma Delta de mayor orden. A continuación se muestra la estructura de un Sigma Delta de segundo orden:

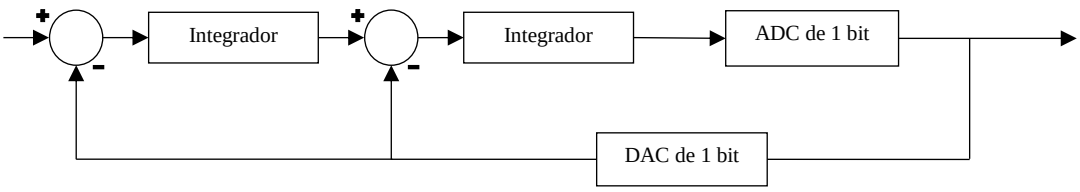

Figura 6: Modulador Sigma Delta de segundo orden

De nuevo nombramos las distintas señales para explicar el funcionamiento del diseño:

- $x(s)$ se corresponde con la señal de entrada.

- $\Delta_2(s)$ es entrada del segundo integrador.
- $y(s)$ es la salida del modulador.
- $\varepsilon(s)$ es el error de cuantización.

De nuevo suponemos $\frac{1}{s}$ como función de transferencia de los integradores de modo que a la salida tenemos:

$$y(s) = \varepsilon(s) + \frac{1}{s}[\Delta_2(s) - y(s)] \quad \textbf{(1.8)}$$

Donde Δ_2 es:

$$\Delta_2(s) = \frac{1}{s}[x(s) - y(s)] \quad \textbf{(1.9)}$$

De modo que al juntar ambas ecuaciones obtenemos:

$$y(s) = \varepsilon(s) + \frac{1}{s}\{\frac{1}{s}[x(s) - y(s)] - y(s)\} \quad \textbf{(1.10)}$$

Despejando la señal de salida obtenemos:

$$y(s) = \varepsilon(s)\frac{s^2}{s^2+s+1} + x(s)\frac{1}{s^2+s+1} \quad \textbf{(1.11)}$$

De nuevo obtenemos un filtrado paso bajo en la señal mientras que en el ruido obtenemos un filtrado paso alto de manera que la respuesta en frecuencia queda de la siguiente manera:

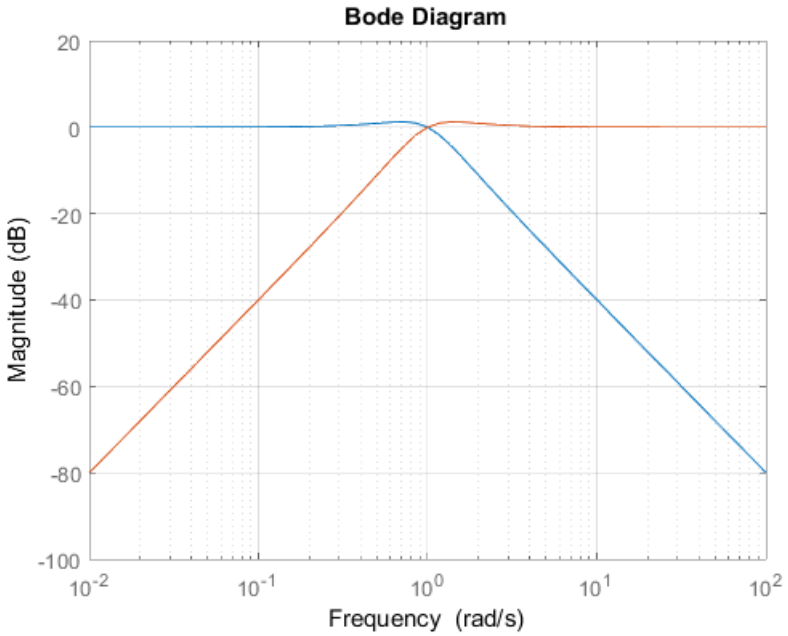

Figura 7: Respuesta en frecuencia de la señal y el ruido tras la modulación en azul y rojo respectivamente

Como podemos ver ahora la pendiente del modelado de ruido pasa a ser de 40 dB por década como podíamos intuir por la función de transferencia ya que es un paso alto de segundo orden. Esto significa que podemos obtener mayor reducción de ruido en nuestro ancho de banda empleando una frecuencia de muestreo menor que la que emplearíamos en un convertidor de primer orden.

Estructuras de orden superior se comportan de manera inestable y necesitan ajuste en la arquitectura por lo que no podemos simplemente añadir integradores en serie al diseño. Aun así la mejora conseguida simplemente añadiendo un integrador adicional al convertidor de primer

orden hace que comúnmente se empleen estructuras como mínimo de segundo orden ya que apenas aumenta la complejidad del diseño.

1.3.4 Variaciones adicionales

Además de aumentar el orden del modulador o aumentar la frecuencia de muestreo otra manera de aumentar la relación señal a ruido es aumentar la resolución del ADC interno empleado en la modulación. El ADC empleado en los convertidores Sigma Delta suele presentar la arquitectura Flash ya que es la más rápida y de fácil implementación siempre que sea de un número de niveles de cuantización reducido. Nótese que se habla de niveles de cuantización y no de número de bits, esto se debe a que al no ser la salida del modulador el elemento final del convertidor no estamos obligados a emplear la codificación binaria clásica y podemos emplear código unario o *thermometer code* de manera que el número de niveles de cuantización no esté restringido a 2^N y pueda tomar cualquier valor. Aun así deberemos tener en cuenta la codificación empleada cuando realicemos el procesado digital utilizando por ejemplo un decodificador a la entrada.

1.3.5 Modulación en cascada MASH

Otra manera de aumentar el número de integradores empleados solucionando los problemas de inestabilidad mencionados para moduladores de orden superior a dos es la implementación de moduladores en cascada en lugar de en serie de manera que se emplea el resultado de la primera modulación restado con la salida del integrador de dicha etapa como entrada de la segunda etapa y así en adelante en caso de haber más etapas. La entrada del segundo modulador es exactamente el error de cuantización de la primera etapa, con esto lo que se consigue es cancelar el error de la etapa anterior.

1.3.6 Filtrado digital

Hasta ahora solo hemos explicado cómo podemos desplazar el ruido de baja a alta frecuencia empleando la modulación Sigma Delta, pero a la salida del bloque continuamos teniendo un tren de pulsos a muy baja resolución y a frecuencia de *oversampling* es por ello que se emplean procesos digitales para obtener la señal a la frecuencia deseada a mayor resolución. Según la aplicación podemos encontrar distintos modos de abordar el filtrado, uno de estos métodos es el uso de un filtro FIR seguido de un diezmado de la señal a la salida. El filtro FIR empleado es un filtro paso bajo con frecuencia de corte igual al ancho de banda, la salida de este filtro es un flujo de datos a la frecuencia de sobremuestreo y la resolución de bits obtenible dependerá de la relación señal a ruido conseguida en banda. Posteriormente como hemos dicho se aplica un diezmado para reducir el número de muestras al correspondiente para nuestro ancho de banda, esto es descartar todos menos uno cada $\frac{f_{os}}{f_s}$ donde f_{os} se corresponde con la frecuencia de sobremuestreo y f_s con la de nuestro ancho de banda, a la relación $\frac{f_{os}}{f_s}$ se le conoce también como relación de sobremuestreo o *oversampling* ratio (OSR).

1.4 Herramientas empleadas

Para la realización de este trabajo se han empleado distintas herramientas ya sea software o lenguajes de programación, estas serán descritas en este apartado.

1.4.1 MATLAB

MATLAB es una herramienta de software matemática desarrollada por MathWorks que permite realizar cálculos con grandes volúmenes de datos y la representación de funciones. Las operaciones se realizan empleando el lenguaje de programación propio de MATLAB (llamado también MATLAB). Además de la aplicación base MathWorks desarrolla distintos paquetes de

contenidos adicionales que agregan funcionalidades al entorno, los hay de muchos tipos cubriendo gran variedad de usos algunos de ellos relacionados con campos bastante específicos y sin relación entre ellos. Una de las herramientas más populares de MATLAB es Simulink que permite modelar y simular en nuestro caso sistemas de señales. Empleamos Simulink pues nuestro punto de partida para realizar el diseño en Verilog-A es un modelo Simulink que debemos adaptar. Además emplearemos dicho modelo para probar las distintas configuraciones y comparar su comportamiento con el conseguido en nuestro diseño.

1.4.2 Verilog-A

Verilog-A es un lenguaje de descripción de hardware que permite describir el funcionamiento de sistemas analógicos mediante ecuaciones matemáticas que relacionan sus entradas y salidas. Existe como subset puramente analógico del lenguaje Verilog-AMS que permite el diseño de sistemas mixtos, es decir que combina el uso de señales analógicas con digitales. Este lenguaje a su vez nace como un derivado del lenguaje puramente digital Verilog.

Las principales características y diferencias con Verilog son:

- Verilog-A permite la descripción de sistemas conservativos, en nuestro caso eléctricos, aunque también sirve para describir comportamientos mecánicos, de fluidos o termodinámicos. Los puertos descritos en los módulos (entradas y salidas) se comportan como nodos eléctricos que cumplen las leyes de Kirchhoff y tienen un valor de tensión y de corriente. Los nodos eléctricos se declaran como "*electrical"*.

- El comportamiento del sistema se define dentro del bloque analog que contiene las distintas ecuaciones, podemos emplear variables auxiliares "*real"* para manejar los valores internamente y asignarlos posteriormente a los nodos eléctricos con el operador de contribución "<+". El acceso a los valores de tensión y corriente se hace con las funciones V() y I() respectivamente (por ejemplo V(nodo A, nodo B)), con la función V() obtenemos la tensión entre dos nodos y si solo introducimos uno obtendremos la tensión entre dicho nodo y el de referencia que es siempre cero.

- Desaparece la asignación no bloqueante "<=" de Verilog que permite que los valores de las variables en un evento no se actualicen hasta la finalización del mismo.

1.4.3 Virtuoso

Virtuoso es una plataforma de la multinacional americana Cadence Design Systems especializada en crear aplicaciones especializadas para el diseño de productos electrónicos, ya sean circuitos integrados, SOCs o PCBs. Virtuoso es una aplicación creada para diseño de circuitos integrados permitiendo desde el diseño esquemático o de layouts a el diseño de módulos mediante la descripción de su comportamiento con Verilog-A. Además incorpora potentes herramientas de simulación que permite realizar gran variedad de análisis en nuestros diseños.

Capítulo 2. Desarrollo de los modelos

En este capítulo se presentan los modelos realizados en Verilog-A de los bloques internos del convertidor sigma delta, la estructura de este capítulo consistirá en alternar los modelos de partida en Simulink con los modelos realizados realizando además comentarios sobre funciones concretas empleadas. Los modelos se han realizado para que sean parametrizables compartiendo los mismos nombres de parámetros que los que se encuentran en el diseño original en Simulink.

2.1 Esquema completo

A continuación se muestra el esquema completo del modulador Sigma Delta en Simulink, en él se observan los distintos bloques que se han creado posteriormente en Verilog-A.

