

Máster en Ingeniería de Sistemas
Electrónicos y Aplicaciones (MISEA)
2023-2024

Trabajo de Fin de Máster

Diseño e implementación de
un comparador de baja
potencia para para redes
neuronales y aplicaciones de
procesamiento distribuido en
tecnología CMOS

Ricardo Carrero Bardón

Rubén Garví Jiménez Ortiz

Agradecimientos

A Rubén Garvi, Javier Granizo, Luis Hernández y Víctor Medina, por haberme dado la posibilidad de trabajar en su grupo de investigación, gracias al que he podido desarrollar este TFM, y en el que he aprendido tanto en los últimos meses.

A Jesús Pérez Amorós por haberme enseñado todo lo necesario en mis primeras semanas aquí, sin el que hubiera estado completamente perdido, y hacer mucho más ameno el aprendizaje.

A mi familia directa y no tan directa, a mi hermana, a mis padres y mis abuelos, mis tíos y todos mis primos, porque tengo el privilegio de tener una gran familia que siempre ha estado y estará para ayudarme y apoyarme en todo en la vida.

A mis amigos cercanos por conseguir siempre sacarme una sonrisa y alegrarme con su simple presencia en cualquier situación.

A Jie Rui Xie, por haberme inspirado a ser mejor persona cada día y ver la vida de otra manera durante este último año, y sobre todo, hacerme recordar lo que es soñar y tener ilusión.

A Diego Morilla y Marcos Carlero, por hacerme siempre pensar a lo grande y de forma profunda sobre cualquier tema. Marcos, se te ha echado de menos en este proyecto, pero estoy seguro de que haremos muchos más en el futuro.

ABSTRACT

Este Trabajo de Fin de Máster (TFM) analiza el diseño, implementación y verificación a todos los niveles (sistema, lógico, transistor, layout) de un circuito para la combinación de las salidas de dos convertidores analógico-digitales basados en osciladores controlados por voltaje (VCO-ADCs), con el objetivo de extender su rango dinámico, para su uso en micrófonos inteligentes. Si bien inicialmente se pretendían utilizar redes neuronales con este fin, de ahí el nombre del proyecto, en última instancia se ha determinado utilizar otra metodología de diseño por considerarse más eficiente.

Se analiza también el diseño a nivel de sistema y lógico de los convertidores completos, explorando diferentes opciones planteadas para el mismo, simulaciones realizadas y justificación de decisiones de diseño tomadas.

Se ha conseguido una extensión efectiva de 6dB SPL en el rango dinámico, con una estimación de consumo de 380 uW en total, y una SNR de 75dB@94dB SPL.

Este TFM se ha realizado en colaboración con el proyecto "Development of Smart Digital Microphones in Nanometer CMOS" de la UC3M y la empresa INFINEON TECHNOLOGIES. El diseño implementado forma parte de un circuito integrado que se encuentra en proceso de fabricación a fecha de la elaboración de este documento, para ser probado en el laboratorio.

ÍNDICE

1 – INTRODUCCIÓN, ESTADO DEL ARTE Y PLANTEAMIENTO DEL PROYECTO	5
1.1 - Que es un VCO-ADC	6
1.1.2 - Como implementar un VCO-ADC.....	10
1.1.2.1 - Como implementar un VCO.....	10
1.1.2.2 - Como implementar un muestreador de fase	13
1.1.2.3 Como implementar un noise-shaper.....	15
1.1.3 – Diferentes clases de osciladores en anillo	16
1.1 - Procesamiento en el Borde de señales de audio	18
2 – DISEÑO DEL SISTEMA IMPLEMENTADO	29
2.1 - Diseño de alto nivel de un sistema de adquisición de audio con extensión de rango y 2 VCO	29
2.2 - Opción 1: optimización del rango dinámico por comparadores implementados con redes neuronales	36
2.3 - Opción 2: Optimización del rango dinámico con estimadores estadísticos de potencia	41
3 – DISEÑO HARDWARE DEL SISTEMA	47
3.1 - Etapa analógica de entrada y osciladores.....	47
3.2 - Contador	48
3.6 - Modelo bit-true completo - simulación del sistema bit-true	61
3.7 - Diseño a nivel de transistor del bloque implementado	64
4 – LAYOUT	65
4.1 Layout del chip completo.....	65
4.3 - Verificación del bloque implementado y estimación de potencias.....	70
5 – PLANIFICACIÓN DEL PROYECTO	75
6 - CONCLUSIONES	77
7 – BIBLIOGRAFÍA Y REFERENCIAS	78
ANEXO	80

1 – INTRODUCCIÓN, ESTADO DEL ARTE Y PLANTEAMIENTO DEL PROYECTO

Este proyecto se incluye en el marco de los VCO-ADCs [1]. Este tipo de convertidores analógico-digitales están empezando a ser utilizados tanto en áreas de investigación como comerciales, por su bajo consumo y buena relación señal a ruido. Estos convertidores están basados, conceptualmente, en los convertidores sigma-delta [2], ampliamente utilizados en la industria desde hace muchos años.

Frente a otros tipos de convertidores, los VCO-ADCs presentan varias ventajas, aunque también algunos inconvenientes. Una ventaja fundamental es que se pueden implementar casi exclusivamente con circuitos digitales, algo que repercute directamente en su bajo consumo y área al ser integrados. Sin embargo, presentan desafíos adicionales en su implementación, tales como el ruido y la distorsión introducidos por los osciladores o la dificultad para extender su rango dinámico, aunque estos problemas pueden ser mitigados con diferentes técnicas, como lazos de realimentación. Por otro lado, no son adecuados para señales de alta frecuencia, por tratarse de convertidores sobre muestrados.

Aunque estos inconvenientes son detrimetnales en ciertas aplicaciones, hay otras en las que no tienen apenas impacto. Tal es el caso del procesamiento de señales de audio, dado que son señales de baja frecuencia. Por ello, tradicionalmente se han utilizado convertidores sigma-delta basados en condensadores comutados para realizar esta tarea. Sin embargo, recientemente se está popularizando el uso de VCO-ADCs, que se presentan como un competidor más fácilmente integrable en tecnologías de fabricación modernas de menos nanómetros.

A lo largo de este Trabajo de Fin de Máster se ha estudiado y contribuido al diseño conceptual y a nivel de sistema de un VCO-ADC de muy bajo consumo basado en múltiples osciladores, utilizado para la digitalización de señales de audio procedentes de un micrófono MEMS, que trata de abordar el problema de la extensión del rango dinámico [4] para su uso en micrófonos inteligentes. También se ha realizado un proceso completo de diseño de uno de los bloques que lo

componen. Dicho bloque, concretamente, se utiliza para combinar las salidas de los múltiples osciladores involucrados en el convertidor. Además, se ha realizado un proceso de verificación del mismo mediante simulaciones a nivel de transistor y layout.

Este documento se divide en diferentes secciones, que reflejan el trabajo realizado. Comienza con una explicación genérica del funcionamiento de los VCO-ADCs. Continúa con el diseño a nivel de sistema del convertidor junto con los modelos, cálculos y simulaciones llevados a cabo para su optimización. Además, se exploran diferentes alternativas para implementar la combinación de las salidas de los múltiples osciladores utilizados, una de ellas utilizando redes neuronales. Después, se analizan los diferentes circuitos digitales propuestos para la implementación del modelo anterior, desarrollando un modelo a nivel de puerta lógica del sistema completo para ello. A continuación, se analiza en profundidad la implementación y verificación del bloque de combinación de salidas, objeto principal de este TFM, hasta conseguir un layout en silicio de este componente. Se analiza también mediante simulaciones el consumo del bloque, incluyendo una comparación con el resto del sistema. Por último, se analizan las conclusiones y trabajo futuro.

1.1 - Que es un VCO-ADC

Un VCO-ADC es un convertidor analógico digital basado en un oscilador controlado por voltaje. De manera intuitiva, puede entenderse que un cambio de voltaje en una señal de entrada produce una variación de frecuencia en el oscilador controlado por voltaje (VCO). El VCO se encuentra conectado a un circuito electrónico con la capacidad de transformar una frecuencia de entrada en un número binario a la salida (Frequency to Digital). De esta forma, variaciones en el voltaje de entrada producen variaciones en la frecuencia del oscilador, que producen variaciones digitales a la salida, por lo que se traslada el problema de la conversión analógico-digital de voltaje-digital a frecuencia-digital. Si bien puede parecer que esto no es una tarea sencilla, y no lo es desde un punto de vista

matemático, los circuitos electrónicos resultantes de la base teórica de los VCO-ADCs no son complejos de implementar y son relativamente pequeños en área. Además, pueden implementarse casi completamente con circuitos digitales, algo que facilita enormemente la integración de estos sistemas. Esta es, precisamente, la virtud de los VCO-ADCs. A continuación, se explica el funcionamiento de un VCO-ADC básico a nivel teórico, que tiene una descripción de sistema como la siguiente:

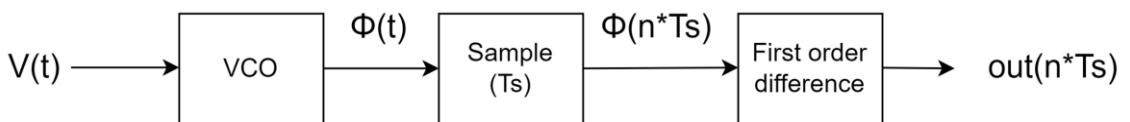


Figura 1: Sistema VCO-ADC

La ecuación fundamental de un VCO lineal con frecuencia en reposo igual a 0 es la siguiente

$$F(t) = K \cdot V(t) \quad (1)$$

donde K es la ganancia del oscilador en Hz / V

Integrando ambas partes de la ecuación obtenemos

$$\int_0^t F(t) dt = \int_0^t K \cdot V(t) dt$$

Como la frecuencia es la derivada de la fase con respecto al tiempo, podemos reescribir la ecuación anterior como

$$\Phi(t) = \int_0^t K \cdot V(t) dt \quad (2)$$

Es interesante convertir la ecuación (1) en (2) debido a que realizar una lectura digital de la frecuencia instantánea de un oscilador es más complicado que realizar una de lectura de su fase. Además, muestrear la fase en lugar de la frecuencia tiene una ventaja muy importante que se desarrolla más adelante.

Suponiendo que existe un dispositivo capaz de muestrear la fase del oscilador de forma digital, la salida de este, si se define un periodo de muestro T_s es:

$$\Phi(n \cdot T_s) = \int_0^{n \cdot T_s} K \cdot V(n \cdot T_s) dt + E(n \cdot T_s) \quad (3)$$

Se ha añadido una función $E(t)$, que representa el error cometido por muestrear digitalmente la fase, al no disponer de infinitos niveles de precisión. Además de por corrección matemática, es útil añadir este término para analizar que ocurre con el error en la siguiente fase.

Si realizamos la primera diferencia, es decir,

$$\Phi(n \cdot T_s) - \Phi((n - 1) \cdot T_s)$$

obtenemos

$$\begin{aligned} \Delta \Phi(n \cdot T_s) &= \int_0^{nT_s} K \cdot V(n \cdot T_s) dt \\ &\quad - \int_0^{(n-1)T_s} K \cdot V((n-1) \cdot T_s) dt + E(n \cdot T_s) - E((n-1) \cdot T_s) \end{aligned} \quad (4)$$

Tras realizar la primera diferencia, $\Delta\Phi$ representa la variación de la fase del oscilador entre dos muestras consecutivas.

Además, si T_s es lo suficientemente menor que el periodo de la señal muestreada, algo que puede hacerse debido al carácter sobremuestreado de los VCO-ADCs [1], podemos realizar una aproximación lineal de la diferencia de integrales:

$$\Delta \Phi(n \cdot T_s) = K \cdot V(n \cdot T_s) \cdot T_s + E(n \cdot T_s) - E((n - 1) \cdot T_s) \quad (5)$$

Esta es, precisamente, la salida digital del sistema

$$Out(n \cdot T_s) = K \cdot T_s \cdot V(n \cdot T_s) + E(n \cdot T_s) - E((n - 1) \cdot T_s) \quad (6)$$

El error de salida es la diferencia de los dos últimos errores, mientras que, por la propia naturaleza del sistema, la primera diferencia es necesaria para recuperar la señal de entrada digitalizada. La salida del sistema es equivalente a un convertidor sigma-delta de primer orden. Como se ve en las siguientes

ecuaciones, el hecho de que el error de salida sea la diferencia entre los dos últimos errores equivale a reducir el ruido de cuantización de baja frecuencia.

$$\begin{aligned}
 Z(E(n \cdot Ts) - E((n-1) \cdot Ts)) = \\
 E(z) - z^{-1} + E(z) = \\
 E(z) \cdot (1 - z^{-1})
 \end{aligned} \tag{7}$$

Vista como función de transferencia en el plano Z, la ecuación (7), tiene la siguiente respuesta:

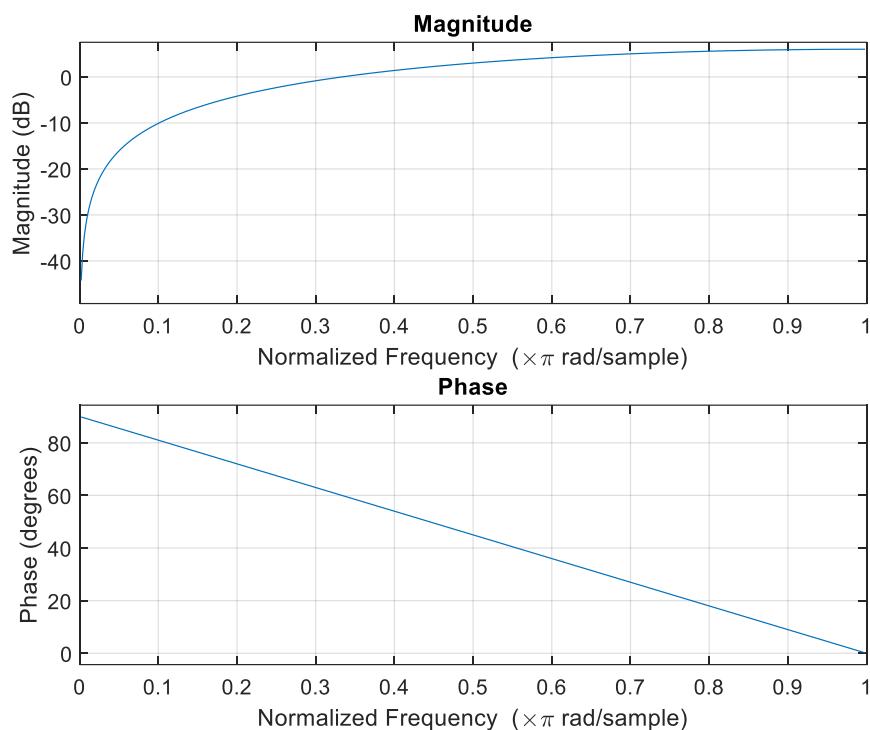


Figura 2: Respuesta en frecuencia de la primera diferencia

Esta respuesta puede ser interpretada como un filtro de paso alto con respuesta lineal en fase.

Por otro lado, el valor digital de salida depende de K y Ts. Fundamentalmente, un sistema equivalente al descrito matemáticamente en este apartado obtiene un valor digital de salida proporcional al voltaje de entrada, y realiza noise-shaping de primer orden del error de cuantización. La información aportada en esta sección

puede complementarse con la lectura del artículo *Time-Encoding Analog-to-Digital Converters: Bridging the Analog Gap to Advanced Digital CMOS-Part 1: Basic Principles* [1].

1.1.2 - Como implementar un VCO-ADC

Como se ha visto, un VCO-ADC consta de tres partes fundamentales: Un VCO, un muestreador de la fase y la realización de la primera diferencia.

1.1.2.1 - Como implementar un VCO

Un VCO puede implementarse de muchas maneras. Para este trabajo, se analiza una implementación con un oscilador en anillo, por ser fácilmente integrable y tener un bajo consumo y área. Sin embargo, esta elección no está exenta de dificultades, pues un oscilador en anillo es sistema altamente no lineal que se ve alterado fácilmente por las condiciones externas.

El oscilador en anillo básico consiste en un número impar de inversores (puertas NOT) conectadas en bucle, como en la siguiente imagen:

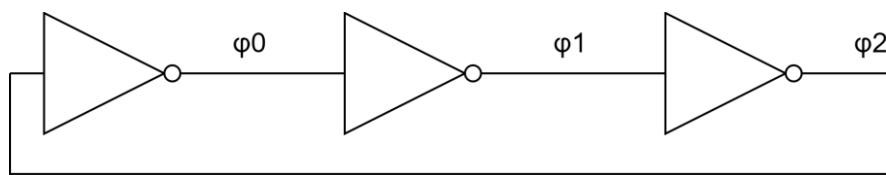


Figura 3: Oscilador en anillo básico

Este sistema oscila, ya que no es posible encontrar una configuración estable para el mismo. El periodo total de oscilación de un oscilador en anillo puede calcularse a partir del tiempo de respuesta de los inversores que lo componen. Si un inversor tiene un tiempo de respuesta T_i , y un oscilador en anillo está compuesto por 3 inversores, su periodo de oscilación será $2 \cdot 3 \cdot T_i$. La siguiente tabla muestra el estado del oscilador en tiempo discreto a lo largo de un periodo

de oscilación, donde ϕ_0 , ϕ_1 y ϕ_2 representan el estado lógico de los tres enlaces del oscilador de la figura 3:

n	ϕ_0	ϕ_1	ϕ_2
0	0	1	0
1	1	1	0
2	1	0	0
3	1	0	1
4	0	0	1
5	0	1	1
6	0	1	0

Figura 4: Estados de un oscilador en anillo de 3 inversores

Donde en rojo se muestra la fase que se encuentra en un estado inestable. Puede observarse como, efectivamente, son necesarios 6 periodos (6 tiempos de reacción de un inversor individual) para que el oscilador vuelva a su estado inicial. De forma genérica, podemos establecer que el periodo del oscilador completo es $2 \cdot N_{\text{inv}} \cdot T_i$, donde N_{inv} es el número de inversores que lo componen y T_i es el tiempo de reacción de un inversor. Por supuesto, para que se pueda aplicar esta ecuación, todos los inversores deben tener características idénticas o muy similares.

Dado que N_{inv} es un parámetro fijo de la arquitectura del oscilador, para variar la frecuencia de un oscilador en anillo, es necesario reducir o aumentar T_i . Para ello, se debe aumentar o reducir la corriente de alimentación de los inversores. Una mayor corriente implica un menor tiempo de respuesta, por tanto, una mayor frecuencia de oscilación y viceversa. No obstante, recordemos que un VCO es un circuito controlado por voltaje y no por corriente, como su propio nombre indica.

Para construir un VCO a partir de un oscilador en anillo, basta con añadir un transductor a la entrada del oscilador, que transforma un voltaje de control en una corriente de control.

Una característica fundamental del VCO resultante para que pueda ser utilizado en un VCO-ADC es que cumpla la ecuación de funcionamiento (1), que básicamente indica un comportamiento lineal. Ciertamente, el resultado de

duplicar la corriente de control de un oscilador en anillo no resulta en un aumento del doble de su frecuencia. Sin embargo, es posible establecer un régimen lineal de operación entre ciertos valores. Cuando se superan estos valores, el oscilador en anillo entra en una zona no lineal, lo que provoca un mal funcionamiento del VCO-ADC. Por tanto, esta zona de operación determina el rango dinámico del convertidor. Un oscilador en anillo tiene una función de transferencia con una forma similar a la siguiente:

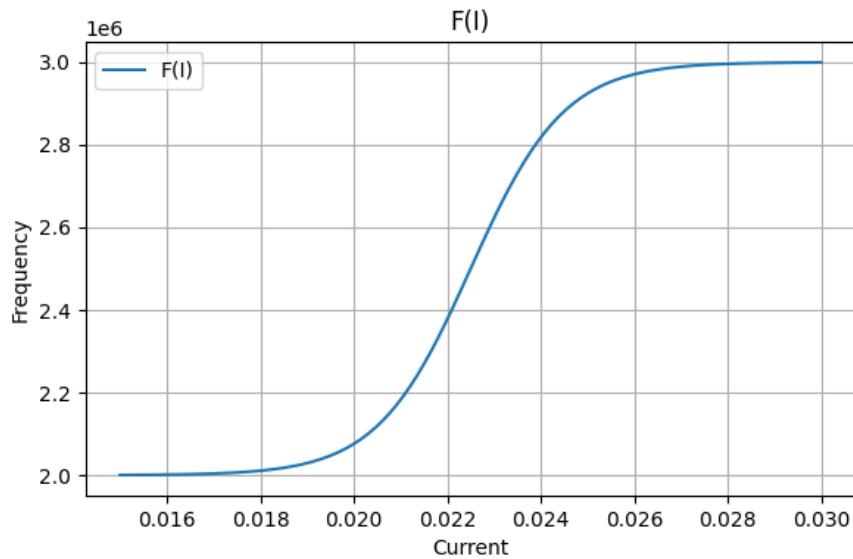


Figura 5: Forma de una función de transferencia de un oscilador en anillo

El oscilador tiene una frecuencia en reposo dependiente de una corriente de polarización fija. Esta corriente es necesaria, ya que llega un momento en el que, si no se proporciona suficiente corriente, el oscilador en anillo deja de oscilar. Alrededor del punto de polarización, se observa un régimen de operación lineal, y una saturación progresiva hacia los extremos.

Puede surgir la duda de por qué en la ecuación (1) se establece una frecuencia en reposo de 0, si una frecuencia en reposo con valor 0 Hz es imposible de conseguir con un oscilador en anillo. La respuesta es que es posible eliminar la frecuencia de reposo, y obtener frecuencias de oscilación “negativas” con una configuración diferencial utilizando dos osciladores, que es precisamente la que se utiliza en

este proyecto. En cualquier caso, si a (1) se le añadiese una frecuencia de reposo, esta se eliminaría en la primera diferencia, por lo que no tiene ningún efecto en el resultado de estas ecuaciones. Descripciones más precisas de modelos matemáticos para osciladores en anillo, así como su implementación, se pueden encontrar en [6],[7] y [8]

1.1.2.2 - Como implementar un muestreador de fase

Gracias a que el estado de un oscilador en anillo puede describirse digitalmente, la fase del oscilador es sencilla de muestrear. De hecho, en la figura 4 (estados de un oscilador en anillo formado por tres inversores) se han numerado los estados de un oscilador en anillo de forma ascendente. De forma efectiva, cada uno de estos números representa, de forma discreta, la fase del oscilador. Realmente, la operación realizada al asignar cada número a un estado en el caso de la figura es:

$$R(t) = \text{floor}(\Phi(t) \cdot (6 / (2\pi))) \bmod 6 \quad (8)$$

Ya que cada vuelta del oscilador, es decir, un aumento de 2π en la fase, se transforma en un aumento de 6 unidades, y se retorna de forma cíclica al estado 0.

$R(t)$ puede realizarse en hardware con un contador que aumente su valor cada vez que cambia el estado del oscilador en anillo. Este contador puede ser un contador binario a doble flanco, conectando su reloj a la salida de la operación XOR de todas las fases del oscilador. De hecho, es posible utilizar un contador de N bits y un oscilador de M fases, tal que su salida sea

$$R(t) = \text{floor}(\Phi(t) \cdot (M / 2\pi)) \bmod 2^N \quad (9)$$

Es conveniente utilizar un contador cuyo módulo sea una potencia de dos, porque es más fácil de implementar y además se puede extender fácilmente, como se verá más adelante.

$R(t)$ es una señal digital, y por tanto fácilmente muestreable mediante un simple registro con un reloj a la frecuencia F_s . Un ejemplo de este sistema, con un contador de 3 bits a doble flanco, se muestra en la siguiente figura:

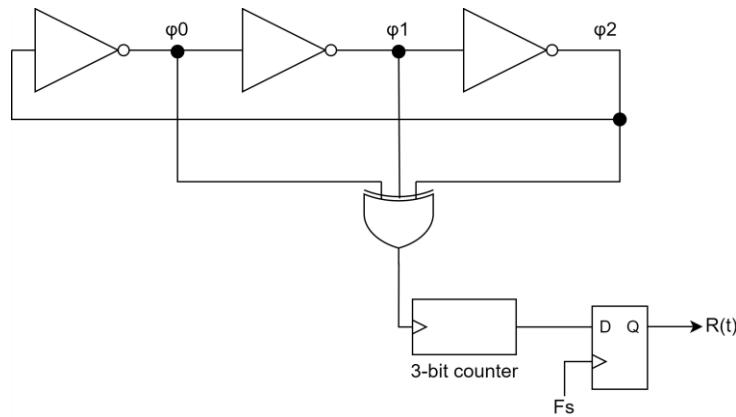


Figura 6: Muestreo de la fase mediante contador

Podría parecer un problema que $R(t)$ se dé la vuelta cada 2^N estados, ya que se debe muestrear la fase del oscilador, como se indica en (9), y la fase crece indefinidamente. Sin embargo, no es tal el caso, ya que lo realmente interesante es la diferencia de fase $\Delta \Phi$ (Eq 5), por lo que siempre que el muestreo de $R(t)$ se realice de forma suficientemente rápida para que el contador no haya dado una vuelta completa entre dos muestras, se obtendrá un resultado $\Delta \Phi$ correcto.

Matemáticamente, esto puede analizarse de la siguiente manera:

La salida del contador es del tipo

$$R2(n \cdot Ti) = n \bmod 2^N$$

donde Ti es el tiempo que tarda en aumentar en una unidad el contador (que es igual al tiempo de respuesta de un inversor del oscilador).

Así,

$$R2(n_2 \cdot Ti) = n_2 \bmod 2^N$$

$$R2(n_1 \cdot Ti) = n_1 \bmod 2^N$$

Por lo que

$$R2(n_2 \cdot Ti) - R2(n_1 \cdot Ti) = (n_2 - n_1) \bmod 2^N \quad (10)$$

Mientras que el valor absoluto de $(n_2 - n_1)$ sea menor que 2^N , el resultado de la primera diferencia será correcto. Este es el límite absoluto para la frecuencia de muestreo, teniendo en cuenta la frecuencia máxima del oscilador. De manera intuitiva, esto indica que el contador no puede realizar más de un periodo completo entre dos muestras, una información que sería imposible de recuperar.

1.1.2.3 Como implementar un noise-shaper

Como se ha visto en las ecuaciones 1-7, con simplemente utilizar la primera diferencia, ya se consigue noise-shaping de primer orden. Es por esto por lo que se considera que los VCO-ADCs tienen noise-shaping de primer orden de manera "inherente" (ver [1]).

Sin embargo, para lograr una buena SQNR en el convertidor, es necesario utilizar una frecuencia de muestreo muy alta, lo que puede resultar impráctico para el tratamiento de la señal de salida. En la industria, el estándar de transmisión de convertidores AD para audio es 3.072 MHz [9]. En el caso de un VCO-ADC, esto supone un problema, ya que la tasa de muestreo puede situarse en decenas o centenas de MHz. Por tanto, es necesario realizar un diezmado, lo que reduciría enormemente la SNR del convertidor si no se realizase de manera adecuada. Una forma de reducir este problema es utilizar un modulador sigma-delta digital [2] en el proceso de diezmado. Este modulador añade noise-shaping al error introducido por el remuestreo:

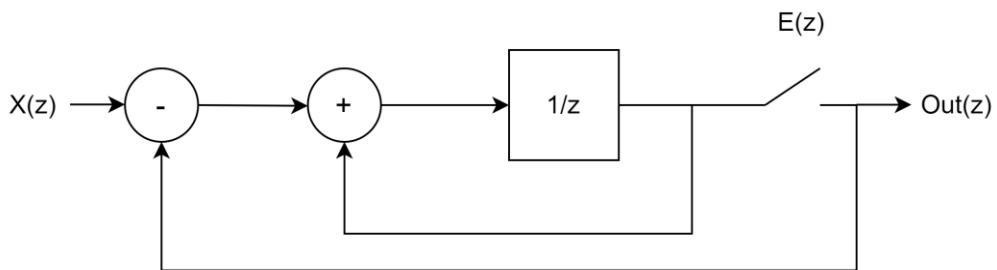


Figura 7: Modulador sigma-delta digital

De forma similar a como se ha hecho en (3), puede interpretarse el remuestreo como un error que se introduce en la señal para cada n , tal que, si no hubiera un modulador entre $Out(n)$ y $X(n)$, entonces

$$Out(n) = X(n) + E(n)$$

Si analizamos el diagrama de bloques del modulador de la figura:

$$\begin{aligned} Out(z) &= (X(z) - Out(z)) \cdot (z^{-1} / (1 - z^{-1})) + E(z) \Rightarrow \\ Out(z) \cdot (1 + (z^{-1} / (1 - z^{-1}))) &= X(z) \cdot (z^{-1} / (1 - z^{-1})) + E(z) \Rightarrow \\ Out(z) &= X(z) \cdot z^{-1} + E(z) \cdot (1 - z^{-1}) \end{aligned} \quad (11)$$

Como se ve, se ha conseguido el mismo resultado que en las ecuaciones 6 - 7 : Mantener la señal de entrada intacta y realizar noise-shaping del ruido introducido.

