Trabajo Fin de Grado
Ingeniería Aeroespacial

Realización de un convertidor Digital/Analógico con 12 bits de resolución y 20kHz de ancho de banda basado en un modulador Sigma/Delta de segundo orden y un bit de salida

Autor: Rosalía Otero Naranjo

Tutor: Francisco Colodro Ruiz

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Ingeniería Aeroespacial

Realización de un convertidor Digital/Analógico con 12 bits de resolución y 20kHz de ancho de banda basado en un modulador Sigma/Delta de segundo orden y un bit de salida

Autor:
Rosalía Otero Naranjo

Tutor:
Francisco Colodro Ruiz
Profesor Titular

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Realización de un convertidor Digital/Analógico con 12 bits de resolución y 20kHz de ancho de banda basado en un modulador Sigma/Delta de segundo orden y un bit de salida

Autor: Rosalía Otero Naranjo
Tutor: Francisco Colodro Ruiz

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría agradecer a mis padres y familiares por todo lo que me han apoyado siempre, así como a todos mis profesores, desde el colegio hasta la universidad, por lo que me han enseñado. Y en especial, agradezco a mi tutor, Francisco Colodro Ruiz, por su paciencia y por ayudarme de principio a fin durante la realización de este proyecto.

Rosalía Otero Naranjo
Sevilla, 2021

Resumen

El propósito de este trabajo es el análisis y desarrollo de un convertidor Digital/Analógico con 12 bits de resolución y 20 kHz de ancho de banda basado en modulador Sigma Delta de segundo orden así como su posterior implementación en una FPGA.

Para ello, se han realizado simulaciones mediante *MATLAB* y *Simulink* con objeto de estudiar el comportamiento del circuito modulador ante distintos valores de los parámetros que lo definen para así determinar las variables de diseño adecuadas que garanticen su correcto funcionamiento.

Tras ello, se codifica el circuito a estudiar en VHDL empleando la herramienta *Vivado Design Suite* - HLx Editions - versión 2019.2 mediante la cuál se llevan a cabo diversas simulaciones para cada componente y para el circuito completo. Todas estas simulaciones previas a la implementación del circuito en la FPGA han tenido como objetivo corregir errores y simplificar los códigos para facilitar una correcta implementación.

Cabe mencionar la parametrización empleada en todos los códigos para permitir la realización de distintos análisis empleando distintos parámetros de diseño.

La última etapa de este proyecto consiste en implementar dicho modulador en una FPGA para así tomar medidas experimentales usando el osciloscopio digital *PicoScope* 6 y comparar las prestaciones obtenidas de estas mediciones con las del caso ideal para concluir si se ha logrado alcanzar los valores objetivos.

Abstract

The aim of this bachelor thesis is to design a second order Sigma-Delta Modulator as well as its subsequent FPGA implementation.

Firstly, a preliminary analysis using *MATLAB* and *Simulink* programs is done with the purpose of establishing the most appropriate parameters values that ensure the correct operation of the electronic circuit.

Following this, a codification of the electronic circuit in VHDL language is carried out using *Vivado Design Suit*- HLx Editions- version 2019.2. This software tool allows running several simulations for each component and for the complete circuit in order to correct possible errors prior to the implementation.

Lastly, the FPGA implementation of the Sigma-Delta Modulator is performed and the experimental results are examined using the digital oscilloscope *PicoScope* 6 concluding if the quality and performance of the circuit correspond to the ideal ones.

Índice Abreviado

Índice

Notación

AD	Analógico Digital
DA	Digital Analógico
SB	Single Bit
SDM	Modulador Sigma Delta
Modulador $\Sigma\Delta$	Modulador Sigma Delta
FPGA	Field Programmable Gate Arrays
CA2	Complemento a 2
DSP	Digital Signal Processing
Δ	Paso de cuantización
SNR	Relación señal ruido
SW	Software
RZ	Retorno a cero
OSR	Factor de sobremuestreo, *Oversampling Ratio*
B	Ancho de banda de la señal
LPF	Filtro paso bajo
LUT	Lookup Table

1 Introducción

En la actualidad, predomina el procesamiento digital de señales debido a las ventajas que este presenta como la disminución del peso de los componentes, un aumento en la robustez de los sistemas y la posibilidad de combinar distintos circuitos hasta llegar a una elevada complejidad. Además, la velocidad a la que estos circuitos trabajan también va en aumento.

De este modo, resulta necesario la existencia de convertidores AD/DA debido a las ventajas que trabajar en el dominio digital presenta, de esta manera se requiere que dichos convertidores funcionen a altas velocidades. Todo lo anterior sirve de motivación para realizar este proyecto donde se lleva a cabo un convertidor Digital/Analógico cuyas principales características son:

- Se basa en un modulador Sigma/Delta de segundo orden, single-bit.
- La salida del convertidor es una señal binaria de 1 bit.
- Número de bits efectivos : $N_{eff} = 12$ bits
- Ancho de banda de la señal : $BW = 20$kHz
- Reloj maestro ($T_{clk} = 10$ns, $f_{clk} = 100$MHz) que rige los dispositivos con lógica secuencial del circuito.
- Se emplea un cuantizador single-bit (SB) simétrico

Dicho modulador emplea aritmética entera a lo largo de todo el circuito con números binarios expresados en complemento a dos.

La realización de este modulador consta de varias etapas muy diferenciadas y dependientes las unas de las otras debido a que los resultados de una etapa serán fundamentales para llevar a cabo la siguiente tal y como se muestra en la Figura 1.1. De este modo, se comienza realizando un análisis ideal mediante simulaciones funcionales en *MATLAB&Simulink* a partir del cuál se define la arquitectura más adecuada para implementar el SDM así como sus parámetros (número de bits de la señal en cada bus de datos del circuito, componentes del circuito ...) y sus prestaciones (SNR y HD2) con las que se compararán los resultados experimentales obtenidos tras la implementación a modo de verificar que el SDM se ha realizado correctamente.

A continuación, se implementa la arquitectura elegida en VHDL empleando el Software *Vivado*, codificando cada componente que forma el circuito completo para finalmente interconectarlos y realizar nuevamente simulaciones funcionales para comprobar el funcionamiento de la arquitectura completa. Además, a las simulaciones funcionales se les añade como comprobación adicional el estudio de los ficheros de datos asociados a la señal de entrada y de salida del SDM en *MATLAB*.

Una vez realizado lo anterior, se procede a efectuar la síntesis e implementación del SDM en VHDL y posteriormente se llevan a cabo análisis tipo timing post-síntesis y post-implementación con el fin de analizar el efecto de los retrasos temporales en la señal a la salida del circuito.

El siguiente paso es la conexión y programación del dispositivo programable (en nuestro caso se emplea una FPGA) lo cual resulta sencillo en el Software previamente mencionado. Finalmente, se conecta la salida

de la FPGA con un osciloscopio digital para analizar los resultados experimentales y extraer ficheros de datos asociados a las señales de salida del SDM que serán por último estudiados en *MATLAB* para determinar sus prestaciones y si estas cumplen con los valores obtenidos en el primer análisis del caso ideal.

Además, para facilitar la correcta comprensión de lo explicado durante este documento, se comienza la memoria con un análisis teórico en el que se exponen los conceptos fundamentales empleados.

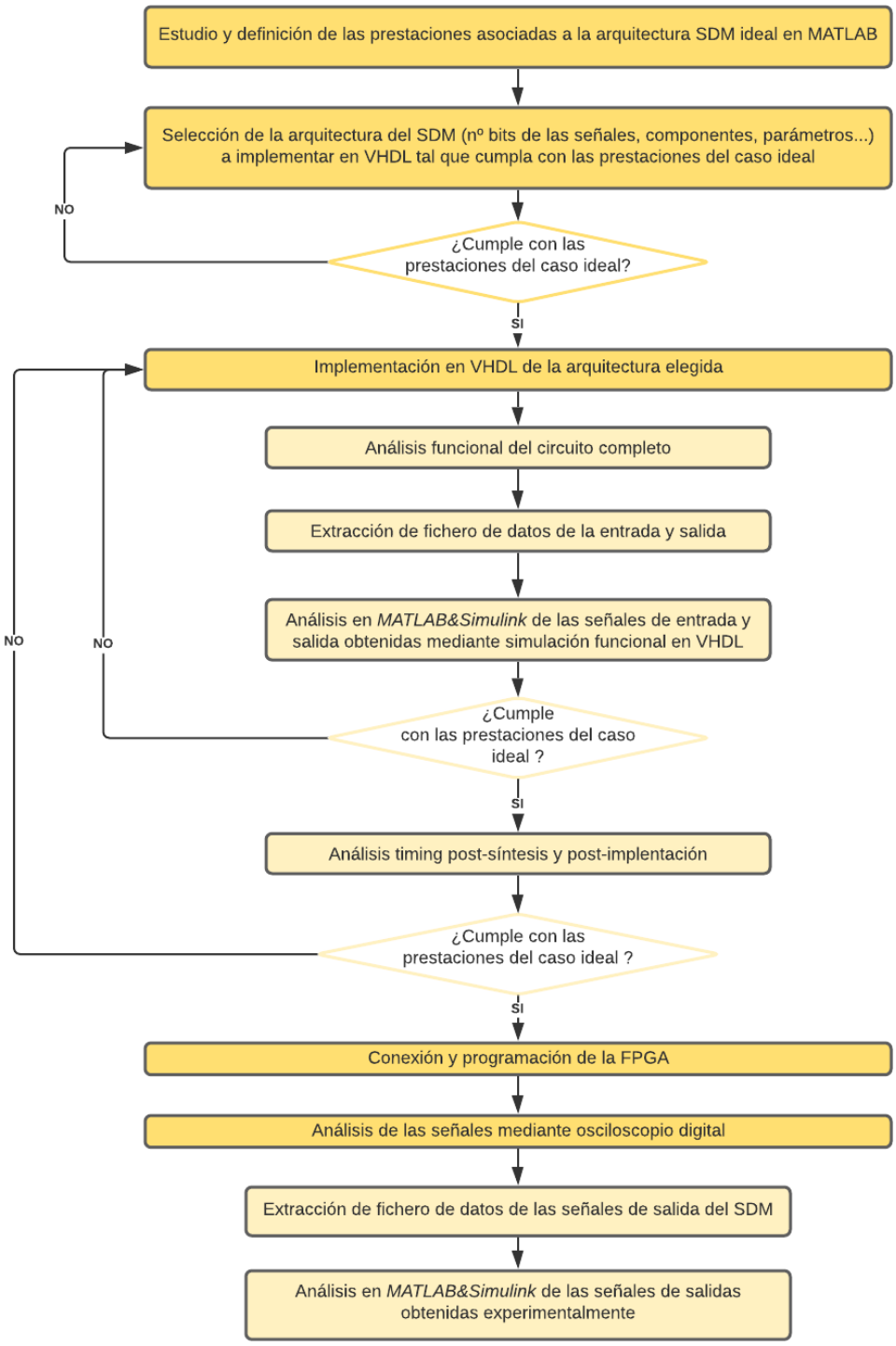


Figura 1.1 Diagrama de flujo de los pasos a seguir durante la realización del convertidor Digital/Analógico .

Un esquema sencillo que permite visualizar el circuito completo así como la interacción entre los distintos componentes, la forma de onda y número de bits de las señales principales es el mostrado en la Figura 1.2 dónde se indica el paso de mundo digital a analógico mediante el convertidor Digital/Analógico (DAC, del inglés digital to analogue converter).

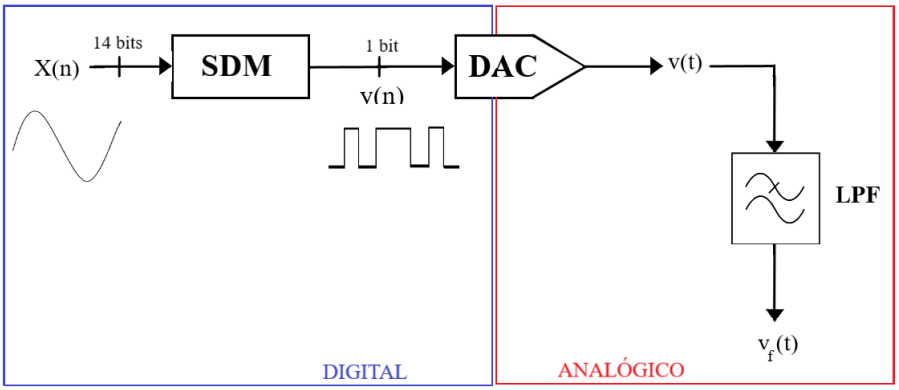


Figura 1.2 Esquema genérico de la obtención de resultados experimentales a partir de una señal de entrada de 14 bits al SDM .

En la Figura 1.2 se muestra la interrelación entre señales analógicas y digitales así como se destacan los principales elementos necesarios para la obtención de la señal analógica a la salida.

- Señal $X(n)$: señal digital de 14 bits cuyo valor corresponde a la toma de muestras de una señal senoidal ideal.
- Señal $y(n)$: señal de salida del SDM, binaria, de 1 bit, correspondiente a una señal senoidal con ruido. La disminución de 14 bits a 1 bit se obtiene gracias al uso de un cuantizador dentro de la arquitectura del Modulador ΣΔ.
- DAC: Convertidor digital analógico, en nuestro caso lo realiza la FPGA.
- LPF: Filtro paso bajo que elimina el ruido a altas frecuencias a la salida del SDM.

2 Marco teórico

A continuación, se detallan los conceptos necesarios para la realización de este proyecto como base teórica para la comprensión de los resultados y conclusiones obtenidos.

2.1 Muestreo de una señal

El muestreo es una operación fundamental en la conversión de una señal analógica en digital y su posterior procesamiento digital o DSP. Se define como la toma de muestras de amplitud de la señal de entrada analógica. Dichos datos se capturan en intervalos de tiempo de valor constante y fijo, lo que resulta en muestras equiespaciadas de la señal de entrada al circuito muestreador. De este modo, se pasa de una señal continua en el tiempo a una señal que toma un conjunto finito de valores como se refleja en la Figura 2.1.

El intervalo de tiempo entre cada muestra recibe el nombre de período de muestreo, $T_s(s)$, tal que su inversa se denomina tasa o frecuencia de muestreo, f_s y representa el número de muestras capturadas en un segundo.

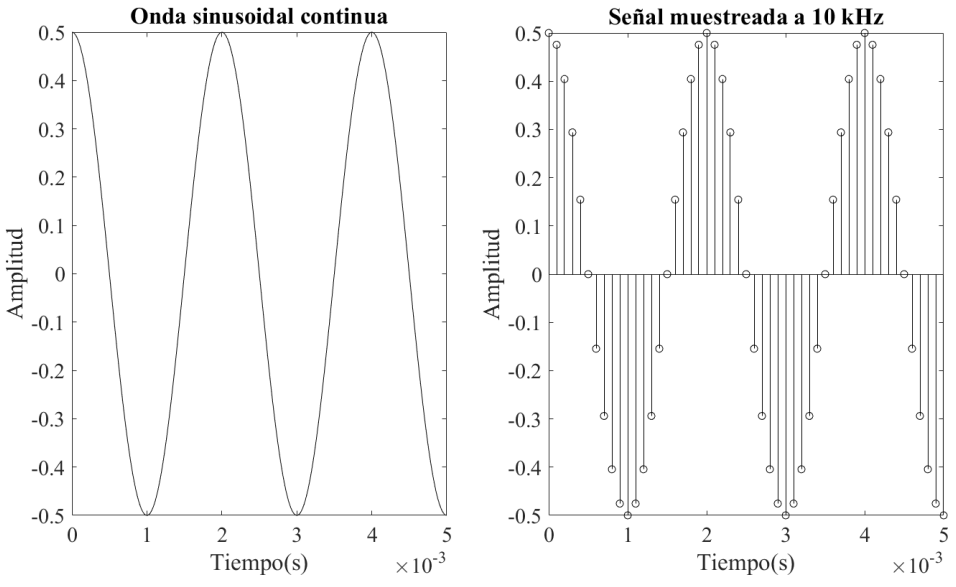


Figura 2.1 Señal sinusoidal muestreada.

Para un análisis más profundo, se expresa matemáticamente este proceso, siendo $x(t)$ la señal de entrada continua e $y(t)$ la señal muestreada.

$$y(t) = x(t) \cdot \sum_{m=-\infty}^{m=\infty} \cdot \delta(t - m \cdot T_s), \tag{2.1}$$

$$Y(f) = f_s \sum_{m=-\infty}^{m=\infty} X(f - m \cdot f_s) \tag{2.2}$$

De la expresión (2.1) se observa como ahora la señal pasa a ser un sumatorio de deltas de dirac con amplitud igual al valor correspondiente a la señal sin muestrear en dicho punto.

En cuanto a la expresión (2.2), se aprecia como la señal muestreada se forma a partir de réplicas del espectro de la señal inicial equiespaciadas un valor igual a la frecuencia de muestreo, poniendo esto último de manifiesto la importanica que tiene la elección adecuada de f_s ya que de no ser así, se pueden producir problemas tanto en la conversión A/D, D/A y posterior reconstrucción de la señal.

2.2 Aliasing

El fenómeno de aliasing o solapamiento aparece cuando la frecuencia de muestreo a la que se trabaja no es la correcta y resulta en la imposibilidad de recuperar la señal original a partir de la digitalizada.

A continuación, se representa dicho efecto, mostrando una señal muestreada sin aliasing (Figura 2.2) y con aliasing (Figura 2.3), observando como el segundo caso se caracteriza por un solapamiento de las réplicas de la señal fundamental. [4]

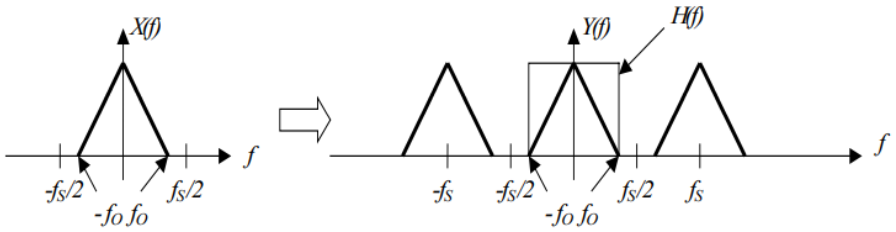


Figura 2.2 Señal muestreada sin aliasing. Fuente:[4].