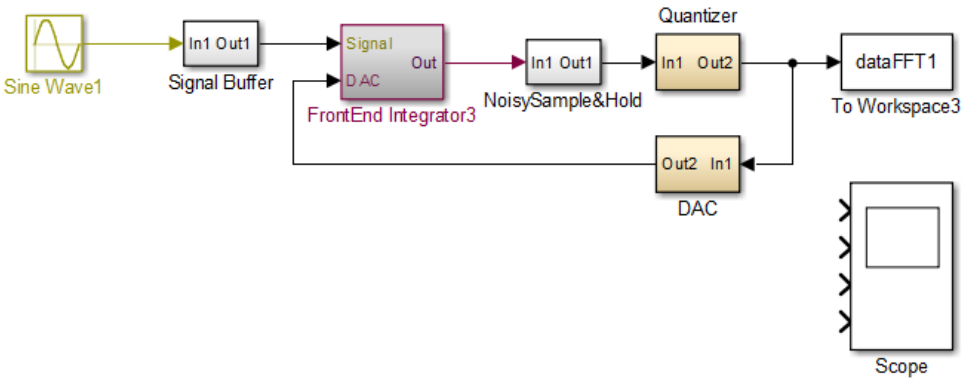

Figura 8: Esquema del modulador Sigma Delta en simulink

En la Figura 8 se observan los siguientes bloques:

- *Sine Wave1*: Es un generador sinusoidal que empleamos para comprobar el correcto funcionamiento del sistema simulando la señal de entrada.
- *Signal Buffer*: Es el bloque con el que la señal entra al circuito, simplemente añade ruido a la entrada del sistema.
- *FrontEnd Integrator3*: Es el integrador del sistema, incorpora la resta entre la señal de entrada y la que proviene del DAC.
- *NoisySample&Hold*: Circuito de Sample & Hold a la entrada del cuantizador, su tasa de muestreo marca la tasa de muestreo del sistema completo.
- *Quatizer*: Cuantizador de un bit.
- *DAC*: Convertidor digital analógico de un bit.
- *ToWorkspace3*: Importa la señal a la salida del cuantizador al área de trabajo de MATLAB para que podamos operar con ella, para por ejemplo calcular la SNR.
- *Scope*: Bloque que hace de osciloscopio pudiéndose conectar a cualquier punto del sistema para ver la señal en dicho punto.

2.2 Signal Buffer

Como hemos mencionado antes el bloque de *Signal Buffer* funciona como puerto de entrada al sistema y su modelo Simulink es el siguiente:

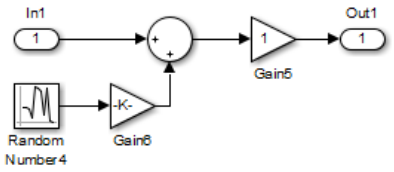

Figura 9: Modelo del *Signal Buffer* en Simulink

El modelo cuenta con los siguientes elementos:

- In1 y Out1: Funcionan como puertos de entrada y salida respectivamente.
- Random Number4: Generador de números aleatorios, sirve para simular el ruido del bloque.
- Gain6: Multiplica la salida del generador aleatorio para que los números tomen la amplitud correspondiente al ruido del bloque.
- Bloque sumador: Se emplea para añadir el ruido a la señal de entrada In1.
- Gain5: Multiplica por 1 el valor anterior, no se ha tenido en cuenta para realizar el modelo en Verilog-A.

2.2.1 Modelo en Verilog-A

```
// VerilogA for prueba, Signal_Buffer, veriloga


`include "constants.vams"
`include "disciplines.vams"

//----------------------
//
// Signal Buffer
// vin:     [V,A]
// vout:    [V,A]
//
// INSTANCE parameters
//    VnSigBuf = Noise amplitude
//    Noise_En = Noise enable
//    bandwidth = bandwidth for noise analysis


module Signal_Buffer(vin, vout);
input vin;
output vout;
electrical vin, vout;

parameter real VnSigBuf = 100e-6;
parameter integer Noise_En = 0 from [0:1];
parameter real bandwidth = 2e6;

electrical noise;
real noisepwr;

analog begin

noisepwr = (VnSigBuf*VnSigBuf)/bandwidth;
V(noise) <+ white_noise(noisepwr);
V(vout) <+ V(vin) + V(noise)*Noise_En;

end


endmodule
```

Este módulo posee los siguientes parámetros:

- VnSigBuf: Es el valor de tensión de ruido correspondiente al bloque.
- Noise_En: Multiplica el valor del ruido antes de sumarlo a la salida, puede tomar como valor 0 o 1, por tanto sirve para activar o desactivar el ruido del bloque.
- bandwidth: Ancho de banda para cálculos de ruido.

Todos estos parámetros se fijan externamente desde el esquemático, de no ser así toman los valores que tienen asignados por defecto.

Como podemos ver tenemos dentro del modelo un valor de ancho de banda, este se corresponde con el ancho de banda del sistema completo, es decir, la frecuencia de *oversampling* esto se debe a que la función empleada para introducir ruido (white_noise( )) funciona con valores de potencia espectral y para obtenerlos es necesario dividir por el ancho de banda. Además hemos creado un

nodo adicional para el ruido pues la función de ruido solo se puede emplear sobre nodos con el operador de contribución. Posteriormente el ruido es sumado (dependiendo del valor de Noise_En) a la tensión de entrada.

Una vez terminado el código y comprobado que no da ningún error al compilarlo generamos el siguiente símbolo que identificará al módulo en el esquemático:

Figura 10: Símbolo del módulo *Signal Buffer*

Una vez tenemos el modelo pasamos a comprobar su funcionamiento, para ello creamos un esquemático de test con los estímulos correspondientes para su simulación:

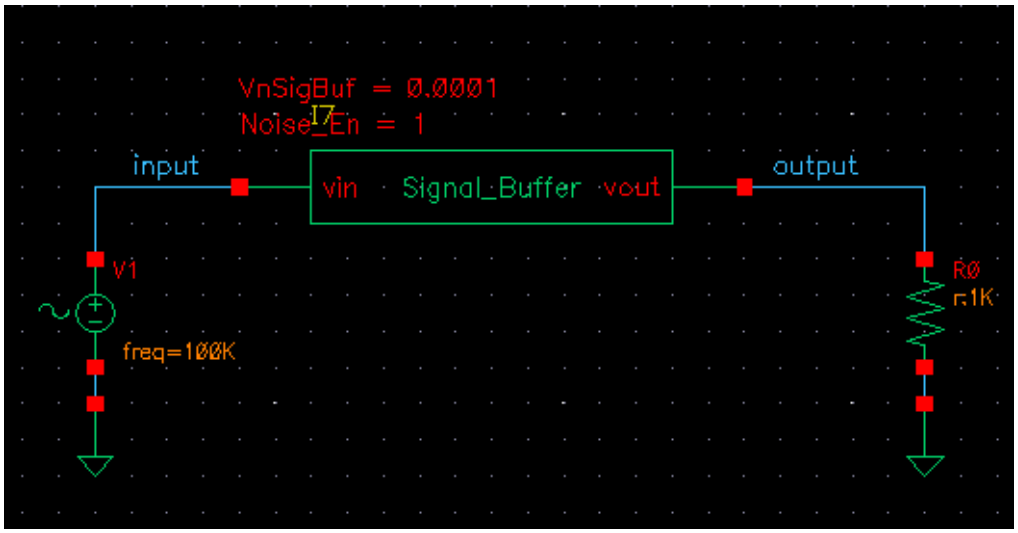

Figura 11: Esquemático de test del módulo *Signal Buffer*

Terminado el esquemático podemos pasar a testear el funcionamiento del módulo, para ello realizamos un análisis transitorio activando la opción de ruido para que el simulador sepa que debe generar el ruido de la función white_noise( ), de no activarlo esta no hace efecto.

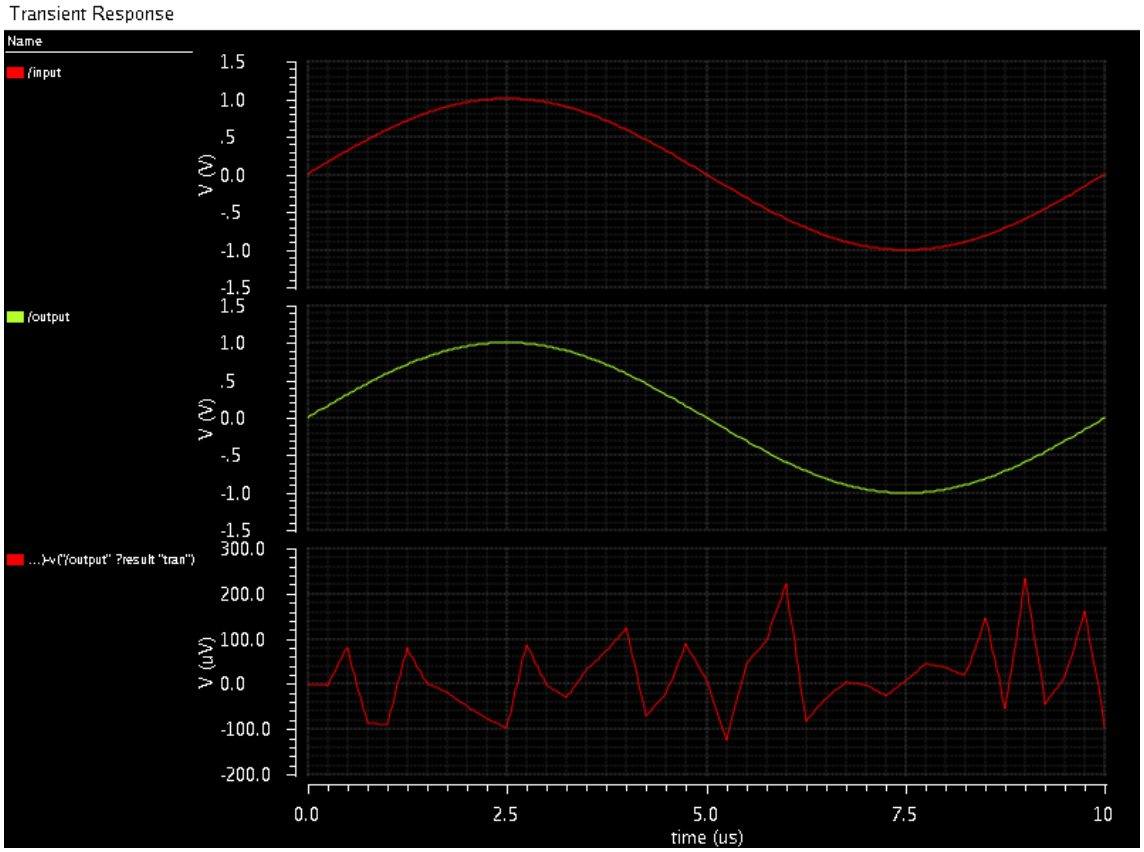

Figura 12: Simulación del módulo *Signal Buffer*

En la figura superior podemos observar los resultados de la simulación sobre nuestro modelo de *Signal Buffer* las señales son (de arriba a abajo) entrada, salida y ruido. El ruido se ha obtenido restando las señales de entrada y salida. Como podemos ver funciona correctamente.

2.3 FrontEnd Integrator

En cuanto al integrador el modelo es el siguiente:

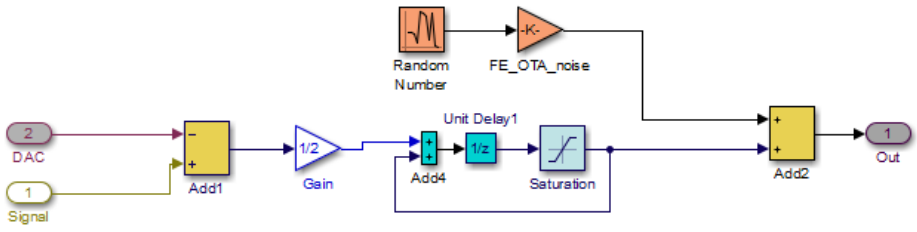

Figura 13: Modelo en Simulink del *FrontEnd Integrator*

El diseño cuenta con los siguientes elementos:

- DAC, Signal y Out: Se corresponden con los puertos de entrada y salida.
- Bloques sumadores.
- Combinación de generador de números aleatorios con amplificador para imitar ruido de igual manera que en el módulo anterior.
- Unit Delay: Retrasa el resultado de la suma una muestra.

- Saturation: Este bloque recorta la señal en caso de que se superen los valores de saturación (1 y -1).