Este modulador es directamente implementable en hardware digital utilizando un registro en el lugar del bloque z^{-1}

Con los bloques descritos en esta sección, es posible implementar un VCO-ADC con el estado del arte actual. Sin embargo, como se verá más adelante, en este proyecto se utilizan ligeras variaciones de las implementaciones descritas anteriormente.

1.1.3 – Diferentes clases de osciladores en anillo

Existen varios tipos de oscilador en anillo, siendo el más simple de todos el mostrado en la figura 3 (oscilador en anillo no diferencial). Este presenta la ventaja de ocupar un área reducida, pero no puede tener un número de fases par por la naturaleza de su propio diseño, algo que resulta muy inconveniente para el diseño de este VCO-ADC en concreto por la forma en que se implementa el contador para muestrear la fase. Por otro lado, encontramos los osciladores en anillo de tipo “Direct Cross Coupled” [26] (CC) y de tipo “Feed Forward Cross Coupled” (FFCC). Estos osciladores son ambos diferenciales, pero presentan estructuras diferentes.

Por diferencial se entiende, en este contexto, que cada celda del oscilador proporciona dos señales de salida, tal que una es la inversa lógica de la otra, ocurriendo lo mismo con la entrada. A continuación, se muestra una figura con las dos estructuras, extraída de [26]:

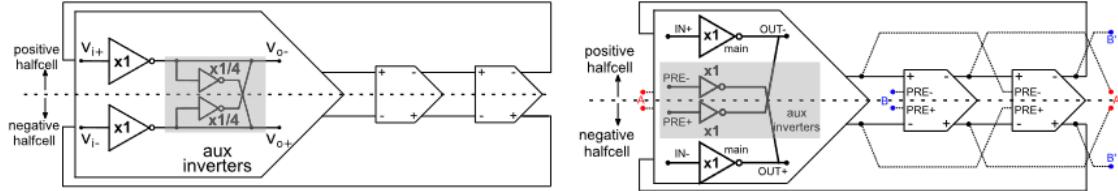


Figura 8: Osciladores en anillo diferenciales (Fuente: The Analog Behavior of Pseudo-Ring oscillators used in VCO-ADCs)

El principio de funcionamiento, intuitivamente, consiste en dos osciladores en anillo simples en los que cada fase está conectada con su fase pareja en el otro oscilador mediante un inversor, de tal forma que ambos osciladores deben mantenerse sincronizados y con señales inversas. Esto es muy fácilmente observable en el oscilador CC (izquierda). En el oscilador FFCC, un principio de funcionamiento similar se utiliza, salvo que, en este caso, cada fase se conecta con la siguiente en lugar de con su pareja (de ahí el nombre “feed forward”). Para diseñar un oscilador con un número de fases par, basta con invertir una de las conexiones entre celdas, conectando la parte positiva con la entrada positiva de la siguiente celda, y la negativa con la negativa, permitiendo ambas arquitecturas esta posibilidad.

El oscilador de la izquierda (CC) tiene la ventaja de ocupar un área más reducida que el de la derecha (FFCC), además de un layout analógico más sencillo de realizar. Sin embargo, el FFCC presenta ventajas en cuanto a ganancia.

1.1 - Procesamiento en el Borde de señales de audio

El procesamiento en el borde (Edge-computing [10]) para señales de audio es un campo de mucha importancia y en el que se ha realizado mucha investigación ([11],[12] y [13] son algunos ejemplos), ya que el procesamiento de audio es una tarea muy relevante en el panorama tecnológico actual. Esto se debe al surgimiento de la IA de procesamiento de voz ([14] entre otros), las interfaces por voz de altavoces inteligentes, y en general de sistemas de interacción humano-máquina mediante voz [15], detección de eventos y presencia mediante sonido, y otros cientos de aplicaciones punteras. Lo que la mayoría de estas aplicaciones tienen en común es que el procesamiento de señal debe realizarse con poca latencia y consumo. Dado que los centros de datos consumen muchos recursos, y el procesamiento de señales en la nube es lento, es muy deseable encontrar maneras, ya sea mediante algoritmos eficientes o hardware específico, de resolver los problemas de procesamiento de señal en el propio dispositivo, mejorando la eficiencia energética y reduciendo la complejidad de los sistemas. Este proyecto se enmarca en el procesamiento en el borde de señales de audio, dado que su objetivo final, como se verá más adelante, es lograr la combinación de dos VCO-ADCs mediante el procesamiento en el borde de sus salidas.

1.2 - Parámetros de audio de un micrófono inteligente

Como este proyecto se enmarca en los VCO-ADCs para señales de audio, es útil para la lectura de la memoria conocer los diferentes parámetros que se utilizan a la hora de medir el rendimiento de micrófonos inteligentes, es decir, micrófonos que tienen convertidores analógico-digitales y/o otras funcionalidades incorporadas. Estos parámetros se han obtenido del artículo [16]. Los más importantes, que además se utilizan en este proyecto, son:

SPL – Sound Pressure Level

Indica la potencia del sonido que entra al micrófono. Se suele utilizar como referencia para otras medidas, por ejemplo, se puede expresar la relación señal a

ruido de un micrófono inteligente para 100db SPL. Se suele indicar al final de una medida con el símbolo @. (por ejemplo, puede indicarse que un micrófono inteligente tiene una SNR de 100dBs @100db SPL)

SNDR – Signal to Noise and Distortion Ratio

Indica la relación señal a ruido del micrófono con respecto al ruido de térmico, de flicker y distorsión. Se suele expresar en función del SPL en dBs. Este ruido proviene tanto de la electrónica de procesamiento de señal como del propio sensor MEMS que incorpora el micrófono inteligente. Normalmente, es el sensor MEMS el mayor causante de ruido, y no tiene mucho sentido optimizar la electrónica de procesamiento más allá del límite del sensor MEMS, lo que permite optimizar su consumo.

SQNR – Signal to Quantization Noise Ratio

Este parámetro indica la relación señal a ruido de la señal con respecto al error producido por la cuantización. Es el parámetro más importante para este proyecto, debido a que se centra en la parte de muestreo y procesamiento digital tras los osciladores en el VCO-ADC implementado. También se expresa en función del SPL.

Rango dinámico

El rango dinámico de un micrófono inteligente viene definido por el rango de valores SPL en los que la señal de salida corresponde adecuadamente a la señal de entrada. Viene determinado por la SNDR y la SQNR, ya que en conjunto marcan la SNR del convertidor. Dicho de otra manera, el rango dinámico son los valores entre los que el micrófono inteligente tiene una buena relación señal a ruido. Normalmente, un buen diseño del convertidor implica que la SQNR es lo suficientemente buena como para que la SNDR sea dominante al determinar el rango dinámico.

Un ejemplo de especificaciones de un micrófono inteligente es el siguiente:

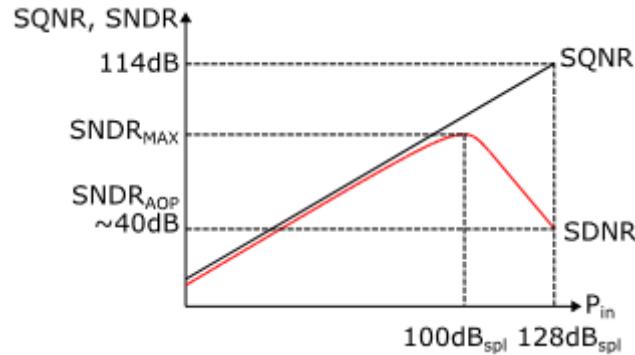


Figura 9: Ejemplo de especificaciones de un micrófono inteligente, Fuente: [16]

Como se ve, la SQNR se mantiene por encima de la SNDR a lo largo de todo el rango dinámico. SNDR-AOP indica la SNDR para el “Acoustic overloading point” [17]. Este valor se define en función de la distorsión armónica producida por el sensor MEMS, concretamente, cuando la distorsión armónica de la señal de salida supone un 10% de la potencia de señal. Para este valor, el micrófono inteligente otorga una SNDR mucho menor, pero la SQNR se mantiene.

Consumo

A pesar de ser un parámetro común en la mayoría de productos electrónicos, el consumo tiene una importancia fundamental en el caso de los micrófonos inteligentes, y por eso se le dedica este apartado. Esto es debido a que, en muchos casos, estos micrófonos se utilizan para tareas tales como el reconocimiento de comandos de voz, por lo que tienen que estar siempre encendidos, suponiendo un consumo estático relevante para los sistemas que los utilizan. En la actualidad, se han conseguido micrófonos inteligentes tales como [16], que apenas tienen un consumo de alrededor de los cientos de uW. La reducción del consumo es una parte muy importante de la investigación alrededor de estos micrófonos inteligentes.

1.3 Planteamiento del proyecto: diseño de un micrófono con extensión de rango dinámico mediante procesamiento en el borde

Como se ha expuesto en el apartado anterior, tanto un rango dinámico amplio como una SNR alta, así como un bajo consumo, son características deseables en los convertidores A/D para micrófonos inteligentes. En el caso concreto de un VCO-ADC que emplea un oscilador en anillo esto supone un problema, ya que para aumentar su SNR en señales pequeñas es necesario aumentar la ganancia del control de su oscilador interno. Aumentar esta ganancia aumenta la SNR debido a que variaciones más pequeñas en el voltaje de entrada del convertidor producen una variación de fase mayor, como se ve en la ecuación (5), por lo que el convertidor ofrece una mayor granularidad. No obstante, aumentar esta ganancia implica reducir el rango dinámico y por tanto perder SNR para señales grandes. Esto es debido a que un oscilador en anillo solamente tiene un comportamiento lineal bajo ciertas condiciones de corriente/voltaje y temperatura, y una ganancia alta implica una saturación más rápida. Como se ha expuesto en la introducción, obviando la temperatura, la frecuencia producida por una corriente de control I tiene una relación similar a la siguiente:

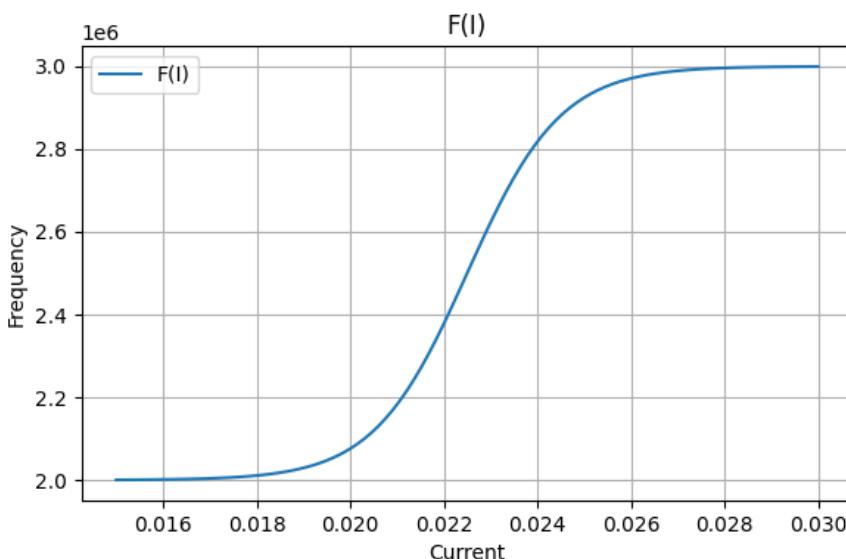


Figura 5 (bis): Forma de una función de transferencia de un oscilador en anillo

En la parte lineal de la función, podemos aproximar el valor de F con:

$$F(I) = K \cdot I + F_0 \quad (12)$$

donde F_0 es la frecuencia en reposo del oscilador

Si asumimos que I es una variable que depende del tiempo, entonces

$$F(I) = K \cdot I(t) + F_0 \quad (13)$$

Por tanto, si para una ganancia de control $K = 1$ el régimen linear del oscilador del ejemplo se encuentra aproximadamente en el intervalo [21,24] (mA), para una ganancia K cualquiera este intervalo se reduce a $[21/K, 24/K]$ (mA), habiendo reducido el rango dinámico de forma proporcional al aumento de ganancia.

La solución propuesta para dicho problema en este proyecto es utilizar varios osciladores en anillo en un mismo convertidor, y combinar sus salidas para obtener un resultado que tiene tanto una alta ganancia para señales pequeñas, y por tanto muy buena SNR en la conversión de estas, como un alto rango dinámico que permite la conversión de señales grandes. Visto desde un punto de vista de especificación, la propuesta consiste en combinar varios convertidores en uno solo. La siguiente figura muestra un ejemplo con dos convertidores

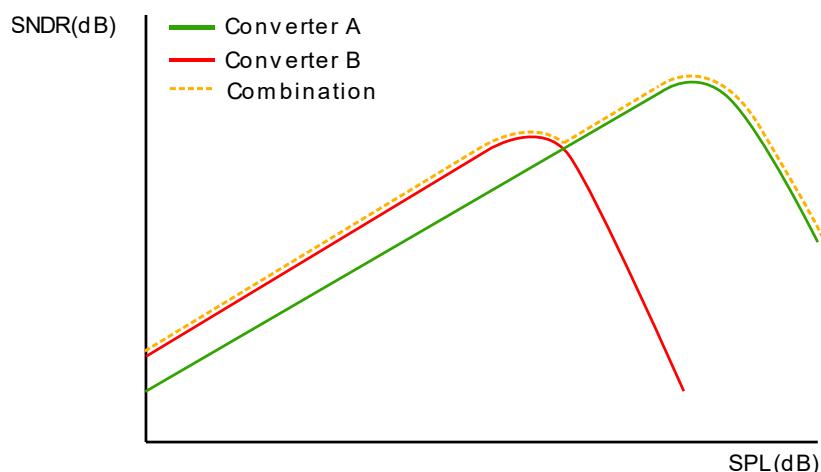


Figura10: Extensión del rango dinámico mediante combinación de convertidores

En el caso de la imagen superior, el convertidor B (rojo) tendría un oscilador con una ganancia mayor al convertidor A (verde). Por tanto, su SNR es mayor para

señales pequeñas, pero disminuye rápidamente al traspasar un umbral de potencia. Por el contrario, el convertidor A (verde), tiene un amplio rango dinámico, pero no tiene una SNR tan buena como el B. Si se consigue hacer una combinación de ambos convertidores, aprovechando al máximo la SNR y rango dinámico de cada uno, la línea discontinua amarilla podría ser el resultado, obteniendo un convertidor con mucho rango dinámico y alta SNR al mismo tiempo.

A continuación, se muestra un diagrama de bloques conceptual que ejemplifica un modelo de sistema para implementar esta solución con tres osciladores

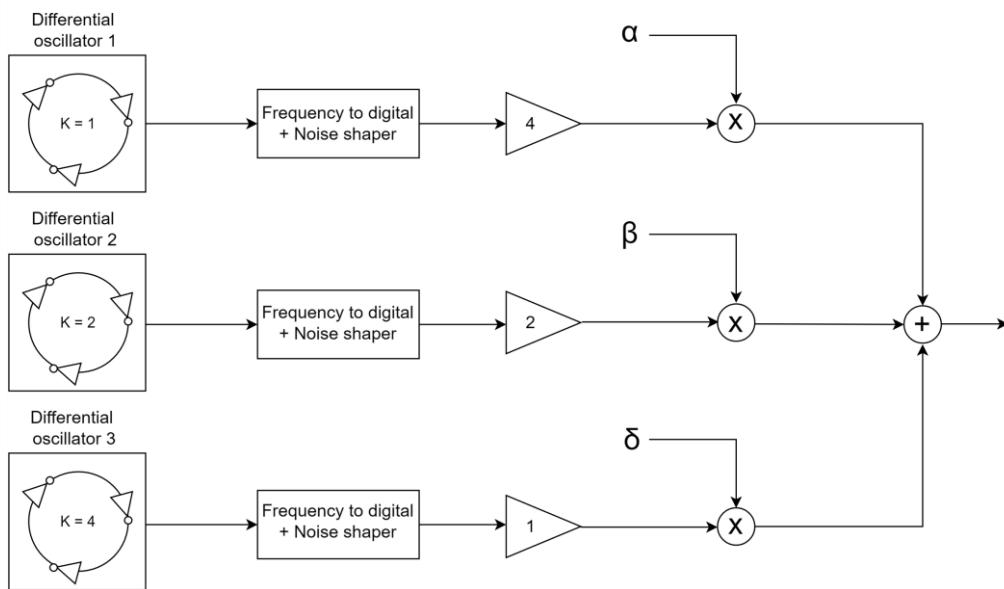


Figura 11: VCO-ADC con varios osciladores y combinación α, β, δ

En la figura superior, se utilizan tres osciladores, cada uno con una ganancia diferente (indicada con K en su interior) y se realiza una combinación de sus salidas de forma ponderada.

Antes de continuar con la explicación, es necesario hacer un inciso para comentar el carácter diferencial del sistema implementado, pues es importante para la siguiente sección.

Es bien sabido que en circuitos electrónicos sensibles al ruido y la distorsión es recomendable utilizar circuitos diferenciales. Tal es el caso para este convertidor

analógico digital. Además, en este caso, ofrece una ventaja extra. Utilizando dos osciladores como los del ejemplo anterior, donde $K1 = -K2$:

$$F1(t) = K1 \cdot I(t) + F0$$

$$F2(t) = -K1 \cdot I(t) + F0$$

Si restamos $F1(t) - F2(t)$, el resultado es

$$F(t) = 2 \cdot K1 \cdot I(t) \quad (14)$$

Puede observarse que el resultado equivale a un nuevo oscilador cuya frecuencia en reposo es 0. De ahora en adelante, las gráficas corriente-frecuencia se realizan con esta propiedad en mente.

Cada uno de los osciladores de la figura 11 puede implementarse de la siguiente manera:

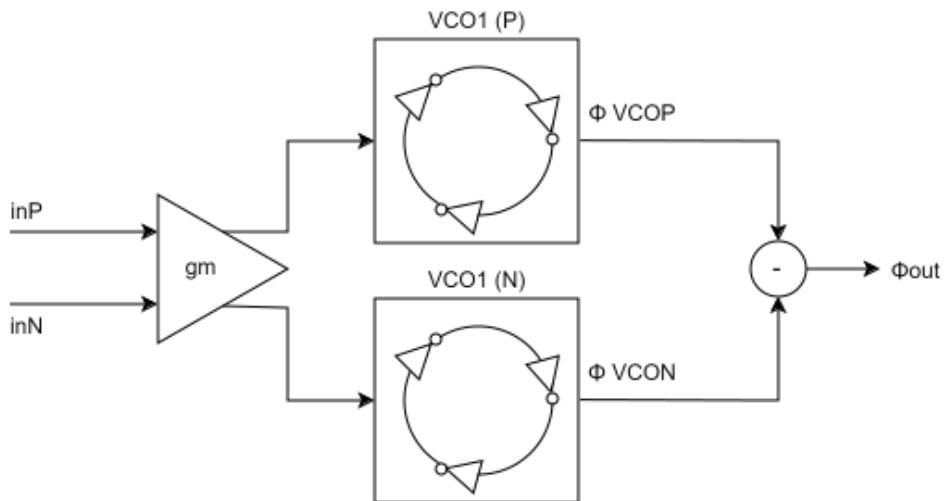


Figura 12: Sistema diferencial de dos osciladores en anillo

Continuando con el planteamiento del sistema, en la figura 11 se observa que cada uno de los tres osciladores tiene una ganancia distinta. Para lograr combinar las salidas de varios osciladores con diferente ganancia, es necesario ajustar la salida de cada oscilador antes de la combinación. De esta forma, para el circuito “combinador”, no habrá ninguna diferencia más allá de la SNR de cada uno. Por supuesto, este ajuste se realiza después de haber digitalizado la señal, ya que de

hacerlo antes empeoraría la SNR de los de mayor ganancia (Sería, a efectos prácticos, equivalente a reducir su ganancia). En el diagrama de bloques conceptual (Figura 11), este ajuste está representado como un bloque de ganancia, que como puede observarse, se encuentra después de la digitalización.

En la siguiente figura se observan las funciones de transferencia de tres osciladores en anillo, cada uno con ganancias distintas. Esta figura pretende ilustrar, junto con la siguiente, el porqué de la necesidad del ajuste digital posterior:

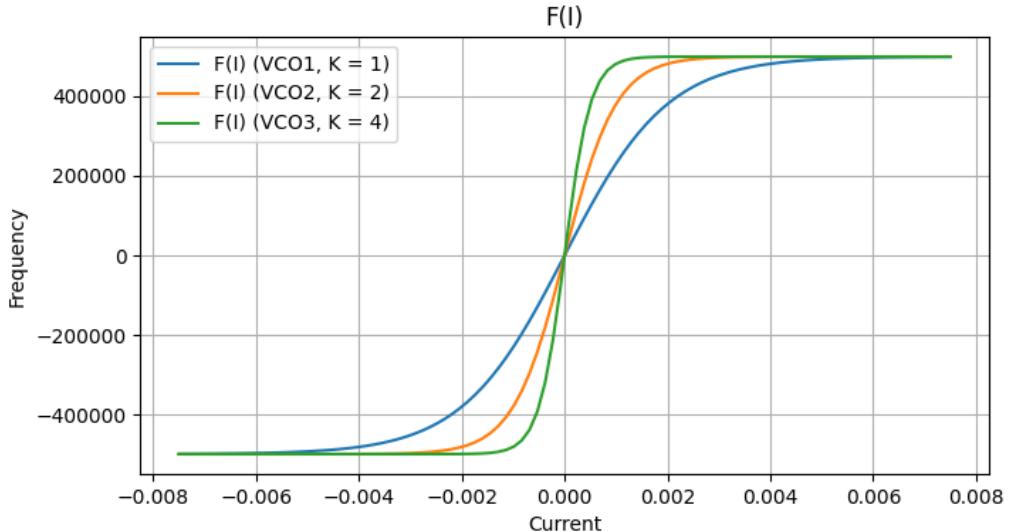


Figura 13: Funciones de transferencia de osciladores con diferente ganancia

Cada una de las funciones de transferencia anteriores, que ejemplifican las funciones de transferencia de los osciladores de la figura 10, pueden ser aproximadas linealmente mediante las siguientes ecuaciones, que son de la forma de la ecuación 1 (Ecuación fundamental de un oscilador en anillo con frecuencia en reposo igual a 0). Así,

$$F1(I) = K \cdot I(t)$$

$$F2(I) = 2 \cdot K \cdot I(t)$$

$$F3(I) = 4 \cdot K \cdot I(t)$$

Para poder realizar una combinación de caminos, es necesario que todos ellos aparenten tener la misma ganancia. Es por esto por lo que se realiza un ajuste, que consiste en multiplicar F1 por 4 y F2 por 2. De esta forma, solamente el punto de saturación y la SNR cambia entre osciladores.

Después del ajuste, el resultado es el siguiente:

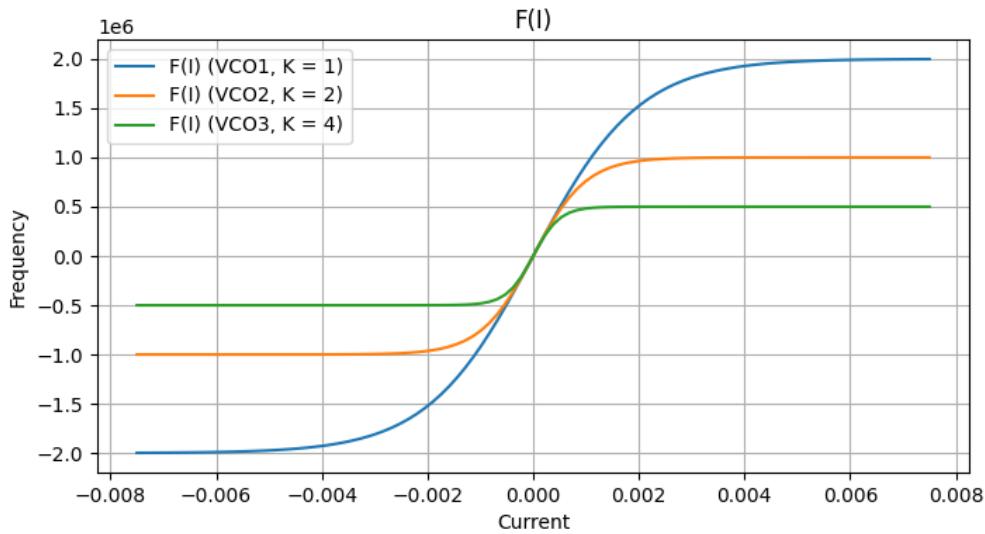


Figura 14: Funciones de transferencia después del ajuste digital

Y en la zona lineal de cada oscilador, todos ellos pueden ser aproximados como:

$$F1(I) = 4 \cdot K \cdot I(t)$$

$$F2(I) = 4 \cdot K \cdot I(t)$$

$$F3(I) = 4 \cdot K \cdot I(t)$$

Por lo que, dentro del régimen lineal, F1, F2 y F3 son indistinguibles.

Como puede observarse, después del ajuste digital de ganancia, es perfectamente posible mantenerse siempre dentro del régimen lineal de alguno de los osciladores a lo largo de todo el rango dinámico, pudiendo aprovechar al máximo la SNR de los de mayor ganancia cuando la señal de entrada es pequeña. Al hacer $K3 = 2 \cdot K2 = 4 \cdot K1$, no es necesario ningún elemento de hardware extra para realizar el ajuste digital, pues basta con hacer desplazamientos.

Ahora bien, para lograr un comportamiento óptimo del convertidor, es necesario generar las señales α , β , δ (ver Figura 10) de tal forma que $\alpha + \beta + \delta = 1$, y asegurarse de que los osciladores que se encuentran fuera de su régimen lineal tengan una ponderación casi nula en cualquier situación, mientras que se aprovechan al máximo los osciladores de mayor ganancia, y por tanto mejor SNR cuando es posible.

La siguiente sección analiza dos posibles métodos para lograrlo.

1.3.2 - Combinación de caminos con redes neuronales

Una prometedora opción a explorar es el uso de redes neuronales, ya que se cree, en base a su buen funcionamiento en problemas similares [18], que podrían conseguir un muy buen resultado.

Una red neuronal simple con tres entradas, una para el valor de cada oscilador, y tres salidas utilizando la función softmax [19] en la capa de salida podría realizar esta labor convenientemente. Es probable que la red neuronal no necesite una gran cantidad de capas internas, al tratarse de un problema relativamente simple.

Sin embargo, la red neuronal presenta varias desventajas. En primer lugar, debe ser entrenada, posiblemente con datos de usuarios y sus percepciones, lo que supone un coste muy grande. Además, y posiblemente más importante, la implementación de una red neuronal es muy costosa en hardware, disparando el área y el consumo de un convertidor que debe ser de muy baja potencia.

1.3.3 - Combinación de caminos con estimadores estadísticos de potencia

Una opción más clásica para la generación de α , β y δ es utilizar la potencia de la señal de entrada para ponderar cada salida. Dicha potencia puede ser comparada con valores preestablecidos, que junto a un algoritmo clásico establecen la ponderación de cada camino. Esta forma de abordar el problema es más sencilla en coste, consumo y área, pero presenta también varios inconvenientes. En primer lugar, para aprovechar al máximo el rango dinámico de cada oscilador, los

valores preestablecidos deben ser cuidadosamente seleccionados. Además, el algoritmo que establece las ponderaciones debe ser simple y a la vez funcional, para no perder rendimiento y tener una implementación sencilla en hardware.