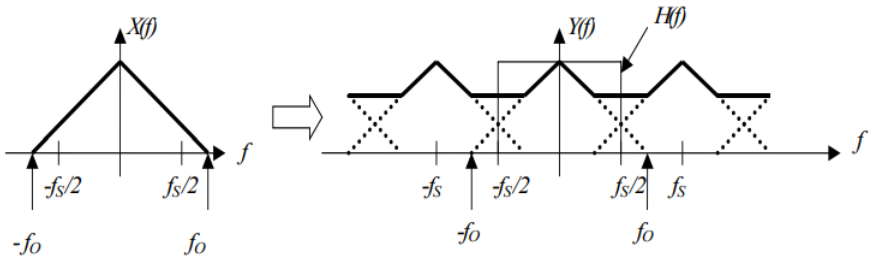


Figura 2.3 Señal muestreada con aliasing. Fuente:[4].

En la Figura 2.2 se observa como las réplicas no se solapan entre sí, lo cual permite que usando un filtro H(f) y por tanto manteniendo únicamente la parte de la señal que se encuentra en torno a la frecuencia nula, se reconstruye la señal original. Por otro lado, la Figura 2.3 sí exhibe una superposición en las réplicas en frecuencia lo cual impide que al usar un filtro se recupere la señal x(t).

El filtro mencionado anteriormente, H(f), empleado para reconstruir la señal deberá ser tal que no altere la señal para valores $f \leq |f_o|$ y que elimine las réplicas para frecuencias fuera de ese intervalo.

2.3 Teorma de Nyquist

A partir de lo expuesto en la Figura 2.2 y Figura 2.3 se define un Teorema de muestreo o Teorema de Nyquist el cual establece un criterio para la correcta elección de f_s. De este modo, la señal x(t) se puede recuperar de manera unívoca a partir de su señal muestreada siempre y cuando se cumplan las expresiones (2.3) y (2.4). [2]

$$f_{s,Nyquist} = 2f_o, \tag{2.3}$$

$$f_s \geq f_{s,Nyquist} \tag{2.4}$$

2.4 Complemento a dos

Se realiza un pequeño paréntesis para definir el Complemento a dos (CA2) de un número binario debido a que será mencionado a lo largo de toda la memoria. La expresión que define el CA2 de un número binario x de m bits es (2.5).

$$\begin{cases} 2^m - x & si \quad x < 0 \\ 2^m & si \quad x \geq 0 \end{cases} \tag{2.5}$$

Siendo ahora x $\in [-2^{m-1},\ 2^{m-1}-1]$.

A modo de ejemplo, se calcula los posibles valores en CA2 que un número binario de 3 bits toma.

Tabla 2.1 Ejemplo CA2 para un número binario de 3 bits.

x_2	x_1	x_0	$x\|_{10}$
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

2.5 Cuantización de una señal analógica

El siguiente paso a realizar para la conversión de una señal analógica (continua) en una señal digital (discreta) es la cuantización de las señales muestreadas como se expone en el esquema Figura 2.4. El dispositivo que realiza dicho procesamiento recibe el nombre de cuantizador y el proceso que realiza se resume en asociar a cada valor del conjunto de muestras uno de los Q niveles del cuantizador. De este modo, el rango de valores que puede representar un cuantizador va desde 0 hasta Q-1, siendo $Q = 2^B$, con B=número de bits. Por ejemplo, un cuantizador Single-Bit, como el que se empleará en este proyecto ($B = 1$) tendrá 2 niveles de cuantización ($Q = 2$).

Figura 2.4 Fuente:[2].

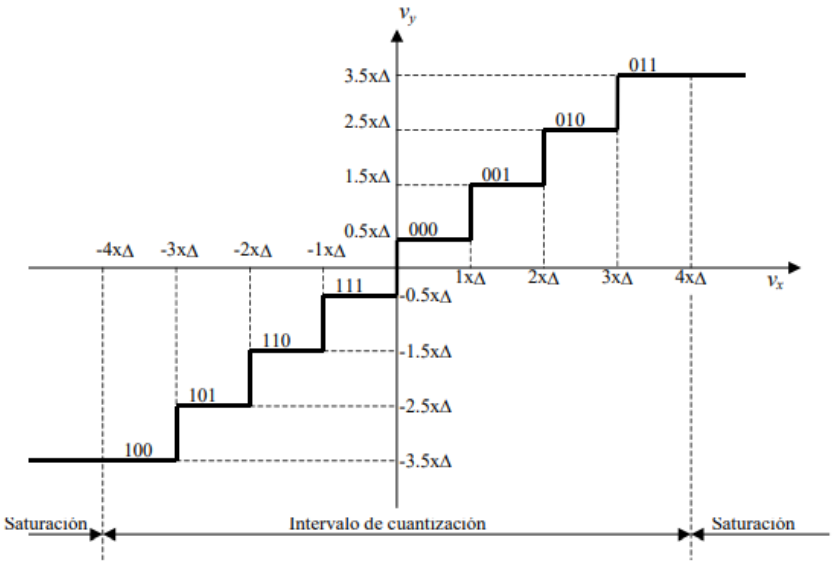


Figura 2.5 Representación gráfica de un cuantizador de 3 bits con codificación en CA2. Fuente:[4].

La señal muestreada es discreta en el tiempo aunque posee continuidad en lo que respecta a sus valores de amplitud. Dicha amplitud se discretiza mediante el cuantizador asignando un mismo nivel de cuantización a todo conjunto de valores continuos que se encuentrem dentro de un determinado rango de valores. De este modo, cada valor de la señal muestreada tendrá asociado un cierto nivel.

Gráficamente queda representado en la Figura 2.5 donde se puede observar como la señal de entrada, denominada v_x puede tomar valores $\forall x \in (-\infty, +\infty)$ mientras que la señal v_y toma valores discretos asociados al número de bits del cuantizador y al paso de cuantización Δ. Además, se distinguen 2 zonas : un intervalo dónde se lleva a cabo el proceso de asignación de niveles y denominado intervalo de cuantización, y una segunda zona tal que toda muestra de la señal que supere un determinado valor, será saturada a un nivel.

Finalmente, se realiza un último paso de codificación tal y como muestra la Figura 2.4 donde a cada nivel de cuantización se le asigna un código binario cuyo número de bits corresponde al del cuantizador como se refleja en la Figura 2.5.

Paso de cuantización

Al igual que en [4], se definen los límites del intervalo de cuantización como V_R y V_s siendo el paso de cuantización definido por la ecuación (2.6)

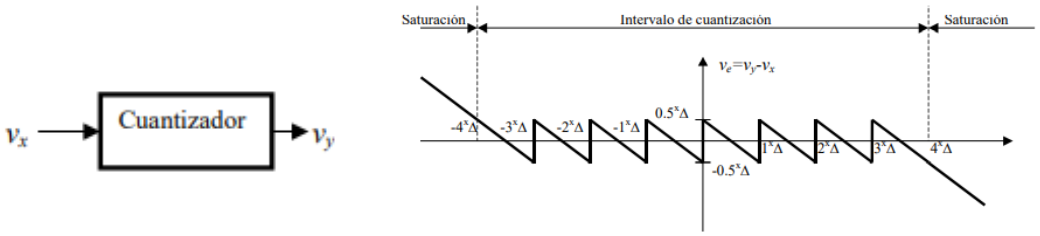
$$\Delta = \frac{V_S - V_R}{2} \tag{2.6}$$

Tipos de cuantizador

En cuanto a los tipos de cuantizadores se definen dos, denominados en inglés *midrise y midthread* en base a si existe o no un nivel situado en 0, siendo el correspondiente a la Figura 2.5 de este primer tipo.
En el modelo *Simulink* expuesto en el siguiente capítulo se emplea un cuantizador uniforme tipo mid-rise tal que no posee un nivel en 0.

Error de cuantización

Cabe destacar el hecho de que tanto el proceso de muestreo como el de cuantización son potenciales fuentes de error debido a la pérdida de información que ambas operaciones conllevan. Esta pérdida de información será menor cuanto mejor se hayan diseñado los componentes del convertidor A/D. Se define como error de cuantización a la diferencia entre la señal de entrada y salida del cuantizador.



(a) (b)

Figura 2.6 Representación del bloque cuantizador de 3 bits y la variación del error de cuantización (v_e). Fuente:[4].

Aplicando la definición del error a la Figura 2.5 se observa como v_e toma valores acotados entre $\pm\frac{\Delta}{2}$ cuando la señal está dentro del intervalo de cuantización, y crece cuando se encuentra fuera de dicho intervalo (Figura 2.6).

En conclusión, para lograr una buena calidad del cuantizador, y por lo tanto un error pequeño de cuantización, se debe diseñar el componente de tal manera que el intervalo de cuantización abarque la señal de entrada al completo.

2.6 Convertidor D/A

Un convertidor digital analógico es un circuito electrónico que recibe por entrada una secuencia digital y discreta en el tiempo y genera como salida una señal analógica y continua en el tiempo.

2.7 Transformada Z

La transformada z se empleará en la arquitectura del circuito SDM a analizar en este proyecto y da lugar a una expresión que permite realizar un análisis frecuencial de señales discretas en el tiempo ($x[n]$). La transformación se describe mediante la expresión (2.7) a partir de una secuencia x[n], donde n es un número entero y z una variable compleja.

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n} \tag{2.7}$$

También se puede determinar una relación (2.8) entre la transformada z y la transformada de Fourier .

$$X(e^{jw}) = X(z) \tag{2.8}$$

2.8 SNR

El parámetro SNR medido en dB (decibelios), *signal to noise ratio* en inglés, equivale al cociente entre la información deseada o potencia de la señal y la no deseada o potencia del fondo de ruido como se muestra en (2.9) y siendo P_{Signal} la potencia de la señal y P_{Noise} la potencia asociada al ruido.

$$SNR(dB) = 10log_{10}\frac{P_{Signal}}{P_{Noise}} \tag{2.9}$$

De este modo, un valor de SNR superior a 0 dB indica que el nivel de señal supera al del fondo de ruido. Luego, a mayor SNR, mayor calidad tendrá la señala analizada.

Además, existe una expresión del SNR (2.10) asociada al CAD. [4]

$$SNR(dB) = 6.02 \cdot N_{bits} + 1.761 \tag{2.10}$$

2.9 HD2

El parámetro HD2, *second harmonic distortion* en inglés, corresponde a la cantidad de contenido asociado al 2^o armónico existente en la señal con respecto a la frecuencia fundamental. Su valor se calcula como se indica en (2.11) donde V_{1F} es el valor de pico del armónico fundamental y V_{2F} el valor de pico asociado al segundo armóinico de la señal.

$$HD2(dB) = 20log_{10}\frac{V_{1F}}{V_{2F}} \tag{2.11}$$

2.10 Convertidor sobremuestreado

Tal y como se ha mencionado en la introducción, el mundo digital es cada vez más predominante frente al analógico en tanto que estas señales permiten desarrollar circuitos de alta complejidad, muy fiables y rápidos, mediante dispositivos muy pequeños y simples. Por ello, la tecnología DSP progresa cada año, dando lugar a un procesado digital cada vez más rápido y la necesidad de convertidores D/A que trabajen a una alta velocidad puesto que estos elementos hacen de interfaz entre el mundo digital y las señales del mundo real que son analógicas.

Se conoce por convertidor sobremuestreado a aquel que funciona a una frecuencia de muestreo superior a la de Nyquist. A partir de su expresión de la densidad espectral de potencia (2.12) [4] se observa como cuanto mayor sea f_s menor será la densidad espectral de potencia por cada frecuencia. Esto último se traduce en que la potencia de ruido se extienda hasta zonas de alta frecuencia dónde al no haber señal, resulta fácil eliminar el ruido mediante un filtro tipo paso bajo.

$$|Q(f)| = \frac{\Delta^2}{12f_s} \tag{2.12}$$

Sabiendo que la potencia de ruido de cuantización se reparte desde $-\frac{f_s}{2}$ hasta $+\frac{f_s}{2}$ se obtiene la representación de la Figura 2.7.

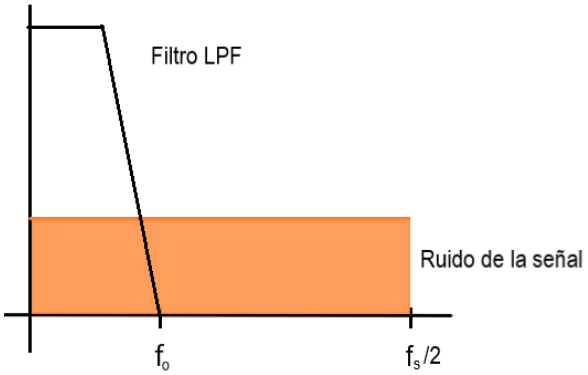


Figura 2.7 Espectro de ruido de una señal sobremuestreada con filtro aplicado y ruido en color naranja.

Por otro lado, a partir del cálculo de la potenica de ruido filtrada y la definición del factor de sobremuestreo (OSR) (2.13) se obtiene una relación entre los valores de SNR del convertidor sobremuestreado y aquel sin sobremuestreo (2.14).

$$OSR = \frac{f_s}{f_{s,Nyquist}} = \frac{f_s}{2B} \tag{2.13}$$

$$SNR_{oversampling} = SNR \cdot OSR \tag{2.14}$$

2.11 Convertidor Sigma Delta

Una vez definidos los convertidores sobremuestreados se puede abordar los convertidores tipo ΣΔ al ser estos una variante de los primeros.

El SDM se caracteriza por traspasar ruido localizado en bajas frecuencias a altas , fenómeno conocido por *Noise Shaping*. De este modo, la eliminación del ruido mediante un filtro es más eficaz que en el caso básico de convertidores sobremuestreados. Los moduladores sigma delta se forman mediante un bucle de realimentación donde se colocan en cascada un integrador o sumador (Σ) con un restador (Δ) que compara los valores de la muestra a la entrada del circuito con su valor esperado como representa la Figura 2.8.

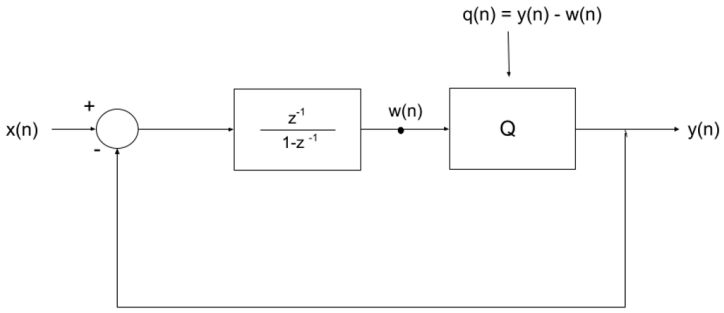


Figura 2.8 Modulador Sigma Delta de 1^{er} orden.

A partir del esquema mostrado en la Figura 2.8 se aprecia la siguiente relación, siendo $q(n)$ el error asociado al bloque cuantizador:

$$Y(z) = z^{-1}X(z) + (1-z^{-1})Q(z) \tag{2.15}$$

$$y(n) = x(n-1) + [q(n) - q(n-1)] \tag{2.16}$$

De este modo, el ruido debido al error de cuantización en el SDM sigue la expresión: $q(n) - q(n-1)$ lo que transformado al dominio z se convierte en $(1-z^{-1})Q(z)$ y empleando la relación (2.8) con $w = 2\pi f T_s$ se obtiene una fórmula asociada a la conformación de la densidad espectral del ruido de cuantización.

Esto último queda representado en la Figura 2.9 dónde se identifica un filtro paso bajo (LPF) que elimina el ruido a altas frecuencias (señalado mediante rayas de color naranja) y deja pasar el ruido asociado a frecuencias bajas (señalado en color naranja). A partir de dicha representación, y comparándola con Figura 2.7 se observa como tras aplicar el LPF se elimina una mayor cantidad de ruido de la señal quedando solo una pequeña proporción en la zona de bajas frecuencias.

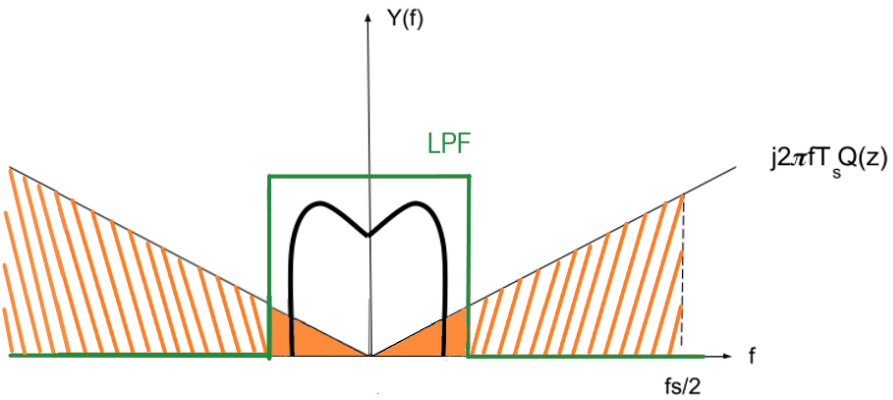


Figura 2.9 Representación espectral de la señal y la potencia de ruido de cuantización.

El orden del modulador va de la mano del número de bloques integradores que este posea. Además, a mayor orden, mayor resolución del SDM.

Extrapolando la relación obtenida para el caso de primer orden a uno de segundo orden y sabiendo que $(1-z^{-1})^2$ en el dominio z equivale a una conformación proporcional a $j2\pi fT_s$, luego la conformación del ruido varía con el orden del modulador como se muestra a continuación (Figura 2.10). [5]

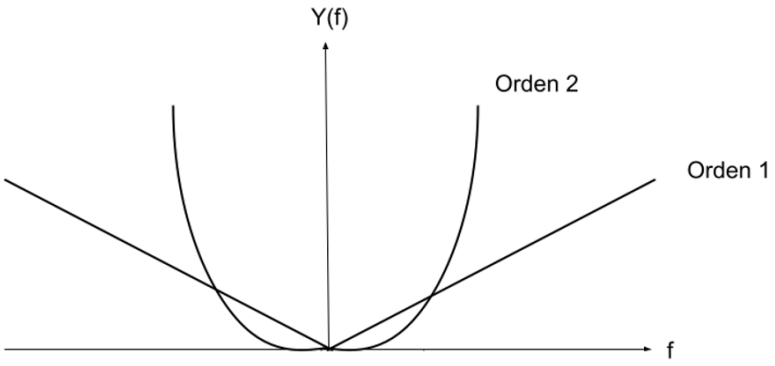


Figura 2.10 Evolución del error de cuantización según el orden del SDM.