2.3.1 Modelo en Verilog-A

```
// VerilogA for prueba, FrontEndIntegrator, veriloga


`include "constants.vams"
`include "disciplines.vams"


//---------------------
// FrontEnd Integrator
//
// vinp:    [V,A]
// vinn:    [V,A]
// vout:    [V,A]
//
// INSTANCE parameters
//    gain = integrator gain;
//    Noise_En = Enable noise
//    bandwidth = noise bandwidth
//    alpha
//    gamma
//    Cload
//    Cint = integration capacitor
//    Cpix = Input total Capacity
//    Cs = S&H capacitor
//


module FrontEndIntegrator(vinp, vinn, vout);


input vinp,vinn;
output vout;

electrical vinp, vinn, vout;
real diff;
real out;
parameter real gain = 1e9;

parameter integer Noise_En = 0 from [0:1];
parameter real bandwidth = 2e6;
parameter real alpha = 2.0;
parameter real Cint = 50e-14;
parameter real Cpix = 2e-12;
parameter real Cs = 2e-12;
parameter real Cload = 21e-12;
parameter real k = 1.38e-23;
parameter real gamma = 1.2;

electrical noise;
real noisepwr;
real integ;

real beta;
```

```
analog begin

    beta = Cint/(Cpix+Cint);

    diff = 0.5 * V(vinp,vinn);
    out = gain * idt(diff,0);
    if(out > 1) out = 1;
    else if(out <-1) out = -1;

    noisepwr = (k*$temperature*gamma*alpha/(beta*(Cs+Cint*(1-
beta)+Cload)))/bandwidth;

    V(noise) <+ white_noise(noisepwr);
    V(vout) <+ out + V(noise)*Noise_En;

end

endmodule
```

Este módulo contiene los siguientes parámetros:

- gain: Ganancia del integrador.
- Noise_En: Multiplica el valor del ruido antes de sumarlo a la salida, puede tomar como valor 0 o 1, por tanto sirve para activar o desactivar el ruido del bloque.
- bandwidth: Ancho de banda para cálculos de ruido.
- alpha, gamma, Cint, Cpix, Cload y Cs: Son distintos parámetros empleados para definir el ruido del integrador.

De nuevo estos parámetros son fijados desde el esquemático. Aun así los valores asignados por defecto a los parámetros necesarios para el cálculo de ruido están fijados por defecto con sus valores correspondientes, por tanto no sería necesario ajustarlos.

El comportamiento del integrador como tal viene definido con la función idt( ), que realiza una integral del comportamiento a la entrada del valor que empleemos, en este caso *diff*, empleamos un parámetro de ganancia que es fijado desde el esquemático y se corresponde con el valor necesario para que el tiempo de integración se corresponda con el de la frecuencia de muestreo. El comportamiento de la saturación del integrador viene definido en las líneas siguientes con el uso de condicionales que limitan el valor dentro del rango [-1,1] V. Nótese también que para realizar la resta entre las tensiones de entrada podemos emplear directamente la función V(vinp,vinn) en lugar de obtener los valores de cada nodo por separado y restarlos.

El símbolo generado para el integrador es el siguiente:

Figura 14: Símbolo del módulo *FrontEnd Integrator*

Para simular el funcionamiento del integrador hemos creado el siguiente esquemático:

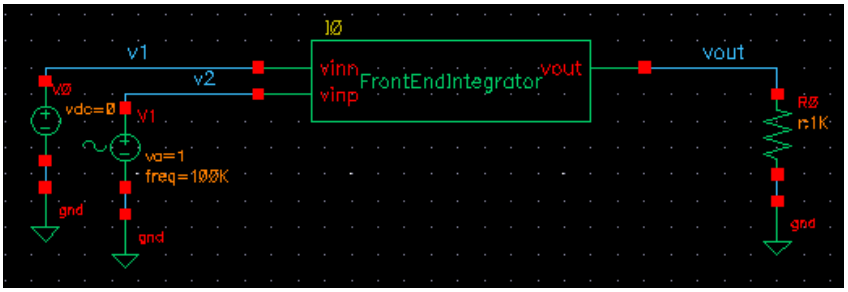

Figura 15: Esquemático de test del módulo *FrontEnd Integrator*

Realizar un testeo certero del comportamiento de este módulo de manera independiente resulta complicado pues las señales que tendríamos a la entrada en su uso real dependen del sistema completo. Esto hace que sean complicadas de emular, una manera de hacerlo sería generar las formas de onda a partir del modelo de MATLAB, pues se pueden introducir formas de onda concretas para simular entradas en simulaciones. Aun así nos limitaremos a comprobar el funcionamiento del integrador como tal ajustando los valores de la ganancia para obtener la salida deseada. En las entradas situamos una tensión nula en la entrada negativa del integrador y una senoidal a la entrada.

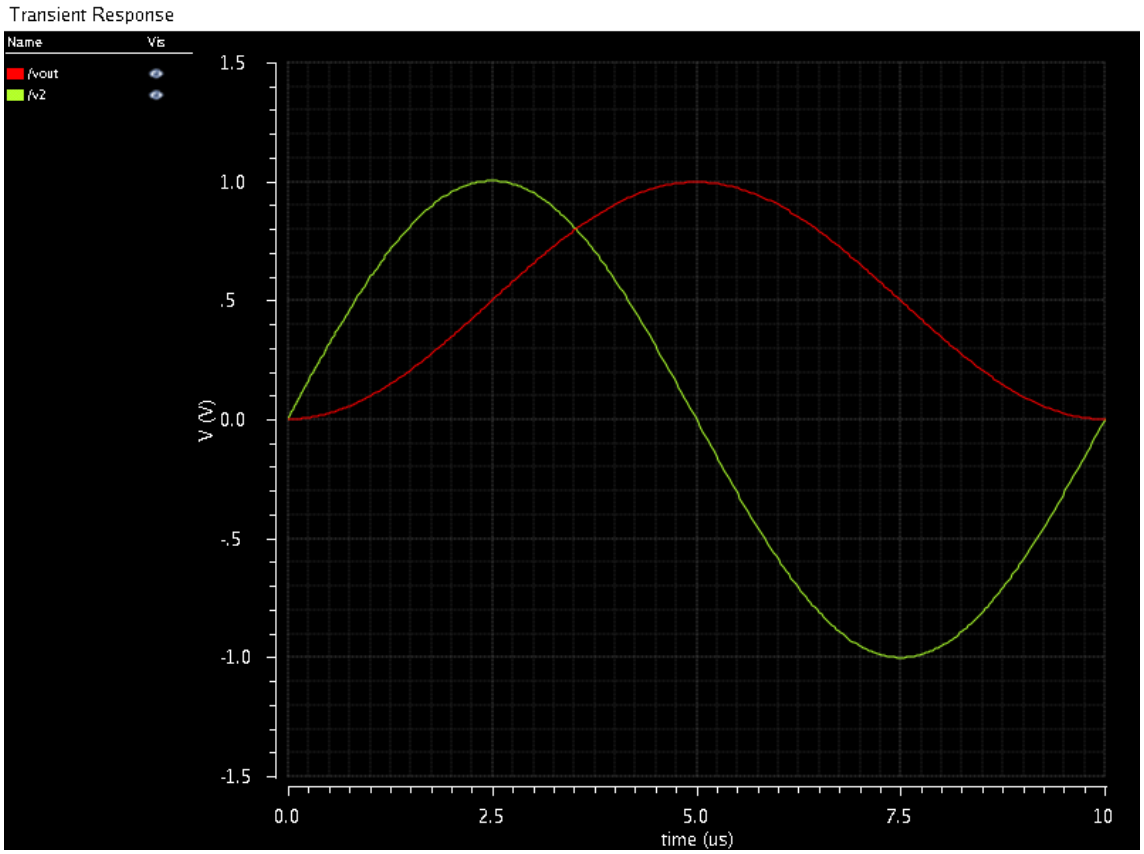

Figura 16: Simulación del módulo *FrontEnd Integrator*

En la imagen superior observamos el resultado de simular el comportamiento del integrador durante el ciclo de una señal sinusoidal. Podemos comprobar que funciona correctamente pues sabemos que la integral del seno es el coseno y tiene la mitad de amplitud porque la resta de la etrada entrada tiene una ganancia de 0.5.

2.4 Noisy Sample & Hold

El siguiente bloque es el de *Sample and Hold* que como hemos mencionado previamente muestrea a la frecuencia de *oversampling.* Su estructura es la siguiente:

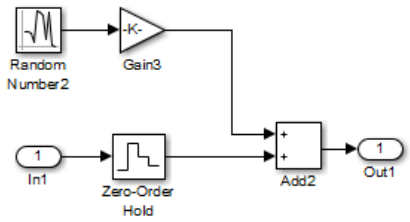

Figura 17: Modelo en Simulink del *Noisy Sample & Hold*

Este módulo cuenta con los siguientes elementos:

- In1 y Out1: Funcionan como puertos de entrada y salida respectivamente.
- Bloques de generación aleatoria más ganancia para simular ruido.
- Zero-Order Hold: Bloque que realiza el *Sample and Hold*
- Add2: Bloque que suma el ruido a la señal de Hold.

2.4.1 Modelo en Verilog-A

```
// VerilogA for prueba, Noisy_SampleandHold, veriloga

`include "constants.vams"
`include "disciplines.vams"

//-----------------------
// Noisy Sample and Hold
//
// vin:     [V,A]
// vout:    [V,A]
// vclk:    [V,A]
//
// INSTANCE parameters
//    vtrans_clk = transition voltage of the clock [V]
//    Cs = sample and hold capacitor
//    Noise_En = Enable noise
//    bandwidth = noise bandwidth

module Noisy_SampleandHold(vin, vclk, vout);
input vin,vclk;
output vout;

parameter real Cs = 2e-12;
parameter integer Noise_En = 0 from [0:1];
parameter real k = 1.38e-23;
parameter bandwidth = 2e6;

electrical vin,vclk,vout;
real out;
```

```
parameter real vtrans_clk = 2.5;

electrical noise;
real noisepwr;

analog begin

noisepwr = k/Cs*$temperature/bandwidth;

V(noise) <+ white_noise(noisepwr);

@(cross(V(vclk) - vtrans_clk, 1))
    out = V(vin);

V(vout) <+ out + V(noise)*Noise_En;
end

endmodule
```

Los parámetros de este módulo son los siguientes:

- vtrans_clk: Tensión de transición del reloj
- Cs: Capacidad del *Sample and Hold*, se emplea para calcular el ruido.
- Noise_En: Multiplica el valor del ruido antes de sumarlo a la salida, puede tomar como valor 0 o 1, por tanto sirve para activar o desactivar el ruido del bloque.
- bandwidth: Ancho de banda para cálculos de ruido.

El Sample and Hold funciona controlado por el reloj, por ello para describir su comportamiento debemos definir el evento de dependiente del reloj. Para ello como en Verilog tenemos @ que marca la definición de un evento, entre paréntesis se encuentran las condiciones de activación. En Verilog-A podemos imitar las descripciones de sensibilidad ante flanco de reloj posedge y negedge con la función cross( ) que devuelve un uno cuando se produce un cruce en el valor indicado, el segundo parámetro nos permite indicar si debe activarse al producirse un cruce ascendente, descendente o en ambos, en nuestro caso como queremos que se active ante el flanco de subida lo fijamos con un 1. Mediante el uso de eventos hacemos que la variable interna out solo se actualice ante flancos de reloj, siendo este el funcionamiento de un *Sample and Hold*.