2 – DISEÑO DEL SISTEMA IMPLEMENTADO

La siguiente sección analiza el sistema implementado a nivel de sistema en detalle. La implementación hardware concreta de cada componente se describe en la sección 3: Diseño hardware del sistema

2.1 - Diseño de alto nivel de un sistema de adquisición de audio con extensión de rango y 2 VCO

El sistema planteado utiliza 2 VCOs para probar el concepto de la extensión de rango de manera sencilla. Sin embargo, el diseño es fácilmente escalable para utilizar más osciladores. Para cada VCO se dispone de un elemento Noise-Shaper, que permite alcanzar una mejor relación señal a ruido en cada canal. Se dispone de dos caminos, de ahora en adelante HDR (High Dynamic Range) y HSNR (High Signal-to-Noise Ratio). La única diferencia efectiva es que la ganancia del oscilador del segundo es cuatro veces la del primero, que es corregida digitalmente como se ha explicado en el apartado 1.3. El diagrama a alto nivel del sistema implementado es el siguiente:

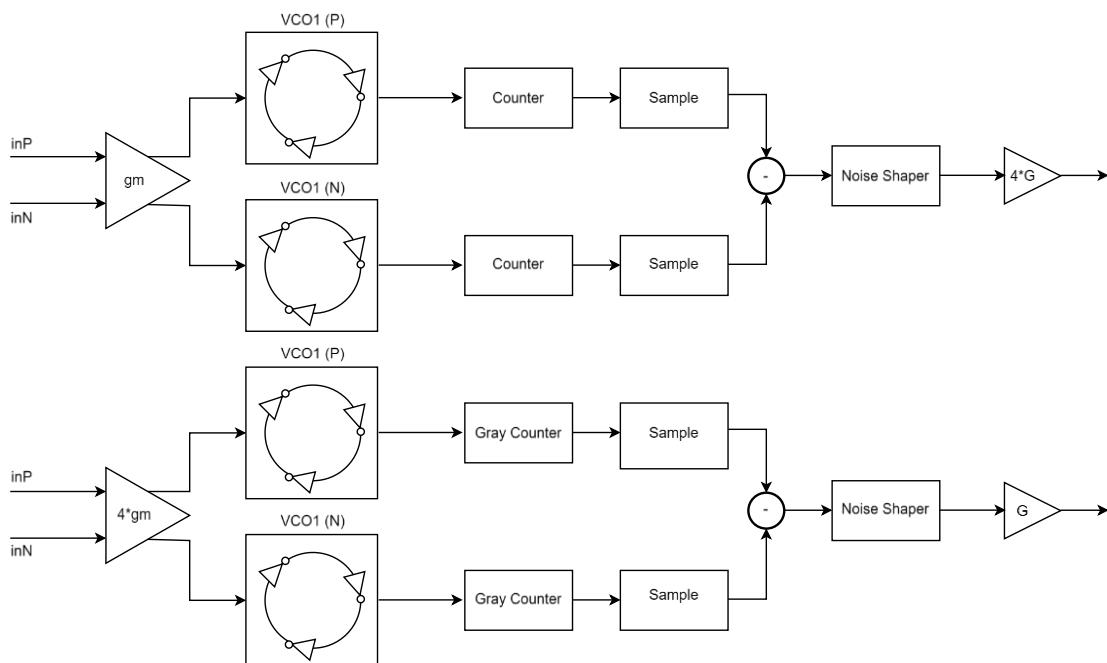


Figura 15: Diseño de alto nivel de cada camino del sistema

Como se ve, el sistema emplea las técnicas explicadas en la introducción para un VCO con $F_0 = 0$ Hz, un muestreo de la fase mediante contador y modulador sigma-delta para aplicar noise-shaping en el diezmado. Otra opción planteada en el diseño a nivel de sistema fue utilizar solamente un noise-shaper, y realizar la corrección de ganancia digital junto con la combinación de caminos antes de este. Un diseño con estas características tendría ventajas de implementación, área y consumo. Sin embargo, debe tenerse en cuenta que el noise-shaper tiene un acumulador interno actuando como integrador, por lo que la señal de entrada a este debe cambiar suavemente, ya que la salida depende de valores anteriores de la señal. Si esto no se cumple, se ocasionan artefactos en la salida del convertidor. Se comprobó por simulaciones que librarse de estos artefactos de forma asegurada suponía una complicación importante, por lo que finalmente se ha utilizado el diseño de la figura superior.

Para probar el desempeño del sistema, se ha creado en Simulink un modelo, adjunto en el anexo “Modelos Matlab/Simulink : Integer-true”. Este modelo ha sido creado con el fin de simular el sistema con valores enteros de un tamaño definido, y ha sido de gran utilidad para optimizar el diseño antes de pasar a una implementación hardware.

En este modelo, se simulan los VCO mediante bloques como los siguientes:

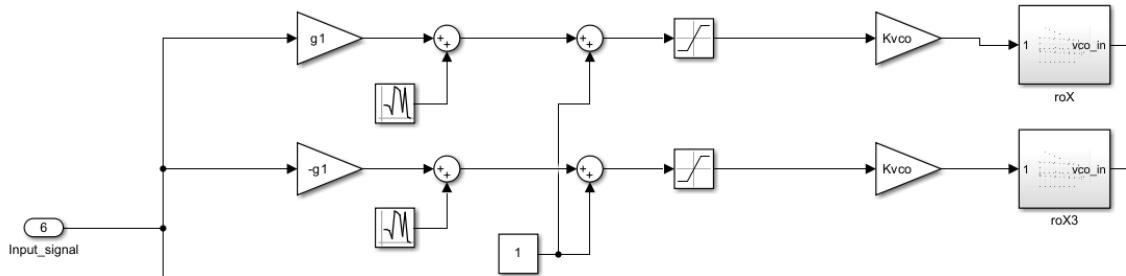


Figura 16: Modelo de VCOs en Simulink

El bloque roX es el oscilador propiamente dicho, y contiene los inversores. Se realiza un tratamiento previo a la señal de entrada donde se le añade el ruido

correspondiente al oscilador, así como la ganancia. Utilizar una ganancia inversa (g_1 y $-g_1$) en sendos VCOs proporciona el comportamiento diferencial buscado.

El contador, muestreo y noise-shaper son fácilmente implementables con acumuladores, y la corrección de ganancia se implementa con un bloque de ganancia de Matlab [20]. Para ver como se implementa cada bloque exactamente, consultar el anexo “Modelos de Matlab/simulink : Integer-true”.

El principal objetivo para diseñar el modelo integer-true ha sido establecer el número de bits y frecuencias de muestreo. La frecuencia de muestreo influye directamente en la SQNR del sistema, por ser la frecuencia utilizada en el noise-shaper. Cuanto más alto sea la ratio de sobremuestreo del noise-shaper, mayor es la SQNR del sistema, acorde con la teoría de los convertidores sigma-delta. Sin embargo, utilizar una frecuencia de muestreo alta en el noise-shaper obliga a que el acumulador interno tenga una cantidad de bits mayor, lo que supone un problema a la hora de la implementación, ya que pueden requerirse registros intermedios en caso de que haya retrasos grandes entre bloques lógicos. De forma general, dado que la frecuencia de muestreo final es de 3.072 MHz, es recomendable que el acumulador interno no pueda sufrir “overflows” entre dos muestras para que funcione correctamente. Como el noise-shaper realiza tantas sumas por periodo de muestreo como F_2/F_s , es una buena regla de diseño que, para una frecuencia F_2 del noise-shaper:

$$3.072E6 \cdot 2^{(N\text{bits} - M\text{bits})} \geq F_2 \quad (15)$$

Donde Nbits es el número de bits del acumulador del noise-shaper y Mbits es el número de bits de su entrada. No obstante, esta es una cifra orientativa, ya que, si el modulador se encuentra en un estado estable, es posible que con menos bits funcione correctamente. También es posible que, debido a tener varias muestras consecutivas altas, aún con este número de bits se produzcan “overflows” en el acumulador. Además, un número de bits pequeño significa perder precisión en el acumulador, por lo que, incluso funcionando correctamente, menos bits implican una pérdida de SQNR. Por ello, se ha recurrido en última instancia a simulaciones para establecer el número de bits de cada bloque del sistema.

Se decidió que las únicas frecuencias F2 posibles eran o bien 24 MHz o 48 MHz, ya que para generar Fs de forma sencilla es conveniente que $F_s \cdot 2^K = F_2$, algo que permite utilizar un divisor de reloj implementado con flip-flops en lugar de un PLL (Phase Locked Loop), y frecuencias por debajo de 24 MHz no pueden utilizarse por estar por debajo de la frecuencia mínima de muestreo, como se verá más adelante. Frecuencias por encima de 48 MHz requieren demasiados bits en el noise-shaper y complican la implementación. Establecer esta frecuencia de muestreo es algo que depende también del diseño del oscilador, ya que un oscilador más lento puede ser muestreado con una frecuencia más lenta. Al final, por motivos de consumo y ganancia, se decidió utilizar una frecuencia de oscilador de 6 MHz, y una frecuencia de muestreo de 24 MHz, que es la mínima que puede proporcionar un funcionamiento correcto del sistema con este oscilador. Una vez establecidas las frecuencias, se procedió a realizar diferentes simulaciones para calcular la SQRN esperada utilizando una señal de fondo de escala para el canal HSNR como entrada. Se ha comprobado por simulación que no cumplir con la regla de diseño de la ecuación (15) lleva a que el convertidor funcione de manera completamente inestable, por lo que el noise-shaper siempre debe tener al menos tres bits más que el contador que lo precede, acorde con la ecuación mencionada. Como ejemplo, este resultado se ha observado en la siguiente simulación, utilizando tan solo dos bits extra para el noise-shaper, y una F2 tal que $F_2 = 8 \cdot F_s$.

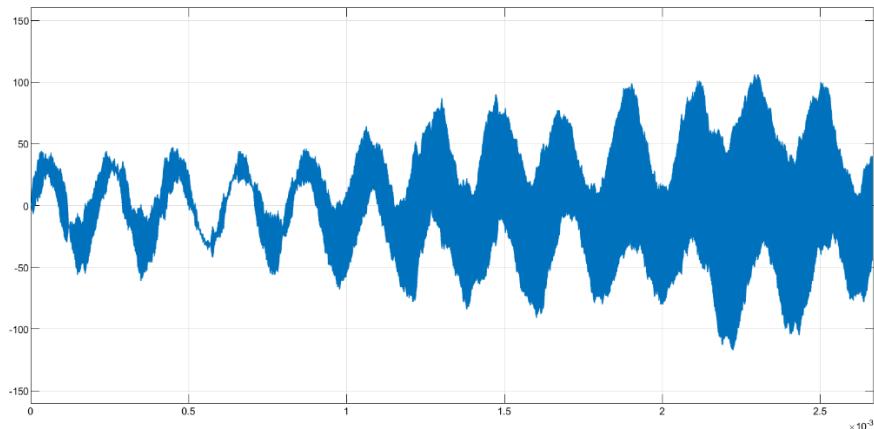


Figura 17: Comportamiento inestable del convertidor en simulación

Para las combinaciones factibles, se ha elaborado la tabla de las siguientes páginas con los resultados de diferentes simulaciones, donde N es el número de bits del contador, y M el número de bits del noise-shaper. Como en este caso el objetivo es optimizar la SQNR del convertidor, se han realizado las simulaciones sin utilizar distorsión ni ruido en las entradas, a excepción de una saturación inmediata (no progresiva) del canal HSNR al llegar a cierto valor. En las gráficas aparece representada la respuesta en frecuencia de cada uno de los canales, siendo el azul el canal HSNR y el rojo el canal HDR. El dato de SQNR que aparece en la tercera columna es el que corresponde a la respuesta del canal HSNR.

N	M	SQNR (dB)	Respuesta en frecuencia
7	10	97.97	
8	11	105.10	

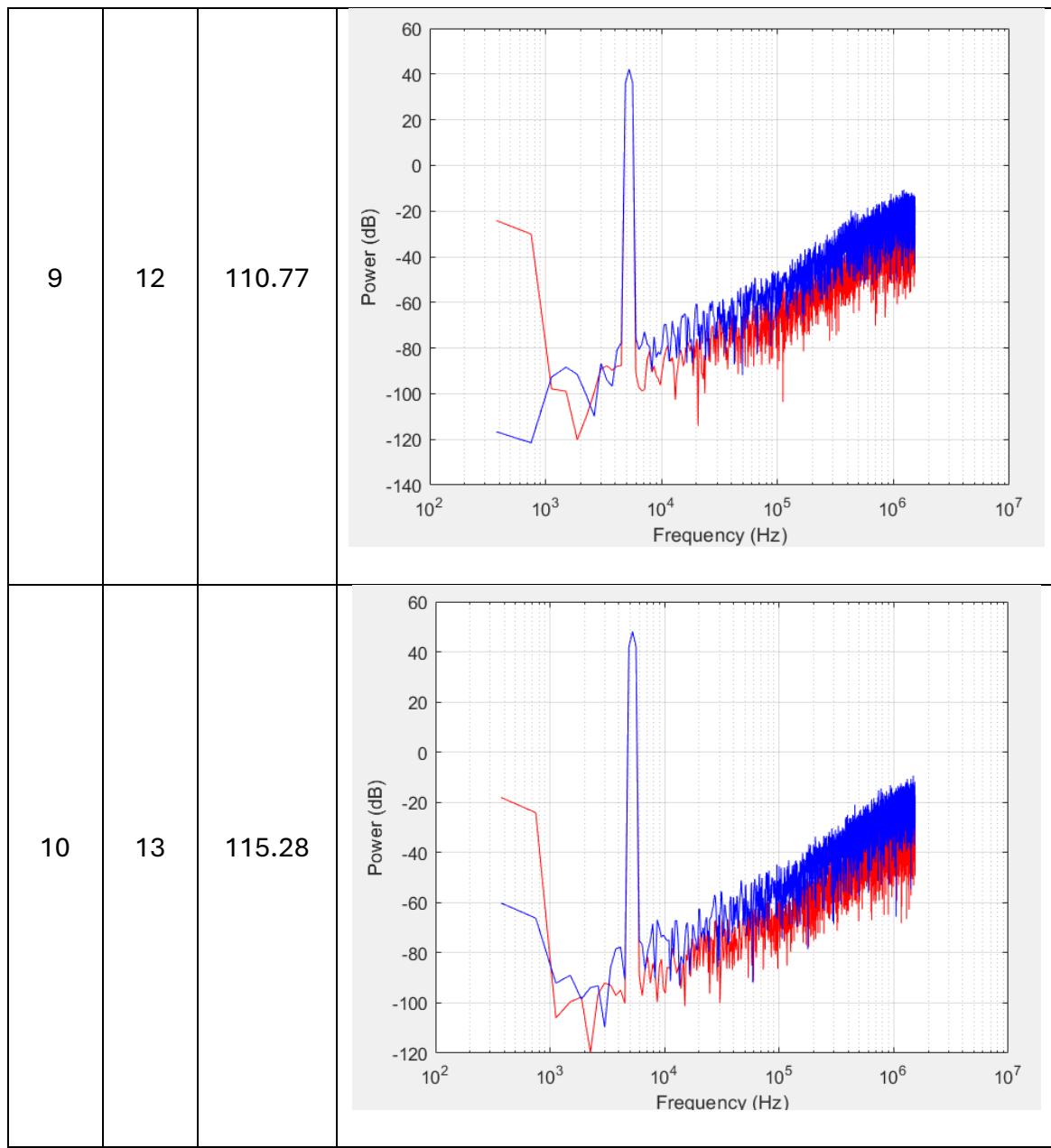


Tabla 1: SQNR frente a número de bits del sistema

En vista de los anteriores resultados, se seleccionaron 9 bits para el contador y 12 bits para el noise-shaper, por suponer un buen compromiso entre sencillez de implementación y consumo frente a SQNR.

Para la salida final, dado que hay que aplicar una ganancia de salida al canal HSNR, se utilizan 11 bits, ya que para multiplicar por cuatro el valor del canal

mencionado es necesario realizar un desplazamiento de dos bits hacia la izquierda.

En resumen, se establecieron para el sistema las siguientes características:

- Frecuencia de muestreo final: 3.072 MHz
- Frecuencia del oscilador: 6 MHz
- Frecuencia de muestreo inicial: 24 MHz
- Número de bits del contador: 9
- Número de bits del noise-shaper: 12
- Número de bits de salida del sistema: 11

Para finalizar este apartado, se muestra a continuación una de las simulaciones ejecutadas en más detalle. En ella puede verse la salida de los canales HDR (arriba) y HSNR (abajo) en el plano del tiempo, para una entrada senoidal en el convertidor.

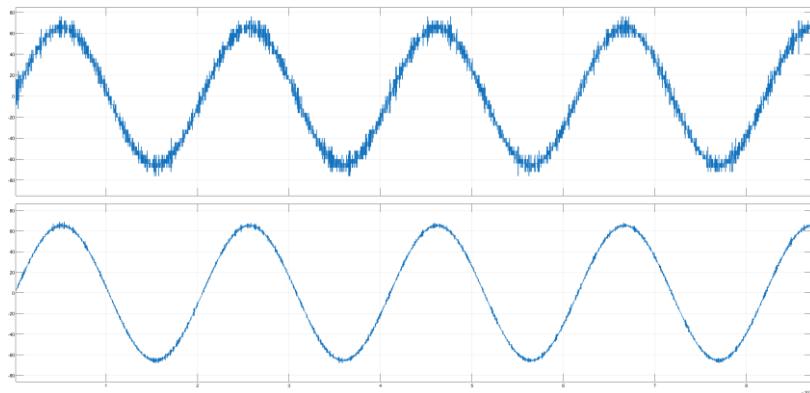


Figura 18: Salidas del canal HDR (arriba) y HSNR (abajo) en el modelo integer-true

Puede observarse que el canal HDR tiene una forma de onda con mucho más ruido de cuantificación, pero ambas representan una señal con la misma amplitud, habiendo conseguido el propósito del sistema. Es fácilmente comprobable en el plano de la frecuencia que la SQNR del canal HDR es inferior a

la del HSNR, en el caso concreto de esta señal, 103.97 dB y 93.37 dB respectivamente. La siguiente figura muestra esta respuesta en frecuencia:

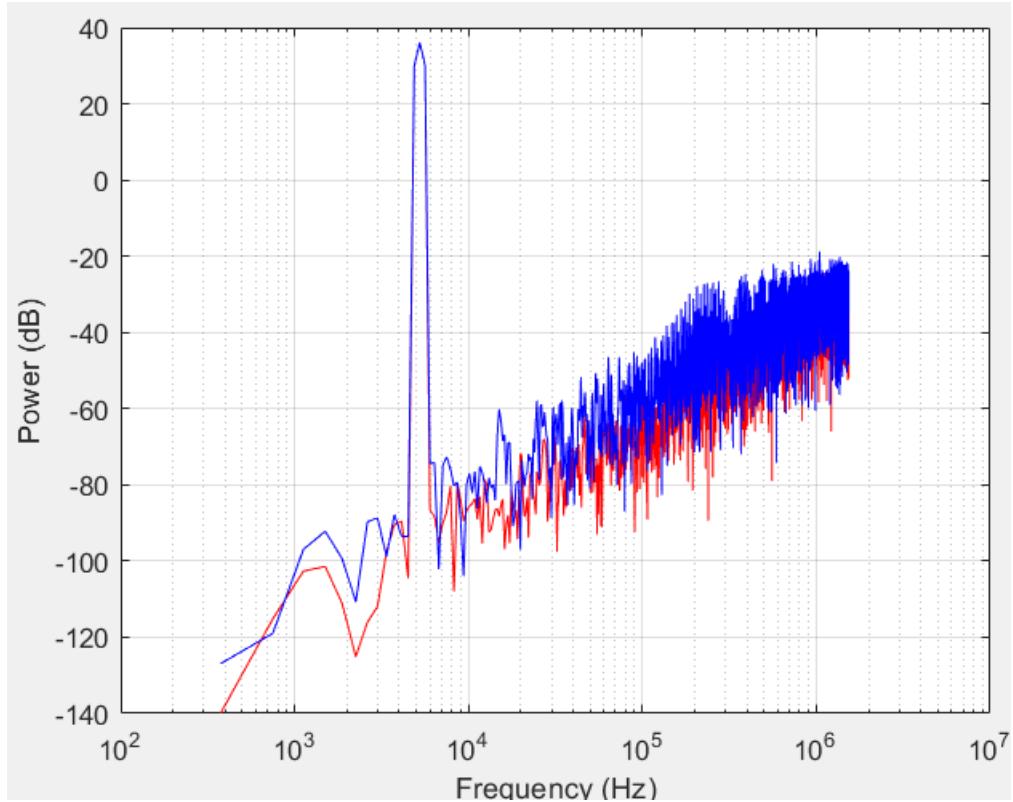


Figura 19: Vista de las salidas HDR (azul) y HSNR (rojo) en el plano de la frecuencia

Para una señal de fondo de escala, el convertidor alcanza una SQNR de 110 dB, algo que cumple el rendimiento buscado.

Una vez comprobado que es posible implementar a nivel de sistema ambos caminos, se procede a decidir el método por el que se va a implementar la extensión de rango dinámico. Se contemplan a continuación las dos opciones planteadas en el apartado 1.3.2 y 1.3.3

2.2 - Opción 1: optimización del rango dinámico por comparadores implementados con redes neuronales

Con el fin de estudiar el uso de redes neuronales para la selección de caminos, se ha desarrollado un programa en Python [21], utilizando Keras [22] y Tensorflow

[23], que entrena una red neuronal para asignar un peso a cada uno de los dos caminos. La red neuronal diseñada tiene dos entradas, correspondientes al valor en tiempo real de los caminos HDR y HSNR, dos capas ocultas, una de 32 y otra de 16 neuronas, ambas activadas mediante ReLU [24], y una capa de salida que utiliza la función softmax [19], produciendo los valores α y $1 - \alpha$.

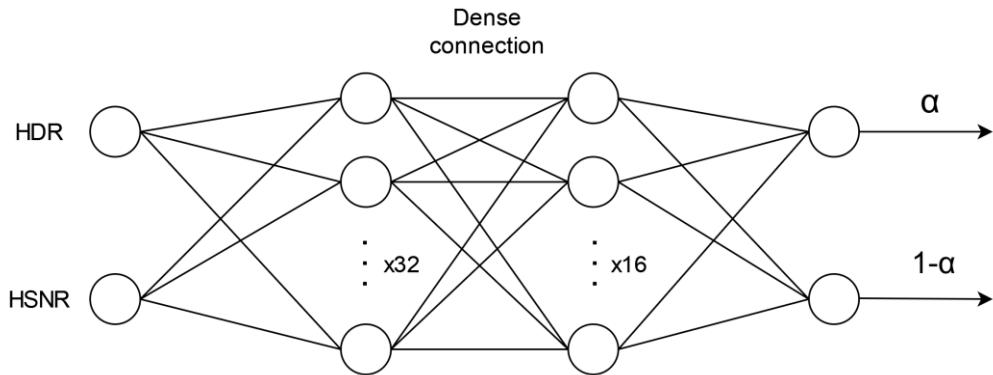


Figura 20: Arquitectura de red neuronal de generación de α

La arquitectura de esta red se ha diseñado de esta manera por varios motivos. En primer lugar, se proporcionan suficientes capas y neuronas intermedias para que su número no suponga una limitación, dado que el objetivo es comprobar la viabilidad del uso de redes neuronales para resolver el problema, y no su optimización. En segundo lugar, la capa de salida proporciona dos pesos que han de sumar uno. Por ello, se utiliza la función softmax, que es típicamente utilizada en labores de clasificación por esta propiedad. En las demás capas se utiliza ReLU (Rectified Linear Unit) por su eficiencia computacional.

La estructura completa del sistema de simulación y entrenamiento de la red neuronal es la siguiente:

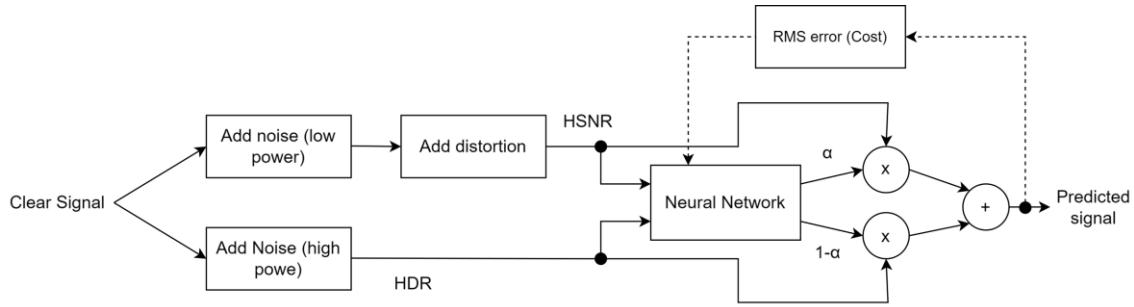


Figura 21: Arquitectura completa de entrenamiento de la red

De forma resumida, se simulan los dos canales de entrada, y se entrena la red minimizando el error de su salida frente a la señal de entrada que la produce. Con ello en mente, se procede a una explicación más detallada del funcionamiento de la red.

Como entradas a la red, se van a utilizar dos señales que representan el canal HDR y el canal HSNR respectivamente. El canal HDR debe tener una distorsión mínima, pero una cantidad de ruido alta. Por el contrario, el canal HSNR debe tener mucha distorsión para una señal de entrada grande, pero una cantidad reducida de ruido.

Para simular el efecto de la distorsión del VCO del canal HSNR, se aplica una distorsión como la siguiente a la entrada correspondiente:

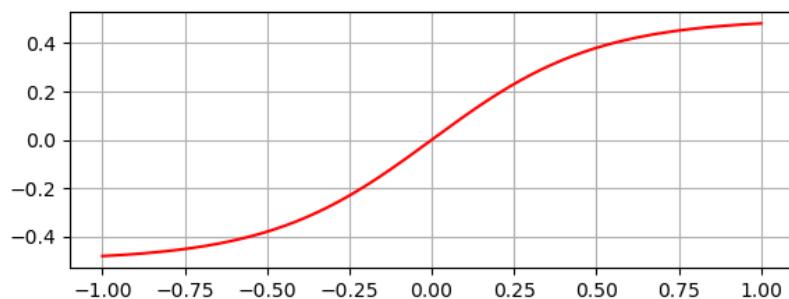


Figura 22: Distorsión aplicada al canal HSNR

Esta función, definida como

$$1 / (1 + e^{-4 \cdot x}) - 0.5 \quad (15)$$

(sigmoide ajustada para pendiente 1 en el origen y asíntotas en [0.5 , -0.5]), produce un comportamiento totalmente lineal en el origen y una saturación progresiva hacia los extremos, de forma similar al comportamiento de un oscilador en anillo. Además de esta distorsión, se añade una pequeña cantidad de ruido uniformemente distribuido a la señal de entrada del canal HSNR. Para el canal HDR, no se añade ningún tipo de distorsión, pero se introduce una cantidad de ruido 10 veces superior a la del canal HSNR. Para una entrada senoidal de amplitud 1, la entrada a la red neuronal de ambos canales es la siguiente, respectivamente:

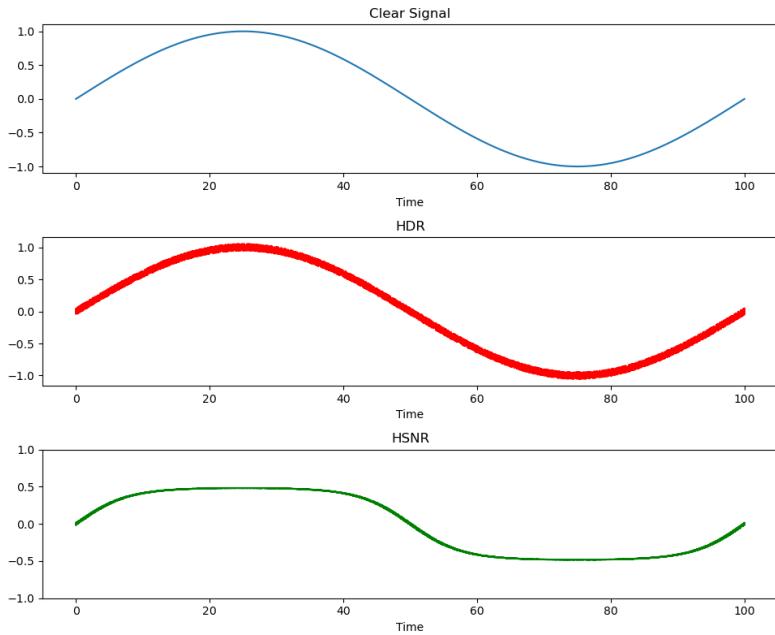


Figura 23: Ruido y distorsión aplicados a las entradas de la red neuronal

Las dos señales generadas mediante los procedimientos descritos anteriormente son las entradas a la red neuronal. La salida de la red son los pesos por los que deben ser multiplicadas cada una de las entradas (α y $1-\alpha$), por lo que, tras realizar una ponderación y suma, se obtiene la señal predicha por la red, que es la salida del sistema. Si se está realizando un entrenamiento de la red, se calcula el error RMS entre esta salida y la señal de entrada (Clear Signal). Este resultado es

utilizado como función de coste por el optimizador de la red neuronal en el entrenamiento.

Tras entrenar la red utilizando como señal de entrada una rampa senoidal ascendente (función de la forma $t \cdot \sin(t)$), se ha probado el desempeño de esta con diferentes señales, obteniendo con todas ellas un comportamiento similar.