Así pues, se observa como a mayor orden del SDM más ruido es trasladado a altas frecuencias y podrá ser eliminado posteriormente mediante un filtro. En este proyecto se trabaja con un modulador ΣΔ de segundo orden.

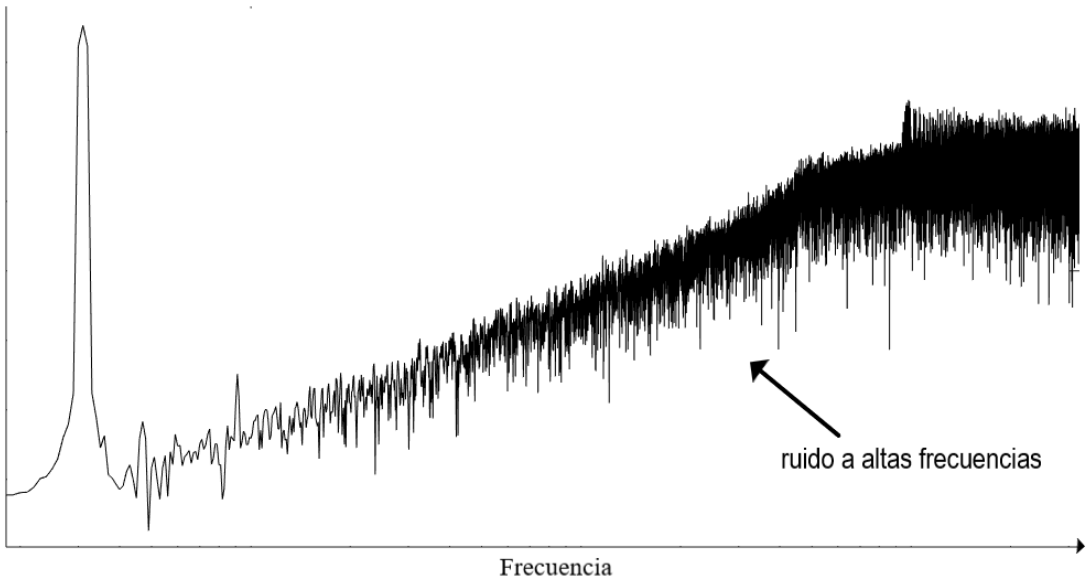


Figura 2.11 Espectro característico del SDM.

El espectro característico del SDM de segundo orden es el mostrado en Figura 2.11 y se comparará con los resultados obtenidos cuando implementemos el circuito en los distintos Software (SW) para verificar si el SDM diseñado funciona correctamente.

En este proyecto se obtendrá a la salida del modulador un pulso cuadrado el cual tras ser filtrado dará lugar a una señal senoidal que deberá ser igual a la de la entrada. La transformación a señal senoidal se debe al uso del filtro que elimina ruido a altas frecuencias como se ejemplifica en la Figura 2.12 y resulta en la suma de los valores en un intervalo N (se promedia), lo cuál se manifiesta como una señal sinusoidal a la salida del

modulador con poco ruido.

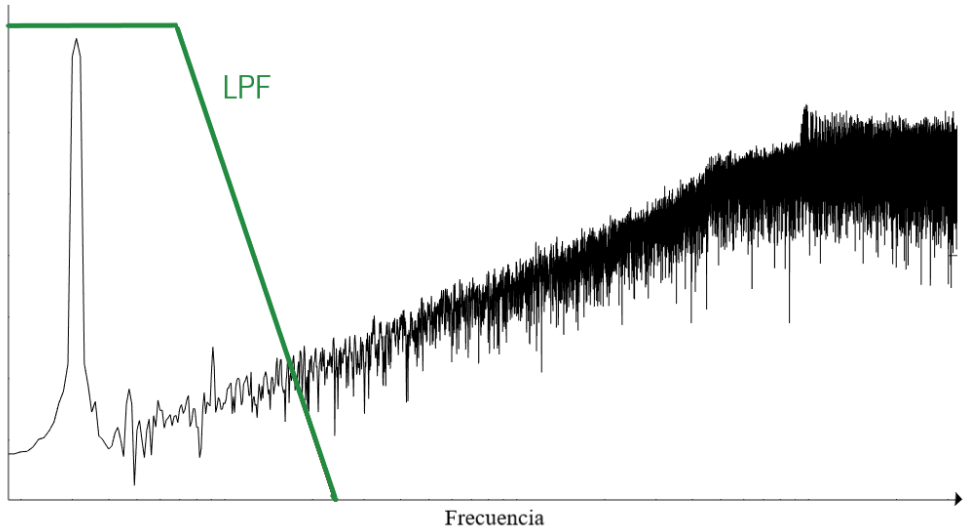


Figura 2.12 Espectro característico del SDM con filtro paso bajo.

2.12 FPGA

FPGA es el acrónimo en inglés de matriz de puertas lógicas programables en campo, *Field Programmable Gate Arrays*, y está formada por dispositivos semiconductores colocados alrededor de una matriz de bloques programables y conectados mediante interruptores también programables.

Las FPGA pueden ser reprogramadas por el usuario tras su fabricación. De hecho, en este proyecto se va a programar para 2 casos distintos variando la señal de entrada tal y como se verá en la última sección relacionada con la toma de medidas experimentales.

Se caracterizan por su rapidez, complejidad no muy elevada, capacidad de realizar diseños de gran volumen y por ser reprogramables. Por ello, se emplean en multitud de aplicaciones y en sectores tales como la industria aeroespacial, automovilística, defensa, medicina entre otros.

3 Simulación MATLAB&SIMULINK

Esta sección tiene por objeto describir el estudio y simulaciones previas a la implementación del circuito SDM en la FPGA y a partir de los cuales quedan determinados los parámetros a usar.

Se parte de un análisis funcional realizado en un SW con lenguaje de alto nivel, en este caso se usa *MATLAB&Simulink* para determinar la arquitectura que cumpla con las prestaciones deseadas. Dichas prestaciones se determinan a partir del análisis de la arquitectura asociada al caso ideal del SDM y a partir de ella se desarrollan distintas versiones de las cuáles se selecciona la más adecuada a partir un estudio exhaustivo de la señal obtenida a la salida de cada uno. Así pues, aquella arquitectura que obtenga los mismos valores de SNR y HD2 que el circuito ideal, será la elegida para su posterior implementación en VHDL.

En resumen, el objetivo de esta sección es determinar los módulos que conforman el circuito a implementar (sumadores, restadores, saturadores, registros...) y el número de bits de las señales y las entradas y salidas del modulador Digital/Analógico.

Esta primera etapa del proceso de realización del modulador es fundamental ya que una buena determinación de los parámetros y de los componentes deriva en un circuito implementado correctamente, sin errores y que cumple las características deseadas.

3.1 Modelo Simulink del Modulador Sigma Delta

El esquema utilizado en *Simulink* consta de cuatro variantes de la arquitectura del SDM (Figura 3.1). A destacar, una de las alternativas corresponde al circuito SDM ideal (y1) de tal modo que las prestaciones obtenidas a partir de él serán las buscadas en el resto de versiones del circuito SDM. El resto (y2,y3,y4), son distintas configuraciones posibles que resultan de poner uno o más saturadores con distinta saturación.

La señal de entrada es una onda senoidal con una amplitud determinada según si el estudio se realiza para el caso de 5 dB o 6 dB por debajo del máximo de la señal. Dicha onda se amplifica y posteriormente se redondea a un valor entero en el bloque cuantizador debido a que en el modulador Digital/Analógico que se desarrolla en este proyecto se trabaja con aritmética entera.

Descripción general del circuito

Mediante un enfoque general del modelo de *Simulink* a emplear, se destacan los siguientes elementos:

- La fuente utilizada genera una señal senoidal con la amplitud definida, la cuál será a continuación amplificada.
- Mediante un cuantizador, se transforma la señal analógica a digital. Dicho cuantizador tiene como parámetro un intervalo de cuantización de 1, tal que la salida corresponde con la señal de entrada redondeada. De modo que tras la cuantización se pasa a trabajar con aritmética entera.
- Se emplean diversos scopes para representar las señales a la salida de los distintos bloques SDM tras ser filtrados mediante un filtro de Butterworth frente a las señales a la entrada del convertidor de tal manera que si ambas coinciden significa que la señal se recupera correctamente a la salida.

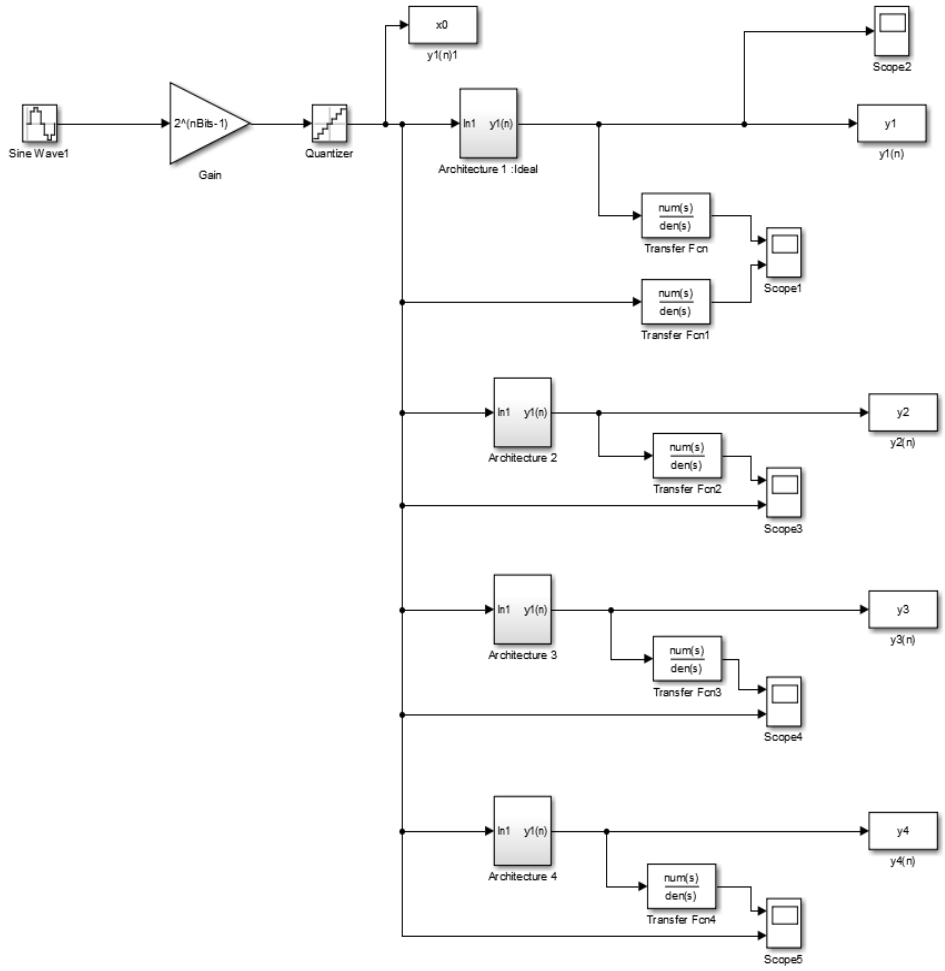


Figura 3.1 Modelo Simulink del SDM.

3.1.1 Descripción particular

En esta etapa inicial del proyecto se saca provecho de las ventajas que ofrece el uso de *Simulink* a la hora de poder realizar varias simulaciones distintas gracias a la sencillez asociada a que los bloques funcionales ya estén determinados, cosa que no ocurrirá en el SW *Vivado*. De este modo, se pueden probar distintas arquitecturas y parámetros para seleccionar en vista a las prestaciones que ofrecen, la arquitectura a implementar.

Arquitectura 1

La primera arquitectura corresponde al esquema del modulador ideal. Trabajando con transformada z, se obtiene:

$$Y(z) = X(z) + (1 - 2z^{-1} + z^{-2})Q(z) = X(z) + (1 - z^{-1})^2 Q(z) \tag{3.1}$$

A partir de la expresión 3.1 se observa como efectivamente se trata de un modulador ΣΔ de segundo orden. Además, la arquitectura de la Figura 3.2 implementa la expresión 3.1.

A partir de un análisis funcional de la arquitectura ideal (Figura 3.2) se determina el valor de OSR : $OSR = 64$. A partir de dicho parámetro, se obtiene la frecuencia de muestreo tal que $f_{SDM} = 2.56MHz$ la cuál se puede obtener usando un valor de Ndiv=40 en el divisor de frecuencias, tal y como se demuestra en (3.2).

$$\text{Si } \; Ndiv = 40, \; entonces : f_{SDM} = \frac{T_{clk}}{Ndiv} = \frac{100MHz}{40} = 2.5MHz \tag{3.2}$$

De este modo, el modulador Sigma Delta tendrá asociado un período 40 veces mayor que el de la señal asociada al reloj maestro : $T_{SDM} = 400ns$.

Finalmente, aplicando la ecuación (2.13) se obtiene el siguiente valor para el ancho de banda: $BW = 19.53kHz$, quedando así definidas las especificaciones del modulador Sigma Delta.

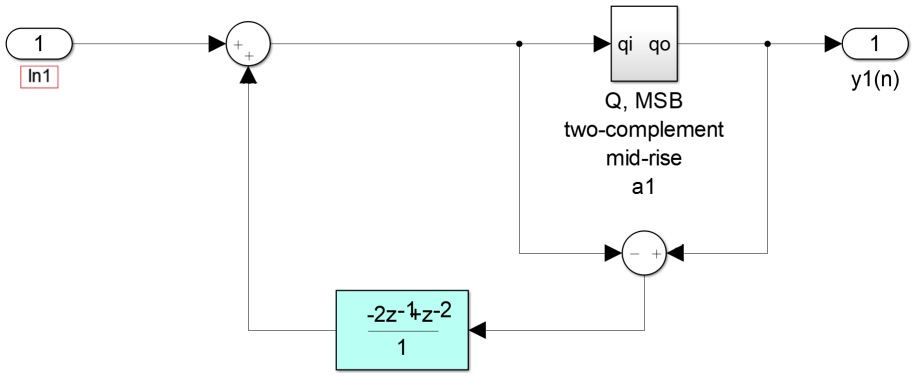


Figura 3.2 Modelo Simulink de la arquitectura ideal del modulador Digital/Analógico.

El uso de 14 bits a la entrada frente a los 12 bits efectivos definidos en la introducción de este proyecto se justifica en base a los resultados obtenidos tras aplicar la ecuación (2.10) para la resolución dada (3.3).

$$SNR_{oversampling} = N_{eff} \cdot 6.02 + 1.761 = 74dB \tag{3.3}$$

Combinando las ecuaciones (2.14) y (3.3) para un valor de 14 bits a la entrada se obtiene en (3.4) que la SNR asociada a la señal de entrada de 14 bits es igual a 104 dB.

$$SNR_{entrada} = (N_{bit} \cdot 6.02 + 1.761) + 10log_{10}OSR = 86 + 18 = 104dB \tag{3.4}$$

Así pues, aunque el valor de SNR para el número de bits a la entrada es mucho mayor que el asociado al número de bits efectivo (3.3) que se desea obtener en el SDM a realizar, se elige trabajar con dicho valor (3.4) sobremuestreado para que los armónicos producidos por la cuantización a la entrada del circuito se encuentren muy por debajo de los posibles armónicos que pueda generar el modulador Sigma Delta que se desea probar y testar.

Arquitectura 2

A continuación, se replica la arquitectura 1 (Figura 3.2) con una serie de modificaciones. Entre estas modificaciones se destaca el empleo de saturadores para controlar el buen funcionamiento del SDM en el caso no ideal. Además, cada bloque de saturación lleva asociado un componente *Scope* que representa la resta de la señal de entrada y salida del saturador. De este modo se verifica que se ha seleccionado de forma adecuada el número de bits al que se realiza la saturación comprobando que la señal en el *Scope* tiene un valor constante e igual a 0 ya que esto significa que no se produce ninguna variación de la señal tras pasar por dicho bloque y consecuentemente ninguna pérdida de información.

Por otro lado, tanto en esta como en el resto de arquitecturas siguientes se ha modificado el bloque marcado en color celeste de la Figura 3.2 por los bloques celestes de la Figura 3.3 viendo como ambos son equivalentes. Además, la señal de entrada, común a todas las arquitecturas, se ha recuadrado en rojo.

Con todo lo anterior, se obtiene una de las posibles estructuras funcionales que se describiría en VHDL quedando definidos sus parámetros, componentes electrónicos (sumadores, restadores, saturadores...) y el número de bits asociado a las señales de cada bus del circuito, habiéndose determinado esto último a partir del uso de saturadores.

Finalmente, la saturación que se ha considerado necesaria para cada uno de los bloques saturadores es de 17 bits para todos, excepto para el bloque Sat3 al que le corresponden 18 bits de saturación. De este modo, las señales de la arquitectura 2 son señales binarias de 17 bits salvo la señal de salida del componenente Sat3.