El símbolo creado para este modelo es el siguiente:

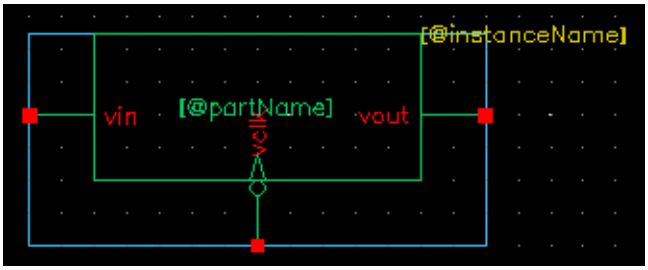

Figura 18: Símbolo del módulo *Noisy SampleandHold*

Para comprobar el funcionamiento del modelo hemos creado el siguiente esquemático de test:

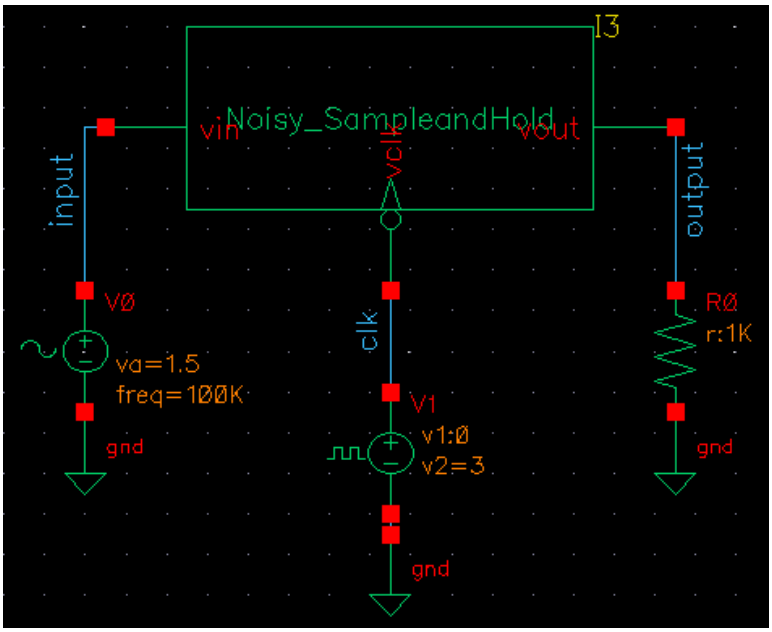

Figura 19: Esquemático de test del módulo *Noisy SampleandHold*

Para comprobar el funcionamiento del *Sample and Hold* simplemente introducimos una señal a la entrada y una señal de reloj. Los resultados de la simulación son los siguientes:

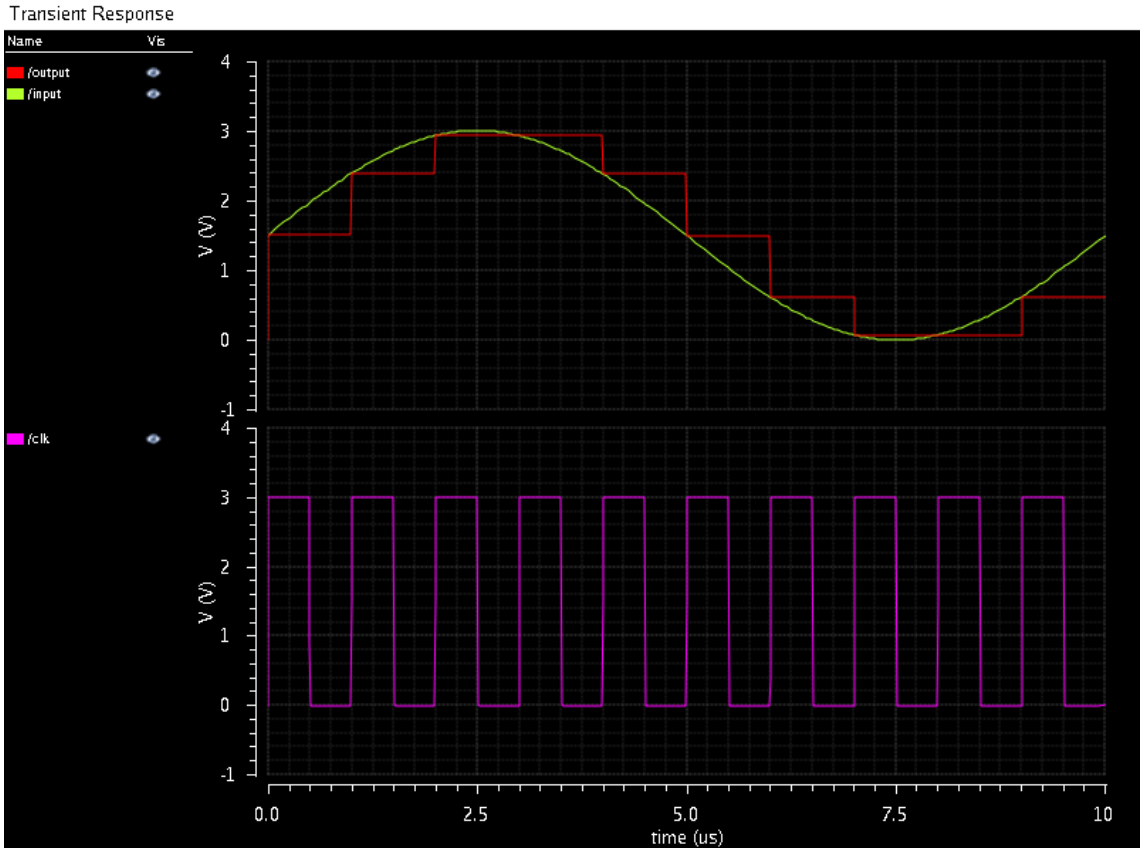

Figura 20: Simulación del módulo *Noisy SampleandHold*

Como podemos ver el módulo funciona correctamente y la señal de salida retiene los valores de la entrada durante un ciclo de reloj completo con cada flanco de subida.

2.5 Quantizer

El cuantizador se encarga de realizar la conversión a un bit de la señal que proviene del *Sample and Hold* y su esquema es el siguiente:

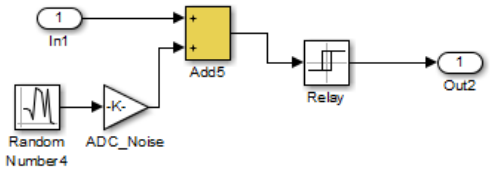

Figura 21: Modelo en Simulink del *Quantizer*

En el cuantizador nos encontramos con los siguientes elementos:

- In1 y Out2: Se corresponden con los puertos de entrada y salida.
- Random Number y ADC_Noise: Se emplean para simular el efecto del ruido
- Add5: Suma el ruido a la entrada.
- Relay: Actua como cuantizador de un bit.

Nótese que el ruido en este bloque viene añadido previamente al cuantizador y no a la salida como en el resto de bloques, esto se debe a que una vez realizamos la conversión digital la señal no es afectada por el ruido.

2.5.1 Modelo en Verilog-A

```
// VerilogA for prueba, Quantizer, veriloga

`include "constants.vams"
`include "disciplines.vams"

//----------------------
// Quantizer
// vin:      [V,A]
// vout:     [V,A]
//
// INSTANCE parameters
//    vhigh = output high    [V]
//    vlow = output low      [V]
//    VnADC = quantizer noise    [V]
//    bandwidth = bandwidth for noise analysis
//    Noise_En = noise enable

module Quantizer(vin,vout);
input vin;
output vout;

parameter real ttr = 1e-12;
```

```
electrical vin, vout;
real out;
parameter real VnADC = 7.17e-5;
parameter real bandwidth = 2e6;
parameter integer Noise_En = 0 from [0:1];
parameter real vhigh = 1;
parameter real vlow = -1;


electrical noise;
real noisepwr;

analog begin

noisepwr = (VnADC*VnADC)/bandwidth;

V(noise) <+ white_noise(noisepwr);

if((V(vin) + V(noise)*Noise En)>0)out = vhigh;
else out = vlow;

V(vout) <+ out;

end

endmodule
```

En este módulo se emplean los siguientes parámetros:

- VnADC: Es el valor de tensión de ruido correspondiente al bloque.
- Noise_En: Multiplica el valor del ruido antes de sumarlo a la salida, puede tomar como valor 0 o 1, por tanto sirve para activar o desactivar el ruido del bloque.
- bandwidth: Ancho de banda para cálculos de ruido.
- vhigh: Valor de la tensión a nivel alto.
- vlow: Valor de la tensión a nivel bajo.

Este módulo simula un convertidor ADC Flash de un bit o dos niveles e igual que en el modelo de Simulink el ruido se tiene en cuenta previamente a la cuantización.

El símbolo creado para este componente es el siguiente:

Figura 22: Símbolo del *Quantizer*

Para comprobar el funcionamiento del cuantizador se ha creado el siguiente esquemático:

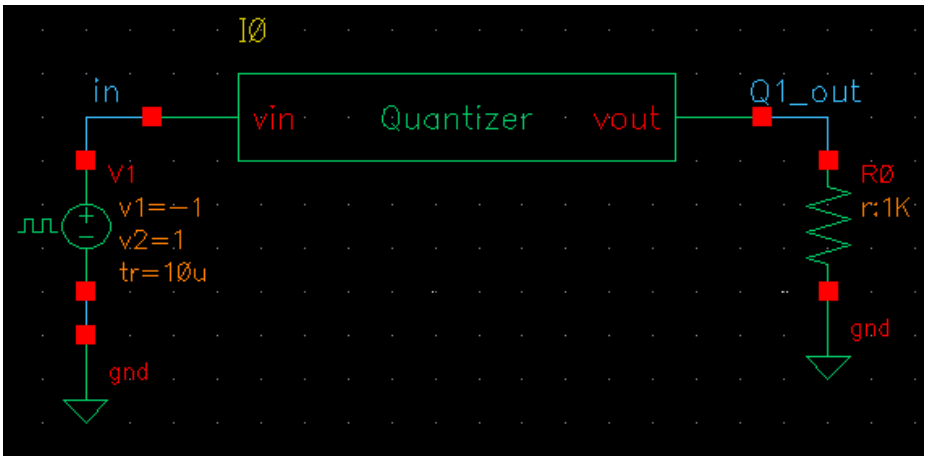

Figura 23: Esquemático de test del *Quantizer*

Para simular el funcionamiento del cuantizador introducimos una señal triangular cuya tensión oscila dentro del rango de funcionamiento de nuestro sistema para ver que la cuantización se produce de manera adecuada. El resultado obtenido es el siguiente