A continuación, se presenta el resultado de aplicar una señal senoidal modulada a la entrada, por considerarse una señal especialmente representativa del comportamiento de la red

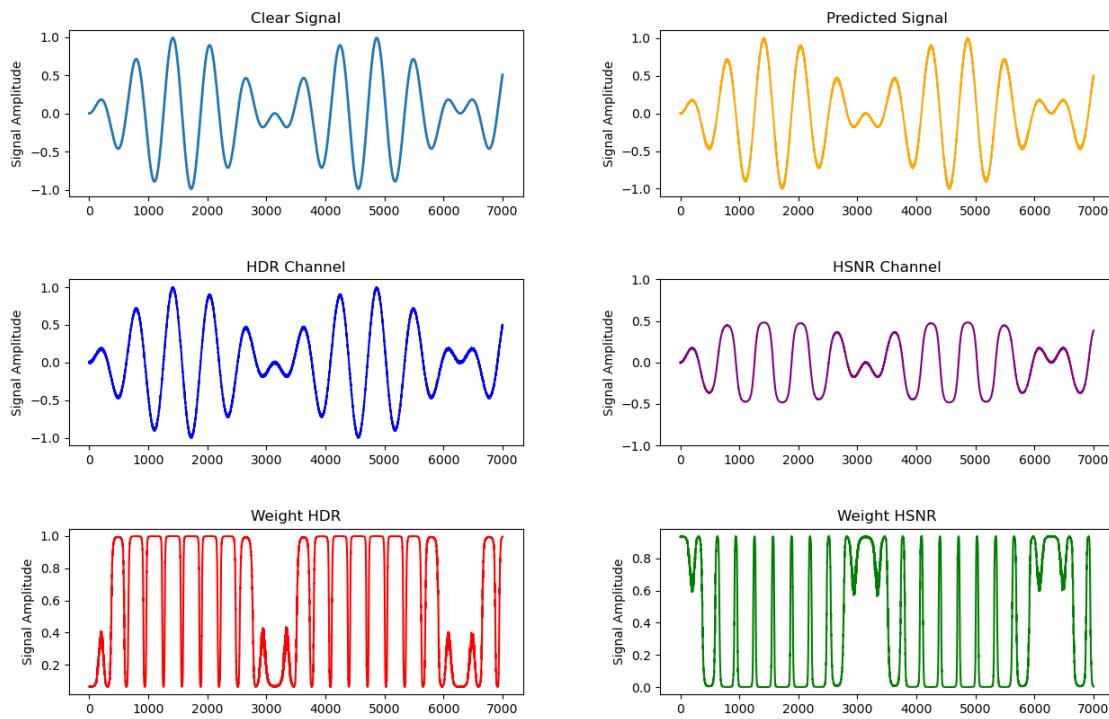


Figura 24: Funcionamiento de la red neuronal tras el entrenamiento

La figura consta de varias señales: la señal de entrada (Clear signal), el resultado de aplicar distorsión y ruido a la misma (HDR y HSNR), los pesos establecidos por la red para cada canal en cada instante de tiempo (Weight HDR y Weight HSNR), y la salida del sistema (Predicted Signal).

Como puede observarse, la red ha aprendido a generar los pesos correctamente, pues la señal predicha es prácticamente igual a la de entrada ($\text{loss} = 6.2080\text{e-}05$). Cuando la señal de entrada es pequeña, el peso del canal HSNR se ve incrementado, mientras que el del canal HDR se ve disminuido. Cuando la señal de entrada es grande, ocurre lo contrario. Sin embargo, la propia red neuronal introduce ruido al variar los pesos con cierta aleatoriedad para valores cercanos, lo cual puede suponer un problema para una implementación real. Esto, posiblemente, se puede solucionar aplicando algún tipo de filtro a la salida de la red para evitar cambios bruscos. Es interesante comprobar como la red neuronal ha aprendido que, alrededor de una amplitud de 0.2, es deseable ponderar más el canal HDR frente al HSNR, pues es donde la distorsión comienza a ser más notoria (ver figuras 22 y 23). Es fácilmente observable que la red neuronal a aprendido a detectar un umbral alrededor de ese valor, y que un cambio de canal ocurre muy rápidamente una vez se atraviesa, teniendo un comportamiento casi de selección binaria a la salida.

La red, a pesar de su correcto desempeño, presenta todos los problemas establecidos en el apartado 1.3.2 (Combinación de caminos mediante redes neuronales) para una implementación real. Hay que destacar que sería necesario realizar un entrenamiento para cada chip fabricado, ya que no todos los osciladores se comportarían igual por variaciones de proceso. Además, se ha comprobado que la red neuronal ha estimado que el comportamiento de selección de camino óptimo es prácticamente una selección binaria dependiente de un umbral, de forma muy similar a como se realizaría mediante estimadores de potencia, que presentan una implementación en hardware mucho más sencilla. Por tanto, se ha decidido utilizar este segundo método.

2.3 - Opción 2: Optimización del rango dinámico con estimadores estadísticos de potencia

Para desarrollar un comportamiento similar al de la red neuronal del apartado anterior, implementando el hardware con comparadores digitales, se ha

desarrollado un circuito que tiene un comportamiento como el de la figura inferior. Para simplificar aún más la implementación y reducir área y consumo, se realiza una selección binaria de camino, es decir, que a solo puede tomar los valores 1 o 0. El comportamiento de sistema buscado se ejemplifica en la siguiente figura:

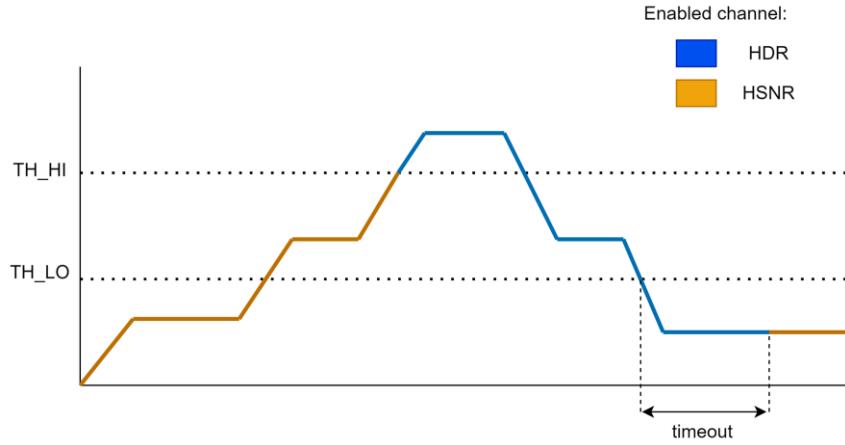


Figura 25: Comportamiento de selección de camino mediante estimadores de potencia

Aplicando la solución propuesta, el canal HDR es seleccionado inmediatamente cuando se sobrepasa un límite TH_HI . Este límite debe corresponderse con el valor en el que la distorsión del canal HSNR comienza a ser notable. Por otro lado, para que se retorne al canal HSNR una vez se ha activado el HDR, debe atravesarse el umbral TH_LO , y se debe permanecer bajo ese umbral un tiempo suficiente (Timeout en la imagen). El propósito de este tiempo de espera es evitar cambios constantes entre el canal HDR y el HSNR, algo que posiblemente podría llegar a causar una distorsión audible.

En una simulación en Matlab [20], utilizando los dos caminos descritos anteriormente y este algoritmo de selección, el resultado es el siguiente:

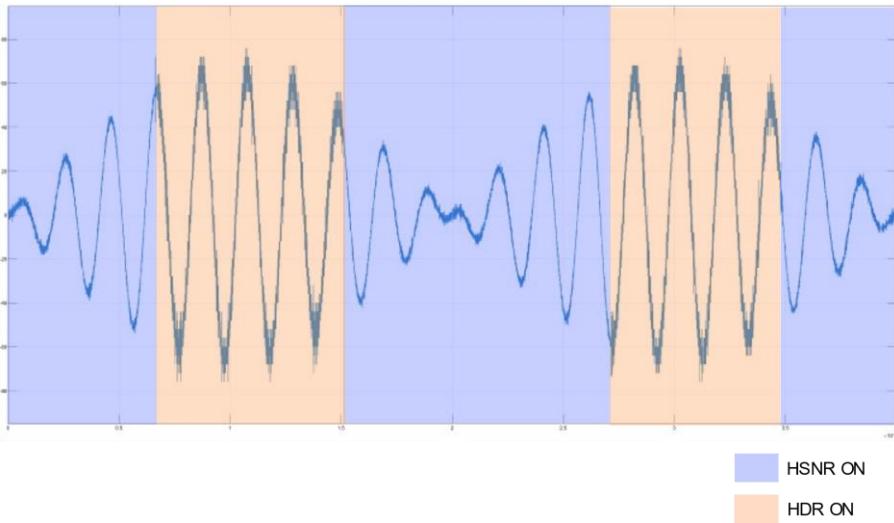


Figura 26: Simulación utilizando estimadores estadísticos de potencia

El modelo de Simulink utilizado para generar esta simulación se encuentra en el anexo “Modelos de Matlab/Simulink: Bit-true”. Este modelo representa el sistema completo implementado a nivel de puerta lógica, y se ha utilizado como base para diseñar los esquemáticos en Cadence Virtuoso [25]. El apartado 3.6 se dedica al desarrollo de este modelo. Algunas simulaciones de comportamiento utilizando este modelo se muestran a continuación, representando cada una un ejemplo del comportamiento de la combinación de caminos implementado. Aparecen marcados los umbrales con líneas horizontales (roja y azul) en el mismo gráfico que la señal, y el canal superior representa el camino seleccionado ($1 = \text{HDR}$, $0 = \text{HSNR}$). La señal propiamente dicha se encuentra marcada en color amarillo.

La primera de estas simulaciones ejemplifica el comportamiento de selección para una señal pequeña, por lo que la salida nunca pasa a estar por encima del umbral, y la salida siempre se asigna al canal HSNR.

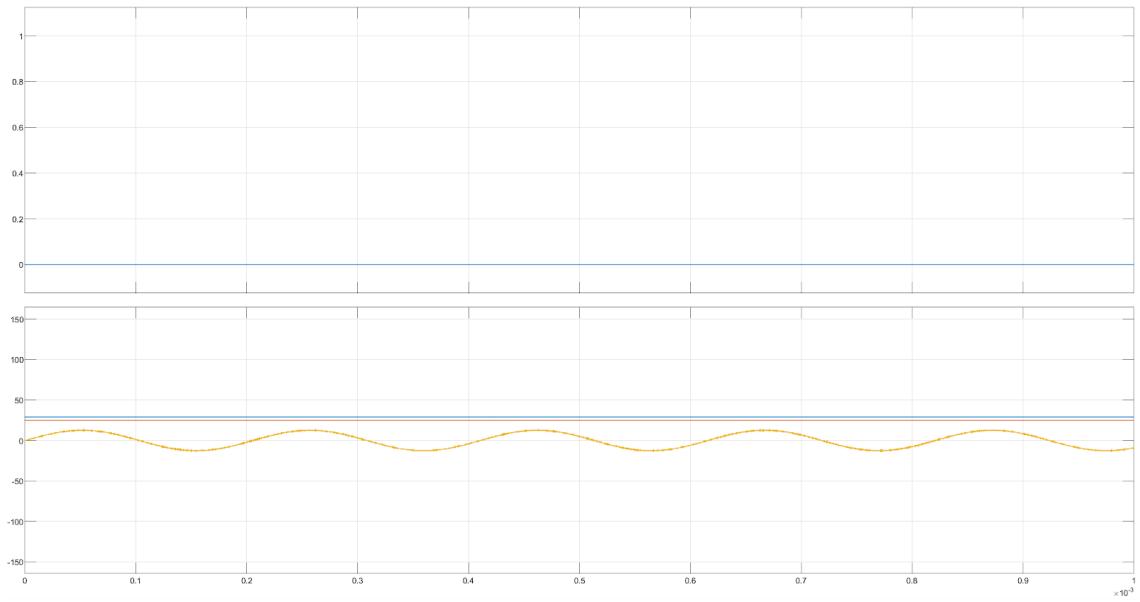


Figura 27: Simulación de selección de camino para señal pequeña (alpha arriba, TH_HI, TH_LO y señal abajo)

Con una señal grande, por encima del umbral TH_HI en valor absoluto, se cambia inmediatamente del canal HSNR al canal HDR y se permanece en él.

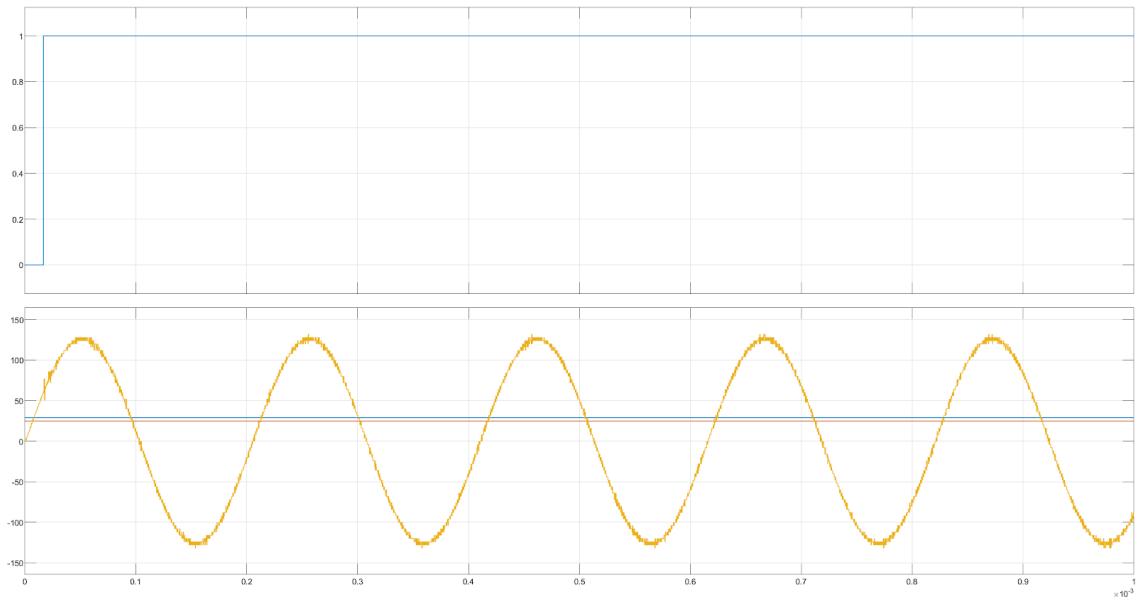


Figura 28: Simulación de selección de camino para señal grande (alpha arriba, TH_HI, TH_LO y señal abajo)

Si se utiliza una señal grande, pero que tiene un algúin de potencia repentino, el canal activo es siempre el HDR. Este es precisamente el propósito del “timeout”

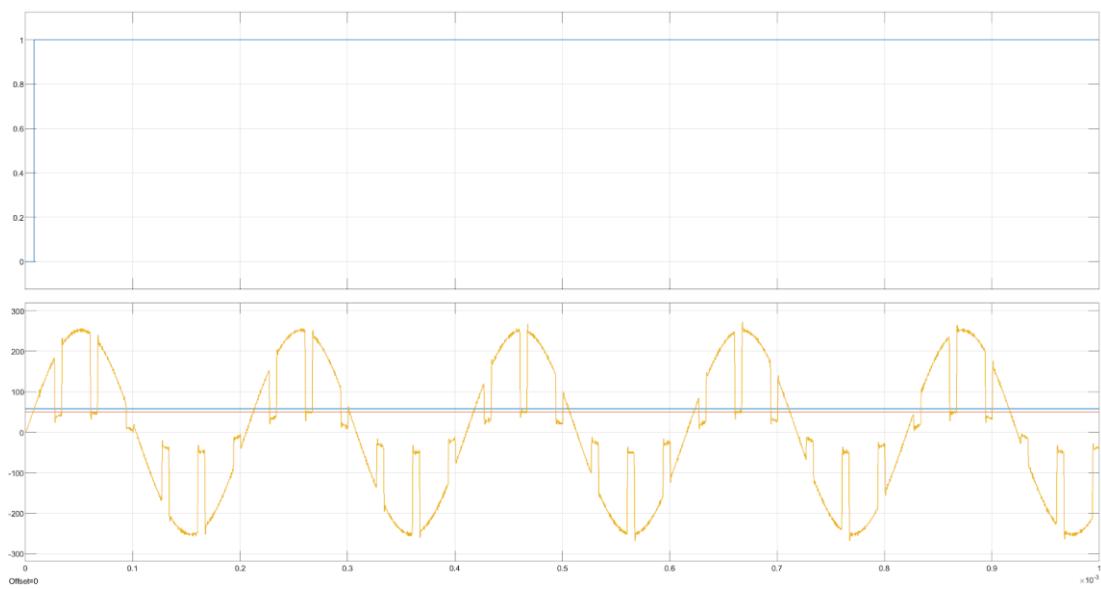


Figura 29: Simulación de selección de camino para señal con reducción brusca de amplitud (alpha arriba, TH_HI, TH_LO y señal abajo)

Por último, si la entrada es una señal que varía en amplitud en el tiempo, y permanece un tiempo considerable por encima y por debajo de los umbrales, se realizan cambios de selección de camino

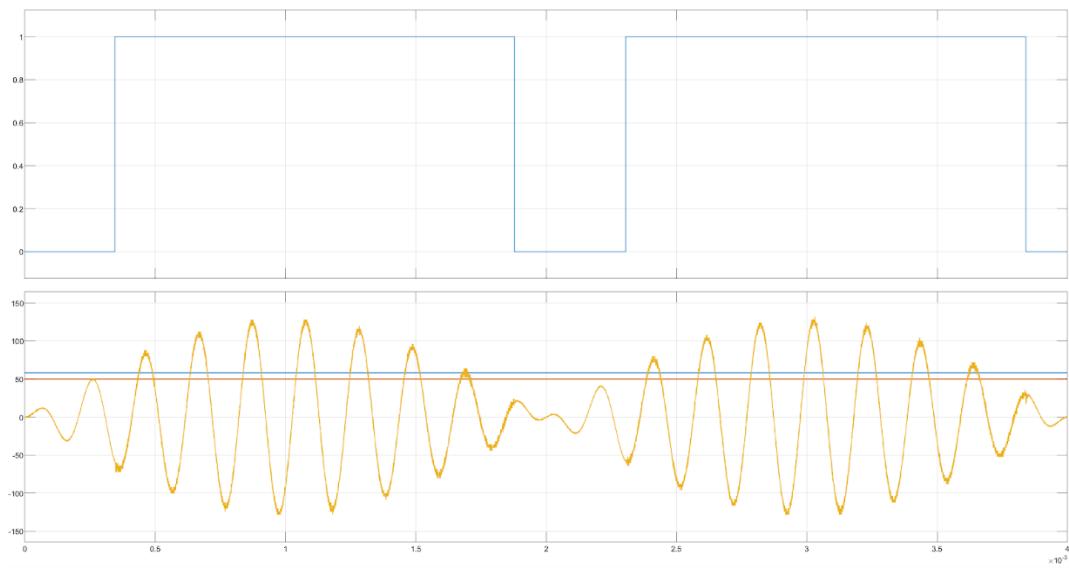


Figura 30: Simulación de selección de camino para señal con cambios de amplitud (alpha arriba, TH_HI, TH_LO y señal abajo)

Durante toda esta sección se han utilizado los términos “grande” y “pequeña” para referirse a la señal de entrada. Esto es debido a que, para poder establecer unos umbrales de potencia adecuados, es necesario realizar experimentos con chips reales. Por ello, en este diseño se utilizan registros programables para establecer tanto los umbrales como el “timeout”, como se verá más adelante.

En base a las simulaciones anteriores, y otras similares realizadas, junto con la posibilidad de programar los umbrales y el “timeout”, se considera que el comportamiento de selección de camino es adecuado para optimizar el rango dinámico del convertidor realizando una combinación de caminos de forma fácilmente implementable, con un coste de área y consumo bajos.

3 – DISEÑO HARDWARE DEL SISTEMA

El objetivo de esta sección es analizar el sistema implementado, con el fin de lograr un comportamiento como el del sistema descrito en 2.1, “Diseño de alto nivel de un sistema de adquisición de audio con extensión de rango y 2 VCO”, utilizando el método de selección de caminos por estimadores de potencia descrito en el mismo apartado. Como se describe en la figura 13 de este apartado, el sistema consta de dos caminos, cada uno con una serie de bloques. (Osciladores (VCOs), Contador, Muestreo y Noise-Shaper). La implementación hardware de cada uno de estos bloques se encuentra descrita por apartados en esta sección. Si bien la implementación de la etapa analógica de entrada es tangencial a este TFM, la implementación ha nivel lógico del sistema completo si es parte fundamental del mismo

3.1 - Etapa analógica de entrada y osciladores

La entrada al sistema real procede de un micrófono MEMS [4], que debe ser adaptada para poder utilizarse en el convertidor. Dado que un oscilador en anillo se controla mediante corriente, es necesario añadir un transductor a su entrada, de forma que las variaciones de voltaje sean transformadas en variaciones de corriente. Para el canal HDR se utiliza un transductor con 4 veces menos ganancia que para el canal HSRN, para lograr un comportamiento como él de la figura 12 (Gráfica corriente vs frecuencia de osciladores con diferente ganancia). La salida de estos transductores se conectan a un oscilador en anillo diferencial de tipo “feed forward cross coupled” [26] de 16 etapas, como los explicados en la introducción. Se seleccionó este oscilador en el proyecto debido a que es ventajoso un número de fases par, y los osciladores diferenciales FFCC presentan ventajas en cuanto a ganancia frente a los CC.

No debe confundirse el carácter diferencial de los osciladores con la idea de que el sistema utiliza una señal diferencial. Cada uno de los osciladores de la figura

13, donde se especifica el diseño a nivel de sistema, se implementa como un oscilador diferencial FFCC, pero ambos forman parte de un sistema diferencial mayor, que utiliza como entrada una señal diferencial, y es este el que corresponde al bloque del diagrama.

3.2 - Contador

Un requisito fundamental para diseñar un VCO-ADC es muestrear la fase del oscilador [1]. En el caso de este chip, la fase del VCO se muestrea y digitaliza mediante código Gray [27]. El código gray se utiliza en lugar de binario convencional por su interesante propiedad de que solamente es necesario cambiar de estado un bit al sumar o restar uno a cualquier número. Como es posible que se realice un muestreo en un cambio de estado del contador, y por tanto muestrear un valor indefinido, con binario convencional se podría cometer un error muy grande. Por ejemplo, si el contador está cambiando del estado 0111 al estado 1000 (7 a 8), todos los bits muestreados tendrían un valor indefinido, pudiendo producirse grandes errores. Utilizando código Gray, por la propiedad anteriormente mencionada, solamente se puede cometerse un error de +1 LSB. Para el ejemplo anterior, 7 es 1100 en código Gray, y 8 es 0100, por lo que únicamente el primer bit podría haberse muestreado de forma incorrecta. En cualquiera de los casos, el valor tras muestrear sería 7 u 8, habiendo cometido un error pequeño, de un LSB. Este patrón se repite para todas las combinaciones posibles. Si bien esta codificación no es apta para realizar operaciones aritméticas, basta con añadir un decodificador de código Gray a binario convencional para disfrutar de ambas propiedades.

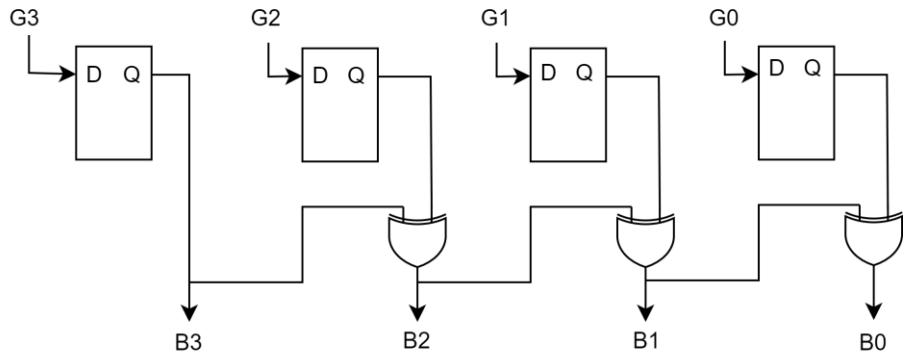


Figura 31: Decodificador de código gray a binario convencional de 4 bits

Es posible que el lector se pregunte como se implementa un contador Gray partiendo de un oscilador en anillo. Para clarificar esto, nombremos cada una de las fases de un oscilador en anillo de 16 etapas como $\phi_0 \dots \phi_{15}$. Si bien en el siguiente diagrama cada celda se dibuja como un oscilador, en el diseño real se trata de las fases positivas de un oscilador en anillo FFCC como el descrito en el apartado anterior. No obstante, esto no tiene ninguna relevancia en la explicación, al margen de solucionar una posible duda sobre la paridad de las fases que pueda surgir.

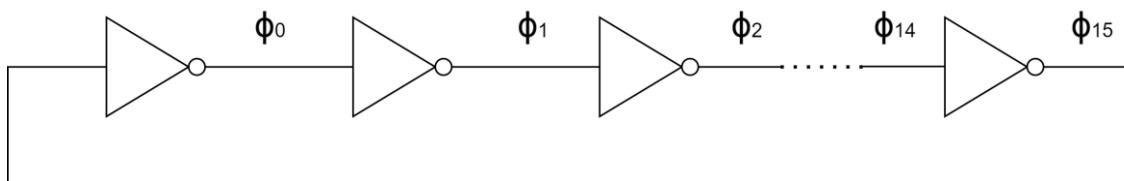


Figura 32: Numeración de las fases (etapas) de un oscilador en anillo de 16 etapas

Basta con establecer:

$$G0 = \phi_1 \oplus \phi_3 \oplus \phi_5 \oplus \phi_7 \oplus \phi_9 \oplus \phi_{11} \oplus \phi_{13} \oplus \phi_{15}$$

$$G1 = \phi_2 \oplus \phi_6 \oplus \phi_{10} \oplus \phi_{14}$$

$$G2 = \phi_4 \oplus \phi_{12}$$

$$G3 = \phi_8$$

$$G4 = \phi_0$$

para obtener un contador de salida ascendente con bits Counter[4..0] = [G4,G3,G2,G1,G0]. Para entender el porqué de estas operaciones, se puede pensar en un oscilador en anillo como una “inestabilidad” que se propaga a lo largo del círculo de inversores (Ver figura 4, estados de un oscilador en anillo de 3 inversores). Solamente una de las fases ϕ n cambia de estado al mismo tiempo, la que se encuentra en dicho estado inestable. Con ello en mente, el operador XOR se utilizan debido a una interesante propiedad. Si se cambia el valor de entrada de una y solamente una de variables en una operación XOR como $(a \oplus b \oplus c \dots \oplus n)$, el resultado siempre se invierte. Dicho de otra manera, la operación XOR de un vector binario $[a, b, c \dots, n]$ es la paridad de dicho vector.

Es posible utilizar esta propiedad para generar cualquier secuencia binaria a partir de un oscilador en anillo. Para hacerlo, es necesario realizar una operación del tipo $\phi_a \oplus \phi_b \dots \oplus \phi_n$, donde a,b .. n son las fases o enlaces entre inversores que corresponden con un cambio de estado en la secuencia binaria a generar. En el caso del código gray, estas fases aparecen marcadas en verde en la siguiente figura, y por ello son las fases seleccionadas en la operación anterior. (Nota: Es posible que el resultado de aplicar esta técnica sea el inverso bit a bit a la secuencia esperada. En tal caso, basta con añadir un inversor a la salida. No ocurre esto con el código Gray. También es posible que sea necesaria multiplexación dependiendo del valor de la fase ϕ_0 . Tampoco es el caso con el código Gray, gracias a su simetría)

Value	G4	G3	G2	G1	G0	ϕ_n
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	0	0	1	1	2
3	0	0	0	1	0	3
4	0	0	1	1	0	4
5	0	0	1	1	1	5
6	0	0	1	0	1	6
7	0	0	1	0	0	7
8	0	1	1	0	0	8
9	0	1	1	0	1	9
10	0	1	1	1	1	10

11	0	1	1	1	0	11
12	0	1	0	1	0	12
13	0	1	0	1	1	13
14	0	1	0	0	1	14
15	0	1	0	0	0	15
16	1	1	0	0	0	0
17	1	1	0	0	1	1
18	1	1	0	1	1	2
19	1	1	0	1	0	3
20	1	1	1	1	0	4
21	1	1	1	1	1	5
22	1	1	1	0	1	6
23	1	1	1	0	0	7
24	1	0	1	0	0	8
25	1	0	1	0	1	9
26	1	0	1	1	1	10
27	1	0	1	1	0	11
28	1	0	0	1	0	12
29	1	0	0	1	1	13
30	1	0	0	0	1	14
31	1	0	0	0	0	15

Figura 33: Código Gray de 5 bits con cambios indicados en verde

En el anexo se encuentra una figura con los 32 estados de un oscilador en anillo de 16 etapas, si se desea explorar más en profundidad esta técnica, que se basa en [36]

Una vez realizado este proceso, se dispone de la fase del oscilador en anillo digitalizada en un número de 5 bits codificado en Gray, y tras utilizar un conversor de Gray a binario convencional como el de la figura 31, se dispone de este número en el formato clásico requerido por los circuitos aritméticos eficientes. Para ello tan solo han sido necesarias algunas puertas XOR y un registro de muestreo.