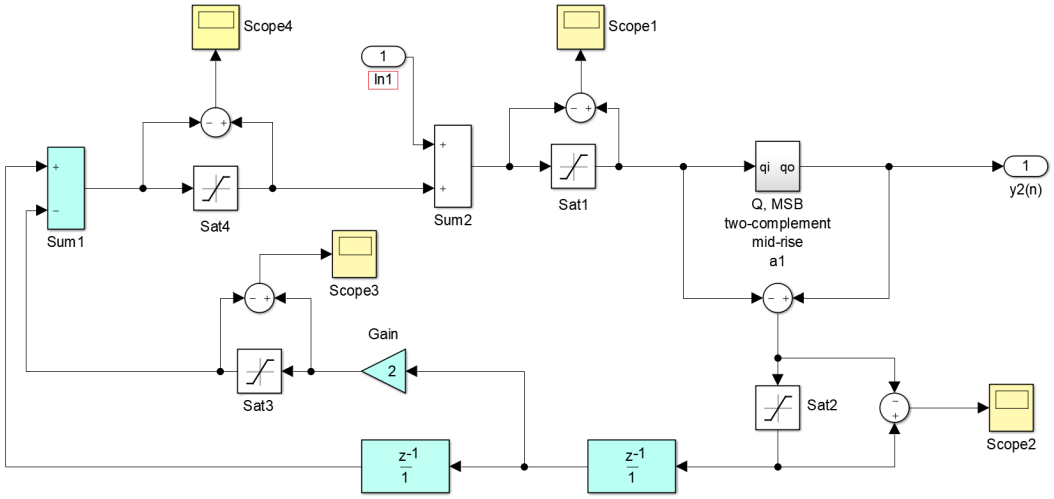


Figura 3.3 Modelo Simulink de la arquitectura 2 del modulador Digital/Analógico empleando 4 saturadores.

Arquitectura 3

Tras realizar un análisis funcional de la Figura 3.4, se concluye que solo es necesario el uso de un saturador, simplificando así considerablemente el circuito y su implementación en VHDL al reducir el número de componentes necesarios.

Por tanto, la arquitectura número 3 (Figura 3.4), es idéntica a la anterior (Figura 3.3) solo que se eliminan todos los bloques saturadores excepto el denominado Sat1. Dicho bloque satura a 17 bits al igual que hacia en la configuración 2.

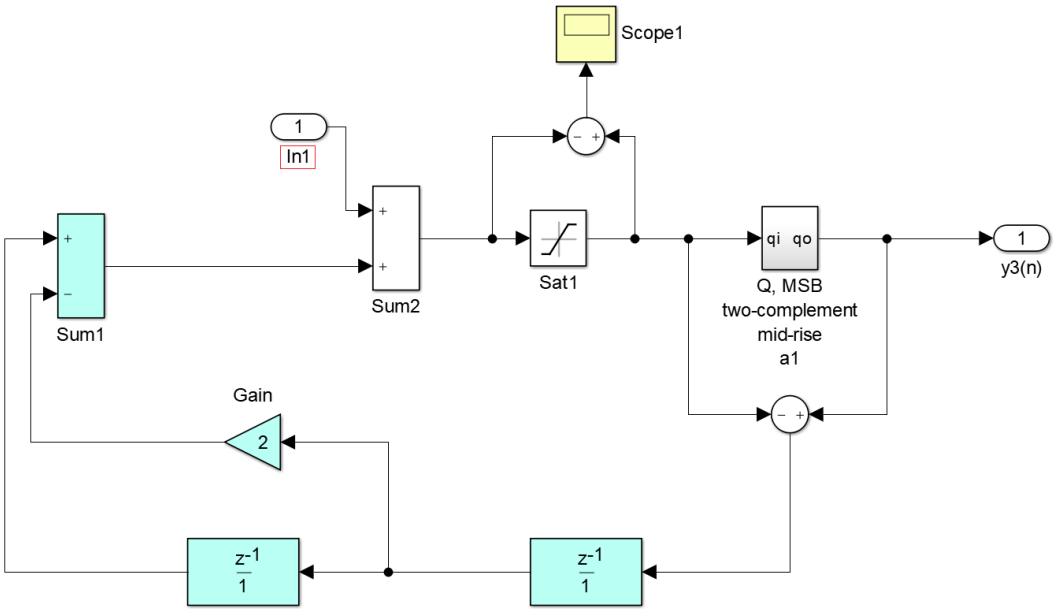


Figura 3.4 Modelo Simulink de la arquitectura 3 del modulador Digital/Analógico empleando 1 saturador de 17 bits.

Arquitectura 4

Tras comprobar que para la arquitectura 3 (Figura 3.4) los resultados obtenidos cumplen los criterios deseados (en base al valor de SNR de la señal de salida), se prueba a reducir el valor de la saturación a 16 bits proponiendo una última variante respecto de la arquitectura original.

Sin embargo, la Tabla 3.1 muestra como esta modificiación resulta en peores prestaciones y por lo tanto una mala solución a la implementación en VHDL del circuito.

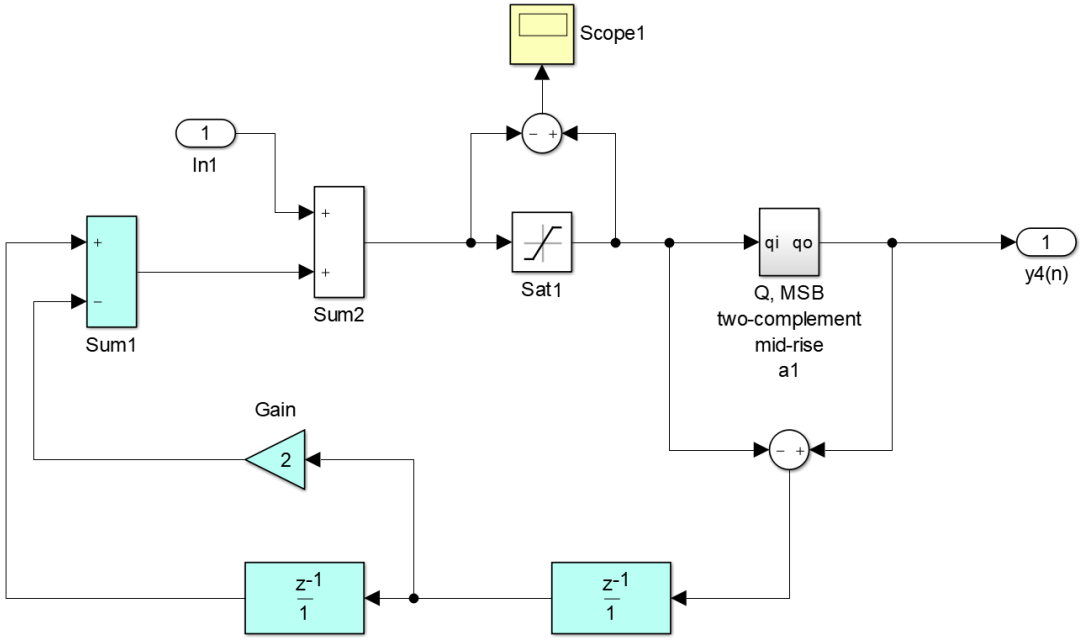


Figura 3.5 Modelo Simulink de la arquitectura 3 del modulador Digital/Analógico empleando 1 saturador de 16 bits.

3.2 Resultados simulación

La simulación funcional asociada al esquema *Simulink* de la Figura 3.1 se realiza para dos casos distintos según el valor de caída de la amplitud a la entrada con respecto a su valor máximo. Dicha amplitud máxima se define teniendo en cuenta que la señal a la entrada senoidal tiene como valores máximos 1 y -1 y posteriormente se amplifica con una ganancia igual a $2^{14-1} = 2^{13}$ siendo ese el valor máximo de la señal. De este modo se obtiene la relación entre el valor de la amplitud de la entrada al modulador Sigma/Delta y el valor del fondo de escala (fs, del inglés *full scale*) como se ejemplifica en (3.5)

$$-5dB(fs) = 10log_{10}\frac{A}{A_{max}} = 10log_{10}\frac{A}{2^{13}} \Rightarrow A = 2^{13}10^{\frac{-5}{20}} \tag{3.5}$$

La ecuación (3.5) es aplicable a cualquier valor. En nuestro caso se realiza un análisis para una señal de entrada con amplitud -5dB y -6dB menos que la amplitud máxima cuyos resultados analíticos se exponen en Tabla 3.1 y los resultados gráficos se muestran en Figura 3.6 y Figura 3.7 respectivamente.

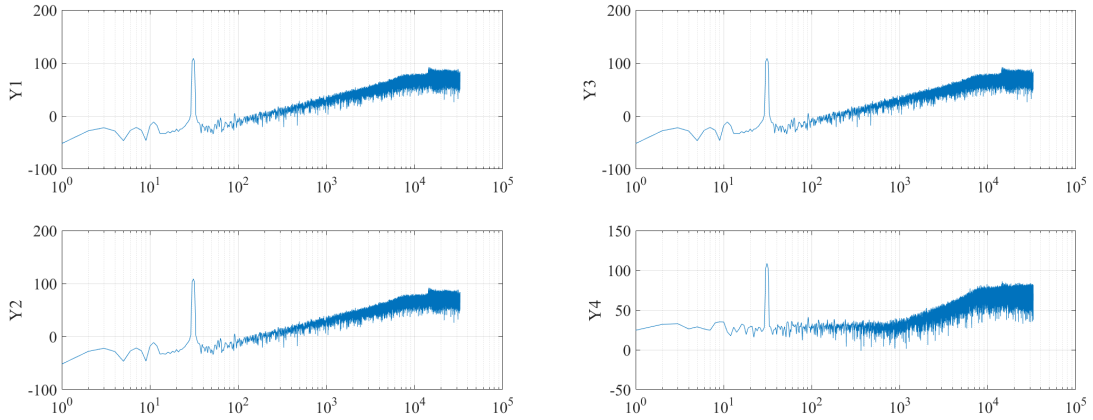


Figura 3.6 Representación espectral de las salidas y_k con k=1,2,3 y 4, correspondientes con las PSD de las señales y_k en la Figura 3.1 para una caída de 5 dB en la amplitud de la señal seno a la entrada.

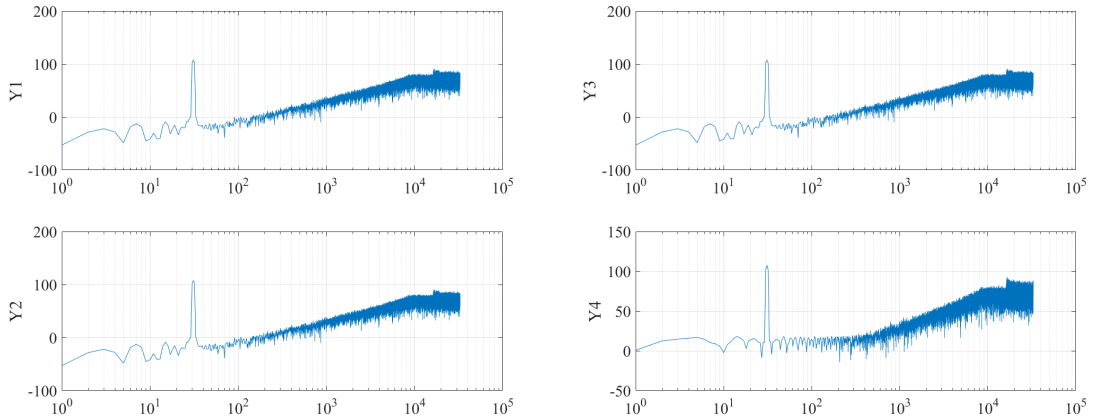


Figura 3.7 Representación espectral de las salidas y_k con k=1,2,3 y 4, correspondientes con las PSD de las señales y_k en la Figura 3.1 para una caída de 6 dB en la amplitud de la señal de entrada.

Tabla 3.1 Valor de SNR de las 4 arquitecturas para distintas caídas de amplitud.

	SNR (dB)			
	Arquitectura 1 (caso ideal)	Arquitectura 2	Arquitectura 3	Arquitectura 4
-5 dB	69.858287	69.858287	69.858287	52.828955
-6 dB	69.928790	69.928790	69.928790	67.017321

3.3 Conclusión

En vista a los resultados expuestos en la Tabla 3.1, se decide trabajar desde ahora con la configuración asociada a la 3^a arquitectura (Figura 3.4) ya que, aunque tanto esta como la segunda variante del SDM, ofrecen unas prestaciones equivalentes al caso ideal (Arquitectura 1), el tercer circuito lo realiza de un modo más sencillo ya que reduce el número de saturadores a usar a uno frente a los cuatro empleados en el circuito

2 (Figura 3.3). Por otro lado, queda descartado emplear la última arquitectura (Figura 3.5) ya que ofrece una calidad inferior al caso ideal.

4 Simulación VIVADO

En el siguiente capítulo de esta memoria se implementa la arquitectura escogida en la sección anterior. Para ello, se desarrollan los distintos códigos (localizados en el Apéndice II) en lenguaje VHDL asociados a cada componente que componen el circuito de la Figura 3.4. Dichos códigos requerirán de los siguientes parámetros, cuyos valores se explicarán detalladamentelos en la sección asociada al código dónde se empleen:

- Nb=14
- Ndiv = 40
- Ncont = 6
- Nduty=21
- Nmuestras = 192
- Nsat = 17

Siendo los parámetros Nb, Ndiv, Nsat definidos a partir de las simulaciones realizadas previamente con *MATLAB&Simulink*.

Es importante destacar el hecho de que se trabajará en Complemento a 2. De este modo, al introducir valores de una señal exterior, estos deberán ser transformados a CA2 antes de entrar en nuestro circuito y lo contrario ocurrirá al extraer señales para analizarlas mediante Matlab.

Por último, para llevar a cabo esta parte del proyecto se han usado esencialmente dos manuales de apoyo [3][6].

4.1 Esquema SDM

De la arquitectura 3 mostrada en la Figura 3.4 se puede obtener el esquema circuital de la Figura 4.1 en el que se indican los nombres de los módulos (CONT, Sum, Res, Sat...), los de las señales (entSD, salidasum, salidasat....) así como su resolución (14 bits, 19 bits, 20 bits...).

Cabe mencionar que el caso mostrado es para un valor de número de bits igual a 14, pero al haberse realizado el código de forma parametrizada, se podría trabajar con cualquier otro valor simplemente cambiando la constante Nb dentro del código.

4.1.1 Descripción general del circuito

A partir del circuito analizado inicialmente en *MATLAB*, se desarrolla el mostrado en Figura 4.1 el cual ha sido sutilmente modificado para poderlo implementar en VHDL.

La señal de entrada se genera internamente en el dispositivo programable en lugar de ser introducida desde el exterior. Para su generación se emplea una Lookup Table con valores asociados a las muestras de un cuarto de período de una señal senoidal, generándose a partir de esos valores una señal senoidal que entra en el SDM.

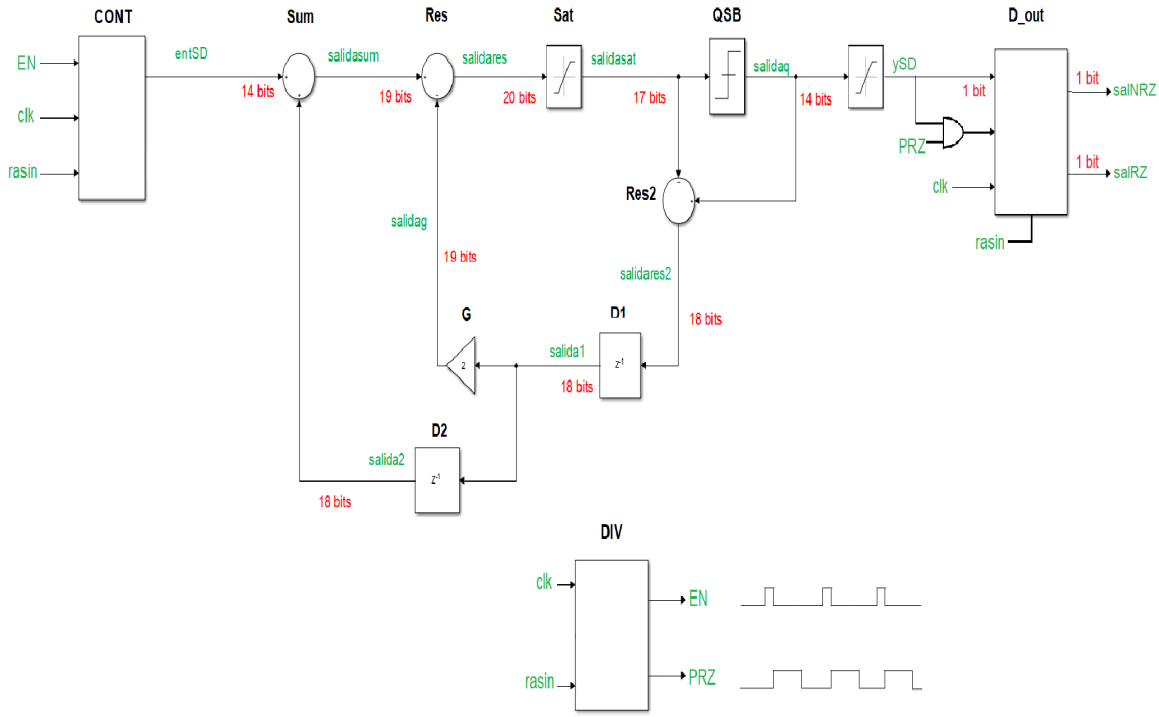


Figura 4.1 Esquema del SDM implentado en VHDL.

Además, se generan también dos pulsos (EN y PRZ) que definen el comportamiento del modulador sigma delta para una configuración con retorno a cero y sin retorno a cero, dado que la actualización de las entradas en los dispositivos secuenciales dependen de los valores de dichos pulsos.

Finalmente, mencionar que a la salida del SDM se obtiene una señal binaria de 14 bits la cual posteriormente se convierte en una señal de 1 bit para obtener una señal pulso cuadrada que toma valores entre '1'y '0'. A su vez, de dicha señal de un bit a la salida se obtendrá su homóloga con retorno a cero para poder así analizar y comparar los resultados asociados a cada caso.

El valor del número de bits de cada señal del circuito se ha determinado realizando un análisis en *MATLAB&Simulink* de cada componente observando la variación del número de bits de la señal de salida con respecto a la de entrada en cada uno de ellos.

4.2 Señal reloj y reset asíncrono

El circuito con el que se va a trabajar esta sincronizado con un reloj maestro de período $T_{clk} = 10\ ns$ y dispone también de una señal reset asíncrona tal que cuando esta tenga un nivel alto (rasin='1') se hagan '0' las señales de los dispositivos con lógica secuencial del circuito.