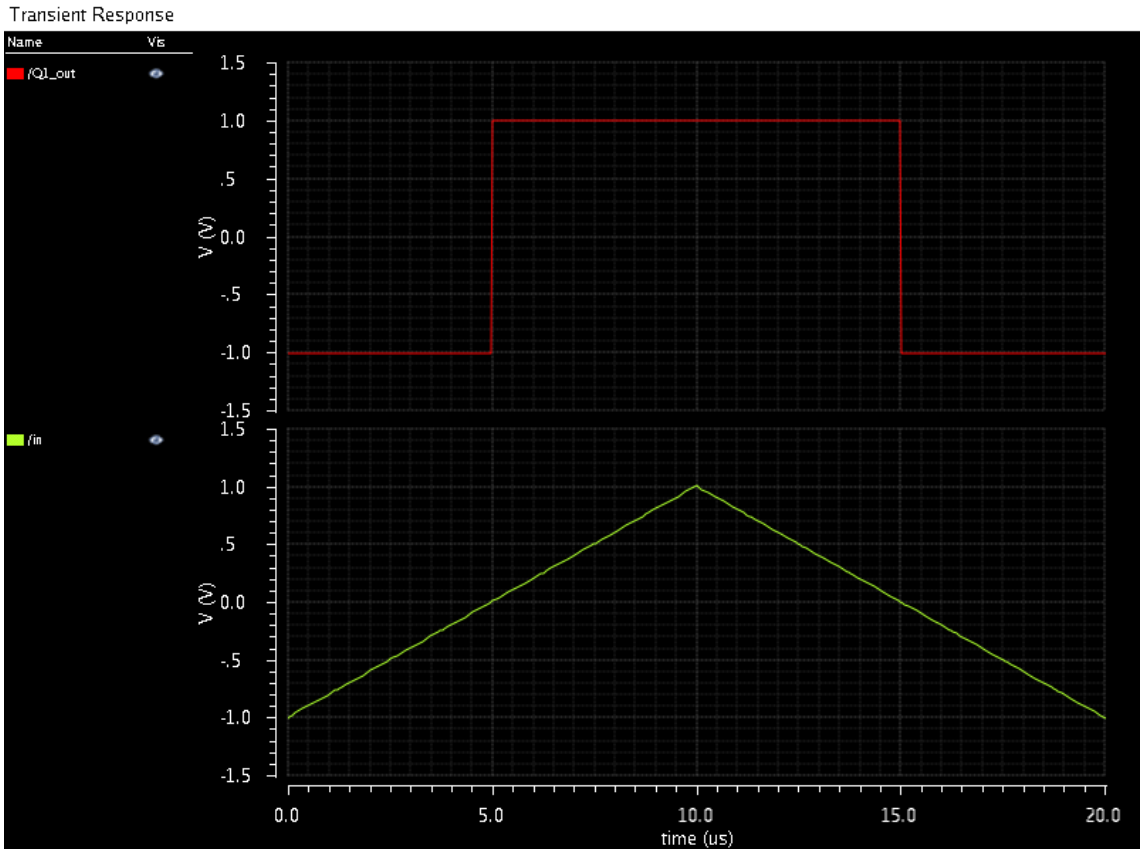

Figura 24: Simulación del módulo *Quantizer*

Como podemos ver el cuantizador funciona correctamente ya que la tensión a la salida mide 1V siempre que la entrada es positiva y -1V cuando es negativa. Cabe mencionar que en nuestro sistema los valores de la entrada no varían de forma continua como sí lo hace en este caso ya que previamente al cuantizador está implementado el *Sample and Hold.*

2.6 DAC

El modelo en Simulink del DAC de un bit es el siguiente:

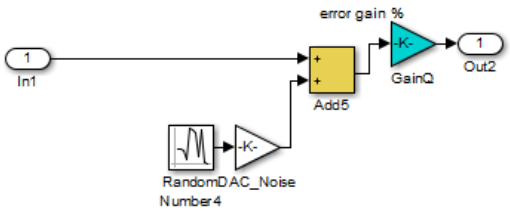

Figura 25: Modelo en Simulink del DAC

No es necesario comentar los bloques de este módulo pues es similar al de Signal Buffer, solo que en este la ganancia es diferente a la unidad y viene afectada por un parámetro de error de ganancia.

2.6.1 Modelo en Verilog-A

```
// VerilogA for prueba, DAC, spectre

`include "constants.vams"
`include "disciplines.vams"
`define MAXINT 2_147_483_647.0

//----------------------
//
// One bit DAC
// vin:           [V,A]
// vout:    [V,A]
//
// INSTANCE parameters
//    Prcnt = percentage of error gain, can be
//          either positive or negative
//    VnDAC = Noise amplitude
//    Noise_En =  noise enable


module DAC(vin, vout);
input vin;
output vout;

electrical vin, vout;

parameter real Prcnt = 0;
parameter real VnDAC = 7.17e-05;
parameter integer Noise_En = 0 from [0:1];
parameter real bandwidth = 2e6;

electrical noise;
real noisepwr;
```

```
analog begin

noisepwr = (VnDAC*VnDAC)/bandwidth;

V(noise) <+ white_noise(noisepwr);
V(vout) <+ (V(vin)) * (1 + Prcnt * 0.01) + V(noise)*Noise_En;
end

endmodule
```

Este módulo contiene los siguientes parámetros:

- VnDAC: Es el valor de tensión de ruido correspondiente al bloque.
- Noise_En: Multiplica el valor del ruido antes de sumarlo a la salida, puede tomar como valor 0 o 1, por tanto sirve para activar o desactivar el ruido del bloque.
- bandwidth: Ancho de banda para cálculos de ruido.
- Prcnt: Porcentaje de error de la ganancia del convertidor.

El símbolo creado para el modelo del DAC es el siguiente:

Figura 26: Símbolo del DAC

Como hemos dicho antes este bloque se comporta de manera similar al Signal Buffer, por ello no es necesario comentar la simulación de su comportamiento. En realidad la definición que se ha hecho de este bloque no es la que corresponde, pues en realidad el DAC debería funcionar por ejemplo mediante comparación y generando el valor según la tensión detectada a la entrada. El motivo por el que no se ha realizado un modelo más fiel a la realidad es porque en nuestro diseño trabajamos con una señal digital que no presenta ruido y sus valores se corresponden con los que encontramos a la salida del DAC, por tanto podemos prescindir de realizar una descripción más precisa, lo que si nos interesa es incluir el ruido generado por el propio DAC.

2.7 Bloques adicionales

Como mencionamos previamente en el apartado 1.3.3 en el que se mencionan variaciones adicionales al modelo del modulador Sigma Delta una de las formas de aumentar el SNR del sistema es aumentar la resolución del convertidor interno, es por ello que se han realizado modelos adicionales para simular este efecto. El modelo creado en Simulink para el convertidor de 3 niveles es el siguiente:

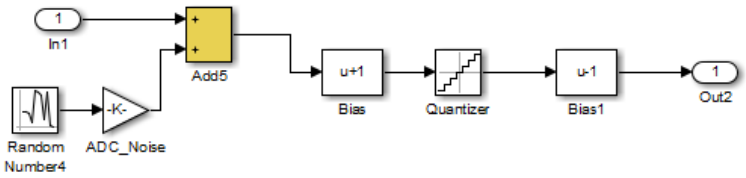

Figura 27: Modelo en Simulink del cuantizador de 3 niveles

Este cuantizador contiene los siguientes elementos:

- In1 y Out2: Se corresponden con los puertos de entrada y salida.
- Random Number y ADC_Noise: Se emplean para simular el efecto del ruido
- Add5: Suma el ruido a la entrada.
- Quantizer: Produce la señal cuantizada a distintos niveles según configuremos sus propiedades.
- Bias: Sirven para introducir un offset a la señal.

Como podemos ver se han empleado dos bloques que introducen y eliminan el offset respectivamente, esto tiene un motivo, el cuantizador funciona según el intervalo introducido contando a partir de cero, esto hace que el cruce por cero siempre lo tome como valor de cuantización, por ello se desplaza la señal para evitar este comportamiento. En este caso concreto no supondría problema pues los tres niveles son -1V, 0V y 1V pero diseñándolo de esta manera podemos reutilizar el diseño para cuantizaciones de más niveles.

Hay que tener en cuenta que las simulaciones que se realicen en MATLAB son orientativas en comparación a las que podamos realizar en Virtuoso, por ello podemos prescindir de realizar la cuantización para que a la salida tengamos los 3 bits correspondientes en *thermometer code* y hacer directamente que la señal a la salida sea el valor de señal que representarían esos bits. Esto nos permite además reutilizar el bloque DAC como bloque que añade ruido ya que del otro modo deberíamos hacer un nuevo modelo de DAC para este caso concreto que fuese capaz de realizar la conversión correspondiente.

Además del bloque de cuantización de tres niveles se ha realizado también el de cuatro niveles, como el proceso de diseño es el mismo no es necesario explicar su diseño. Además el convertidor de cuatro niveles sí que convierte la señal al código binario habitual en lugar de a *thermometer code* al ser cuatro potencia de dos.

Capítulo 3. Funcionamiento

En este capítulo comprobaremos el funcionamiento del modulador Sigma Delta en distintas configuraciones en Simulink y posteriormente en virtuoso. Las configuraciones testeadas son las siguientes:

- Modulador Sigma Delta de primer orden.
- Modulador Sigma Delta de segundo orden.
- Modulador Sigma Delta de segundo orden con cuantizador de tres niveles.
- Modulador Sigma Delta de segundo orden con cuantizador de cuatro niveles.

Todas estas configuraciones serán probadas con un *oversampling ratio* de 32 y de 64. El motivo de realizar estas simulaciones en MATLAB previamente a las simulaciones en Virtuoso es que en MATLAB son cálculos mucho menos intensos y apenas necesitan tiempo en comparación a las que realicemos posteriormente en Virtuoso que aunque serán más fiables ocupan bastante tiempo. Por ello consideramos apropiado partir de simulaciones en Simulink para obtener una idea de los resultados y comprobar que la teoría detrás de las configuraciones se sostiene.

3.1 Modulador Sigma Delta de primer orden

Aunque el modelo de esta configuración nos ha sido dado de partida realizaremos las simulaciones para comprobar los efectos de variar el *oversampling ratio*. A continuación se muestra de nuevo el esquema en Simulink del modulador de primer orden:

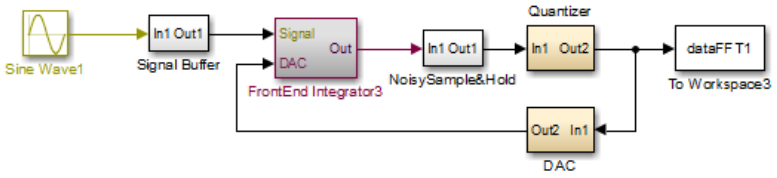

Figura 28: Esquema del modulador Sigma Delta de primer orden en Simulink

Las simulaciones mostradas a lo largo de este capítulo se corresponderán con la respuesta en frecuencia obtenida tras modular una señal de 1MHz y en algunos casos a 10 KHz adicionalmente a la salida del cuantizador. A continuación se muestran las simulaciones correspondientes a este apartado con un *oversampling* de 32:

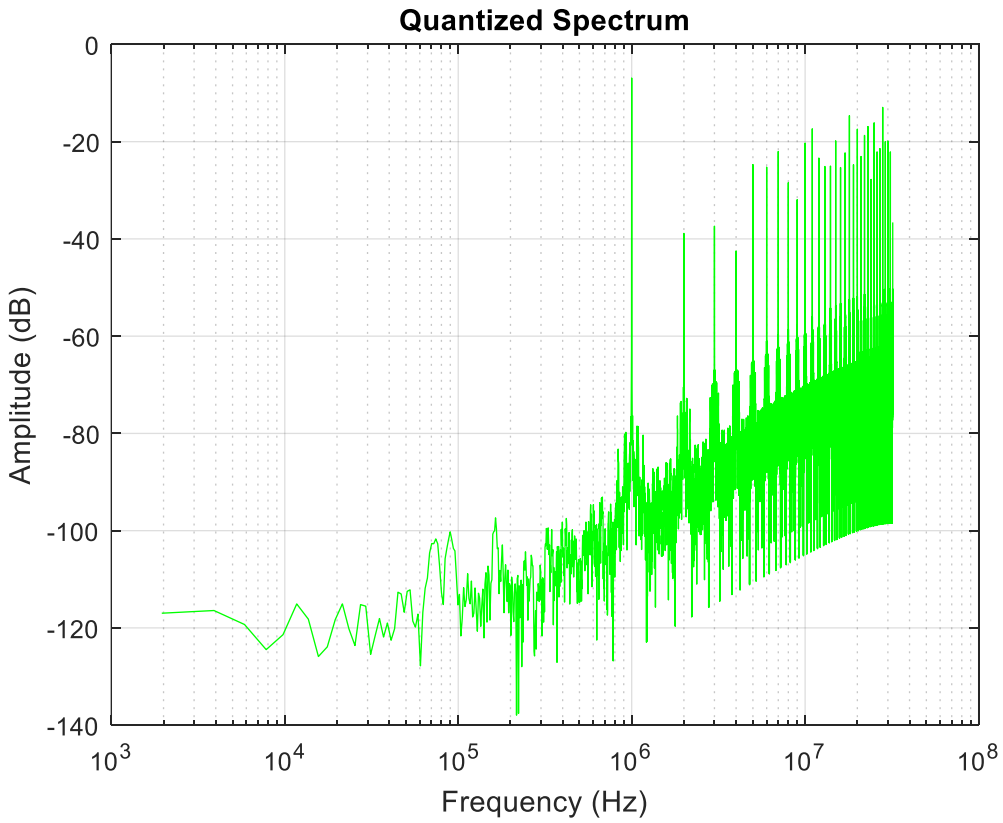

Figura 29: Respuesta del modulador Sigma Delta de primer orden ante una señal de 1 MHz (OSR = 32)

En la figura superior observamos la respuesta que presenta un tono de 1 MHz al ser modulado en el Sigma Delta de primer orden con un OSR de 32. Se observa la presencia del tono y como el ruido asciende a medida que aumenta la frecuencia. Podemos observar además la presencia de tonos adicionales que por su posición en el espectro podemos suponer que son armónicos de la señal muestreada. Vemos además que de igual manera que ocurre con el ruido base la amplitud de los armónicos también aumenta conforme aumentamos la frecuencia. En el caso de la señal muestreada no supone un problema pues se corresponde con el ancho de banda del sistema por

tanto los armónicos no ensuciarían la señal, veamos ahora que ocurre si cambiamos el tono de 1 MHz por uno dos décadas por debajo, es decir de 10 KHz.