Aunque ya se haya digitalizado la fase del oscilador en un número binario de 5 bits, para el resto del sistema es conveniente utilizar números de 9 bits, tal y como se ha explicado en el diseño del sistema. Para lograrlo, se realiza una extensión utilizando un contador binario adicional tras el muestreo. Así, el bit más significativo de la salida del muestreo se conecta con la entrada de un contador

binario asíncrono de 4 bits, de tal forma que se consiguen los 9 bits objetivo. El diagrama completo de este subsistema, que es el que finalmente se integra en el chip, se muestra a continuación:

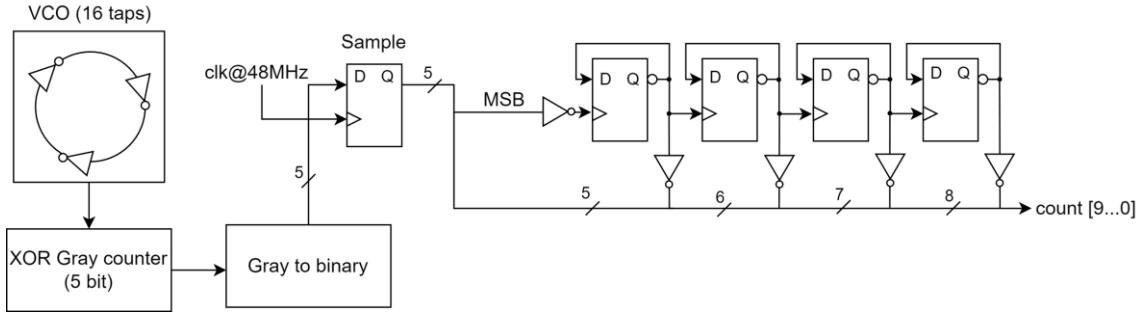


Figura 34: Oscilador en anillo, contador gray, muestreo y extensión a 9 bits

Como curiosidad, dado que se utiliza una extensión en el contador, la frecuencia mínima de muestreo debe ser el doble de la frecuencia máxima del oscilador. Esto es debido a que una transición en el contador binario asíncrono de extensión se produce cuando el MSB del valor muestreado pasa de ser 1 a ser 0. Por ejemplo, si la muestra $n-1$ es 11100, y la muestra n es 00001, se detecta que el contador gray ha dado una vuelta, y por tanto se añade una unidad al contador extendido correctamente. Pero si la muestra $n-1$ es 11111, y la muestra n es 10000, entonces no cambiaría el valor del contador extendido. Por tanto, la frecuencia mínima de muestreo debe ser tal que el contador Gray pueda dar la mitad de una vuelta como máximo, es decir, el doble de la frecuencia del oscilador. Es por esto que se estableció en el diseño a nivel de sistema que la frecuencia mínima de muestreo es 24 MHz, y no 12 MHz como cabría a esperar (frecuencia máxima del oscilador).

3.3 - Noise shaper

Como se ha especificado anteriormente, la frecuencia de salida del convertidor ha de ser de 3.072 MHz [9], y para reducir el ruido de cuantización debido al remuestreo, se utiliza un elemento de noise-shaping completamente digital similar al utilizado en los DACs sigma-delta.

La implementación del noise-shaper se realiza de la siguiente manera, que es una traducción elemento por elemento a hardware del sistema descrito en la figura 7 de la introducción:

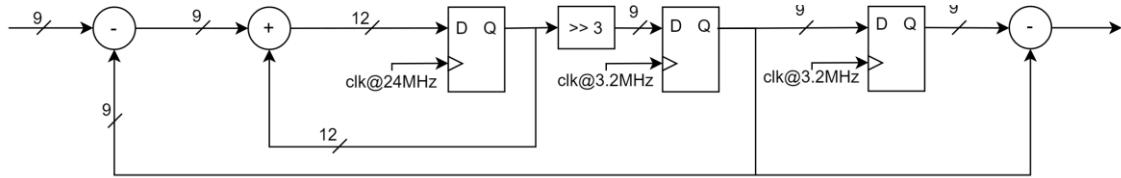


Figura 35: Implementación del noise-shaper

La única diferencia entre ese sistema y esta implementación es la precisión. Es imposible utilizar una cantidad ilimitada de bits en el acumulador interno, por lo que se utiliza la cantidad de bits mínima para que no se produzcan overflows entre dos muestreos a 3.072 MHz.

Esta cantidad resulta en 3 bits más que los circuitos que funcionan a 3.072Mhz, ya que el acumulador puede, como máximo, multiplicar su contenido por 8 ($24.57/3.072 = 8$), como se ha explicado en el apartado de diseño del sistema. Utilizar esta cantidad de bits, no obstante, reduce la precisión, ya que en la realimentación solo se utilizan los 9 MSB de los 12 bits del acumulador.

3.4 - Comparador de umbral y selector de camino

Para conseguir el comportamiento descrito en el apartado 2.3, se propone el siguiente circuito, que es el principal objeto de este TFM:

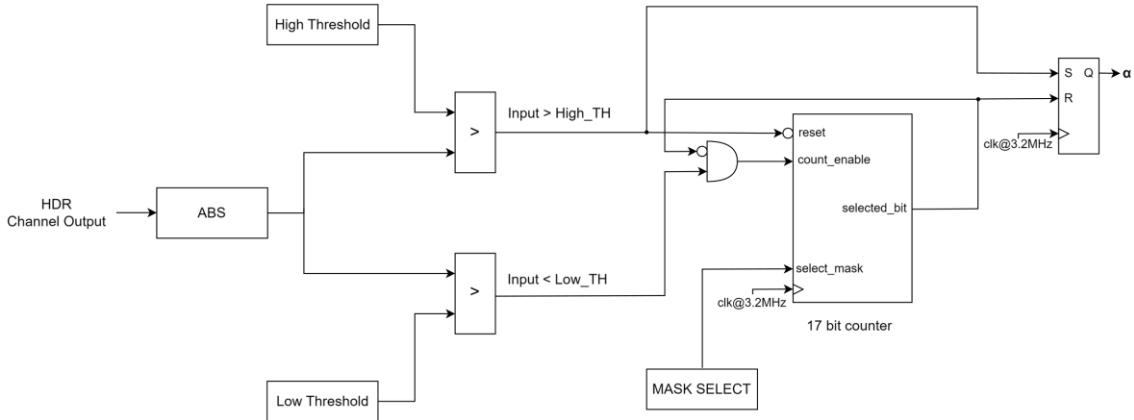


Figura 36: Comparador de umbral, bloque a

El circuito compara el valor absoluto de la entrada con los dos umbrales. En caso de que dicho valor se encuentre por encima del umbral superior, el flip-flop de salida es se pone a ‘1’ incondicionalmente. Además, se realiza un reset del contador de forma asíncrona. En caso de que el valor caiga por debajo del umbral inferior, el contador cuenta hasta alcanzar un valor programable mediante el registro MASK_SELECT. Este valor puede ser desde 2^{13} hasta 2^{17} . Estos valores han sido elegidos para que, utilizando un reloj de 3.072 MHz, el “timeout” pueda encontrarse entre 0.04 s (24 Hz) y 0.0025 s (390 Hz). Cuando se alcanza este valor, el contador deja de contar y se activa el canal HSNR. En caso de que el valor se encuentre entre ambos umbrales, el circuito simplemente mantiene su estado. Se ha escogido este rango de valores de “timeout” con el fin de probar en el laboratorio, una vez fabricado el chip, si se producen artefactos no deseados al realizar cambios de canal demasiado rápido. El más lento de los posibles valores es 0.04 s, que se encuentra en la frontera de las frecuencias subsónicas, por lo que en principio, no debería suponer ningún problema y proporciona una opción conservadora. Por otro lado, 390 Hz permitirá determinar si realmente una frecuencia elevada de cambios de canal produce artefactos audibles.

Para implementar este circuito de forma eficiente, ha sido necesario diseñar el bloque de que realiza la operación de valor absoluto, los comparadores y el contador.

El bloque para calcular el valor absoluto sigue la siguiente estructura:

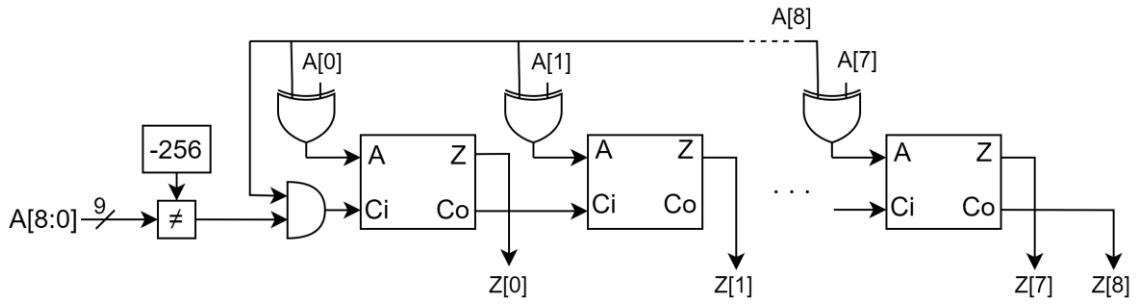


Figura 37: Bloque para valor absoluto

Las puertas XOR actúan como NOT controlada. De esta forma, si $A[8]$ es igual a 1 se invierte todo A. En ese caso, y siempre que el valor de entrada sea diferente a -256, se suma uno al valor obtenido, habiendo completado la operación de valor absoluto para complemento a dos. Esta forma de calcular el valor absoluto es más eficiente en términos de área y consumo que utilizar un restador y un multiplexor.

El output de este circuito es el valor absoluto de la entrada excepto cuando esta tiene el valor de -256. En este caso, la salida es 255. Esto se hace para poder mantener una salida de 9 bits, ya que para representar +256 sería necesario otro bit extra. Añadir este bit significaría tener que complicar el hardware de los comparadores, y no existe apenas ninguna diferencia a efectos prácticos entre utilizar 255 y 256 en este caso.

Pasando a los comparadores de umbral, estos se implementan como restadores de 10 bits en complemento a dos que solamente producen como resultado el bit de signo. Es necesario realizar una extensión en signo de 1 bit, ya que el valor de la resta de dos números de 9 bits necesita 10 bits para ser representado.

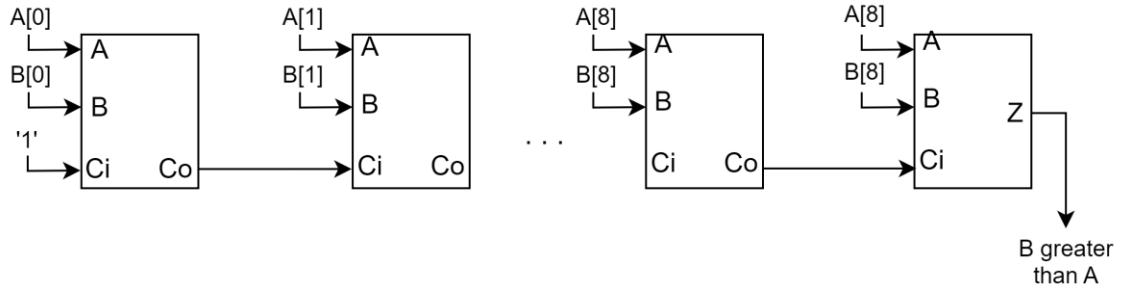


Figura 38: Comparador mediante restador en Ca2

Por último, el contador para establecer el “timeout” se realiza con la siguiente estructura:

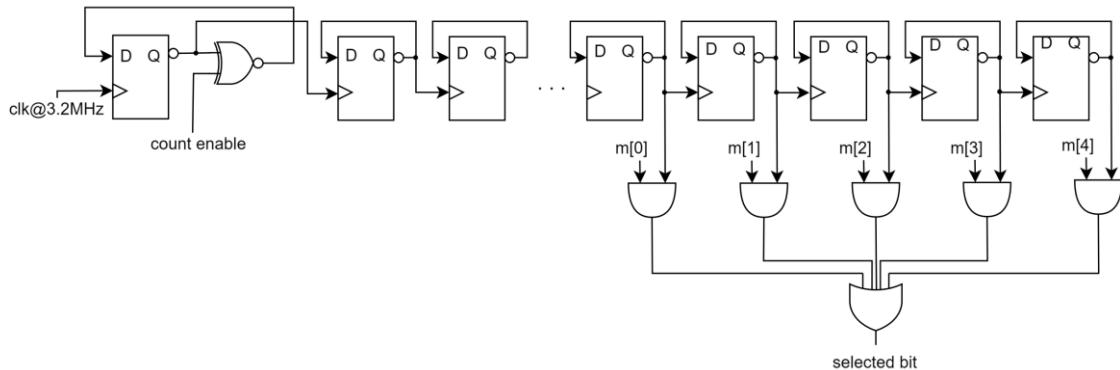


Figura 39: Contador con selección de salida

Como en otros componentes de este chip, el contador es asíncrono debido a que tiene un consumo mucho menor que su equivalente síncrono, además de ocupar un área menor y ser más fácil de implementar a nivel de “layout”. Dado que la frecuencia del contador es de 3.072 MHz, esto no supone ningún problema. Para contadores con frecuencias más altas, sería necesario utilizar estructuras síncronas. La función “count enable” se implementa con una simple puerta XNOR a la entrada del primer flip-flop. Otra opción de implementación hubiera sido utilizar “clock gating” [29] en el primer flip-flop, es decir, realizar la operación AND entre “count enable” y el reloj. Lo que resultaría más eficiente en términos de consumo. Sin embargo, podría ocasionar problemas en caso de que el flanco de reloj llegase antes que la señal de control a la entrada, o que la función “count

enable” tenga resultados de computación intermedios que produzcan falsos flancos de subida. Utilizar OR a la entrada y mantener la señal de reloj a ‘1’ tampoco solucionaría el problema. Para evitar este problema, se ha decidido optar por la opción más segura, a pesar de un ligero aumento de consumo.

La función “count enable (Ce)” se encuentra descrita en la siguiente tabla de verdad

$Q(n)$	Ce	$Q(n + 1)$	$D(n)$	$\overline{Q(n)}$	$(\overline{Q(n)} \oplus Ce)$
0	0	0	0	1	0
0	1	1	1	1	1
1	0	1	1	0	1
1	1	0	0	0	0

Tabla 2: Función count enable

Donde se puede ver que efectivamente $(\overline{Q(n)} \oplus Ce)$ es la operación necesaria para el comportamiento deseado.

El problema de utilizar “clock gating” en lugar de una función síncrona se exemplifica en el siguiente cronograma

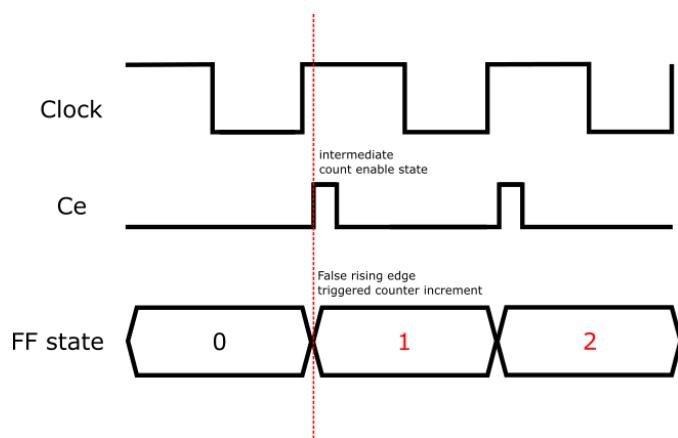


Figura 40: problemas del “clock gating”

Como se puede ver en la figura, un resultado intermedio de “count enable”, que es calculado de forma síncrona, produce un falso flanco de subida de reloj, que puede llevar al contador a un estado incorrecto. Una posible solución para este problema es utilizar flancos de bajada para activar el contador y flancos de subida para calcular “count enable”, pero es algo no deseable puesto que se dispondría solo de la mitad del tiempo para la computación de todo aquello que dependa del valor del contador. Dado que el contador utilizado es asíncrono, apenas tiene impacto en cualquier caso.

Por otro lado, la lógica de selección elige la salida del contador, asignando el bit 13,14,15,16 o 17 en función de la máscara.

El reset no se muestra en el diagrama, pero es activo a nivel bajo y asíncrono.

3.5 - Combinador de salida progresivo

Uno de los posibles problemas de este diseño de convertidor, como se ha expuesto, es que se produzcan artefactos al cambiar del canal HSNR al HDR y viceversa. Una posible solución a este problema es combinar las salidas de los dos convertidores de manera progresiva, a pesar de utilizar una selección de camino binaria. De esta manera, cuando se produzca un cambio, se puede cambiar suavemente de uno al otro. Para que el cambio del canal HDR al canal HSNR se produzca de forma progresiva, hay que hacer que

$$Z(n) = \alpha(n) \cdot HSNR(n) + (1 - \alpha(n)) \cdot HDR(n) \quad (18)$$

Donde $Z(n)$ es la salida del convertidor, y $\alpha(n)$ es una progresión que va desde 0 hasta 1 de forma más o menos lineal. Como el bloque produce un α de ‘1’ o ‘0’, es necesario desarrollar un circuito que construya una progresión de este tipo cuando una entrada binaria cambia su valor. La respuesta inmediata es un contador de N bits, similar al anterior, produciendo 2^N valores en la progresión y deteniendo el conteo cuando llega a su valor máximo. Si estos valores son interpretados como números decimales, se consigue una progresión desde 0 hasta 1.

Para simplificar la implementación del combinador, es posible reescribir (18) como:

$$Z(n) = \alpha(n) \cdot (HSNR(n) - HDR(n)) + HDR(n) \quad (19)$$

De esta forma, una posible implementación es la siguiente:

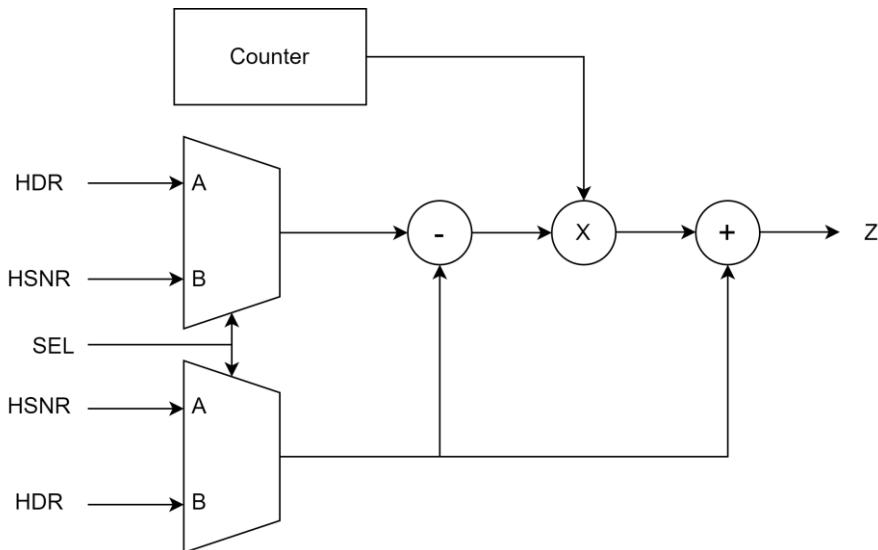


Figura 41: Combinador de salida progresivo (Datapath)

SEL debe tomar el valor de 1 cuando se está realizando un cambio del canal **HDR** al canal **HSNR**, y 0 cuando sea al revés.

Si se quiere realizar una implementación más sencilla en hardware, es posible sustituir el multiplicador por un barrel-shifter [30], utilizando potencias de dos generadas por un registro de desplazamiento como secuencia en lugar de un contador. En este caso, α toma los valores 0.125, 0.25, 0.5, 1 (para un registro de desplazamiento de 4 bits)

Se ha observado mediante simulación que la diferencia entre realizar un cambio progresivo entre los canales a utilizar un valor binario de α es prácticamente inexistente cuando las ganancias de los osciladores mantienen una relación correcta, como se ve en la siguiente simulación:

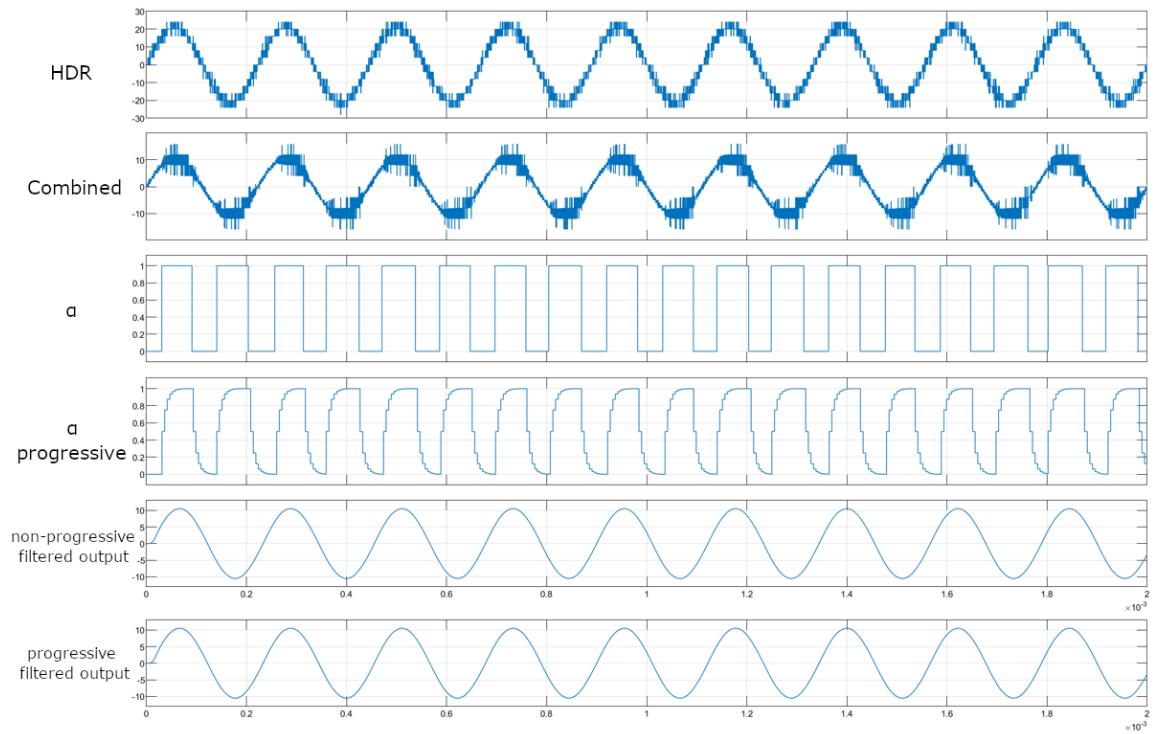


Figura 42: Simulación de combinador de salida progresivo frente a no progresivo

El combinador progresivo no supondría pues una gran ventaja, pero consumiría energía y área de manera innecesaria, por lo que no se implementa en este chip.

3.6 - Modelo bit-true completo - simulación del sistema bit-true

Con el fin de servir como referencia para la implementación de la parte digital del chip en Cadence Virtuoso [25], se ha desarrollado un modelo a nivel de puerta lógica del mismo en Matlab-Simulink [20]. Este modelo se encuentra en el anexo “Modelos de Matlab/Simulink: Bit-true”, y utiliza los circuitos descritos en la sección anterior, por lo que no se procederá a su explicación bloque por bloque.

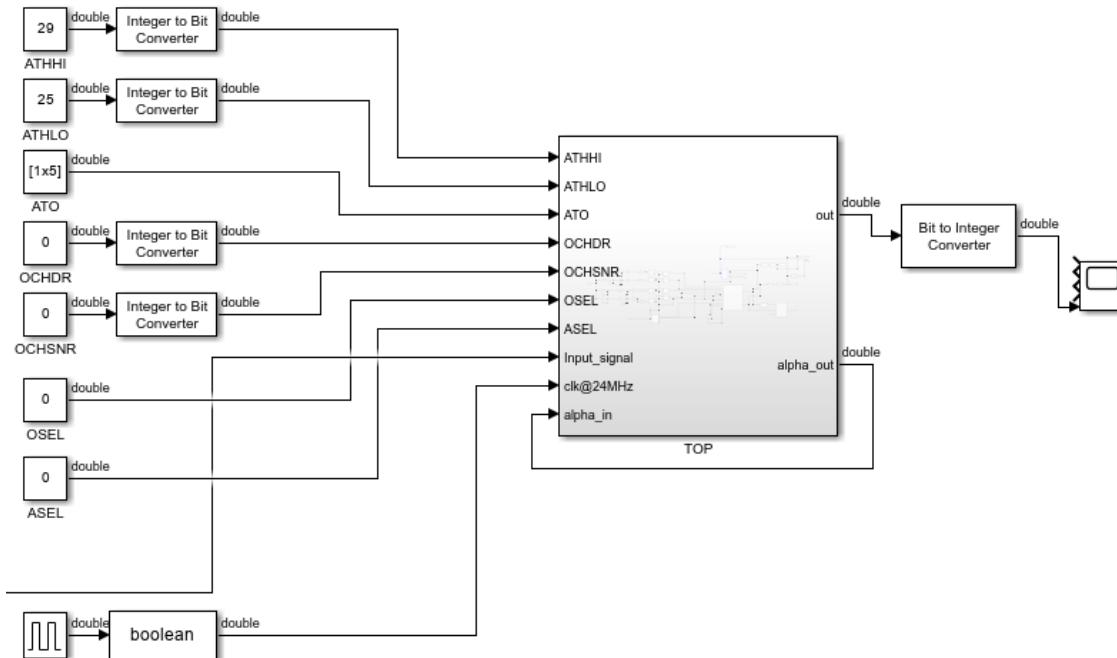


Figura 43: Vista global del modelo bit-true (representación lógica del chip)

Gracias a este modelo, se puede comprobar el comportamiento del chip, tanto los SNR de los dos canales como el cambio de canal y el funcionamiento del bloque generador de α .

Utilizando una señal senoidal de fondo de escala, podemos obtener la SQNR (Relación señal a ruido de cuantización) del convertidor implementado en el modelo bit-true. Esta relación no tiene en cuenta el ruido introducido por los osciladores, solamente el ruido producido por el muestreo y remuestreo. La SQNR resulta en 107.2414 dB para una señal de fondo de escala. El noise-shaping del

convertidor puede observarse claramente en la siguiente imagen, producto de aplicar la transformada discreta de Fourier a la salida del convertidor (Eje x: frecuencias de la señal de salida (Hz), Eje y: Potencia (dB)):

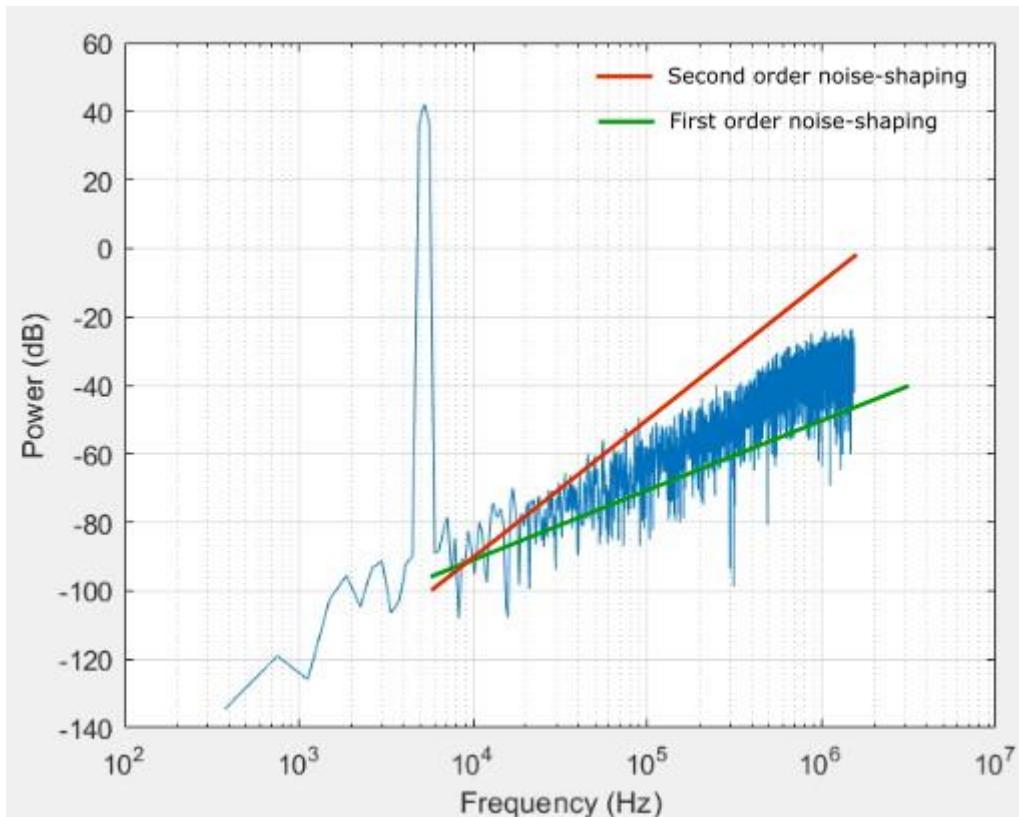


Figura 44: Noise-shaping del convertidor (modelo bit-true)

El pico se corresponde con la señal de entrada, de 5Khz. Se observa noise-shaping superior a primer orden, lo que equivale a un aumento de potencia de ruido de 20 dB por década. (Ver la función de transferencia del noise-shaping de primer orden de la sección de introducción y estado del arte, Figura 2). Debido a que se está aplicando noise-shaping dos veces, una primera vez utilizando el noise-shaper, y una segunda con la primera diferencia, la curva frecuencial del ruido no es completamente recta en escala semilogarítmica, sino que se observa una pendiente mayor hacia las altas frecuencias, y no se corresponde exactamente ni con un primer orden ni con un segundo orden.