4.3 Pulsos Enable y PRZ

4.3.1 Pulso EN

En primer lugar, partiendo del objetivo de definir nuestro SDM tal que posea una frecuencia $f_{SDM} = 2.5MHz$ como se explicó en la sección anterior, se define la señal Enable, EN, que actualiza los valores de todos los registros empleados en el circuito de tal modo que cuando EN='1'y se da un flanco de subida del reloj, la entrada del registro D se actualiza como se muestra en el Código 4.1.

Código 4.1 Fragmento del código VHDL.

```
D<= entrada when EN='1'
else Q;
```

Esta señal se genera mediante un divisor de frecuencia tal que la señal del SDM se divide por un factor denominado $Ndiv$: $f_{SDM} = \frac{f_{clk}}{Ndiv}$. Haciendo así que el modulador trabaje a una frecuencia inferior a la del reloj maestro. En nuestro caso, $Ndiv = 40$ tal que cada 40 pulsos de reloj $EN =' 1'$. De este modo, para lograr generar la señal enable se emplea un registro que hace de contador con un comparador a su salida tal y como se muestra en la Figura 4.3.

A modo de ejemplo ilustrativo y para facilitar la comprensión, se analiza como varía la señal EN para un valor de Ndiv=4 (Figura 4.2).

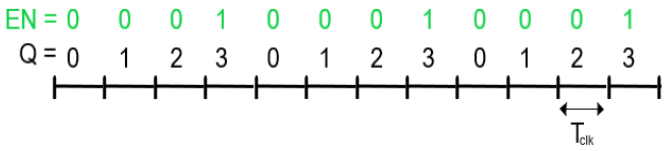


Figura 4.2 Comportamiento señal EN para Ndiv=4.

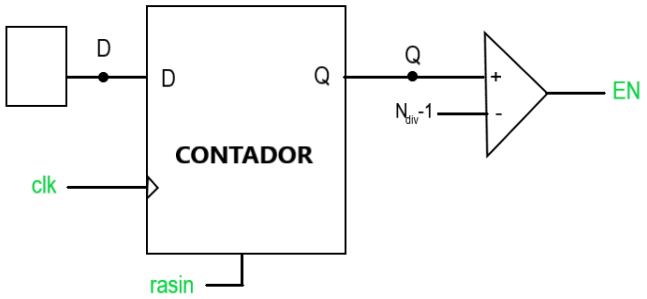


Figura 4.3 Representación esquemática de cómo generar la señal EN.

- Entrada : clk, rasin
- Salida : EN
- Parámetro : Ndiv

El divisor de frecuencia es un componente más, o *entity*, que se añade al fichero top generándose así la señal EN dentro de este fichero, se hace de este modo en lugar de generarla a partir de estímulos porque el circuito real tiene que generarla así.

4.3.2 Pulso PRZ

Definición

La señal con retorno a cero ('Return to zero', RZ) toma valor '0' en algún instante de tiempo dentro del período de muestreo del SDM cuando se dan varios valores '1' consecutivos.

Motivación del uso del Retorno a cero

En caso de que existan transiciones debido al uso del pad de salida de la FPGA y no se disponga de una configuración con RZ, ocurre lo mostrado en la Figura 4.4: aunque se den dos o más pulsos consecutivos con valor alto, esta forma de onda no es la misma en todos los casos existiendo en el primer valor alto unas pequeñas oscilaciones mientras que en los siguientes no porque la señal ya se ha estabilizado para entonces. Esta desigualdad da lugar a distorsiones y un aumento del ruido de cuantificación del SDM lo que provoca una pérdida de información a la salida del modulador.

Cuando se produce una transición de un valor bajo (’0’) a un valor alto (’1’) aparecene perturbaciones. La transición mencionada se manifiesta en las oscilaciones que aparecen al principio hasta que la señal se estabiliza al valor ’1’.

Por otro lado, cuando sí hay retorno a cero, ambos pulsos son iguales, pudiendo además corregir facilmente las pequeñas oscilaciones que aparecen. Luego en este último caso no habrá pérdida de información.

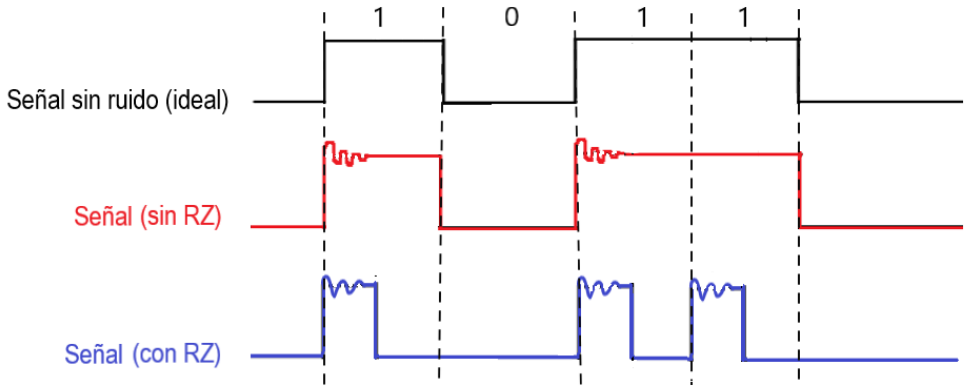


Figura 4.4 Comparación de pulsos con y sin RZ respecto al pulso ideal sin transiciones.

Generación pulso PRZ

En cuanto a su generación, se realiza mediante una filosofía similar a la utilizada para la señal EN . También se usa un contador y un comparador, salvo que en este caso se impone que la señal PRZ este a 1 siempre que el contador tenga un valor inferior al parámetro Nduty, donde: $Nduty = round(\frac{Ndiv}{2} + 0.5)$. En nuestro caso, Nduty=21.

Su diagrama de bloques sería idéntico al presentado en la Figura 4.2 comparando en este caso con el parámetro Nduty en lugar de con (Ndiv-1) lo que se refleja en la Figura 4.5.

4.3.3 Divisor de frecuencia

Agrupando los diagramas de bloques planteados para la generación de ambos pulsos por separado, se obtiene el diagrama asociado al componente divisor de frecuencia (Figura 4.5), dónde se procesan ambas señales de manera simultánea.

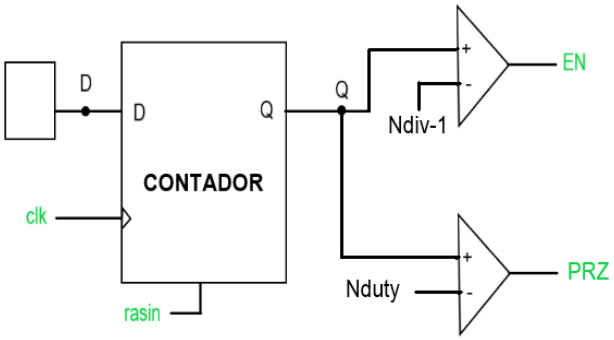


Figura 4.5 Representación esquemática del componente Divisor de frecuencia.

- Entrada : clk, rasin
- Salida : PRZ
- Parámetro: Ndiv, Nduty

Este componente se implementa en el código 8.1 del Apéndice II (Códigos VHDL) y tras su simulación se obtiene lo mostrado en la Figura Figura 4.6.

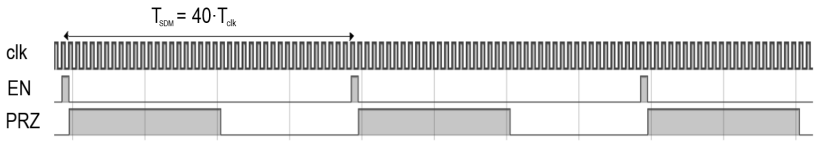


Figura 4.6 Simulación componente divisor de frecuencia donde T_{clk} es el período de la señal reloj (10ns) y T_{SDM} el período del SDM (400ns).

4.4 Generación de la señal de entrada

La generación de la señal de entrada se hace a partir de unos valores de entrada dados por una Lookup Table, leídos mediante un bloque contador que cuenta un número igual al número de valores de la LUT y lee las entradas asociadas a cada número.

Inicialmente, se trabajó con un numero de muestras tal que abarcasen un período completo de la señal seno. Sin embargo, esto conllevaba realizar una LUT con un número muy elevado de valores lo cual generaba un circuito combinacional con abundantes multiplexores (MUX) dando lugar a un retraso temporal considerable en las señales. Esto último provocó la saturación de la señal de salida a partir de cierto tiempo de simulación.

La solución propuesta ha sido trabajar con una LUT que incluye un cuarto de período de la señal sinusoidal a la entrada disminuyendo de este modo el número de muestras de la LUT lo cual genera un número menor de MUX en el *schematic* obtenido tras el RTL Analysis. De este modo, el período completo de la señal senoidal de entrada se genera a partir de dichos valores analizando si la señal crece o decrece o si se encuentra en el semiperíodo positivo o negativo.

La lectura de valores de la LUT y la generación de la señal senoidal se realiza como se muestra en la representación de bloques del código 8.2 en la Figura 4.7 donde se aprecia el uso de varios multiplexores, un contador y un registro. Además, en la Figura 4.7 aparecen señales no definidas en el esquema global de la Figura 4.1 las cuales son fundamentales para la reproducción de la señal seno y que se generan dentro de este bloque CONT.

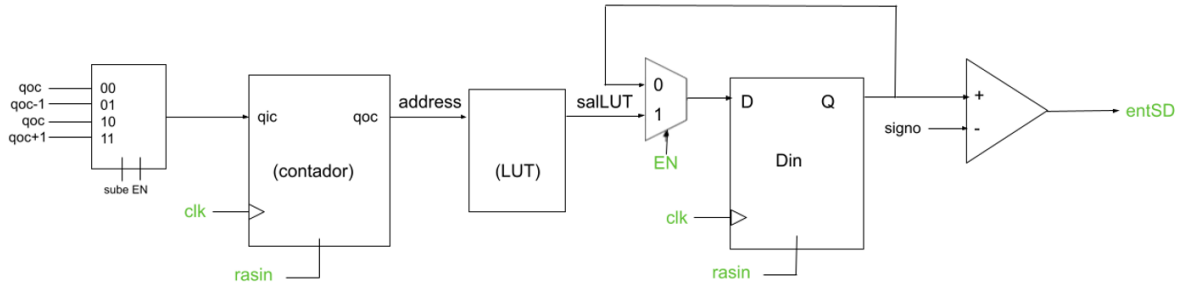


Figura 4.7 Representación esquemática del componente CONT de la arquitectura mostrada en la Figura 4.1.

Por último, la amplitud y frecuencia de la señal de entrada al SDM son:

- Amplitud: $A = 2^{Nbits-1}10^{\frac{-5}{20}}$
- Frecuencia: $f = f_{SDM} = \frac{f_{clk}}{Ndiv} = 400ns$

4.4.1 Lookup Table

Los valores de la Lookup Table empleada se obtienen a partir de un código *MATLAB* que genera los valores binarios en CA2 asociados a las muestras de 1/4 de período de nuestra señal senoidal, y los asocia con la

señal address empleada por el contador. En nuestro proyecto se trabajan con 2 amplitudes distintas de la señal senoidal luego se usarán dos LUT diferentes asociadas a cada caso. Aún así el proceso es idéntico para ambas. Además, la señal address tiene como función indicar la posición en la LUT del valor que debe entrar al SDM.

4.4.2 DSube

A partir de este registro se genera la señal sube. Esta señal afecta a la señal address, y por lo tanto, determina la dinámica de lectura de la LookUp Table. De este modo, cuando la señal sube='1', se leerán los valores de la LUT de manera ascendente y descendente en el caso contrario como se observa en la Figura 4.8 y Figura 4.9.

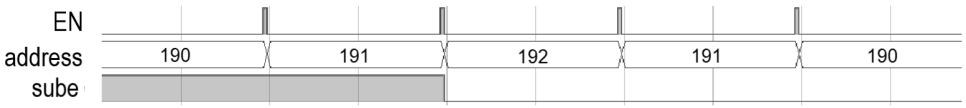


Figura 4.8 Simulación del componente DSube.

Además, la Figura 4.10 muestra la variación de la señal sube en función de los valores de la señal senoidal. Así, cuando el seno tiene pendiente positiva la señal sube toma un valor alto y bajo en caso de pendiente negativa.

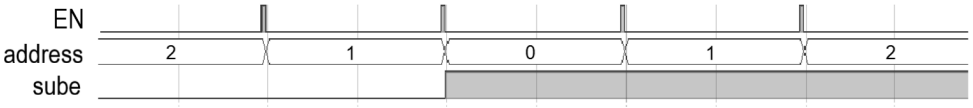


Figura 4.9 Representación de la señal sube.

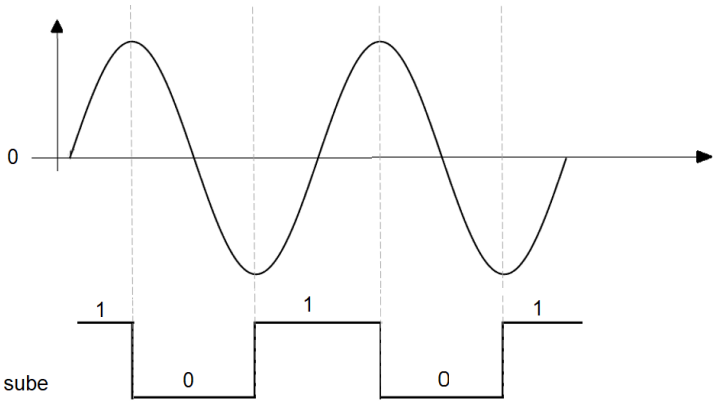


Figura 4.10 Esquema de la variación del pulso sube según el signo de la pendiente de la señal senoidal.

4.4.3 DSigno

El objetivo es generar una señal que provoque un cambio de signo en la muestra leída de la LUT cuando nos encontremos en el semieje negativo de la señal seno a generar.

De este modo, se hace el valor de la LUT negativo siempre que signo='0'.

En la Figura 4.11 se muestra como la señal signo cambia su valor cuando el seno toma el valor asociado a la primera muestra (address='0'), lo que se traduce en que la señal signo cambia cuando el seno pasa por 0 como se aprecia en la Figura 4.12.

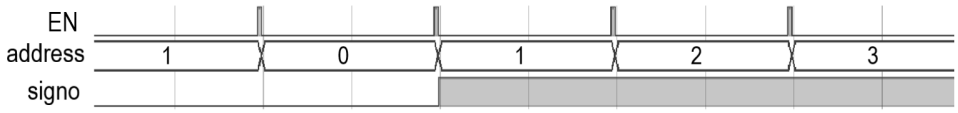


Figura 4.11 Simulación del componente DSigno.

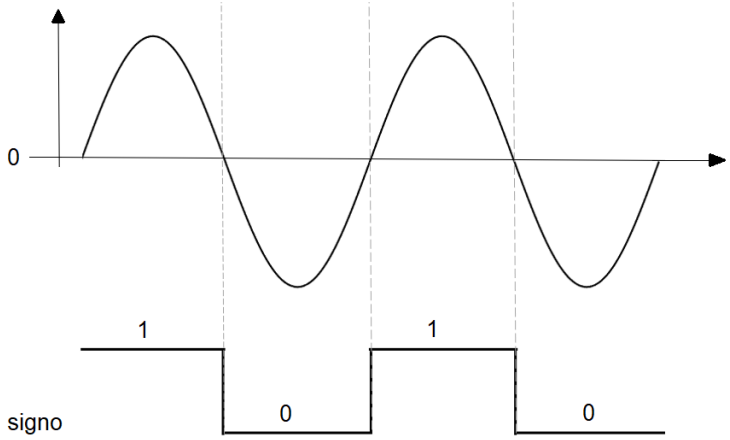


Figura 4.12 Esquema de la variación del pulso signo según el semiperíodo de la señal senoidal.

4.4.4 Contador y Din

El contador está definido dentro del código 8.2 basándose en un registro tipo D que cuenta hasta un número igual al número de muestras de la señal seno en la LUT de manera ascendente o descendente según si la señal crece o decrece. De este modo el contador aumenta en uno su valor a la salida cuando la señal EN y la señal sube están a nivel alto, mientras que si ocurre la situación anterior pero cuando sube toma un nivel bajo, la señal a la salida decrece en una unidad. El valor del contador se identifica con la señal address y para cada posible valor de ella se asocia una muestra como se ha explicado anteriormente.

Esta señal address entra en un registro, también tipo delay, denominado Din cuya finalidad es romper con la lógica combinacional y evitar que se propaguen errores por retraso temporal en la señal.
Finalmente, se cambia de signo o no a la salida del componente Din según el valor que tome la señal signo, teniendo en cuenta la conversión asociada a la configuración en CA2 como se muestra en las últimas líneas del código 8.2.

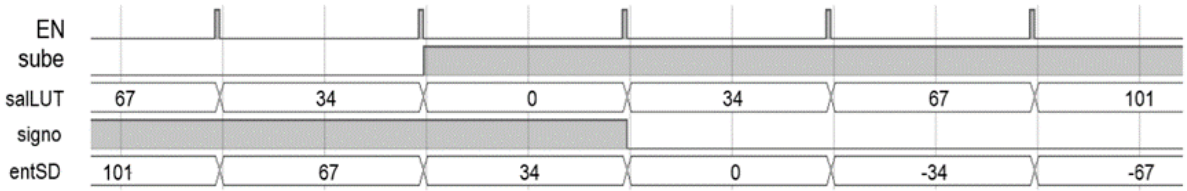


Figura 4.13 Simulación de las señales asociadas al contador y al registro Din.

En la simulación expuesta en la Figura 4.13, se observa como cuando la señal sube está a '1', efectivamente el valor de salLUT crece mientras que en el caso contrario decrece. También se aprecia la simetría de la señal seno.

Por otro lado, dicha señal salLUT es la entrada al registro Din y su valor se cambia de signo siempre que la señal signo sea '0'tal y como queda demostrado en la Figura 4.13. Finalmente, también se aprecia como

añadir un registro hace que la señal que entra a nuestro circuito SDM sea la correspondiente a salLUT un período de EN después, pero este retraso no genera ningún error en nuestro circuito.