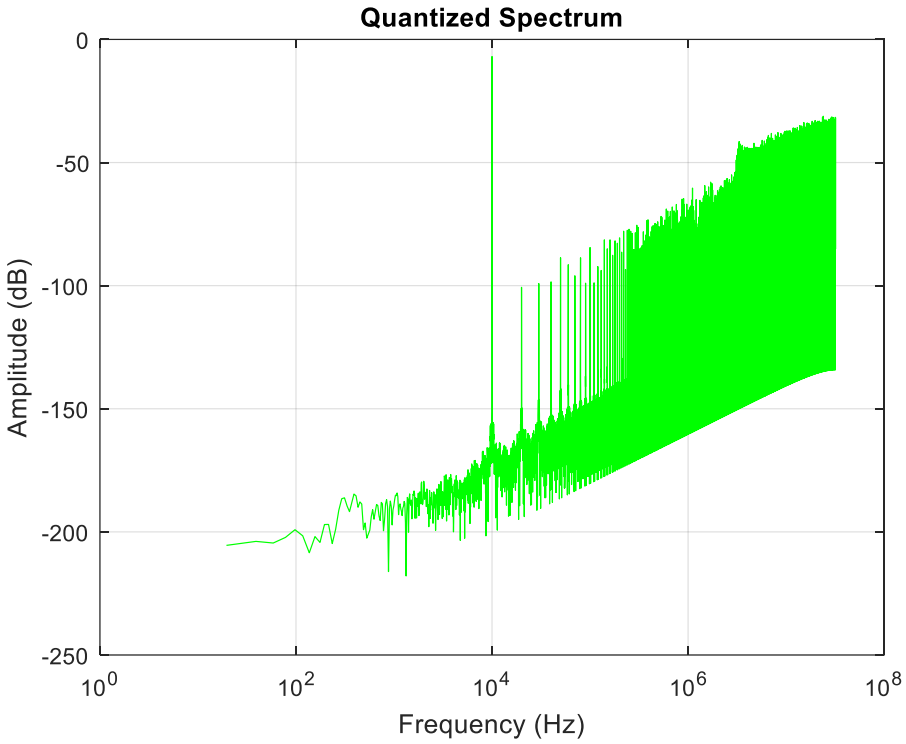

Figura 30: Respuesta del modulador Sigma Delta de primer orden ante una señal de 10 KHz (OSR = 32)

Como era de esperar los armónicos se desplazan con la señal y en este caso aparecen en gran cantidad dentro de nuestro ancho de banda y por tanto no se pueden eliminar con el filtrado digital. Por ello no sería recomendable trabajar con esta configuración y este *oversampling ratio.*

A continuación se muestran las simulaciones con un *oversampling ratio* de 64:

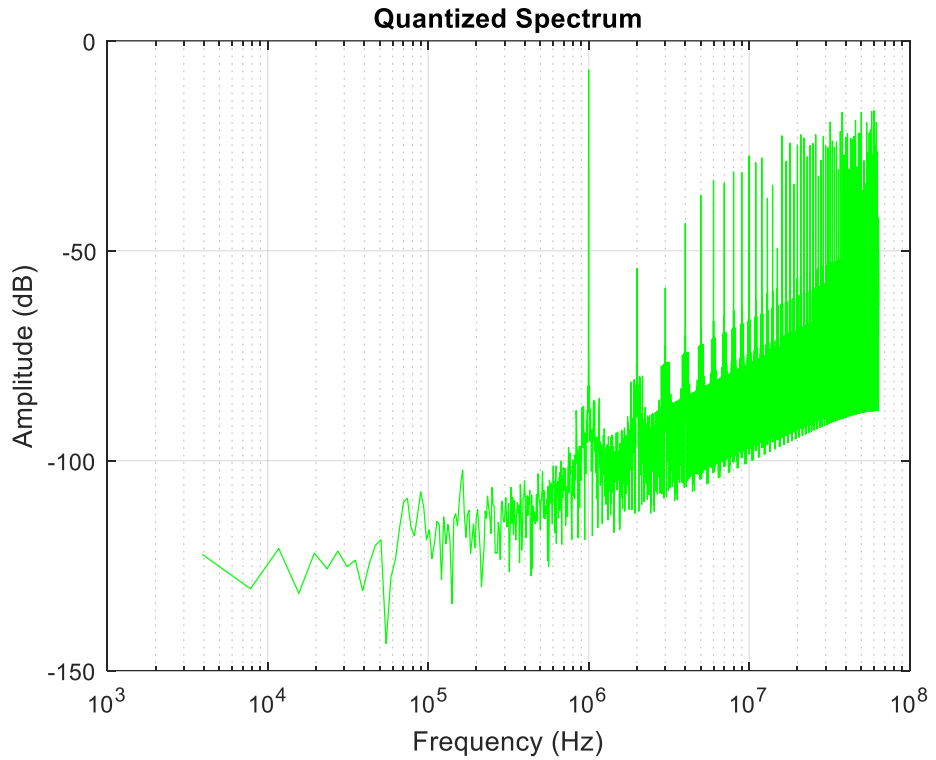

Figura 31: Respuesta del modulador Sigma Delta de primer orden ante una señal de 1 MHz (OSR = 64)

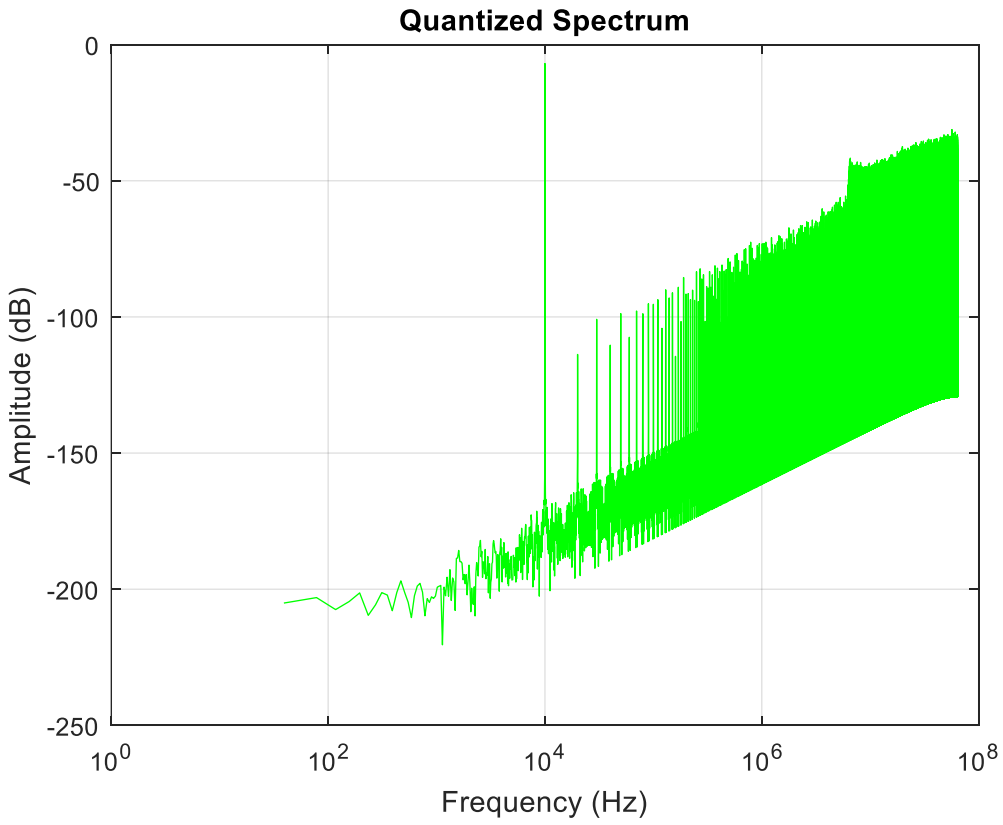

Figura 32: Respuesta del modulador Sigma Delta de primer orden ante una señal de 10 KHz (OSR = 64)

Como se puede observar en las simulaciones, en el caso de la señal a 1 MHz la reducción de ruido es apreciable así como una reducción del nivel de los armónicos, sin embargo al bajar la frecuencia a 10 KHz podemos observar como persiste el problema de los armónicos con una figura de ruido casi idéntica a la que encontrábamos con un OSR de 32.

3.2 Modulador Sigma Delta de segundo orden

La siguiente configuración cuyo funcionamiento se ha verificado es la del modulador Sigma Delta de segundo orden. Como ya sabemos consiste en incorporar un segundo integrador en serie, a continuación se muestra su esquemático en Simulink:

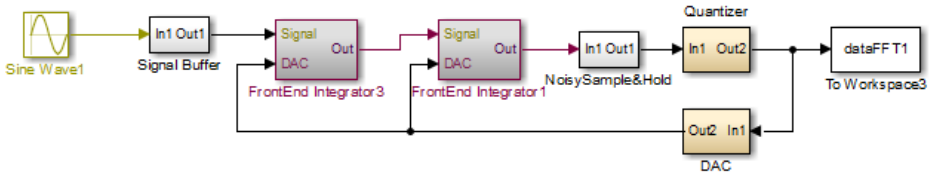

Figura 33: Esquema del modulador Sigma Delta de segundo orden en Simulink

Se han obtenido los siguientes resultados para 1 MHz y 10 KHz con un *oversampling ratio* de 32:

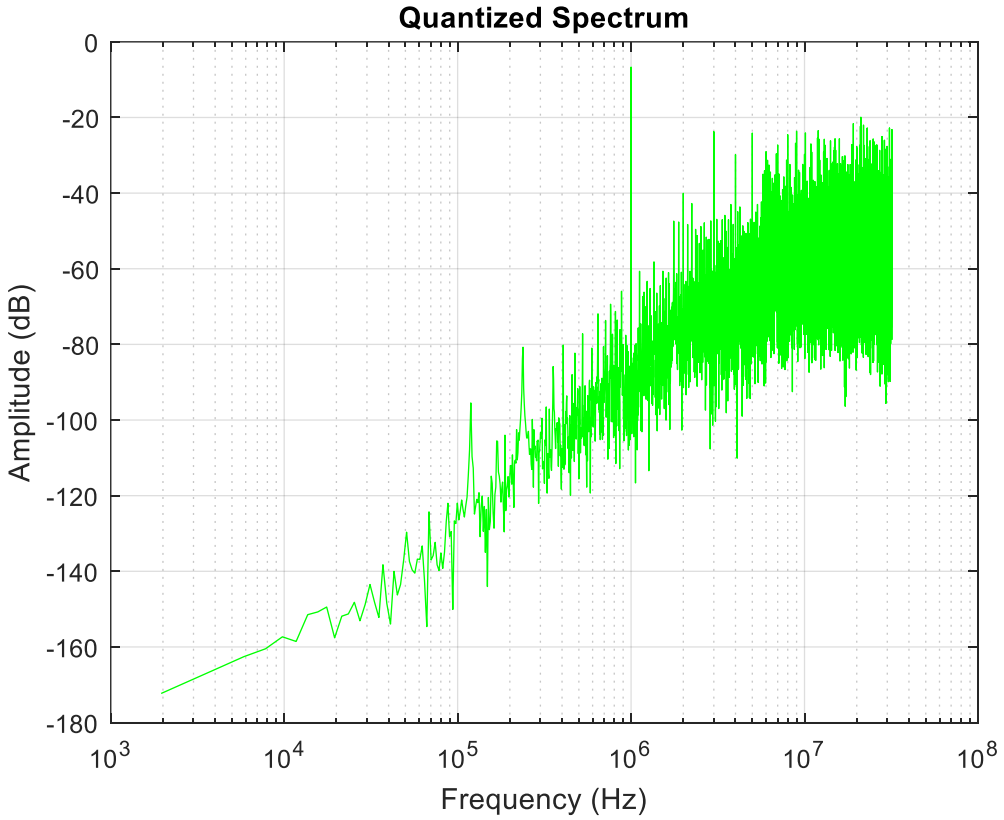

Figura 34: Respuesta del modulador Sigma Delta de segundo orden ante una señal de 1 MHz (OSR = 32)

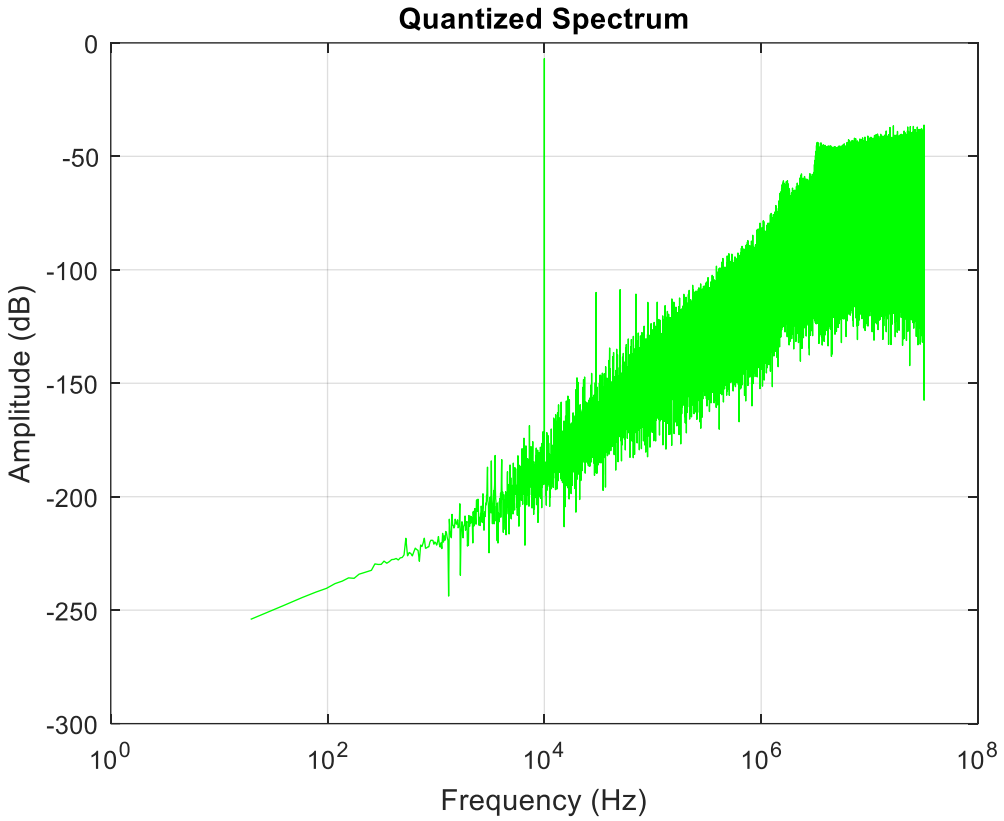

Figura 35: Respuesta del modulador Sigma Delta de segundo orden ante una señal de 10 KHz (OSR = 32)

Como se puede observar en las imágenes se ha conseguido reducir bastante la presencia de los armónicos que encontrábamos en el convertidor de primer orden. Veamos ahora que ocurre al doblar el *oversampling ratio.*

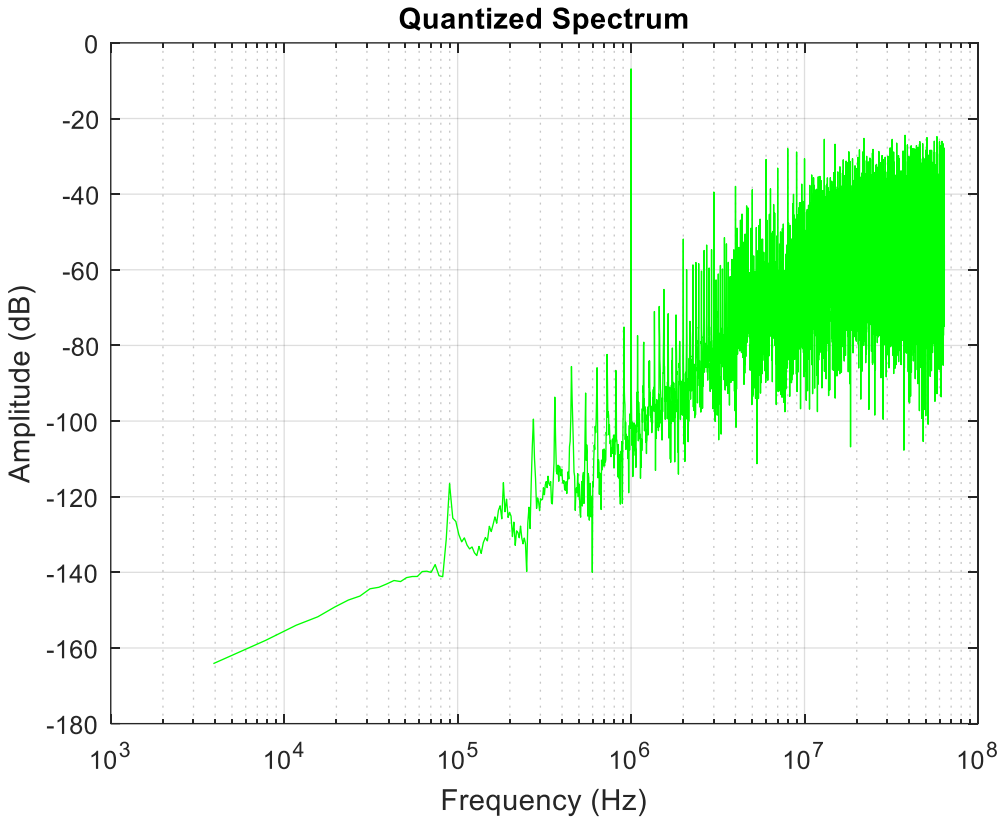

Figura 36: Respuesta del modulador Sigma Delta de segundo orden ante una señal de 1 MHz (OSR = 64)

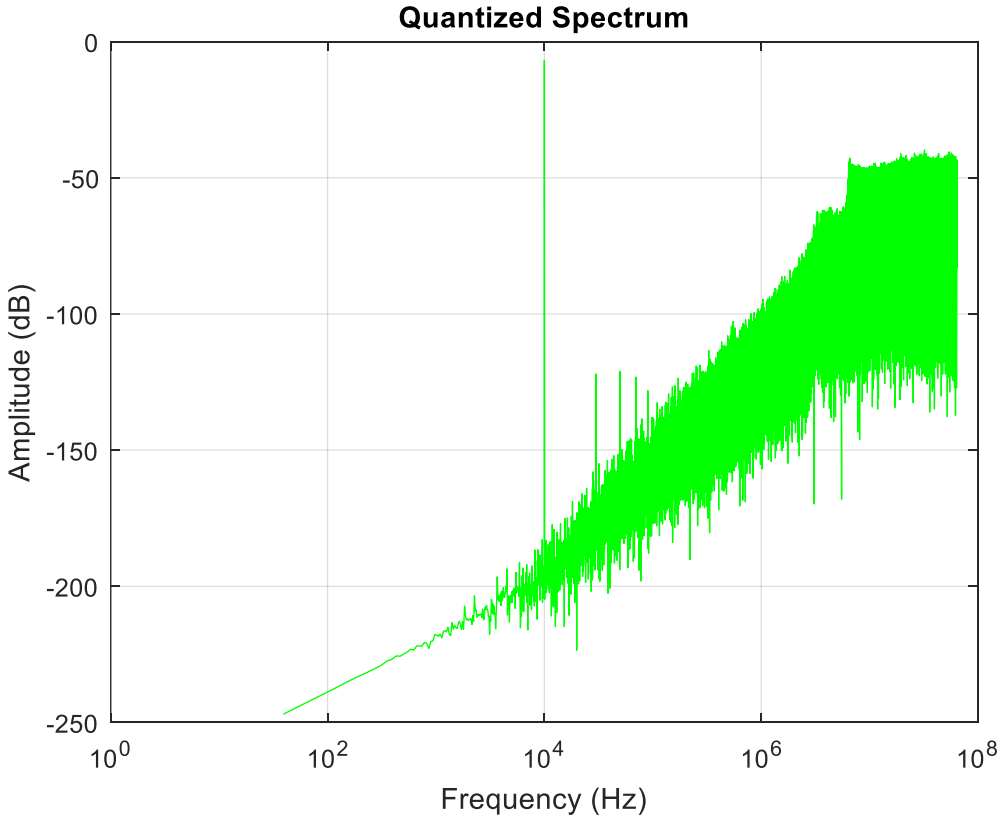

Figura 37: Respuesta del modulador Sigma Delta de segundo orden ante una señal de 10 KHz (OSR = 64)

Como podemos ver, se ha conseguido reducir el ruido bastante en el ancho de banda y los armónicos ya no son un problema, si bien aparecen picos a frecuencias puntuales lo hacen con amplitudes mucho menores. Vemos además que se cumple la caída de 40 dB por década del ruído.