Para cerciorarse de la validez de la implementación se han realizado simulaciones temporales para diferentes entradas y se han comparado con el modelo integer-true, además de comprobar que se obtiene una SNR similar para entradas senoidales puras. Una de estas simulaciones, para entrada senoidal, se muestra en la siguiente, previo a la selección de caminos, siendo el canal superior el HDR y el inferior el HSNR:

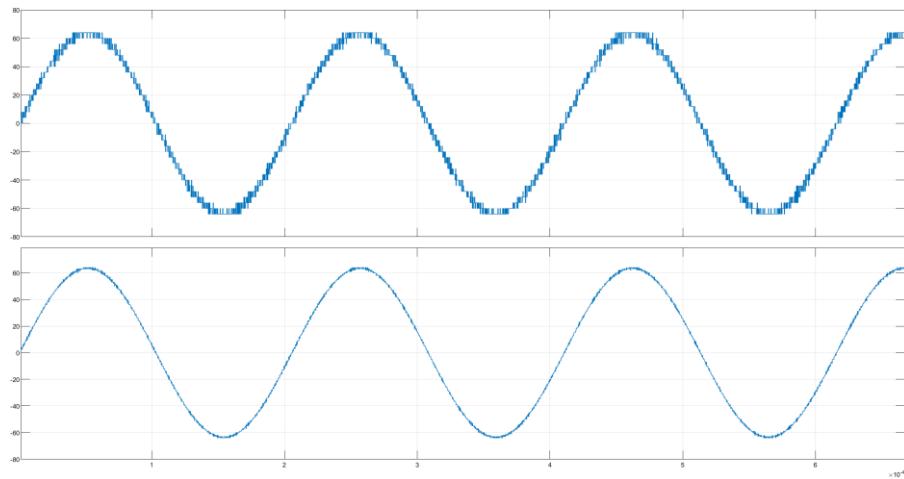


Figura 45: Simulación temporal modelo bit-true (HDR arriba, HSNR abajo)

Las simulaciones del bloque de combinación de caminos del modelo bit-true se mostraron en el apartado 2.3, y por ello no se repiten aquí.

Una vez el resultado de las simulaciones tanto temporales como frecuenciales del modelo bit-true se correspondieron con las simulaciones realizadas en el modelo integer-true, y se comprobó el buen funcionamiento del bloque de combinación de caminos, se tomó como referencia el modelo bit-true para su implementación a nivel de transistor en Cadence Virtuoso.

3.7 - Diseño a nivel de transistor del bloque implementado

Una parte fundamental de este TFM es la implementación y verificación del bloque α en CMOS para su integración en el chip que se corresponde con el modelo bit-true anteriormente descrito. Para ello, se ha empleado cadence virtuoso, primero para diseñar a nivel de transistor tomando como referencia el modelo bit-true, y para posteriormente la implementación de layouts y realización de simulaciones de verificación.

La tecnología empleada en el desarrollo de este chip es CMOS 130 nm [31]. Todos los bloques implementados a nivel de transistor del chip, incluyendo el bloque objetivo del TFM, es decir, el bloque α , se realizan con este proceso.

Se dispone también de un PDK [32] (Process Design Kit) que incluye los modelos necesarios para simular el comportamiento de los transistores reales tras la fabricación. También incluye una gran cantidad de celdas estándar digitales, que permiten agilizar el proceso de implementación, haciendo que la transformación del modelo bit-true (nivel de puerta lógica) a la implementación a nivel de transistor sea casi automática, al disponer de los bloques digitales de construcción básicos (puertas lógicas, biestables y multiplexores) para ello. Gracias a la metodología de diseño utilizada, los circuitos implementados son prácticamente iguales a los expuestos en la parte de diseño hardware, por lo que solamente se adjunta la vista de esquemático de más alto nivel.

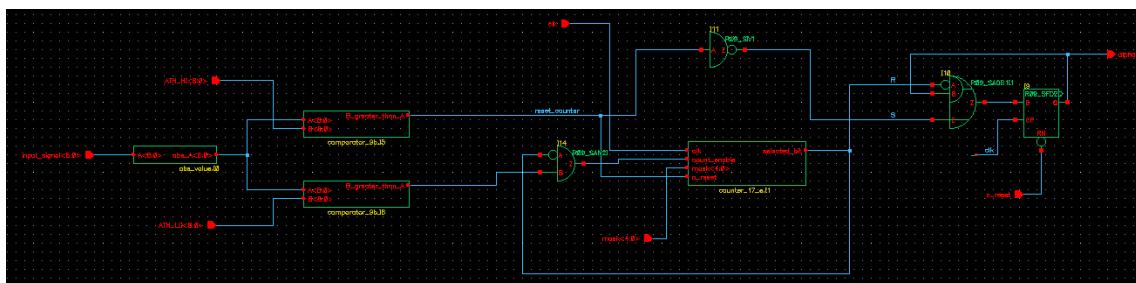


Figura 46: Implementación del bloque α a nivel de transistor

En el anexo pueden encontrarse capturas de pantalla de los diferentes subbloques implementados, que se corresponden con los circuitos expuestos en el apartado 3: “Diseño Hardware del sistema”.

4 – LAYOUT

Un objetivo importante del TFM es la implementación completa del bloque a, desde nivel de sistema hasta el desarrollo de un layout fabricable y funcional.

El bloque a se integra dentro del sistema descrito en apartados anteriores, que ha sido implementado completamente a nivel de layout para su fabricación. La integración del layout de los diferentes bloques, realizada por Javier Granizo Cuadrado, da lugar al layout del chip completo fabricable en 130 nm.

4.1 Layout del chip completo

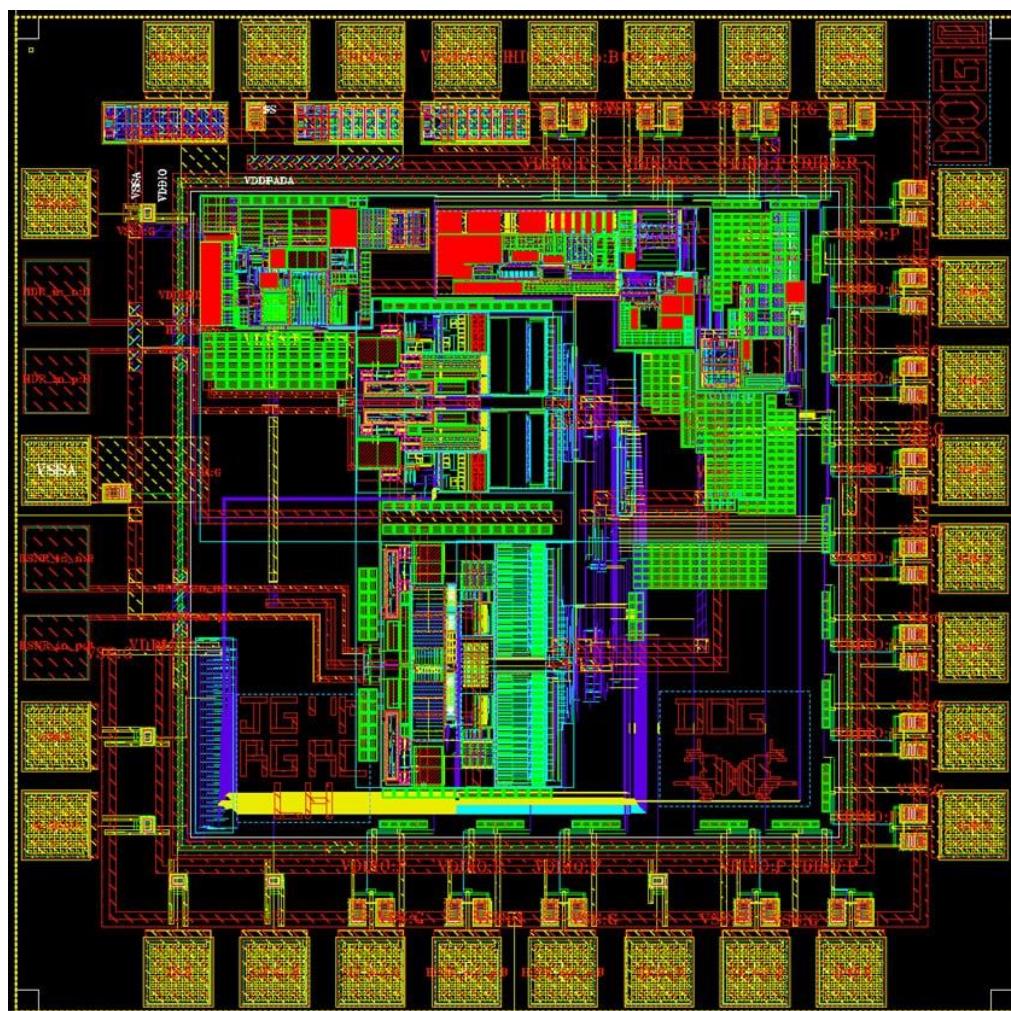


Figura 47: Layout del chip completo

El chip completo desarrollado del que forman parte los componentes de este Trabajo de Fin de Máster se muestra en la imagen superior.

Las diferentes partes del chip se encuentran marcadas en la siguiente imagen:

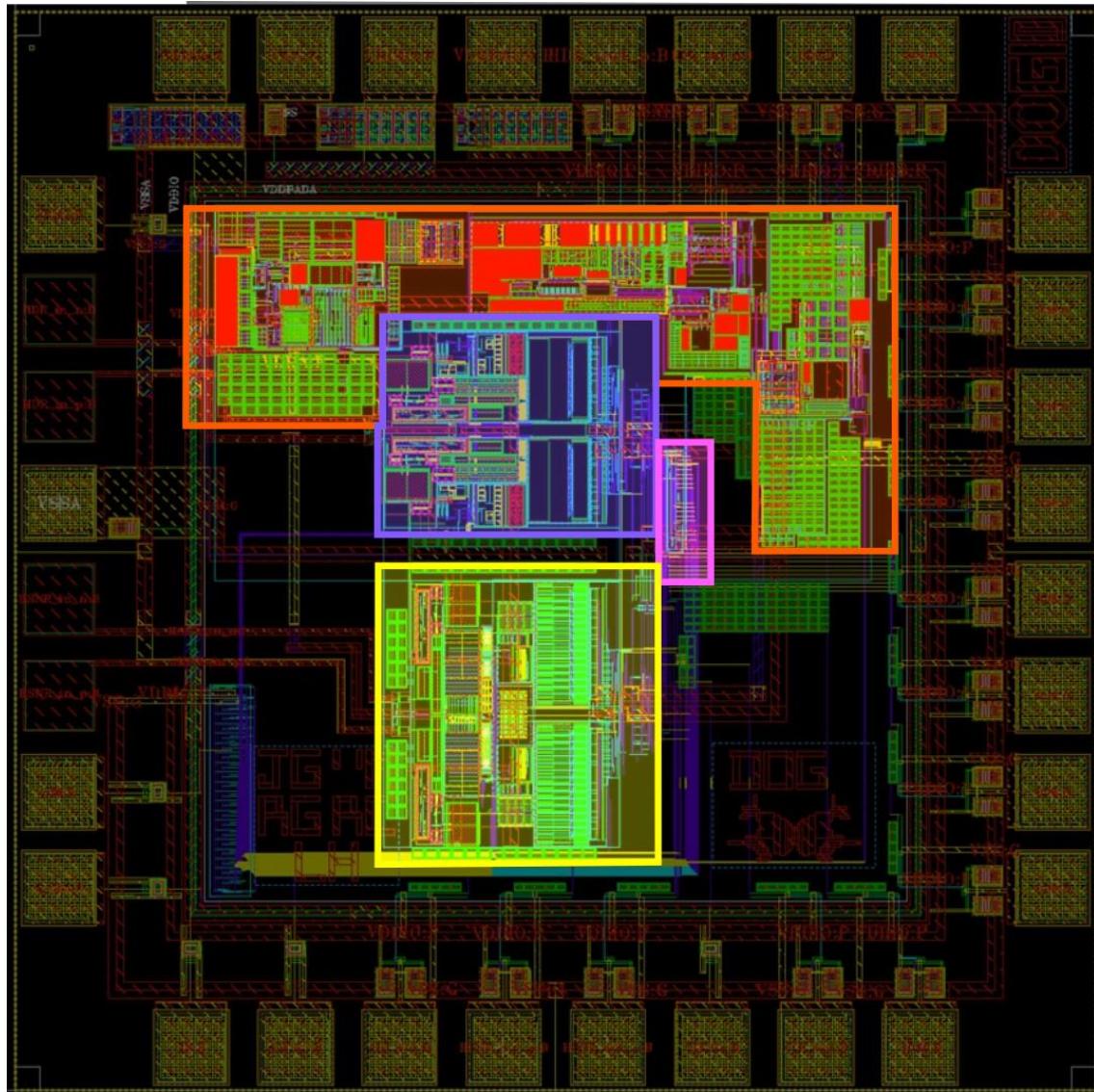


Figura 48: Diferentes bloques del chip completo en layout

En naranja se encuentra marcada la parte de potencia del chip. Esta incluye un regulador de voltaje (LDO), y condensadores de desacoplo. El canal HDR se encuentra marcado en morado, y el canal HSNR en amarillo. El bloque a está marcado en rosa, y se encuentra a la derecha de los canales HDR y HSNR.

Resulta interesante comprobar en este layout la gran diferencia de área entre los componentes analógicos y los componentes digitales, poniendo de manifiesto las

ventajas que se han mencionado de los VCO-ADCs implementados con circuitos digitales. De toda el área ocupada por el convertidor propiamente dicho, es decir, excluyendo la parte de potencia, prácticamente toda está consumida por los osciladores, ocupando el bloque a, los contadores, noise-shapers y muestreadores un porcentaje muy pequeño de la misma.

4.2 - Layout del circuito objeto del TFM

La implementación del bloque a se ha realizado mediante el uso de celdas estándar, pero, al no disponer de una herramienta de síntesis y place & route, este se ha realizado completamente a mano. Para implementar este layout se dispone de un paquete de celdas estándar y cuatro niveles de metal, siendo preferible utilizar los de nivel más bajo siempre que sea posible.

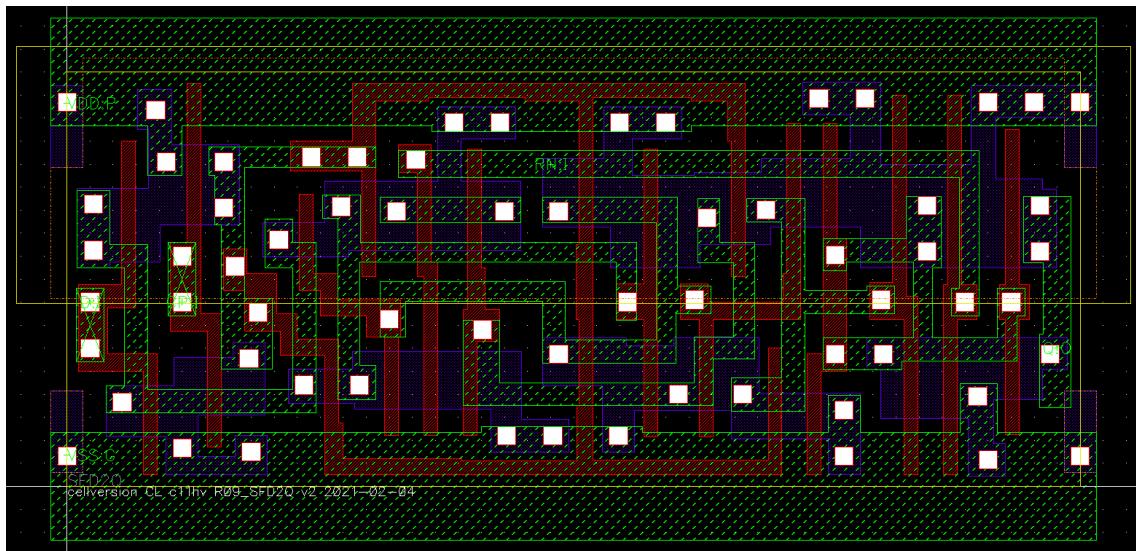


Figura 49: Celda estándar flip-flop tipo D

El espacio que ocupa el layout del bloque a es de 106 x 16.28 um, resultando en un área bastante compacta para la cantidad de conexiones que se han tenido que realizar.

En los diagramas de implementación hardware puede verse que muchos circuitos tienen lazos (por ejemplo, el componente de valor absoluto debe calcular el primer acarreo a partir de todas las entradas). Estos lazos producen dificultades a

la hora de hacer un layout compacto, lo que ha supuesto un reto y ha complicado la implementación del mismo.

En primer lugar, se ha tenido que desarrollar el layout de los comparadores. Este es posiblemente el más sencillo de los cuatro componentes necesarios para el bloque a, que gracias a su estructura regular se ha podido diseñar con el siguiente patrón:

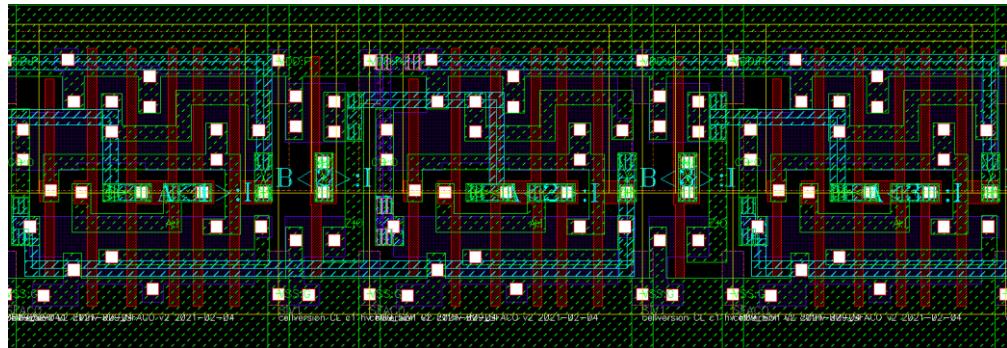


Figura 50: Estructura del layout de los comparadores

A continuación se ha implementado el layout del contador. Este layout se ha dividido en dos líneas de celdas estándar para evitar que sea demasiado largo. El resultado mide 100 x 9 uM, que se encuentra dentro de las especificaciones:

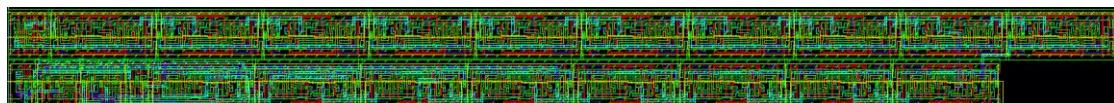


Figura 51: Layout del contador

Por último, se ha implementado el bloque de cálculo del valor absoluto, que es el que ha supuesto un reto mayor, a pesar de ocupar un área más pequeña:



Figura 52: Layout del bloque de valor absoluto

Finalmente, se ha realizado la interconexión de todos los componentes para disponer del bloque completamente implementado y funcional, siguiendo el siguiente “floorplan”:

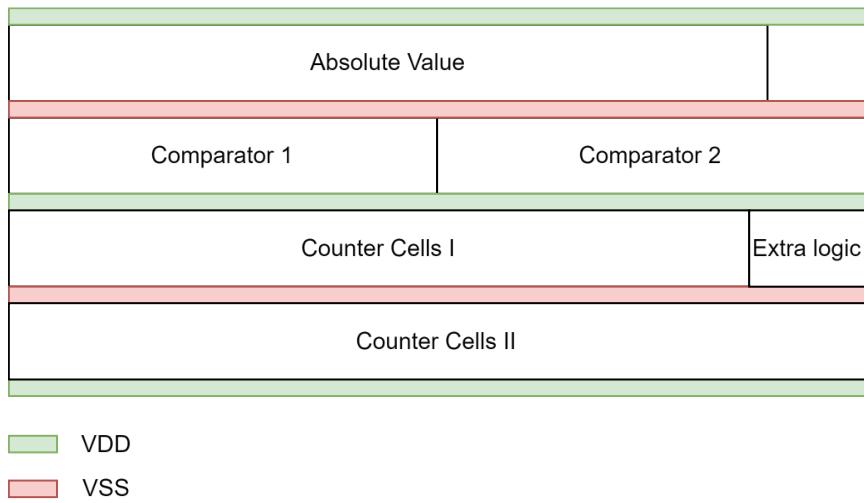


Figura 53: Floorplanning bloque a

El “floorplan” se ha diseñado de esta manera con la idea de tener una direccionalidad vertical en el avance de los datos. Las salidas del bloque de valor absoluto son utilizadas en exclusiva por los comparadores. El contador necesita a su vez los datos producidos por los comparadores, y la lógica de control debe interaccionar con ambos. Las celdas estándar se apilan de manera que comparten VDD y VSS entre ellas, algo que optimiza el área.

El siguiente layout ha sido implementado siguiendo el “floorplan” anterior

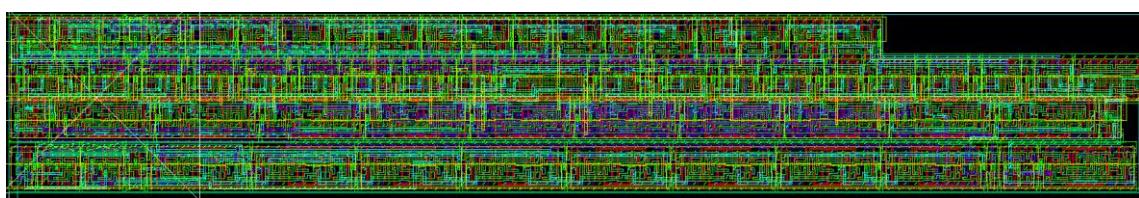


Figura 54: Layout del bloque a

Todas las imágenes de los layout se encuentran adjuntas en alta resolución en el anexo “Layouts”.

4.3 - Verificación del bloque implementado y estimación de potencias

La verificación es un proceso fundamental en el desarrollo de circuitos integrados para aplicaciones específicas (ASICs) [32]. En el caso de Cadence, se dispone del lenguaje de descripción hardware Verilog-A [33], que utiliza una sintaxis muy similar a Verilog [34], pero puede ser utilizado para generar señales analógicas. Para comprobar el correcto funcionamiento del bloque a, se ha desarrollado un test para cada componente, así como un test de funcionamiento general del bloque. En el caso de los componentes, se han realizado simulaciones a nivel de transistor, y para el bloque completo se ha realizado una simulación post-layout, que permite estimar el consumo más adecuadamente.

Para el bloque de cálculo de valor absoluto, simplemente se ha realizado una prueba con todos los valores posibles de entrada. Se ha comprobado en simulación que para todos los valores de entrada el resultado es, efectivamente, el resultado de aplicar la función valor absoluto, a excepción del valor -256, que tiene como salida 255. El bloque de valor absoluto es, por lo tanto, lo suficientemente sencillo para realizar una verificación formal completa, algo que rara vez es posible en este tipo de verificaciones. Una parte del resultado de este test se puede observar en la siguiente figura. El código para utilizado generarlo se encuentra en el anexo, en la sección “Códigos Verilog-A”, bajo el nombre “va_abs_val.va”

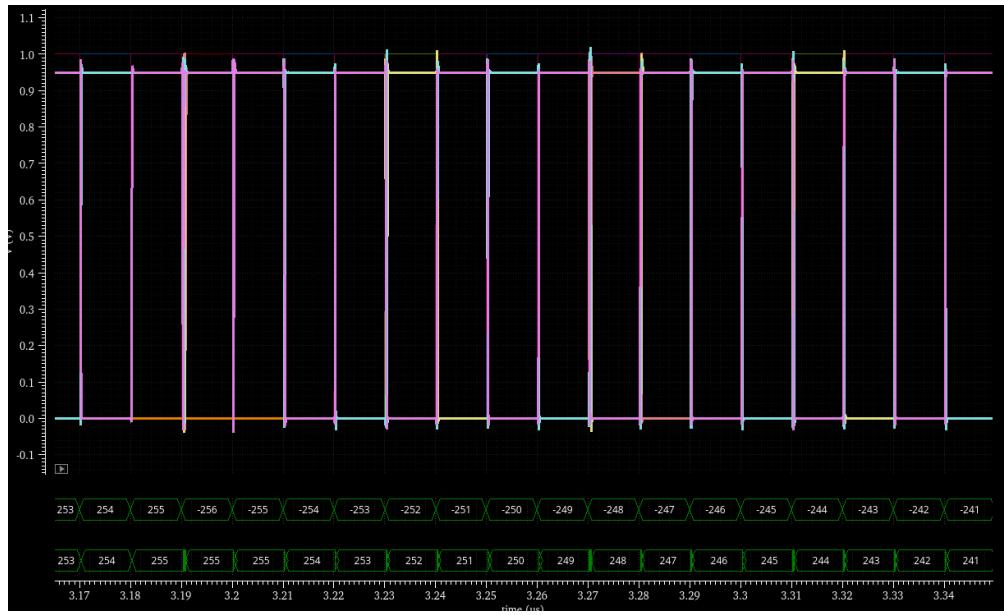


Figura 55: Simulación a nivel de transistor del bloque de valor absoluto

En el caso de los comparadores, se han generado una señal de conteo que se compara con valores fijos. Una parte de esta simulación de verificación se encuentra a continuación, y se adjunta en el anexo el código para producirla (va_comparator.va)

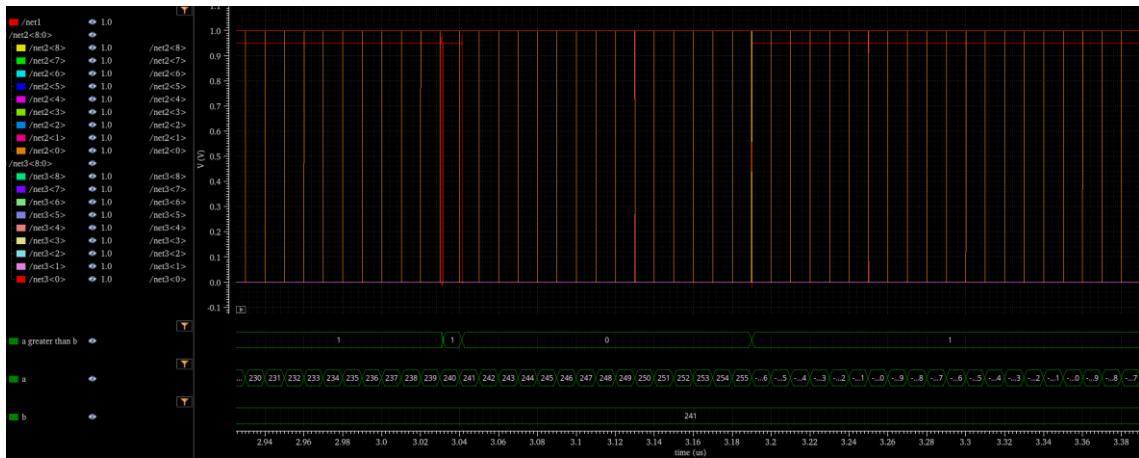


Figura 56: Simulación a nivel de transistor del comparador

Para la verificación del contador, se ha realizado una simulación que prueba el comportamiento del contador utilizando diferentes máscaras de selección de bit

de salida en el proceso. Este archivo se encuentra en el anexo “Códigos Verilog-A: va_counter”.

La verificación del comportamiento del bloque a completo es más compleja de realizar que las anteriores. A diferencia de Verilog, Verilog-A no permite generar señales digitales fácilmente, y tampoco la sintaxis “#n” para especificar un retraso de n segundos en una señal. Por tanto, ha sido necesario desarrollar un script en Verilog-A que permita leer señales especificadas en un archivo. Dicho archivo contiene un numero de N bits en cada línea, seguidos por un delay especificado en segundos. El script de Verilog-A lee línea por línea el archivo especificado, y genera un bus de N bits de salida con la palabra especificada. De esta forma, es posible generar señales digitales de una manera mucho más conveniente y sencilla. Un ejemplo del archivo de datos a partir del cual el script podría generar una señal digital es el siguiente:

```
0 0 1 0 0 0 1 1 0 0.00003
0 0 1 1 0 1 1 1 0 0.00003
0 0 1 1 1 1 1 0 1 0.00003
0 0 1 1 0 1 1 1 0 0.00003
0 0 0 0 1 0 1 0 0 0.0015
0 0 1 1 1 1 1 0 1 10.0
```

Figura 57: Estructura de archivos de test desarrollada

Las señales producidas por este archivo de la figura superior serían 0010001100 (30 us) 001101100 (30 us) 0011111010 (30 us) ... etc

El script responsable de esta funcionalidad se encuentra en el anexo con el nombre de `va_signals_from_file`. Para facilitar aún más el proceso, se ha desarrollado un script en Python que convierte un archivo similar al anterior, pero con números en base 10 en el equivalente que puede leer el archivo de Verilog-A, añadiendo retardos por defecto si no se especifican, adjunto como `converter.py` en el anexo “Códigos Python”.

Otra versión del script de Verilog-A permite ignorar los retardos y disparar un cambio con una señal de reloj externa. Este script se encuentra adjunto bajo el nombre `va_signals_from_file_clocked` en el anexo “Códigos Verilog-A”.