4.5 Lógica combinacional

A continuación, se van a comentar los dispositivos que forman el sistema combinacional de nuestro circuito analizando sus entradas y salidas, el código que lo implementa y por último las simulaciones mediante las cuáles se verifica el buen funcionamiento de los componentes.

4.5.1 Sumador: Sum

Este componente se implementa mediante el código 8.6, donde se trabaja con señales CA2 y sin desbordamiento lo que conlleva que se añada 1 bit adicional a la salida tras analizar la suma de los valores máximos a y b.

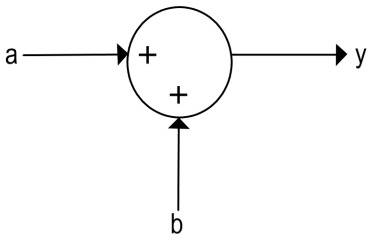


Figura 4.14 Representación esquemática del componente Sumador.

Tabla 4.1 Identificación señales circuito SDM con las entradas y salidas del componente Sumador.

Componente	Circuito SDM
a	entSD
b	salida2
y	salidasum

Identificando a que señales de nuestro circuito corresponden las entradas y la salida del componente sumador y realizando una simulación a partir de su código, se obtiene lo mostrado en la Figura 4.15.

entSD	-168	-201	-235	-268	-302	-336	-369
salida2	-3040	6069	7154	248	1768	-4635	-2545
salidasum	-3208	5868	6819	-21	1466	-4971	-2914

Figura 4.15 Resultado de simular el componente Sumador.

4.5.2 Restador

El componente es el mismo para el caso del Restador1 y Restador2 salvo que difiere el número de bits a las entradas y por tanto también a la salida, de este modo se utilizan dos códigos distintos para su implementación: 8.7 y 8.8.

De nuevo se trabaja en CA2 y en este caso, la señal b asociada al puerto '-' debe ser el negado de su valor inicial, estando dicho número negativo en configuración CA2.

A continuación, se identifican las entradas y salida del componente restador con las señales de nuestro circuito SDM así como se representa su simulación para cada restador que forma el circuito.

Restador1:Res

El primer restador tiene asociado las señales de la Tabla 4.2, cuya evolución temporal se muestra en la simulación de la Figura Figura 4.17.

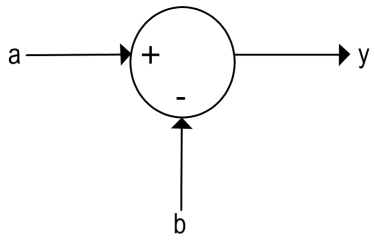


Figura 4.16 Representación esquemática del componente Restador.

Tabla 4.2 Identificación señales circuito SDM con las entradas y salidas del componente Restador1.

Componente	Circuito SDM
a	salidasum
b	salidag
y	salidares

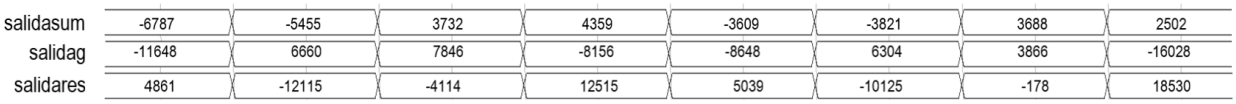


Figura 4.17 Representación esquemática del componente Restador1.

Restador2:Res2

Tabla 4.3 Identificación señales circuito SDM con las entradas y salidas del componente Restador2.

Componente	Circuito SDM
a	salidaq
b	salidasat
y	salidares2

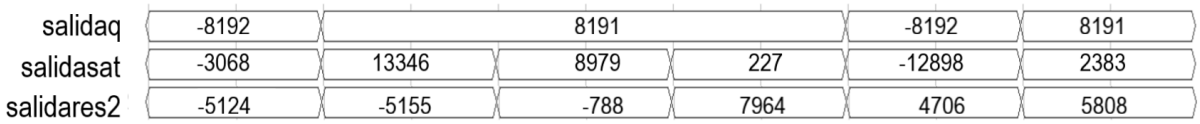


Figura 4.18 Representación esquemática del componente Restador 2.

En la Figura 4.18 se observa como la señal salidaq varía entre dos valores: -8192 y 8191. Esto se debe a su configuración Single Bit que se explicarán con más detalle en el apartado relacionado con el componente cuantizador.

4.5.3 Saturación: Sat

El valor del parámetro Nsat se ha determinado a partir del análisis del circuito en *Simulink* y de cómo varía la SNR según el valor de saturación, eligiendo aquel tal que la SNR este entorno a los 70 dB obtenidos inicialmente.

Además, al trabajar en CA2, con la saturación se gestiona o evita que el valor de la señal de la vuelta, es decir, al realizar la suma de dos números positivos/negativos podría darse el caso en el que la suma se haga negativa/positiva debido a un cambio en el bit de signo. A modo de ejemplo, se realiza la suma de dos números binarios de 4 bits, sabiendo que con dicho número de bits y de nuevo trabajando en complemento a dos, se puede obtener números decimales dentro del intervalo [-8, +7]. En él se observa cómo el número más positivo en decimal se vuelve el más negativo en CA2 ya que supera el valor máximo positivo posible para dicho número de bits.

$$\begin{array}{rll} & 0110 & (6|_{10}) \\ + & 0010 & (2|_{10}) \\ \hline & 1000 = (8|_{10}) & \longrightarrow \text{CA2 : -8} \end{array} \qquad \begin{array}{rll} & 0110 & (6|_{10}) \\ + & 0101 & (5|_{10}) \\ \hline & 1011 = (11|_{10}) & \longrightarrow \text{CA2 : -5} \end{array}$$

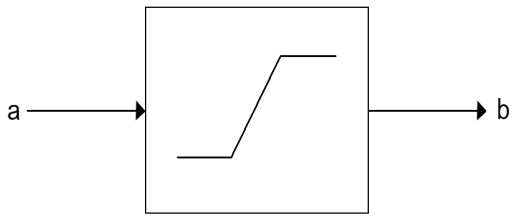


Figura 4.19 Representación esquemática del componente Saturación.

Por último, el código 8.9 es el asociado a este componente (Figura 4.23). Se observa como se satura a 17 bits tal que cuando los valores de entrada superen los valores límites del intervalo en CA2 de números para esta configuración de bits, se satura con el valor más positivo o negativo según si el valor de la entrada sea mayor o menor que cero.

Tabla 4.4 Identificación señales circuito SDM con las entradas y salidas del componente Saturacion.

Componente	Circuito SDM
a	salidares
b	salidasat

salidares	955	-7747	10858	7646	-975	17786	14804
salidasat	955	-7747	10858	7646	-975	17786	14804

Figura 4.20 Simulación componente Saturacion.

4.5.4 QuantizerSB: QSB

El cuantizador que se implementa en nuestra arquitectura del SDM es tipo Single Bit que se comporta de tal manera que cuando la señal de entrada tome un valor positivo, a la salida se obtendrá el número más positivo posible según un número de bits dado. En caso de entrar una señal negativa, la salida será el número más negativo posible para la configuración de bits que acontezca. Además, no es simétrico y por lo tanto los valores a la salida son +8191 y -8192.

Esta dinámica que se acaba de explicar se plasma en el código 8.10 mediante el cuál se obtienen los resultados de simulación mostrados en la Figura 4.22. Además, dicho código se asocia a un esquema (Figura 4.21) tal que el componente cuenta con una señal de entrada y dos señales de salida, una con 14 bits y otra de 1 bit. Esta última se denomina ySD y toma valores '1' y '0'cuando la señal a la entrada sea positiva o negativa

respectivamente.

Por otro lado, también cabe destacar como la señal de salida del circuito sufre una modificación intermedia antes de entrar en el cuantizador tal que la señal con la que trabajaremos será de 1 bit el cuál corresponde al bit más significativo de la señal de salida del saturador, que al trabajar en CA2, representa el signo de la señal. De este modo se optimiza el código al trabajar con un valor inferior de bits a la entrada ya que al fin y al cabo sólo nos interesa dicho bit para determinar el valor a la salida. Esta transformación de la señal de entrada se muestra enmarcada en una línea de puntos en la Figura 4.21.

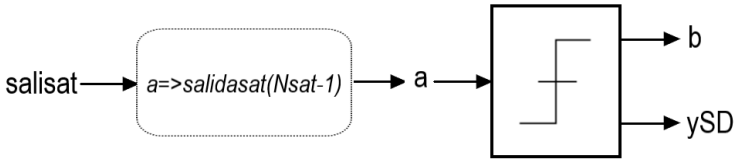


Figura 4.21 Representación esquemática del componente Quantizer SB.

Tabla 4.5 Identificación señales circuito SDM con las entradas y salidas del componente QSB.

Componente	Circuito SDM
salidasat	salidasat
a	salisat(Nsat-1)
b	salidaq
ySD	ySD

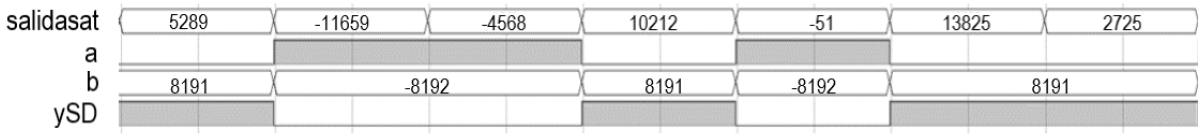


Figura 4.22 Simulación componente Quantizer SB.

Todo lo explicado anteriormente queda reflejado en la Figura 4.22.

4.5.5 Ganancia: G

Este componenente forma parte de la arquitectura de SDM y con él se duplica el valor de la señal binaria a la entrada de este como se muestra en el código 8.11.

Al igual que en los apartados anteriores, a continuación, se muestra el esquemático asociado a este componente (Figura 4.23), se identifican las señales de nuestro circuito SDM en dicho esquema (Tabla 4.6) y se ilustra parte del resultado obtenido tras su simulación Figura 4.24.

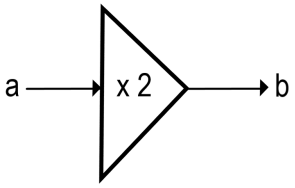


Figura 4.23 Representación esquemática del componente Ganancia.

Tabla 4.6 Identificación señales circuito SDM con las entradas y salidas del componente Ganancia.

Componente	Circuito SDM
a	salida1
b	salidag

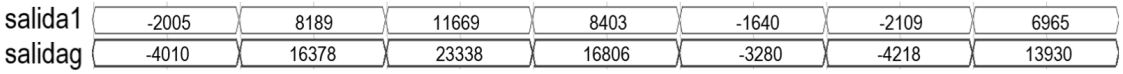


Figura 4.24 Simulación del componente Ganancia.

4.6 Lógica secuencial

Se consideran dentro de esta clasificación a aquellos componentes del circuito cuyas salidas son función de sus entradas actuales y de su estado anterior. En nuestro caso serán síncronos, y sus valores se actualizan en los instantes de tiempo en los que la señal clock tiene un flanco de subida. De este modo los cambios de estado se darán sincronizados con la señal reloj.

Además de lo anterior, el circuito tendrá como entrada un reset asíncrono (rasin), tal que todas las variables se harán cero cuando rasin='1'.

Los componentes que cumplen estas características son los registros y biestables, en nuestro caso tipo D.

4.6.1 Componente D_ff

Para los bloques z^{-1} se trabaja con biestables tipo D (delay). Esta relación se debe a que, trabajando con transformada z, dividir entre z equivale a retrasar un intervalo de muestreo, lo cual llevan a cabo este tipo de registros que permiten sincronizar la entrada con el reloj imponiendo un retraso. Dicho retraso se observará en las simulaciones mostradas posteriormente.

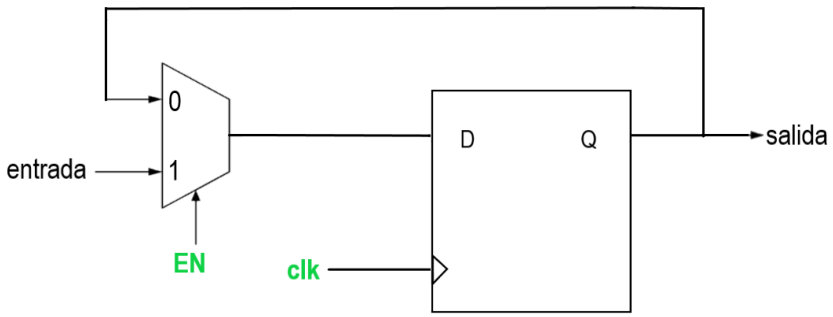


Figura 4.25 Representación esquemática del componente Flip-flop tipo D.

Se parte del componente genérico mostrado en la Figura 4.25 el cuál consta de una entrada D y la entrada asociada al reloj, y una salida. Además, mediante un multiplexor, se impone que la entrada al registro se actualice con la señal de entrada siempre que EN tenga un valor alto. Dicha representación de bloques se plasma en el código 8.12.

A partir de dicho esquema general, se especifican los dos registros que conforman el circuito a estudio según las señales que entran a cada uno de los dos.

D1

El primero de los dos registros empleados en el circuito tiene asociado las señales de la Tabla 4.7 y su simulación da los resultados mostrados en la Figura 4.26 donde se aprecia el retraso que este registro presenta.

Tabla 4.7 Identificación señales circuito SDM con las entradas y salidas del componente D1.

Componente	Circuito SDM
entrada	salidares2
salida	salida1

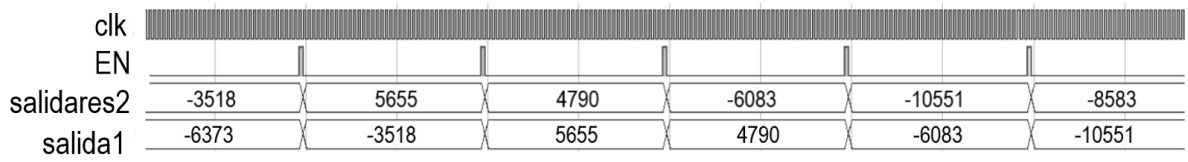


Figura 4.26 Simulación del componente D1.

D2

Para poder observar mejor el retraso que se produce en las señales una vez estas entran en los registros se ha realizado la simulación asociada al componente D2 para el mismo intervalo de tiempo que el mostrado en la Figura 4.27.

Tabla 4.8 Identificación señales circuito SDM con las entradas y salidas del componente D2.

Componente	Circuito SDM
entrada	salida1
salida	salida2

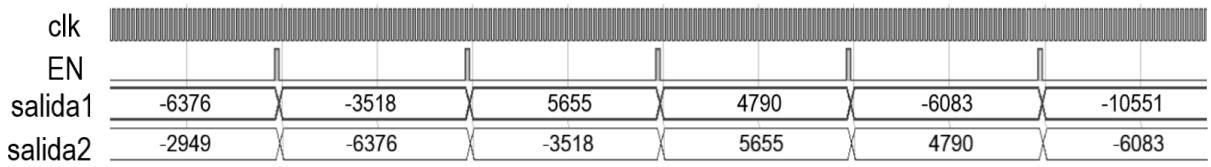


Figura 4.27 Simulación del componente D2.

4.6.2 Componente D_out

El objetivo de este biestable es almacenar las señales de salida del SDM para su posterior extracción y análisis bien mediante Matlab o midiendo dichas señales con un osciloscopio digital.

En este caso, sería erróneo incluir como entrada a señal EN dado que la señal ySD que sale del cuantizador y entra en Dout ya está sincronizada con la señal Enable.

El componente mostrado en Figura 4.28 se implementa a través del código 8.13. Consiste en un biestable que consta de cuatro entradas y dos salidas las cuales son a su vez salidas del bloque circuital completo.

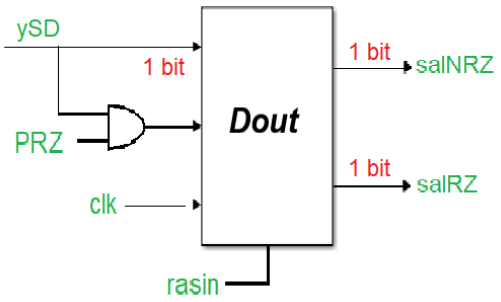


Figura 4.28 Representación esquemática del componente Dout.

Simulación componente Dout

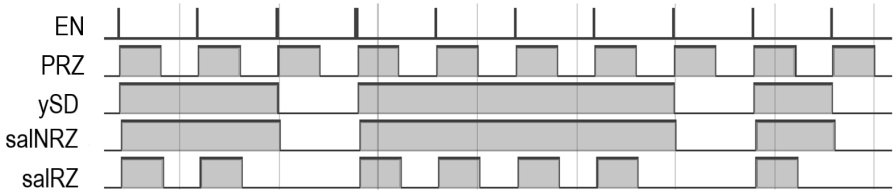


Figura 4.29 Representación esquemática del componente Dout.

A partir de la Figura 4.29 se observa como las señales salNRZ y salRZ son idénticas salvo que las segunda sufre un retorno al valor cero.

4.7 Design Sources

4.7.1 Archivo top

En este archivo se conforma el circuito completo a partir de los componentes descritos anteriormente. El código 8.14 constituye lo que se denomina *Design Source* y en él se realiza la conexión de los distintos elementos, asociado las entradas y salidas de cada uno al igual que se definen las constantes y señales internas.

De este modo, se describe todo nuestro diseño dentro de la entidad (entity) top3 que actúa como una caja negra, la cual solo consta de dos señales de entrada y tres señales de salida como se observa en la Figura 4.30. La selección de dichas entradas y salidas no ha sido arbitraria, sino que se han elegido tal que solo apareciesen las imprescindibles para poder realizar el análisis final.

Las señales intermedias restantes no serán ni extraídas en *MATLAB* ni representadas mediante un osciloscopio digital ya que no son de interés en el estudio final, pero si han sido debidamente analizadas en los pasos anteriores para comprobar que todo funciona correctamente y así minimizar la posibilidad de error en la etapa final de este proyecto. Todo lo descrito aquí se ejemplifica en la Figura 4.30

Además, cabe mencionar que este código primario se alimenta de una serie de códigos secundarios que son los relativos a los distintos componentes que conforman el circuito completo mostrado en la Figura 4.1 y que se han ido explicando con detalle en las Sección 4.5 y Sección 4.6 de la memoria.