3.3 Modulador Sigma Delta de segundo orden con cuantizador de tres y cuatro niveles.

Los resultados al aumentar la resolución del cuantizador son los siguientes:

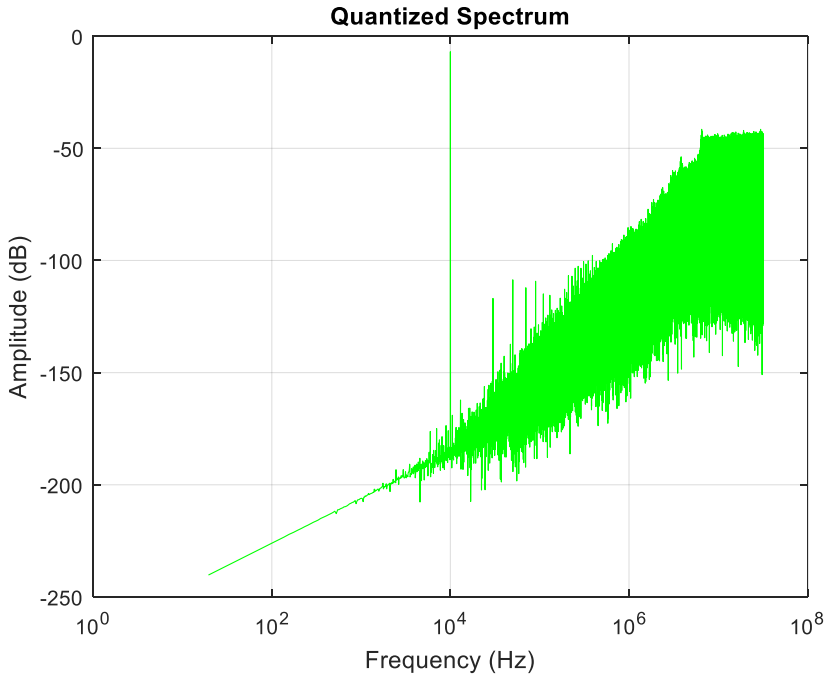

Figura 38: Respuesta del modulador Sigma Delta de segundo orden con 3 niveles ante una señal de 10 KHz (OSR = 32)

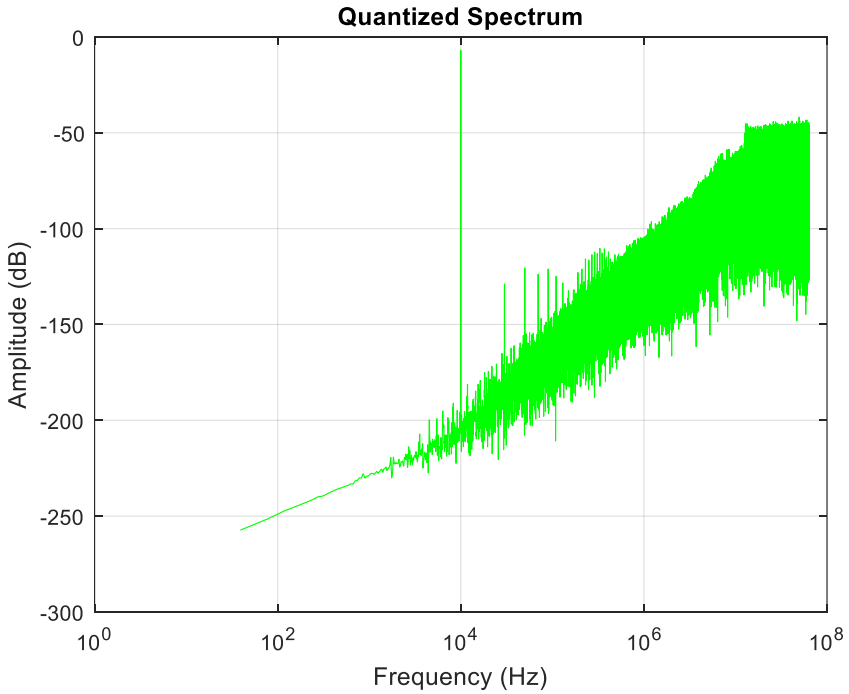

Figura 39: Respuesta del modulador Sigma Delta de segundo orden con 3 niveles ante una señal de 10 KHz (OSR = 64)

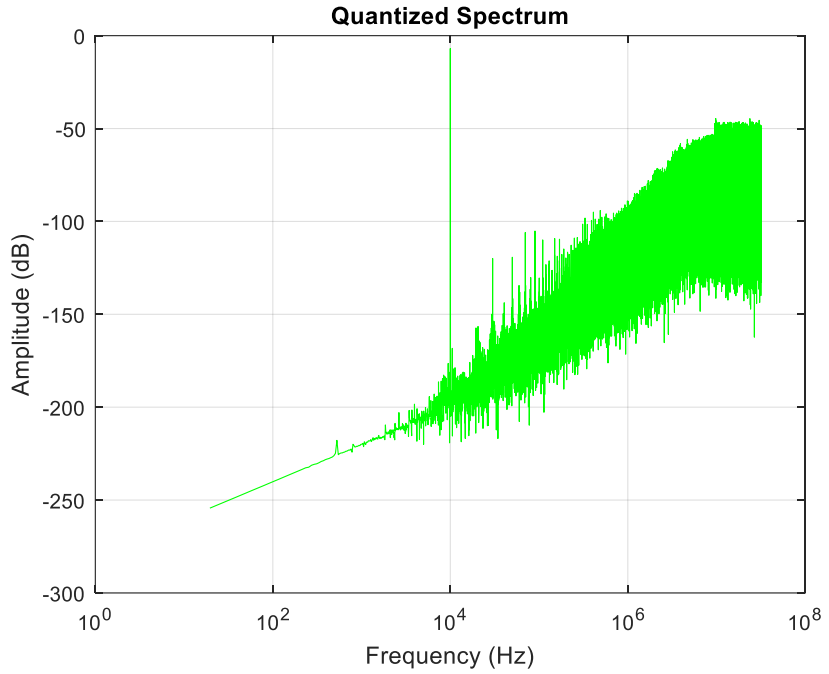

Figura 40: Respuesta del modulador Sigma Delta de segundo orden con 4 niveles ante una señal de 10 KHz (OSR = 32)

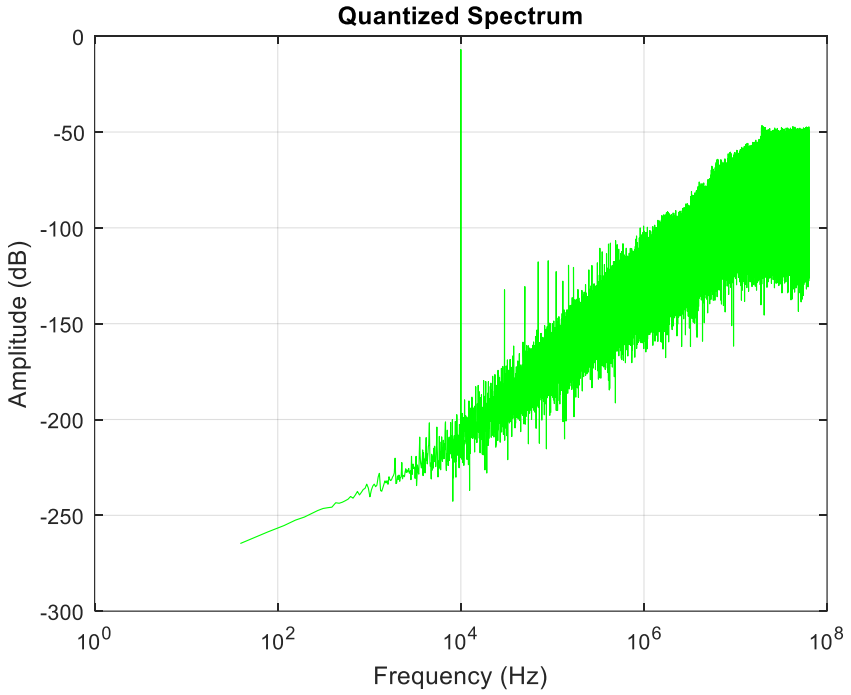

Figura41: Respuesta del modulador Sigma Delta de segundo orden con 3 niveles ante una señal de 10 KHz (OSR = 64)

3.4 Simulaciones en Virtuoso

Una vez comprobado en Matlab el funcionamiento hemos realizado las simulaciones con un *oversampling ratio* de 64 obteniendo los siguientes resultados:

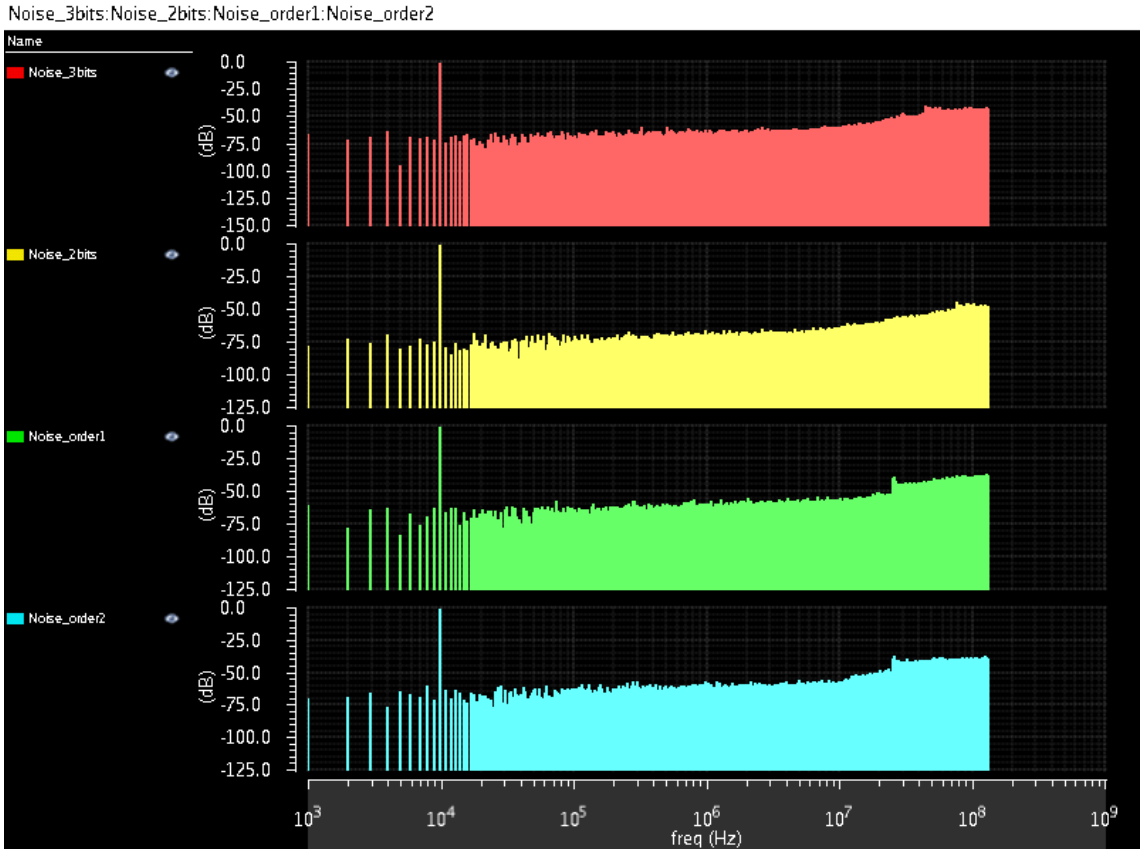

Figura41: Respuesta de las distintas configuraciones en Virtuoso

Estas simulaciones se han realizado finalmente con el ruido activado, y se ha sustituido la configuración de tres niveles de cuantización por una de 8 niveles, es decir 3 bits.

Como podemos ver, los resultados distan bastante de los obtenidos con Matlab, por lo que sería conveniente realizar un ajuste de los modelos, aun así son apreciables las variaciones entre las distintas configuraciones.

Capítulo 4. Conclusiones y propuesta de trabajo futuro

Mediante la realización de este trabajo se ha podido comprobar las posibilidades que permite el uso de lenguajes descriptivos en el diseño de circuitos analógicos. Si bien la filosofía detrás de ellos a la hora de programar es bastante similar a la de lenguajes de diseño digital como Verilog o VHDL las capacidades de descripción que ofrecen son bastante más amplias, al basarse en la relación de variables mediante ecuaciones. Esto nos permite en diseños complejos realizar modelos de todos los bloques para tener un banco de pruebas en el que probar cada bloque al diseñarlo finalmente con componentes reales. Además permite al usuario profundizar tanto como quiera en la descripción de cada componente, dejando a decisión de este la cantidad de efectos de segundo orden que afectan al sistema, pues a más detallada sea la descripción obtendremos un modelo cuyo comportamiento se acerca más a la realidad.

En cuanto a posibles mejoras del trabajo una de ellas de haber dispuesto de mayor cantidad de tiempo hubiese sido profundizar en los moduladores Sigma Delta de orden superior y en la arquitectura MASH.

Bibliografía

[1] “Understanding Delta-Sigma Data Converters” Richard Schreier, Gabor C. Temes.

[2] “CMOS Circuit Design, Layout, and Simulation” Jacob Baker, Harry W. Li, David E. Boyce.

[3] “Verilog-A Language Reference Manual” Open Verilog International.