Disponiendo de estas herramientas, se ha podido realizar una verificación adecuada del bloque a. Para ello, se generan señales de entrada por arriba y por debajo de los valores umbral, lo que permite observar el comportamiento del contador. También se prueban todos los valores de máscara para observar el punto de parada del contador, así como el reset. El resultado (satisfactorio) de la simulación a nivel de transistor del bloque a es el siguiente:

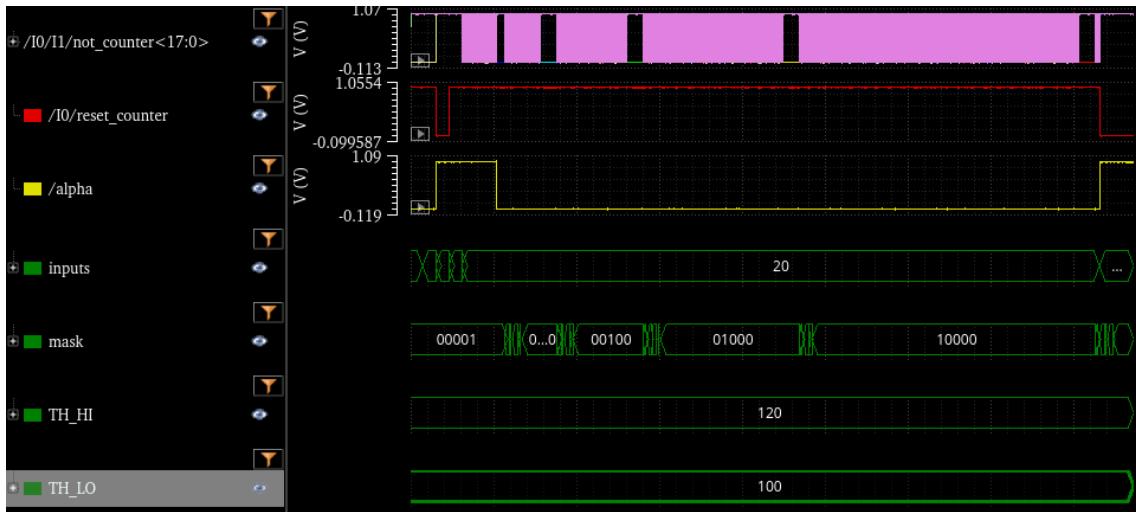


Figura 58: Resultado de la verificación del bloque a

En la simulación se aprecia el cambio del valor de α , los momentos en los que el contador está contando y los que no frente a todos los valores de la máscara de selección y diferentes valores de entrada. Con esta información puede comprobarse que el desempeño funcional es correcto.

Aunque simulaciones a nivel de transistor son importantes, es necesario realizar también simulaciones post-layout [35]. Estas simulaciones, además, permiten estimar el consumo del bloque implementado.

Para realizar una simulación de este tipo, se ha extraído un modelo capacitivo del layout del bloque a, y se ha procedido a realizar una simulación idéntica en forma a la anterior. Se utiliza un modelo capacitivo en lugar de un modelo RLC (Resistivo, Inductivo, Capacitivo), por tratarse de un circuito digital. El uso de los modelos RLC es común en la simulación de circuitos analógicos, donde el ruido tiene una importancia mayor.

Tras ejecutar la simulación, se ha comprobado que el resultado de la simulación post-layout es idéntica a la de nivel de comportamiento (transistor) en cuanto a comportamiento. Si bien los retardos aumentan ligeramente, estos se encuentran muy lejos de suponer un problema de timing para una frecuencia de 3.072 MHz como la utilizada en este chip. De hecho, para los casos peores de retardos, es decir, cuando cambia el bit de mayor peso del contador asíncrono, se ha comprobado que el circuito podría funcionar en torno a frecuencias de 150 MHz resultando en un amplio margen para la frecuencia objetivo. También aumenta ligeramente el consumo de corriente estimado, y por tanto el consumo.

La potencia varía mucho en este circuito según la señal de entrada, ya que el contador es el elemento que más consume, y si la señal de entrada no varía mucho en amplitud, su consumo es mínimo. Sin embargo, para una señal de entrada que este constantemente saltando del canal HDR al HSNR, el consumo será mayor.

Como estimación, la potencia media utilizada en la simulación post-layout es de 7.39 uW. Este consumo puede entenderse como una cota superior, ya que el contador se encuentra prácticamente siempre activo. El consumo del bloque a representa solamente un 2% de los 380 uW de consumo estimado por el chip completo. Esto se debe a que opera a una frecuencia muy reducida en comparación a otras partes del chip, utiliza contadores asíncronos y es completamente digital. Se considera, por tanto, que se ha logrado un buen resultado en cuanto a consumo del bloque a.

5 – PLANIFICACIÓN DEL PROYECTO

Para la realización de este TFM se ha seguido el siguiente plan de trabajo, organizado por semanas:

Semanas 1 – 3

Durante las primeras semanas se ha realizado un estudio de los convertidores sigma-delta y su implementación analógica. Para ello, se ha hecho uso del libro T. C. Carusone, K. W. Martin, and D. Johns, *Analog Integrated Circuit Design*, 2nd ed. S.l.: John Wiley & Sons, 2012, específicamente el capítulo dedicado a los convertidores sigma-delta. Además, se ha realizado un estudio de los artículos [1][2][6], dedicados a VCO-ADCs y osciladores en anillo.

Semanas 4 - 5

Durante esta semana, se han puesto en práctica los conocimientos adquiridos en las semanas 1-4, mediante simulaciones en Matlab/Simulink. Además, se ha realizado formación en Cadence Virtuoso, con el fin de poder llevar a cabo el resto del proyecto.

Semanas 6 - 12

Estas semanas se han dedicado a participar en la elaboración del sistema de alto nivel llevada a cabo por todo el equipo de trabajo que ha desarrollado el chip y desarrollo de los modelos integer-true y bit-true de Matlab.

Semanas 13 - 14

Las semanas 13 y 14 se han dedicado a la implementación y verificación a nivel de transistor del bloque implementado.

Semanas 15 – 16

Estas semanas se han dedicado al desarrollo del layout del bloque implementado, así como las simulaciones post-layout.

Semanas 17 – 20

Se han dedicado a la escritura y revisión de la memoria.

6 - CONCLUSIONES

A lo largo de este TFM se ha realizado un estudio de los convertidores sigma-delta, VCO-ADCs, microelectrónica analógica y digital, además del aprendizaje de las herramientas de diseño y simulación de Cadence, Verilog-A y redes neuronales. Se ha diseñado un componente de un chip real completamente funcional, realizando el proceso de diseño a todos los niveles: Sistema, Transistor y layout, explorando diferentes opciones a nivel de sistema. Se han estudiado diferentes formas para combinar de salida del convertidor y extender el rango dinámico, llegando a la conclusión de que utilizar redes neuronales no es la mejor opción frente a otras alternativas de más bajo consumo y rendimiento similar. Además, se han llevado a cabo los procesos de verificación correspondientes para asegurar un funcionamiento adecuado y un consumo de potencia reducido del bloque implementado. Todos los objetivos planteados al inicio del proyecto se han completado satisfactoriamente.

A la fecha de finalización de este TFM, el chip explicado a lo largo de este documento se encuentra en proceso de fabricación y como líneas futuras de trabajo se encuentran la medición y comparación con simulaciones del rendimiento del convertidor, así como un diseño con más osciladores que permita extender aún más el rango dinámico u obtener una SNR más alta.

Todo ello ha supuesto un enriquecimiento de mi conocimiento personal y profesional muy grande.

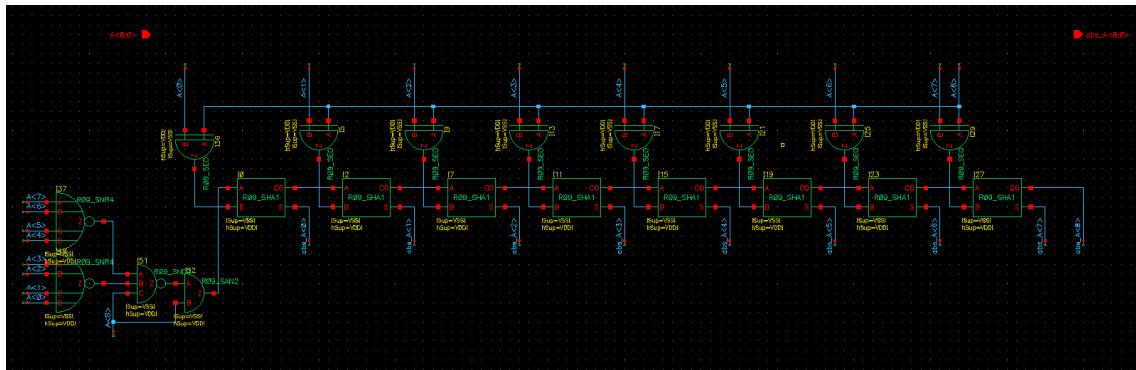
7 – BIBLIOGRAFÍA Y REFERENCIAS

- [1] - G. G. E. Gielen, L. Hernandez and P. Rombouts, Time-Encoding Analog-to-Digital Converters: Bridging the Analog Gap to Advanced Digital CMOS-Part 1: Basic Principles
- [2] - Convertidores sigma-delta - T. C. Carusone, K. W. Martin, and D. Johns, Analog Integrated Circuit Design, 2nd ed. S.l.: John Wiley & Sons, 2012. Capítulo 5 - Oversampling Converters
- [3] - "Design, Simulation Techniques, and Architectures for Oversampling Converters" in Oversampling Delta-Sigma Data Converters: Theory, Design, and Simulation, J. C. Candy and G. C. Temes, Eds. Publisher, Year, pp. pages.
- [4] - Introduction to MEMS Microphone Technology—Analog vs Digital Microphones - All about circuits
- [5] - G. G. E. Gielen, L. Hernandez and P. Rombouts, "Time-Encoding Analog-to-Digital Converters: Bridging the Analog Gap to Advanced Digital CMOS?Part 2: Architectures and Circuits"
- [6] - G. G. E. Gielen, L. Hernandez and P. Rombouts, "Time-Encoding Analog-to-Digital Converters: Bridging the Analog Gap to Advanced Digital CMOS?Part 2: Architectures and Circuits"
- [7] - A. A. Abidi, "Phase Noise and Jitter in CMOS Ring Oscillators,"
- [8] - Fernando Cardes ,Andres Quintero,Eric Gutierrez,Cesare Buffa,Andreas WiesbauerandLuis Hernandez - "SNDR Limits of Oscillator-Based Sensor Readout Circuits"
- [9] - Audio Application Kit - Infineon Technologies
- [10] - What is Edge Computing? - IBM - (<https://www.ibm.com/es-es/topics/edge-computing>)
- [11] - M. Todisco, S. K. Datta and C. Bonnet, "Audio analysis using edge computing for road side assistance systems"
- [12] - C. Lee, H. -M. Kang, Y. Jeon and S. J. Kang, "Ambient Sound Analysis for Non-Invasive Indoor Activity Detection in Edge Computing Environments"
- [13] - V. S. Raj, J. V. M. Sai, N. A. L. Yogesh, S. B. K. Preetha and L. R, "Smart Traffic Control for Emergency Vehicles Prioritization using Video and Audio Processing"
- [14] - R. Sumikawa, A. Kosuge, Y. -C. Hsu, K. Shiba, M. Hamada and T. Kuroda, "A183.4-nJ/Inference 152.8- μ W 35-Voice Commands Recognition Wired-Logic Processor Using Algorithm-Circuit Co-Optimization Technique"
- [15] - S. Mondal, L. Ruan, M. Maier, D. Larrabeiti, G. Das and E. Wong, "Enabling Remote Human-to-Machine Applications With AI-Enhanced Servers Over Access Networks"

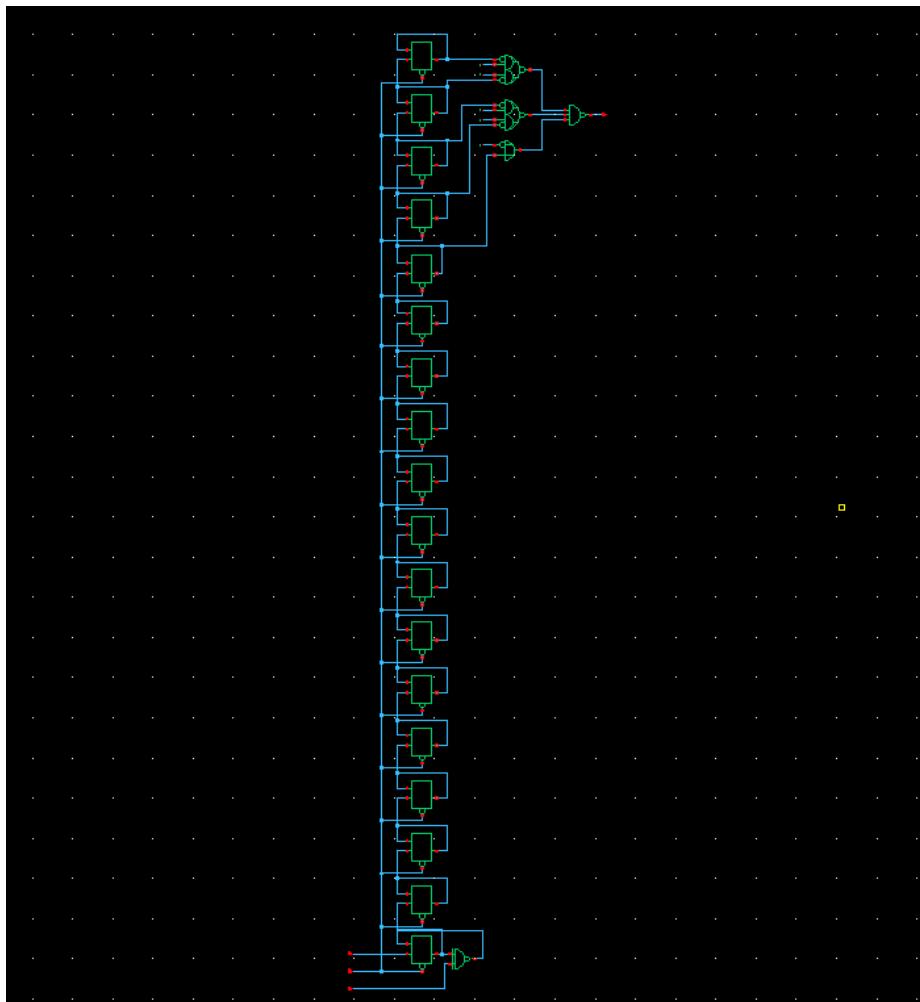
- [16] - C. Perez et al., "A VCO-Based ADC With Direct Connection to a Microphone MEMS, 80-dB Peak SNDR and 438- μ W Power Consumption"
- [17] - Microphone specifications explained (AN-1112), InvenSense
- [18] - J. Xu, J. Yu, X. Liu and H. Meng, "Mixed Precision DNN Quantization for Overlapped Speech Separation and Recognition"
- [19] - Softmax function definition - DeepAI - (<https://deepai.org/machine-learning-glossary-and-terms/softmax-layer>)
- [20] - Matlab & Simulink - <https://es.mathworks.com/>
- [21] - Python - <https://python.org>
- [22] - Keras - <https://keras.io/>
- [23] - Tensorflow - <https://www.tensorflow.org/>
- [24] - Jason Brownlee - "A gentle introduction to the Rectified Linear Unit (ReLU)"
- [25] - Cadence Virtuoso - https://www.cadence.com/en_US/home.html
- [26] - J. Borgmans, R. Riem and P. Rombouts, "The Analog Behavior of Pseudo Digital Ring Oscillators Used in VCO ADCs"
- [27] - What is Gray Code - <https://www.eeeguide.com/what-is-gray-code/>
- [28] - V. Medina, R. Garvi, E. Gutierrez, S. Paton and L. Hernandez, "A Gray-Encoded Ring Oscillator for Efficient Frequency-to-Digital Conversion in VCO-Based ADCs,"
- [29] - Clock Gating - T. Chindhu S. and N. Shanmugasundaram, "Clock Gating Techniques: An Overview,"
- [30] - What is a Barrel Shifter ? - Technopedia - www.techopedia.com/definition/17863/barrel-shifter
- [31] - J. Y. Chen, "CMOS — The emerging VLSI technology"
- [32] - Duen-Jeng Wang and S. Narayan, "SoC Verification"
- [33] - Verilog-A (Cadence) - <https://verilogams.com/tutorials/vloga-intro.html>
- [34] - Verilog - <https://verilog.com/>
- [35] - Spectre Simulation Platform – Cadence
- [36] - V. Medina, R. Garvi, E. Gutierrez, S. Paton and L. Hernandez, "A Gray-Encoded Ring Oscillator for Efficient Frequency-to-Digital Conversion in VCO-Based ADCs,"

ANEXO – CAPTURAS DE LOS ESQUEMATICOS EN CADENCE VIRTUOSO

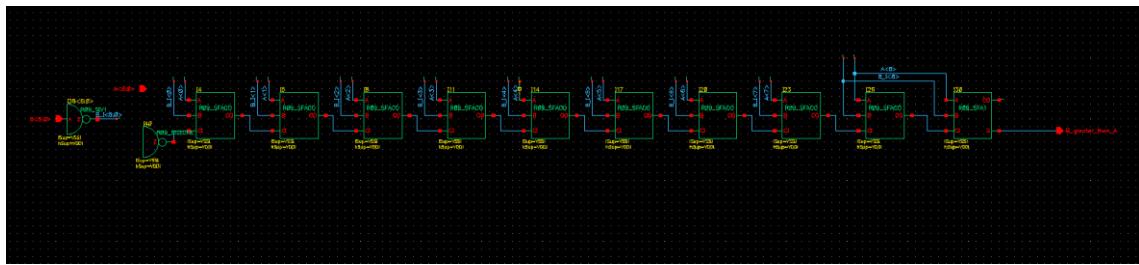
Valor absoluto



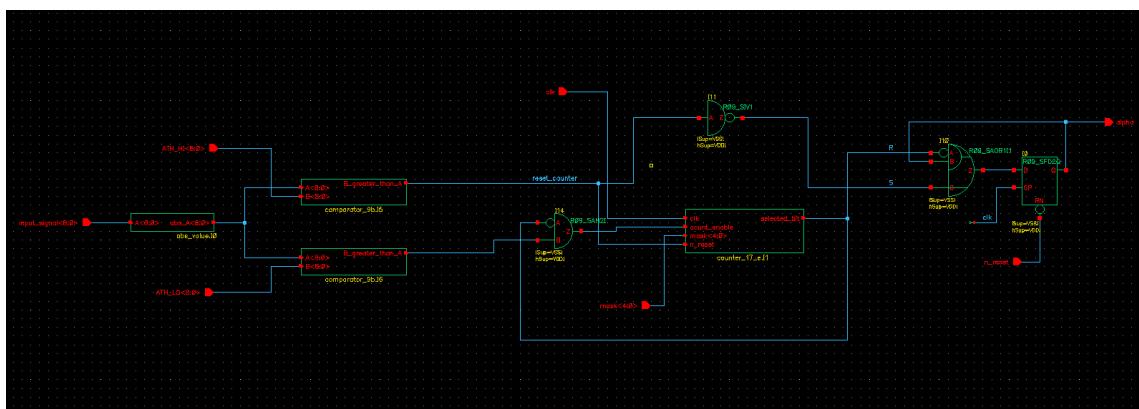
Contador con selección de máscara y desactivación de conteo



Comparador

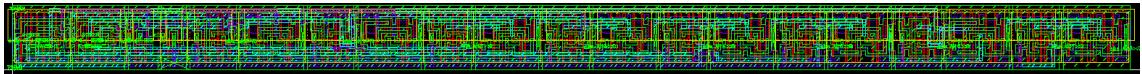


Bloque Alpha completo

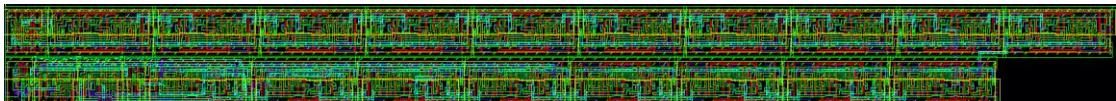


ANEXO – CAPTURAS DE LOS LAYOUTS EN CADENCE VIRTUOSO

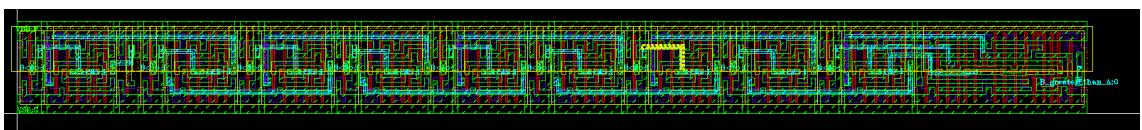
Valor absoluto



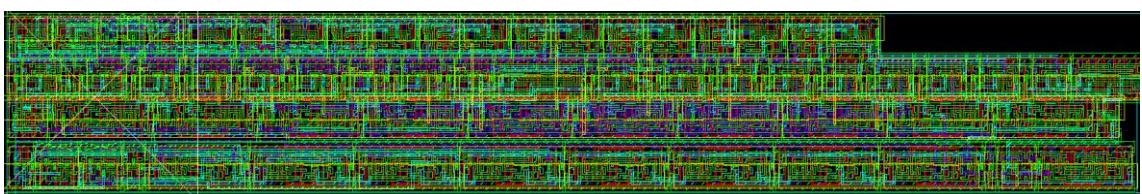
Contador



Comparador



Bloque Alpha



ANEXO – TODOS LOS ESTADOS DE UN OSCILADOR EN ANILLO DE 16 ETAPAS

16 tap VCO phases [15..0] evolution																	GRAY
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1
0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0
1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	1	1
1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

ANEXO – CÓDIGOS VERILOG-A

```
// t_va_abs_value, verilog-a

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

module t_va_abs_value(out_val[8:0]);

output [8:0] out_val;
electrical [8:0] out_val;

parameter real period = 10.0;
parameter integer initial_value [8:0] = {1,1,1,0,0,0,0,0};

parameter real vlogic_high = 1.0;
parameter real vlogic_low = 0;
parameter real vtrans = 0.6;
parameter real tdel = 1e-12;
parameter real tfall = 50e-12;
parameter real trise = 50e-12;

genvar k;
genvar l;
integer i = 0;
integer j = 0;
integer h = 0;
integer t;

integer out_val_state [8:0];

analog begin

@(initial_step) begin
    i = 0;
    for(k = 0; k < 9; k = k + 1) begin
        out_val_state[i] = initial_value[i];
        i = i + 1;
    end
end

@(timer(0,period)) begin
    out_val_state[0] = !out_val_state[0];
    i = 1;
    for(k = 1; k < 9; k = k + 1) begin
        for(l = 0; l < k; l = l + 1) begin
            if(out_val_state[j] == 1) begin
                h = 1;
            end
            j = j + 1;
        end
        if(h == 0) begin
            out_val_state[i] = !out_val_state[i];
        end
        h = 0;
        j = 0;
    end
end

```

```

        i = i + 1;
    end
end

generate t (0, 8, 1 ) V(out_val[t]) <+
transition(out_val_state[t]*vlogic_high, tdel, trise, tfall);

end

endmodule

// va_t_counter_17_e, veriloga

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

module va_t_counter_17_e(clock,n_reset,mask,enable_count);

output clock;
electrical clock;

output [4:0] mask;
electrical [4:0] mask;

output n_reset;
electrical n_reset;

output enable_count;
electrical enable_count;

parameter real vlogic_high = 1.2;
parameter real vlogic_low = 0;
parameter real vtrans = 0.6;
parameter real tdel = 1e-12;
parameter real tfall = 50e-12;
parameter real trise = 50e-12;

parameter real clk_period = 50n;
integer mask_bit_enabled = 0;
parameter real reset_period = 45u;
parameter real enable_count_period = 20u;
real next_mask_change = 0.0003000;

genvar k;
integer j = 0;
integer i = 0;
integer mask_out [4:0];
integer clock_state = 1;
integer n_reset_state = 1;verilog
integer enable_count_state = 1;
real n = 0;

analog begin

@(initial_step) begin
    for(k = 0; k < 5 ; k = k + 1) begin
        mask_out[j] = j == mask_bit_enabled ? 1 : 0; // generate correct
mask

```

```

                j = j + 1;
            end
            mask_bit_enabled = -1;
        end

        @(timer(0,clk_period / 2)) begin
            clock_state = !clock_state; // generate clock
        end

//@(timer(0,reset_period / 2)) begin
//    n_reset_state = !n_reset_state; // generate reset signal for test
//end

//@(timer(0,enable_count_period / 2)) begin
//    enable_count_state = !enable_count_state; // generate eable count for
test
//end

@(timer(0,next_mask_change)) begin
    j = 0;
    mask_bit_enabled = mask_bit_enabled + 1;
    for(k = 0; k < 5 ; k = k + 1) begin
        mask_out[j] = j == mask_bit_enabled ? 1 : 0; // generate correct
mask
        j = j + 1;
    end // generate mask for test
    next_mask_change = 0.0003 * (2 ** (n)) - $abstime ;
    n = n + 1;
end

V(clock) <+ transition(clock_state*vlogic_high, tdel, trise, tfall);
V(n_reset) <+ transition(n_reset_state*vlogic_high, tdel, trise, tfall);
V(enable_count) <+ transition(enable_count_state*vlogic_high, tdel, trise,
tfall);
generate i (0, 4, 1 ) V(mask[i]) <+ transition(mask_out[i]*vlogic_high, tdel,
trise, tfall);

end

endmodule

// va_signals_from_file_clocked, veriloga

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

`define DATA_WIDTH 9

module va_signals_from_file_9b(data[`DATA_WIDTH-1:0]);
    output [`DATA_WIDTH-1:0] data;
    electrical [`DATA_WIDTH-1:0] data;

```

```

parameter string file_path = "input_signal.txt";

parameter real vlogic_high = 1.2;
parameter real vlogic_low = 0;
parameter real vtrans = 0.6;
parameter real tdel = 1e-12;
parameter real tfall = 50e-12;
parameter real trise = 50e-12;

integer file_descriptor;
real next_launch = 0.0;
genvar k;
integer j;
integer i = 0;
integer current_read [`DATA_WIDTH-1:0];

analog begin

@(initial_step) begin
    file_descriptor = $fopen(file_path, "r");
end

@(timer(next_launch)) begin
    i = 0;
    for(k = 0; k < `DATA_WIDTH; k = k + 1) begin
        $fscanf(file_descriptor,"%d",current_read[`DATA_WIDTH-1-i]);
        i = i + 1;
    end
    $fscanf(file_descriptor,"%e",next_launch);
    next_launch = next_launch + $abstime;
end

generate i (0, `DATA_WIDTH-1, 1 ) V(data[i]) <+
transition(current_read[i]*vlogic_high, tdel, trise, tfall);

end

endmodule

// va_signals_from_file, veriloga

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

`define DATA_WIDTH 9

module va_signals_from_file_clocked_9b(clk,data[`DATA_WIDTH-1:0]);

output [`DATA_WIDTH-1:0] data;
electrical [`DATA_WIDTH-1:0] data;

input clk;
electrical clk;

parameter real clk_threshold = 0.6;
parameter integer clk_edge = 1 from [-1:1];
parameter integer using_file_with_delays = 0;

parameter string file_path = "input_signal.txt";

```

```

parameter real vlogic_high = 1.2;
parameter real vlogic_low = 0;
parameter real vtrans = 0.6;
parameter real tdel = 1e-12;
parameter real tfall = 50e-12;
parameter real trise = 50e-12;

integer file_descriptor;
genvar k;
integer j;
integer i = 0;
integer current_read [`DATA_WIDTH-1:0];
real trash = 0.0;

analog begin

@(initial_step) begin
    file_descriptor = $fopen(file_path, "r" );
end

@(cross(V(clk) - clk_threshold, clk_edge)) begin
    i = 0;
    for(k = 0; k < `DATA_WIDTH; k = k + 1) begin
        $fscanf(file_descriptor,"%d",current_read[`DATA_WIDTH-1-i]);
        i = i + 1;
    end

    if(using_file_with_delays == 1) begin
        $fscanf(file_descriptor,"%e",trash);
    end
end

generate i (0, `DATA_WIDTH-1, 1 ) V(data[i]) <+
transition(current_read[i]*vlogic_high, tdel, trise, tfall);

end

endmodule