4.8 Simulation Sources

A continuación se comentan los ficheros VHDL empleados para simular el componente Top gracias a que estos incluyen la definición de los distintos estímulos que describen las señales clk y rasin, las cuales provocarán cambios en las señales que intervienen en nuestro circuito global.

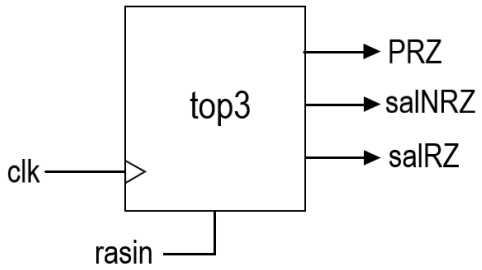


Figura 4.30 Representación esquemática del componente Dout.

Este tipo de código recibe el nombre de banco de pruebas o *testbench* en inglés debido a su función de verificar si las señales, tanto de salida como de entrada, son las esperadas en un primer análisis teórico comprobando así el correcto funcionamiento del top. En nuestro proyecto se han definido dos archivos de esta clase abordados en las siguientes subsecciones, teniendo cada uno asociado una misión concreta.

4.8.1 Archivo sim_top

El código 8.15 hace referencia a un fichero VHDL del tipo *Simulation Source* que tiene por finalidad simular el componente top a partir de la definición de los estímulos asociados a la señal clk con un período de 10 ns y la señal rasin, activa el primer cuarto de período e inactiva durante el tiempo restante de simulación.

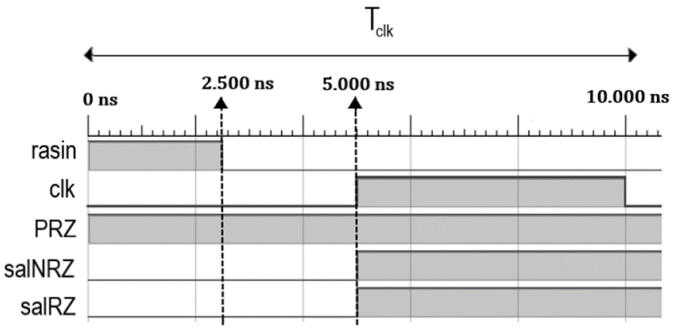


Figura 4.31 Simulacion corta del componente top.

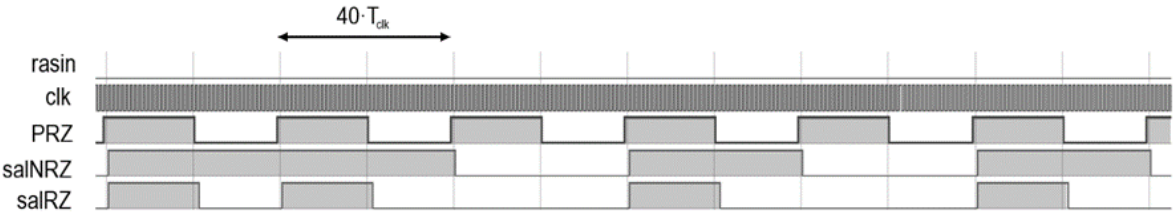


Figura 4.32 Simulación del componente top.

En la Figura 4.31 se ha realizado una simulación corta del archivo VHDL con objeto de mostrar como se cumple la definición de los estímulos explicada anteriormente. Al pricipio, la señal rasin toma un valor alto y por lo tanto las señales de salida salNRZ y salRZ están a cero. Una vez que rasin toma un nivel bajo, las señales comienzan a evolucionar.
También se destaca de dicha representación como la señal rasin está a '1' durante un tiempo igual a $\frac{T_{clk}}{4} = 2.5ns$ tal y como se expone en la simulación.

Por otro lado, la señal PRZ no se ve, aparentemente, afectada por la variación de la señal rasin. Esto no

significa que no tenga dependencia con ella, si no que, recordando como se generaba dicha señal intermedia, PRZ toma un valor alto siempre y cuando la salida del contador (diseñado mediante un registro tipo D) sea inferior a Nduty, por lo que cuando la señal rasin provoca que el registro sea 0, la salida del contador seguirá siendo inferior a Nduty y por tanto PRZ será '1'.

Adicionalmente, se ha realizado una simulación durante un período de tiempo mayor tal y como se refleja en la Figura 4.32, lo cual permite ahora analizar cómo varían las señales de salida con respecto a las entradas. Efectivamente se sigue cumpliendo que las señales salNRZ y salRZ se actualizan cada período de la señal EN, es decir cada $40 \cdot T_{clk}$. Además, se observa como la señal salRZ es análoga a salNRZ pero aplicando retorno a cero. Por último, se aprecia un cierto retraso de las señales anteriores con respecto a PRZ debido al uso de registros tipo D en el circuito completo, aunque este retraso no compromete el buen funcionamiento del circuito.

Finalmente, cabe mencionar que todas las simulaciones de los componentes realizadas anteriormente se han obtenido a partir de esta simulación del SDM al completo, ya que el Software permite añadir señales internas a la ventana dónde se representan las ondas (*Wave Window*).

4.8.2 TB_top_SDM

Este archivo también tiene como objetivo realizar una simulación del fichero top pero con la finalidad de extraer los datos a la salida del SDM para posteriormente analizarlos mediante *MATLAB* y se realiza mediante el código 8.17.

De este modo se respaldan las conclusiones obtenidas tras la simulación realizada en 8.15 con un análisis de las prestaciones de circuito. Dichas prestaciones se calculan haciendo uso de los códigos 7.1 para analizar la señal de salida sin retorno a cero y 7.2 para analizar la señal a la entrada.

Así pues, se comprueba que todo se ha realizado correctamente observando su espectro (cumple con el típico espectro de SDM: señal en Nfs y ruido a altas frecuencias) así como la SNR obtenida con simulink inicial ($\sim 70\ dB$). Adicionalmente, se calcula el valor del *Second Harmonic Distorsion* o TH2 para poder comparar lo obtenido en esta primera simulación ideal con los resultados medidos en *PicoScope* al final de este proyecto.

También resulta de interés comprobar que la señal que entra al SDM es la correcta. Es decir, una entrada senoidal. Para ello se realiza el mismo proceso relacionado con la extracción de datos de salida pero aplicado a la entrada. Todo lo relacionado a los códigos requeridos para llevar a cabo esta sección se ha explicado con más detalle en ambos anexos.

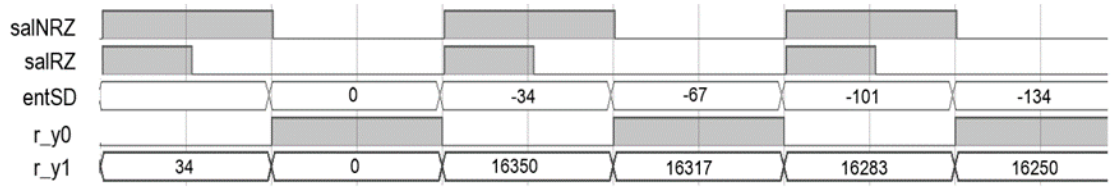


Figura 4.33 Simulación del archivo TB_top_SDM.

Tras realizar una simulación de tipo Behavioral se obtiene lo mostrado en Figura 4.33. En dicha representación se han obviado las señales clk y rasin debido a que ya se explicó anteriormente como las salidas estaban sincronizadas con el reloj y el efecto del rasin.

Por un lado, se puede observar como la señal r_y0 es una copia de la señal salNRZ con un pequeño retraso. Este retraso temporal no tendrá importancia ya que el análisis que se realizará con Matlab se aplica individualmente a cada señal, luego no afecta a los resultados que se obtengan. La señal es la misma, las prestaciones son las mismas debido a que el retraso no implica ninguna pérdida en la señal.

Por otro lado, se comprueba como la señal r_y1 es igual a la señal entSD solo que escrita como un valor decimal sin signo. Dicho signo se impondrá en el código 7.2 para poder representar y analizar correctamente dicha señal. Además, en este caso no se observa ningún retraso.

4.9 Extracción de datos de Vivado

A continuación, se explica el proceso asociado a la extracción de datos del SW Vivado al ser el paso siguiente a realizar según el digrama de flujo de actividad presentado al principio del documento en la Figura 1.1.

A partir del fichero TB_top_SDM mencionado previamente y llevado a cabo en el código 8.17, se obtiene un fichero de texto que recibe el nombre de salida_RON.txt en el caso que acontece, y en el cuál se recogen 2 columnas de datos asociada la primera a la salida y la segunda a la entrada. Dicho fichero de datos se carga en los códigos de *MATLAB* respectivos, obteniendo los resultados que se muestran a continuación para el caso amplitud 5 dB por debajo de la amplitud máxima.

4.9.1 Análisis salNRZ

El análisis de la señal de salida del SDM sin retorno a cero empleando *MATLAB* alcanza los siguientes resultados.

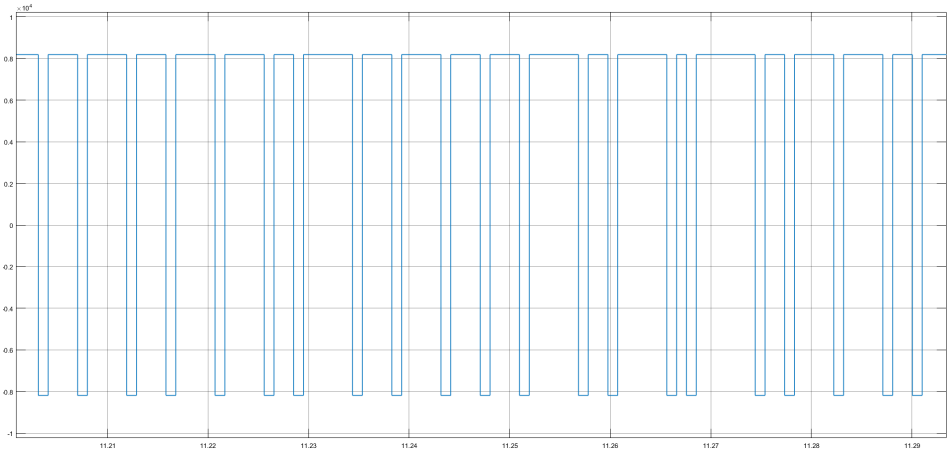


Figura 4.34 Señal temporal de 1 bit a la salida del SDM para la configuración sin retorno a cero (salNRZ).

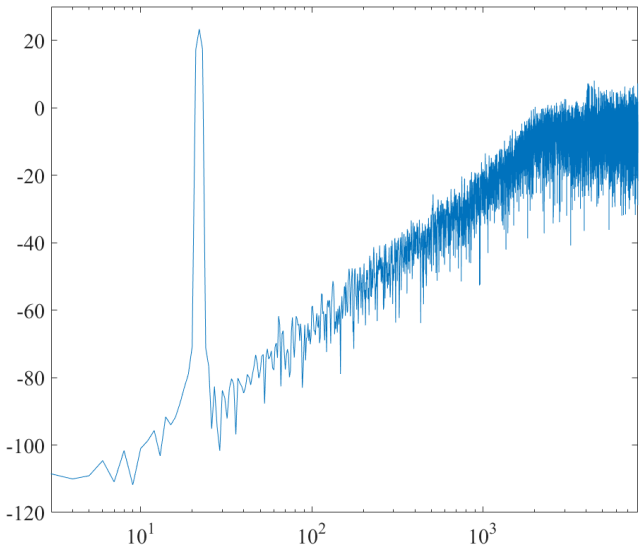


Figura 4.35 Espectro asociado a la señal salNRZ.

En la Figura 4.34 se muestra una señal que toma valores en el intervalo [-1,1] tal y como se especificó en el código 7.1 del Apéndice I. Además, dicha representación se ha obtenido mediante *Simulink*. Así pues, se observa como no sigue un patrón, si no que la señal se mantiene a 1 o a -1 distintos períodos de tiempo.

Por otro lado, la Figura 4.35 muestra el espectro característico de un SDM con ruido a alta frecuencia como se mostró en la Figura 2.11, lo que a priori da una buena impresión y sugiere que funciona todo correctamente. Esto último queda respaldado con los valores numéricos obtenidos en este análisis Tabla 4.9 ya que se obtiene un valor de SNR alrededor de 70 dB tal y como sucedía en la simulación preliminar previa a la implementación del circuito en VHDL.

Tabla 4.9 Prestaciones de la señal de salida del SDM sin retorno a cero medidas a partir del fichero de datos extraído de *Vivado*.

	SNR (dB)	HD2 (dB)
salNRZ	70.011312	75.278451

4.9.2 Análisis entSD

También resulta interesante analizar la señal de entrada al modulador con objeto de comprobar que se está generando correctamente la señal seno dentro del dispositivo programable. Para ello, a continuación, se muestran una serie de figuras resultantes de ejecutar el código 7.2 del Apéndice I en *MATLAB* . Comparando Figura 4.36 con Figura 4.37 se aprecia la necesidad de realizar la conversión aplicada en el bucle for, tal que la señal pasa de tomar valores decimales sin signo a tener valores positivos y negativos, permitiendo esto último obtener una representación de una señal senoidal perfecta tal y como era de esperar.

Adicionalmente, la representación espectral de esta señal (Figura 4.38) coincide con el espectro característico de una señal senoidal y nos da una idea del valor del HD2 comparando los valores de la señal y del segundo armónico al igual que de la SNR comparando la señal con el fondo de ruido.

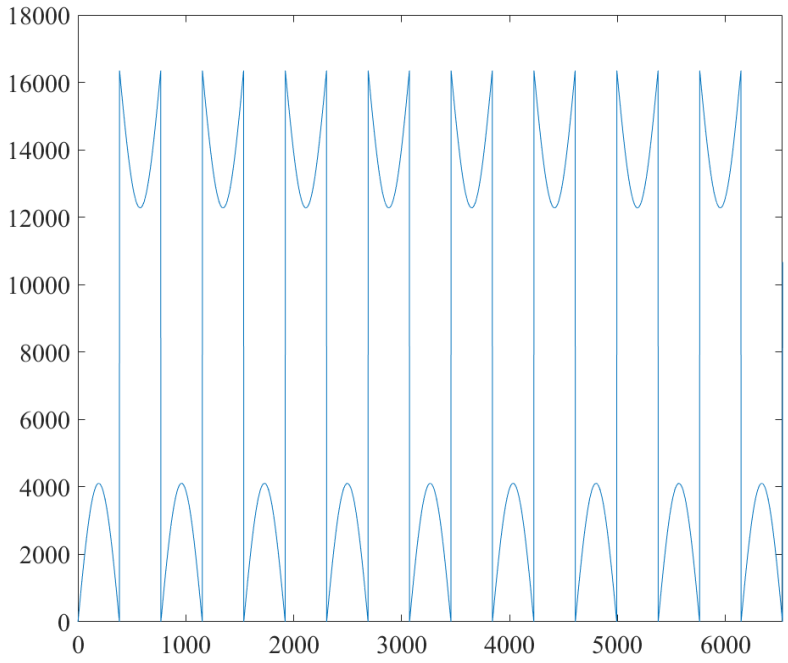


Figura 4.36 Señal a la entrada del SDM con valores decimales sin signo.

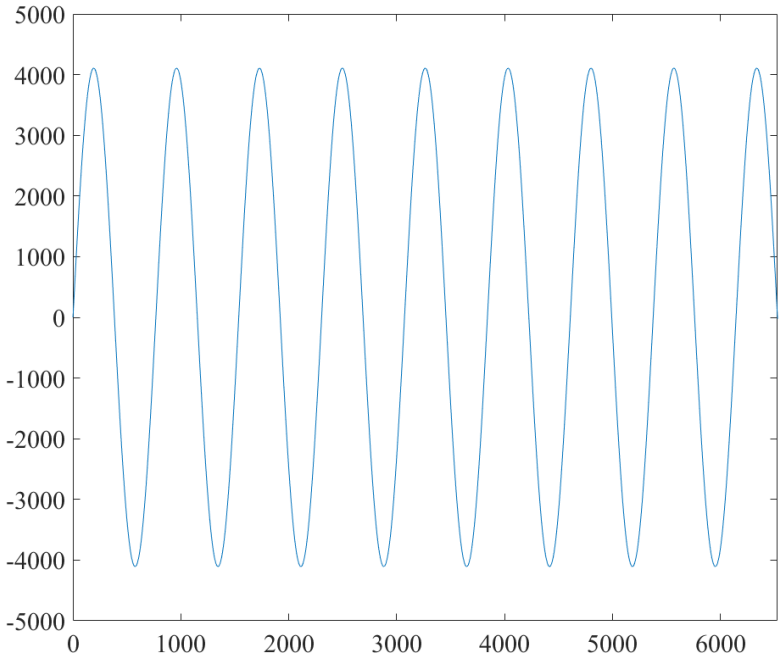


Figura 4.37 Señal senoidal a la entrada del SDM.

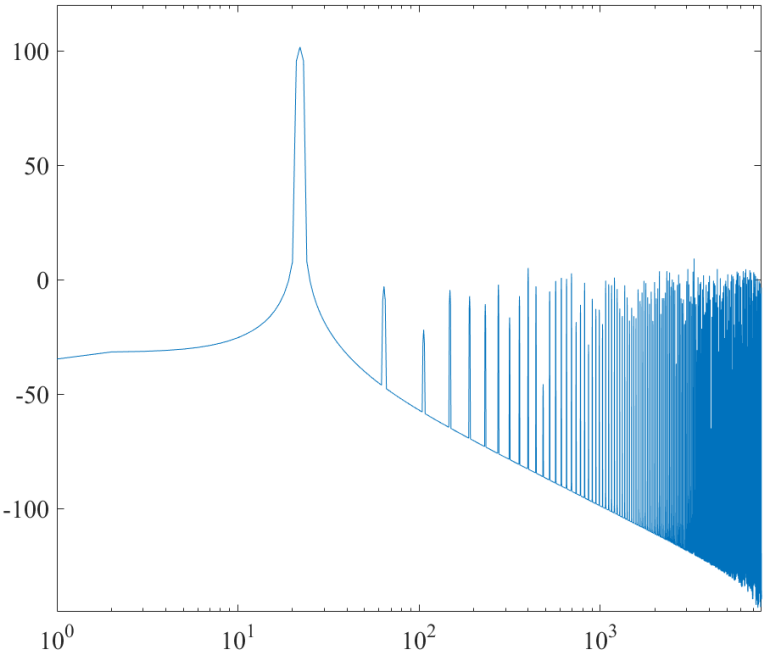


Figura 4.38 Espectro asociado a la señal senoidal.