```

ANEXO – CÓDIGOS PYTHON

nn.py (red neuronal)

```
from matplotlib import pyplot as plt
import numpy as np
import keras
from keras import layers
import tensorflow as tf
import siggen as sg

class CustomModel(keras.Model):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.dense1 = layers.Dense(32, activation='relu')
        self.dense2 = layers.Dense(16, activation='relu')
        self.output_layer = layers.Dense(2, activation='softmax')

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return self.output_layer(x)

    def train_step(self, data):
        x, y = data

        with tf.GradientTape() as tape:
            y_pred = self(x, training=True) # Forward pass
            # Compute our own custom loss
            out1 = y_pred[:, 0]
            out2 = y_pred[:, 1]
            predicted_signal = out1 * x[:, 0] + out2 * x[:, 1]
            loss = tf.reduce_mean(tf.square(predicted_signal - y)) # MSE Loss

        # Compute gradients
        trainable_vars = self.trainable_variables
        gradients = tape.gradient(loss, trainable_vars)

        # Update weights
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))

        # Return a dict mapping metrics names to current value
        return {"loss": loss}

# Instantiate and compile the model
model = CustomModel()
model.compile(optimizer='adam')
```

```

# Generate the dataset
time = np.arange(32000) * 0.01
clear_signal = sg.generate_growing_sine(time)
s_hdr = sg.add_noise(clear_signal, 0.0001)
s_hsnr = sg.add_noise(clear_signal, 0.00001)
s_hsnr = sg.distort(s_hsnr, 1, 2)

# Prepare input and output data
inputs = np.stack((s_hdr, s_hsnr), axis=-1) # Creates a (10000, 2) array
outputs = clear_signal # Already in the correct shape (10000,)

# Train the model
model.fit(inputs, outputs, epochs=10, batch_size=32)

#####
#####

# model test with sine wave:

time = np.arange(7000) * 0.01
clear_signal = sg.generate_modulated_sine(time)
s_hdr = sg.add_noise(clear_signal, 0.0001)
s_hsnr = sg.add_noise(clear_signal, 0.00001)
s_hsnr = sg.distort(s_hsnr, 1, 2)

# Prepare input and output data
inputs = np.stack((s_hdr, s_hsnr), axis=-1) # Creates a (10000, 2) array
outputs = clear_signal # Already in the correct shape (10000,)

predictions = model.predict(inputs)

# Calculate the weighted sum of s_hdr and s_hsnr using the model's predictions
weighted_sum_predictions = predictions[:, 0] * inputs[:, 0] + predictions[:, 1] * inputs[:, 1]

# Plotting the comparison of clear signal and predicted signal

fig, axs = plt.subplots(3, 2, figsize=(15, 20)) # Creates 6 subplots in a 3x2 grid

# Column 1
axs[1, 0].plot(s_hdr, label='HDR Channel', color='blue')
axs[1, 0].set_title('HDR Channel')
axs[1, 0].set_ylabel('Signal Amplitude')
#axs[1, 0].legend()

```

```

    axs[1, 1].plot(s_hsnr, label='HSNR Channel', color='purple')
    axs[1, 1].set_title('HSNR Channel')
    axs[1, 1].set_ylabel('Signal Amplitude')
    axs[1, 1].set_ylim([-1,1])
    #axs[1, 1].legend()

    # Column 2
    axs[0, 0].plot(clear_signal, label='Clear Signal', linewidth=2)
    axs[0, 0].set_title('Clear Signal')
    axs[0, 0].set_ylabel('Signal Amplitude')
    #axs[0, 0].legend()

    axs[0, 1].plot(weighted_sum_predictions, label='Predicted Signal', color='orange')
    axs[0, 1].set_title('Predicted Signal')
    axs[0, 1].set_ylabel('Signal Amplitude')
    #axs[0, 1].legend()

    axs[2, 0].plot(predictions[:, 0], label='Weight Out1', linestyle='-', color='r')
    axs[2, 0].set_title('Weight HDR')
    axs[2, 0].set_ylabel('Signal Amplitude')
    #axs[2, 0].legend()

    axs[2, 1].plot(predictions[:, 1], label='Weight Out2', linestyle='-', color='g')
    axs[2, 1].set_title('Weight HSNR')
    axs[2, 1].set_ylabel('Signal Amplitude')
    #axs[2, 1].legend()

fig.subplots_adjust(hspace=0.5, wspace=0.3) # Adjust horizontal and vertical spacing

plt.show()

```

siggen.py (Utilidad creada para generar señales, ruido, etc)

```

from matplotlib import pyplot as plt
import numpy as np

def sigmoid(x,gain,gainmax):
    return (gainmax/(2*gain)) *(1 / (1 + np.exp(-4*x * gain)) - 0.5)

def distort(x,gain,gainmax):
    return sigmoid(x,gain,gainmax)

def generate_growing_sine(xvector):
    return (np.arange(len(xvector)) / len(xvector)) * np.sin(xvector)

```

```

def generate_sine(xvector):
    return np.sin(xvector)

def generate_one_period_sine(xvector):
    return np.sin(xvector * 2* np.pi / xvector[-1])

def generate_level(xvector, level):
    return np.full(xvector.shape, level)

def generate_modulated_sine(x):
    return generate_sine(x/10) * generate_sine(x)

def add_noise(x, amount):
    d = np.sqrt(3 * amount) # use amount as power for a uniform distribution.
    noise = np.random.uniform(-d, d, size=x.shape)
    return x + noise

```

converter.py

```

import argparse

def twos_complement_binary(number, bits=8):
    if number < 0:
        number = (1 << bits) + number
    binary_representation = f"{number:0{bits}b}"
    spaced_binary = ' '.join(binary_representation)
    return spaced_binary

def process_file(input_filepath, output_filepath, num_bits):
    with open(input_filepath, 'r') as file:
        lines = file.readlines()

    processed_lines = []
    for line in lines:
        if line.strip():
            integer_part, decimal_part = line.split()
            binary_string = twos_complement_binary(int(integer_part),
bits=num_bits)
            processed_line = f'{binary_string} {decimal_part}\n'
            processed_lines.append(processed_line)

    with open(output_filepath, 'w') as file:
        file.writelines(processed_lines)
    print(f'Processed data written to {output_filepath}')

def main():
    parser = argparse.ArgumentParser(description="Process a file to convert integers to binary representation.")

```

```
parser.add_argument('-input', type=str, help='Path to the input
file', default='input.txt')
parser.add_argument('-output', type=str, help='Path to the output
file', default='output.txt')
parser.add_argument('-nb', type=int, help='Number of bits for the
binary conversion', default=8)

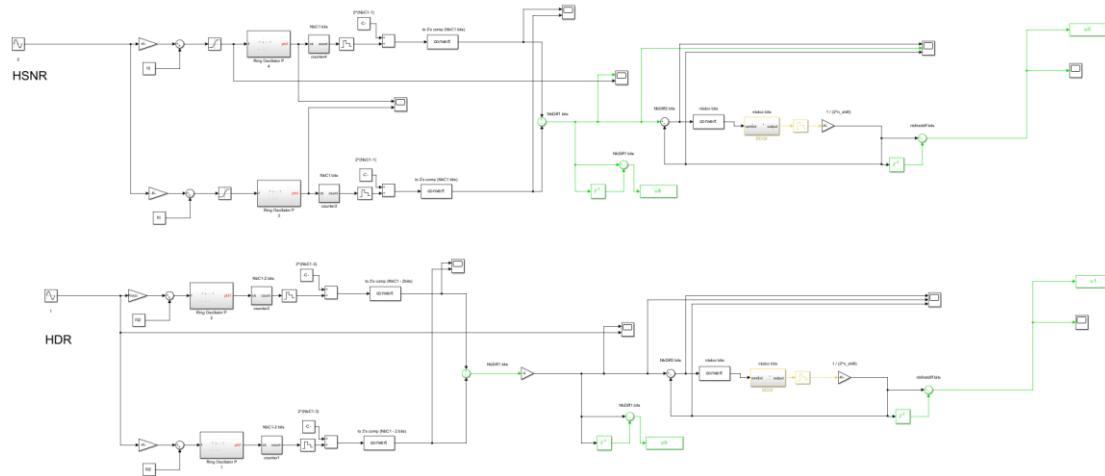
args = parser.parse_args()

process_file(args.input, args.output, args.nb)

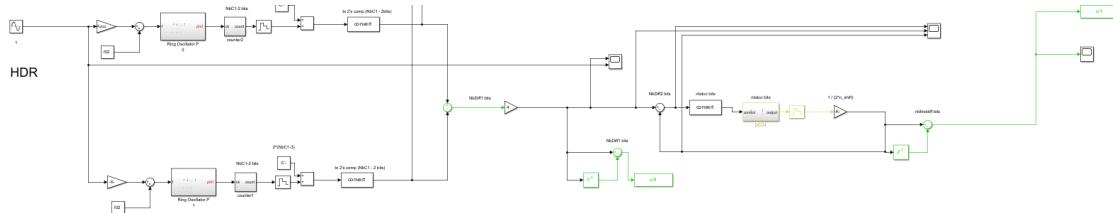
if __name__ == "__main__":
    main()
```

ANEXO – MODELOS MATLAB/SIMULINK: INTEGER-TRUE

Vista TOP

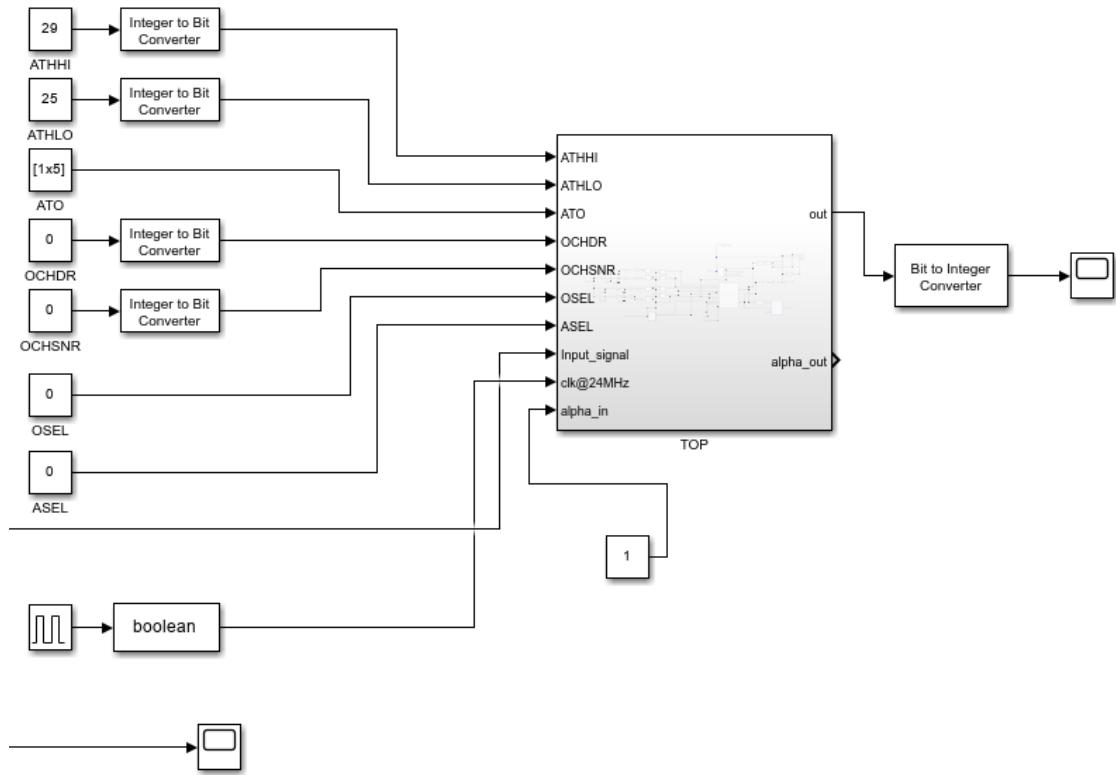


Canal HDR/HSNR

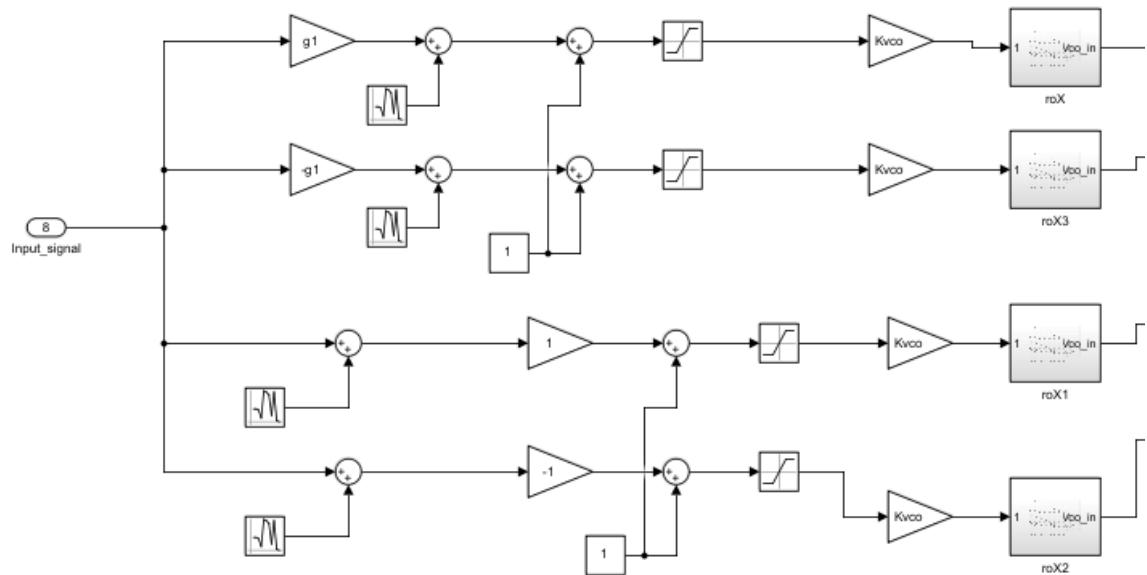


ANEXO – MODELOS MATLAB/SIMULINK: BIT-TRUE

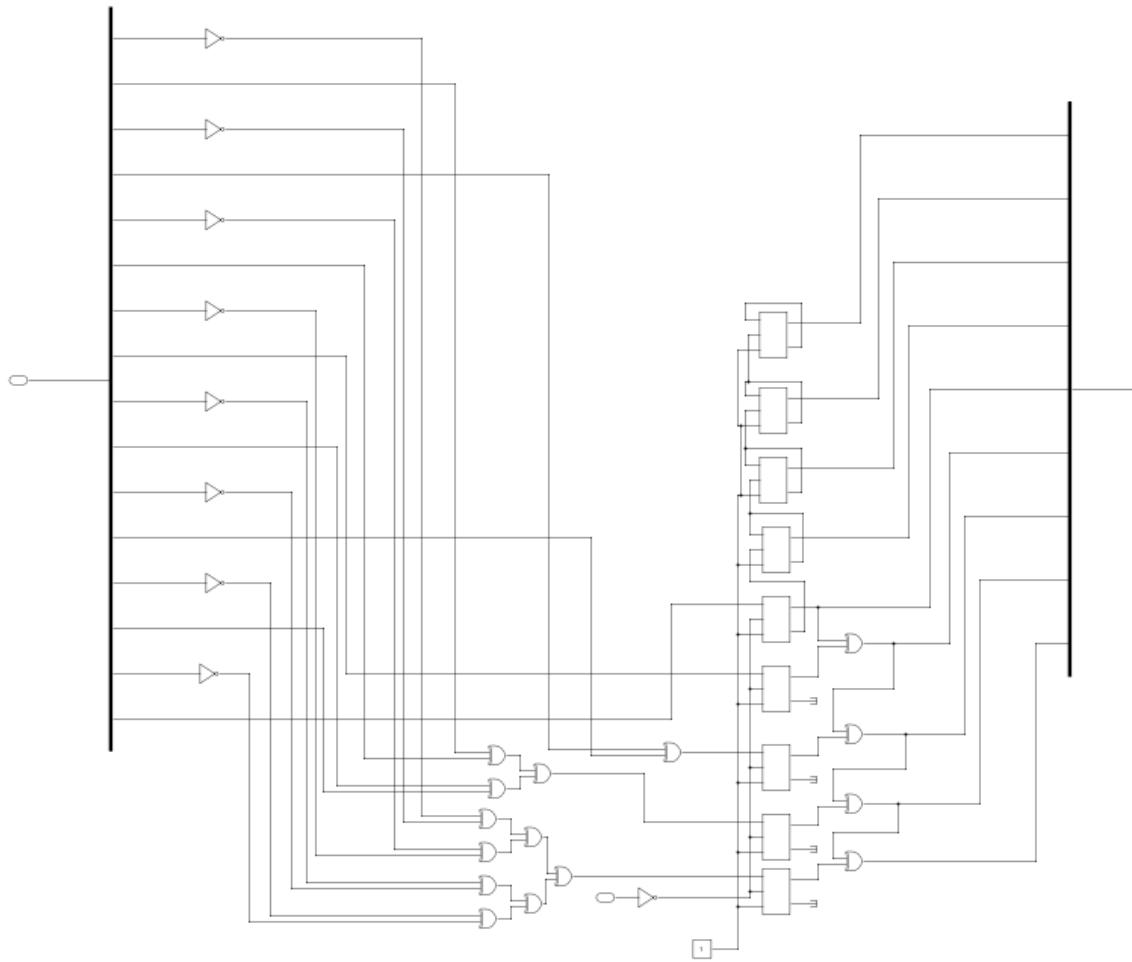
Vista TOP



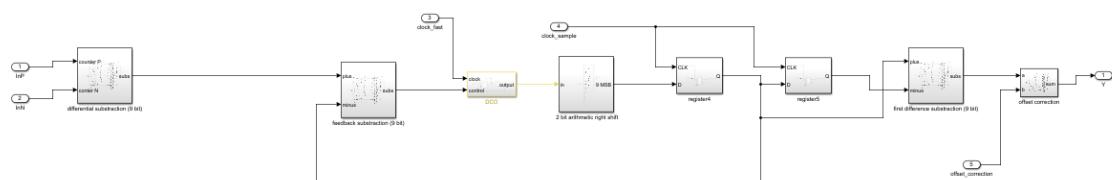
Osciladores



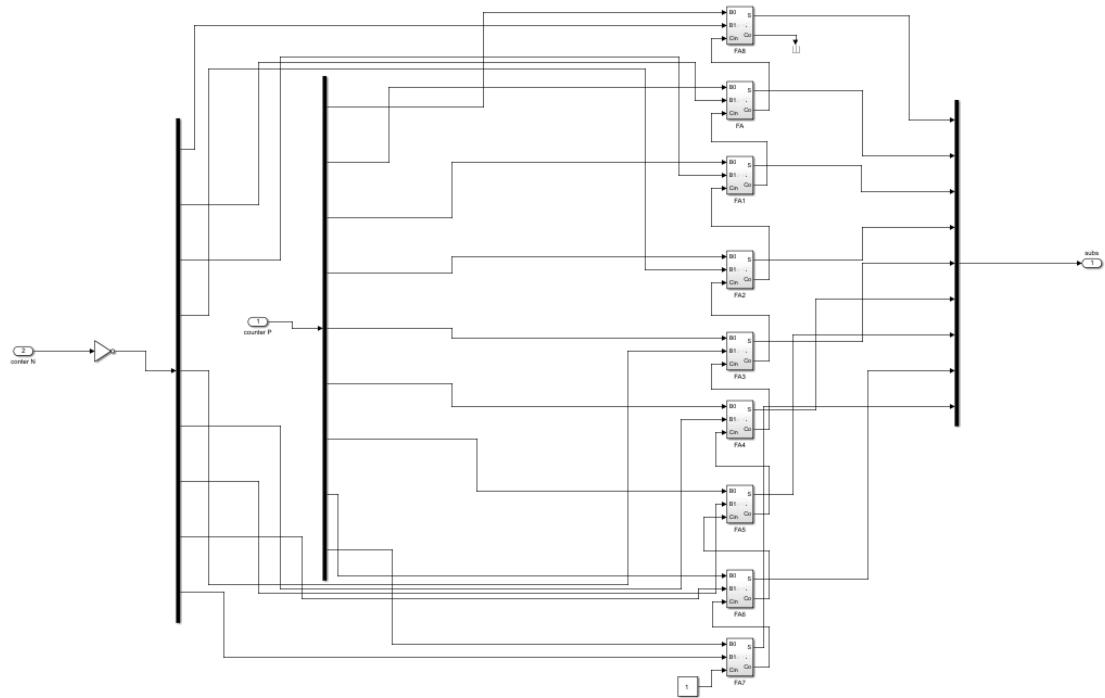
Contadores Gray + extensión + conversión a binario



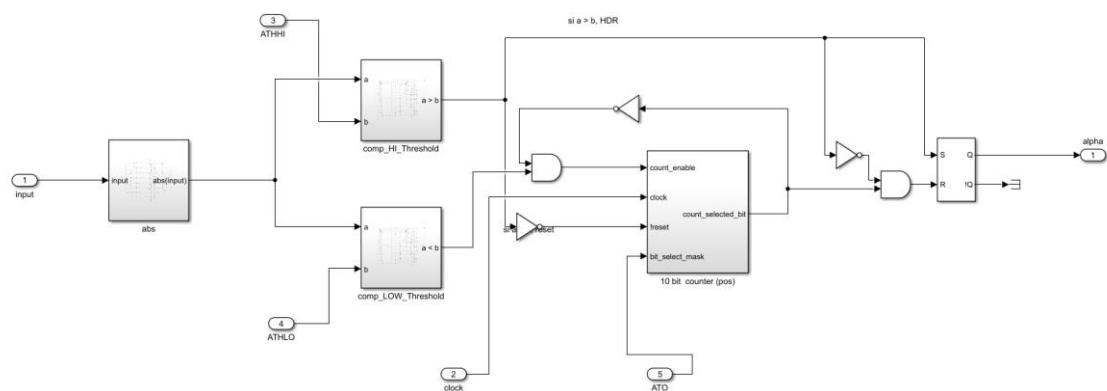
Noise-Shaper



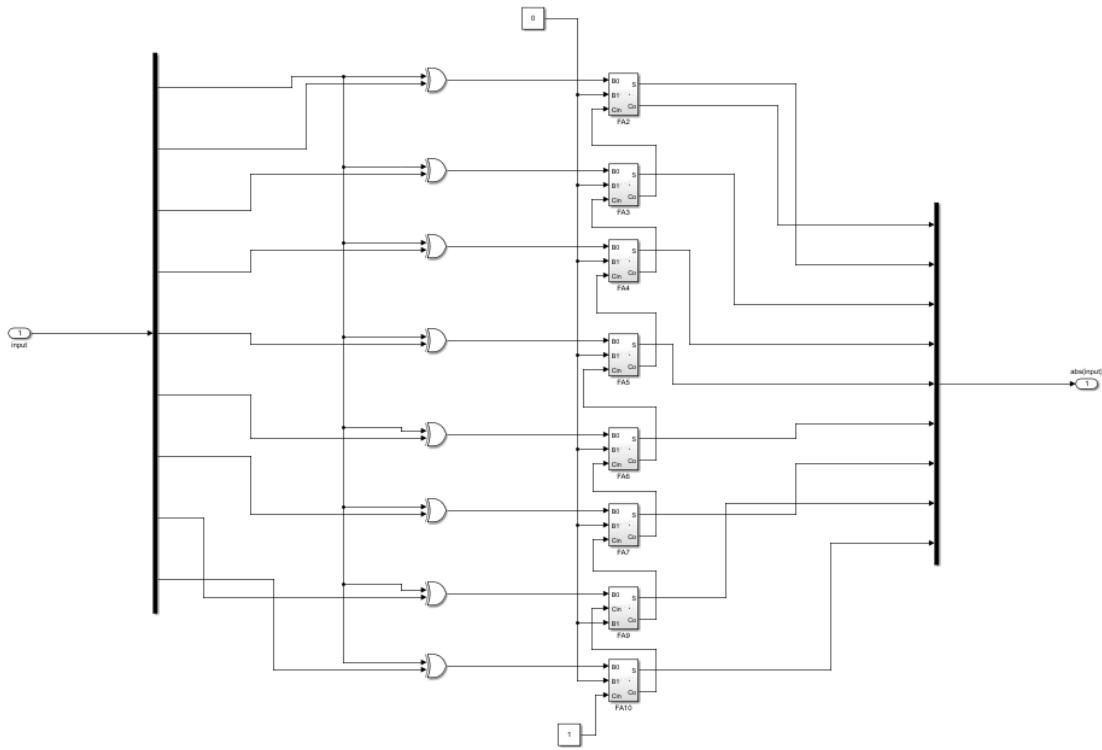
Sumadores utilizados en el Noise-shaper



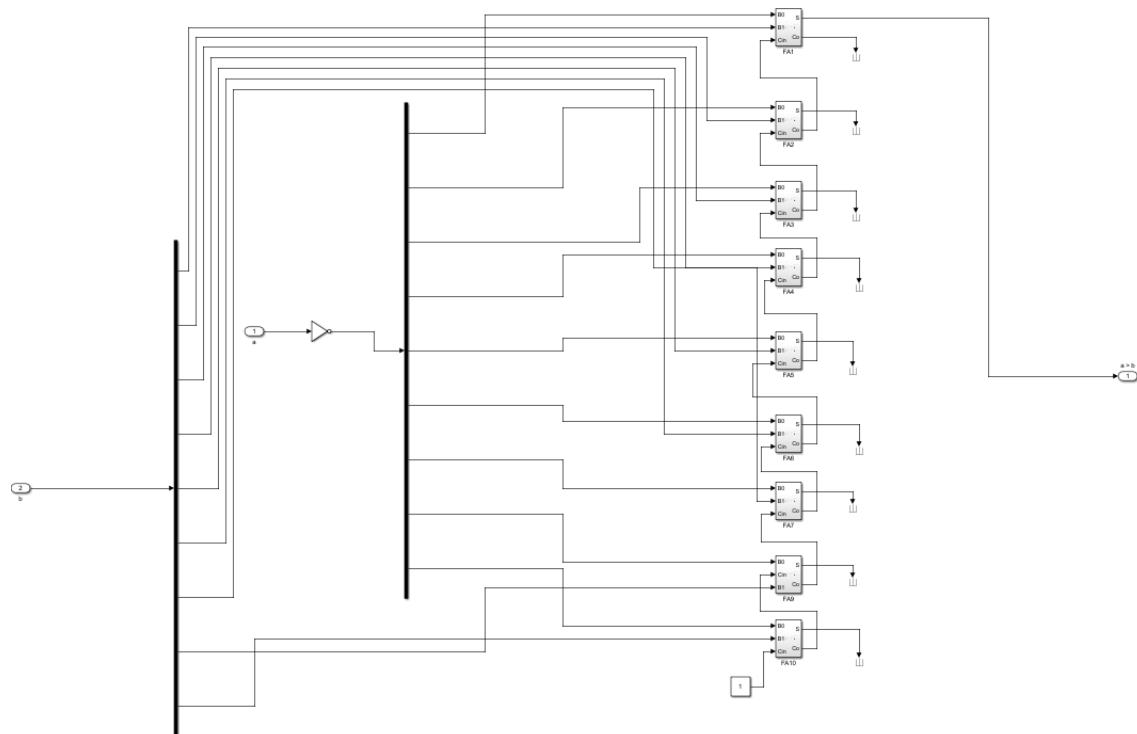
Bloque Alpha (TOP)



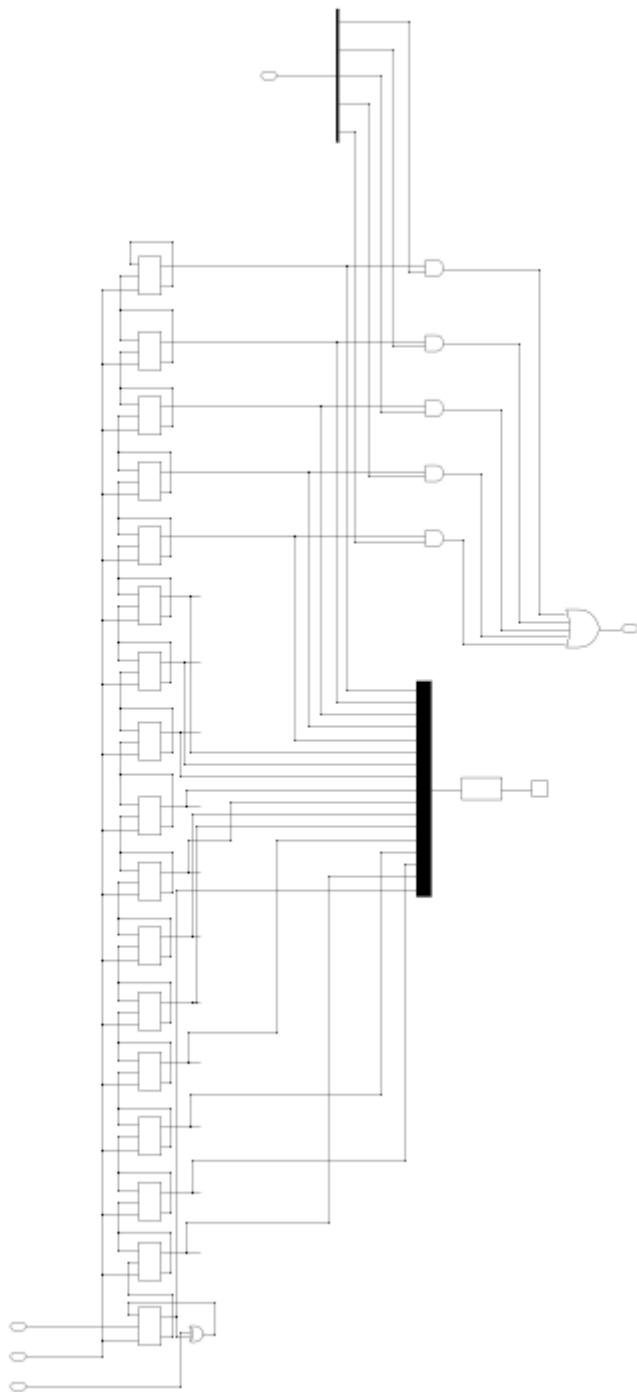
Valor absoluto



Comparadores



Contador con selección de máscara y desactivación de conteo



Generador de relojes