Tabla 4.10 Prestaciones de la señal de entrada al SDM a partir del fichero de datos extraído de *Vivado*.

	SNR (dB)	HD2 (dB)
entSD	99.689415	104.465874

4.9.3 Conclusiones

En vista a los resultados obtenidos, se concluye que la señal de entrada se genera correctamente dando lugar a una señal senoidal lo cual se respalda tanto gráficamente como analíticamente ya que se obtienen unos valores muy elevados de SNR y HD2 como es de esperar. Además, el valor de SNR obtenido en la Tabla 4.10 coincide con el calculado teóricamente en la expresión (3.4). Por último, comparando los resultados obtenidos con los del análisis preliminar en *Simulink* se espera que la señal de salida en VHDL tenga las mismas prestaciones que las observadas en el estudio en *Simulink*. Así pues, tanto el espectro como el valor de SNR coinciden con los del diseño inicial.

4.10 Programación de la FPG

En esta última sección se explican los distintos pasos a seguir para poder implementar lo expuesto en el capitulo Capítulo 4 en la FPGA. Es fundamental tener en cuenta que para realizar el proceso de programación de la FPGA se trabajará con el archivo top3 y sim_top3, luego ambos han de establecerse como *Top* en el SW *Vivado*.

4.10.1 Modelo FPGA

El modelo de FPGA empleado cumple las siguientes características, las cuáles se deben definir en el SW para poder realizar la conexión y programación del dispositivo.

PRODUCT FAMILY	Zynq-7000
PROJECT PART	xc7z020clg484-1

4.11 Elección pines I/O. Archivo de constraints

Lo primero que se debe hacer es crear un archivo dónde se asocian las entradas y salidas del SDM con los pines de la FPGA por los que se desee obtener dichas señales. Dicho archivo se ha de definir con una sintaxis específica tal y como se ha realizado en el código 8.18.

La elección de los pines asociados a las señales y que se expone en dicho código ha sido tal que a la hora de tomar medidas a la salida se evitase la aparición de acoplo de ambas señales y diese lugar a un fenómeno de diafonía también denominado *crosstalk* en inglés. Por ello se han elegido pines que estuviesen alejados entre ellos, dentro de lo posible.

A partir de la guía de usuario [1] (Figura 4.39) se aprecia como se distribuyen dichos pines en la FPGA. Nuestro modelo de FPGA consta de 5 conectores de regletas de pines Digilet Pmod, de los cuales nosotros emplearemos dos: JA1 y JB1 (Figura 4.40). Dentro de dichos Pmod hay que seleccionar y asociar los pines que se van a usar y que señales salen de ellos.

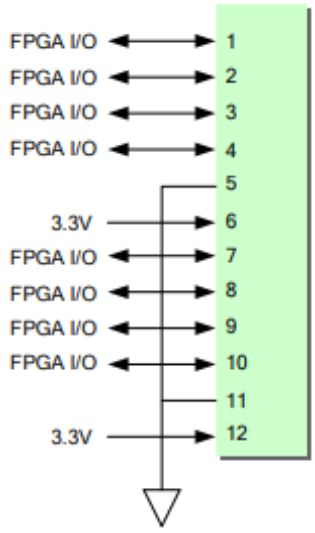


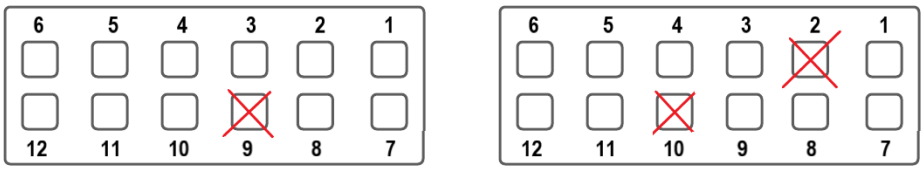
Figura 4.39 Conexiones Pmod. Fuente [1].

Haciendo uso del esquema mostrado en la Figura 4.39 y de las tablas de conexiones Pmod también indicadas en el mismo manual de usuario, se decide imponer la siguiente relación de señales de salida y pines, indicadas en la Tabla 4.11 teniendo en cuenta que el pin número 5 se reserva para la conexión a tierra mientras que a los pines 6 y 12 se les asigna una entrada de 3.3V.

Tabla 4.11 Asociación de señales de salida con pines I/O de la FPGA..

Señal	Pmod	Pin de Zynq	N^o pin
salNRZ	JA1	AB9	JA9
salRZ	JB1	V8	JB10
PRZ	JB1	W11	JB2

Para concluir con la explicación, en la Figura 4.40 se expone la distribución aplicada a las señales de salida del SDM implemetado en este proyecto.



(a) JA1. **(b)** JB1.

Figura 4.40 Esquemas de los pines empleados para cada Pmod.

4.12 Pasos a seguir

Una vez realizado lo anterior, y tras haber comprobado el correcto funcionamiento de los códigos a lo largo del Capítulo 4, se realizan los pasos pertinentes para la programación de la FPGA.

La secuencia de pasos a seguir es bastante sencilla y fácil de ejecutar, siempre y cuando se haya realizado todos los pasos anteriores correctamente.

I. **RTL Analysis**
Se elabora el diseño ejecutando el apartado RTL Analysis. Una vez finalizado este, el SW generará un esquemático del circuito implementado.

II. **Synthesis**
Se realiza la síntesis del circuito y posteriormente dos simulaciones asociadas a esta: $Post-Synthesis\ Functional\ Simulation$ y $Post-Synthesis\ Timing\ Simulation$ debiendo comprobar que ambas dan resultados adecuados, sin producirse ninguna saturación en las señales u otros errores.

III. **Implementation**
Este tercer paso se ejecuta de manera análoga al anterior, habiendo que comprobar también las simulaciones funcional y timing correspondientes a este.

IV. **Program and debug**
Lo primero a realizar en este último paso es generar el *Bitstream* que es un archivo que contiene la información de programación para una FPGA.
A continuación, se conecta la FPGA empleada al ordenador para poder proceder a su conexión, seleccionando la opción de $Open\ Target \longrightarrow Auto\ connect$ y finalmente se programa la FPGA ejecutando la opción de $Program\ device$.

Tras realizar todo lo anterior, ya se pueden tomar medidas experimentales midiendo las tensiones en los pines de salidas de la FPGA que se hayan asociado con las señales a estudio. Esta toma de medidas se realiza en nuestro caso mediante el osciloscopio digital *Picoscope* y se analizan mediante códigos de Matlab.

5 Resultados Experimentales

A continuación, se muestran los resultados experimentales obtenidos tras realizar todos los pasos anteriores y una vez generado el Bitstream en el Software Vivado tras la exitosa conexión y programación de la FPGA con el ordenador. Estos resultados experimentales se analizan mediante el sistema de medición *PicoScope*. Concretamente, usando el Software PicoScope 6. Además, para medir correctamente los resultados a la salida del SDM se ha diseñado un filtro paso bajo que elimine el ruido a altas frecuencias.

Se va a proceder a analizar estos resultados experimentales tanto mediante el propio osciloscopio digital, realizando un análisis de la señal gráficamente (observando si su período y amplitud es el adecuado para el caso temporal así como la frecuencia del armónico fundamental y el fondo de ruido para el caso espectral) y mediante *MATLAB* analizando los ficheros de datos asociados a la señal temporal y su FFT.

5.1 Diseño del filtro

En primer lugar, se analiza el filtro paso bajo diseñado para eliminar el ruido a altas frecuencias tal como se ejemplificó en la Figura 2.12. Como SW de apoyo se emplea *PSpice*, donde se dibuja el esquema del filtro a estudiar (Figura 5.1) y se realiza un análisis tipo AC de este (Figura 5.2).

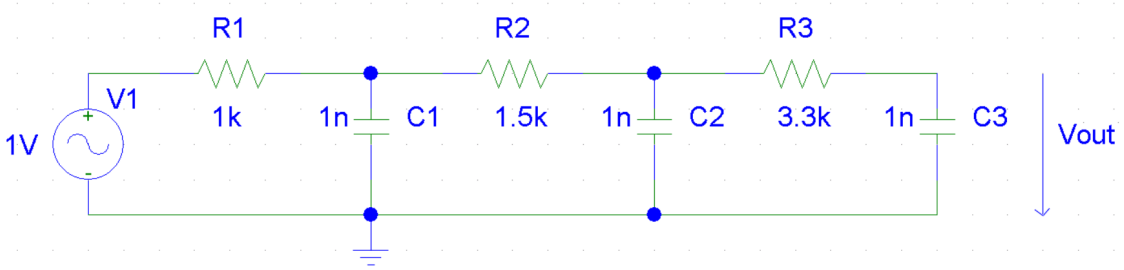


Figura 5.1 Filtro paso bajo para eliminar ruido a altas frecuencias de la señal de salida del SDM.

En la Figura 5.2 se ha representado el valor en dB del voltaje Vout. Además, se han añadido varios cursores en los valores relevantes a destacar:

- Frecuencia $f_s = \frac{f_{clk}}{Ndiv} = 2.5MHz$: frecuencia de muestreo de la señal. Es la frecuencia en la que se haya la imagen del espectro de bajas frecuencias, recordando los valores de los parámetros $Ndiv = 40$ y $f_{clk} = 10ns$
- Frecuencia $f_s/2 = 1.25MHz$: frecuencia tal que cualquier componente espectral que se encuentre por encima de ella crea aliasing.
- Ancho de banda de la señal: $BW = \frac{f_{clk}/Ndiv}{2 \cdot OSR} = 19.5KHz$, donde $OSR = 64$.

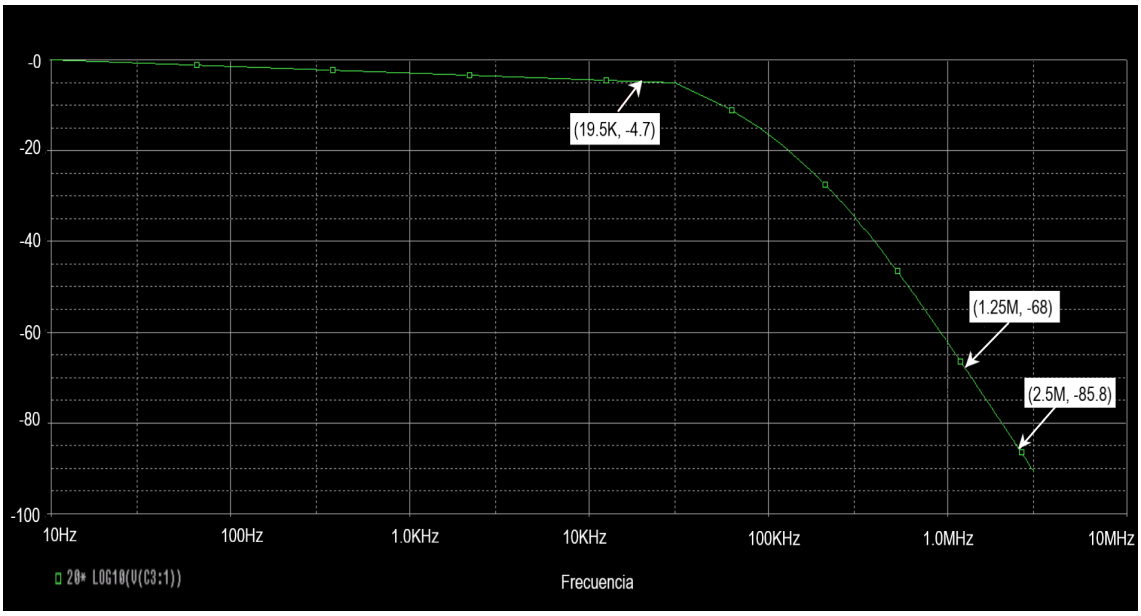


Figura 5.2 Simulación AC en PSpice del filtro.

5.2 PicoScope

La observación y análisis de las señales a la salida del SDM se realizan mediante el osciloscopio digital *PicoScope* tal y como se ha mencionado previamente. Dicho análisis será tanto de la señal temporal (Modo de osciloscopio) como la correpondiente FFT (Modo de espectro).

Además, se trabaja con dos modelos de PicoScope: de 12 bits y de 16 bits. El primer elemento de medición se emplea para capturar las señales de salida del SDM previas a ser filtradas gracias a la mayor rapidez de este instrumento frente al de 16 bits que será empleado para capturar y generar ficheros de puntos de las señales anteriores una vez filtradas que se analizarán empleando los Códigos 7.3 y 7.4

5.2.1 Configuración de captura y opciones de canal

A continuación se muestra sucintamente los valores empleados para configurar la representación temporal y espectral de la señal y a partir de los cuales se han extraído los ficheros de datos posteriormente empleados en *MATLAB* (Tabla 5.1).

5.3 Valoración de las prestaciones del circuito implementado en VHDL

Para evaluar el buen funcionamiento del diseño del circuito realizado en VHDL se realiza un estudio espectral de las señales representadas en el osciloscopio digital. Dicho estudio se basará en los valores de dos parámetros: SNR y HD2. Estos parámetros se calculan empleando un código de Matlab, el cuál carga un fichero de datos generado a partir de las medidas obtenidas en PicoScope.De este modo, un valor de SNR próximo a 70-71 dB demuestra que se ha realizado una correcta implementación del circuito a estudio.

Tabla 5.1 Valores asignados a las distintas configuraciones y opciones de canal de PicoScope.

Modo de Osciloscopio	
Control de base de tiempo	5 ms/div
Control de muestras	1 MS
Modo de Espectro	
Control del rango de frecuencia	2 MHz
Colectores de espectro	32768
Función de ventada	Hann
Eje y. Escala	Logarítmica
Unidad logarítima	dBu
Eje x. Escala	Lineal
Opciones de canal	
Sonda	x10
Modo de acoplamiento	CA
Resolución de hardware	16 bits
	12 bits
Filtro paso bajo	20 kHz
Límite de ancho de banda	200 kHz

5.4 Análisis de las señales a la salida del SDM para distintas amplitudes de entrada

Como se ha mencionado previamente, se realizan análisis de los resultados experimentales asociados a la señal de salida para el caso en que la señal de entrada tenga una amplitud 5 dB por debajo de la amplitud máxima (A=-5dBfs) y para una amplitud 6 dB por debajo de la máxima (A=-6dBfs), concepto explicado en la expresión (3.5) del Capítulo 3.

Además, los valores asociados a las prestaciones de las señales de salida se comparan con los del caso ideal Tabla 3.1 como ya se ha mencionado previamente.

5.4.1 Señal de entrada a 5 dB por debajo de la amplitud máxima

Señales salNRZ y salRZ. Sin filtrar

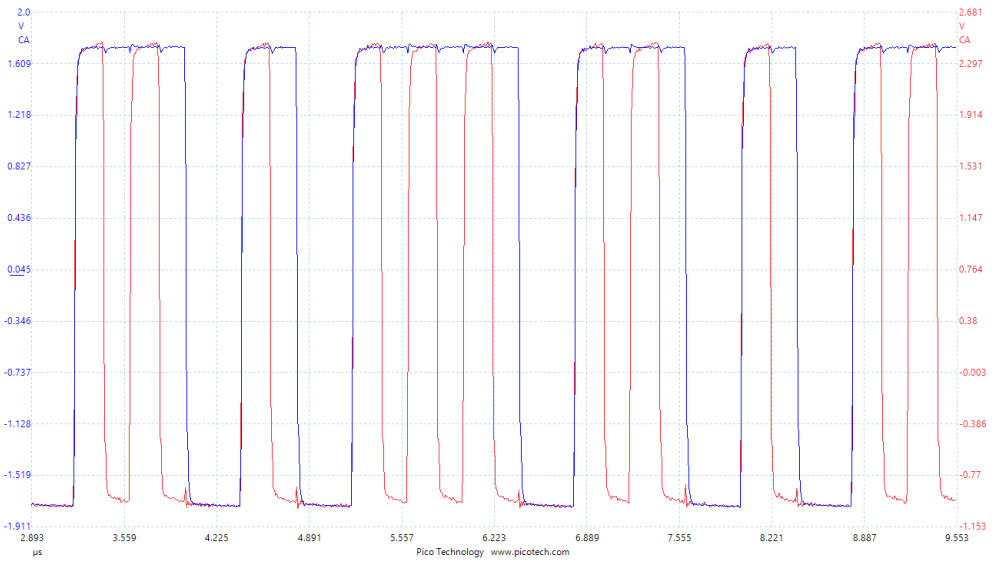


Figura 5.3 Representación temporal de salNRZ (azul) y salRZ (rojo) sin filtros medidas con PicoScope 12 bits para una amplitud Amp=-5dBfs.

Los resultados (Figura 5.3) muestran lo esperado. Ambas señales son pulsos que al ser superpuestos se identifica la señal salRZ como una copia de salNRZ con retorno a cero. Cabe destacar que estas medidas se han tomado con el medidor PicoScope con resolución hardware de 12 bits debido a que al ser este más rápido, permite realizar una mejor captura de estas señales. El resto de medidas se realizarán con el de 16 bits, justificándose esta decisión en el apartado 5.6.

Señal salNRZ.Filtrada

Empleando el filtro paso bajo (LPF) diseñado en *PSpice* (Figura 5.2) se obtienen las señal temporal (Figura 5.4) y espectral (Figura 5.5) para la salida del modulador con configuración sin retorno a cero.

En el caso de la representación temporal se observa como se ha obtenido una señal senoidal a la salida, luego se ha recuperado la forma de onda correctamente.

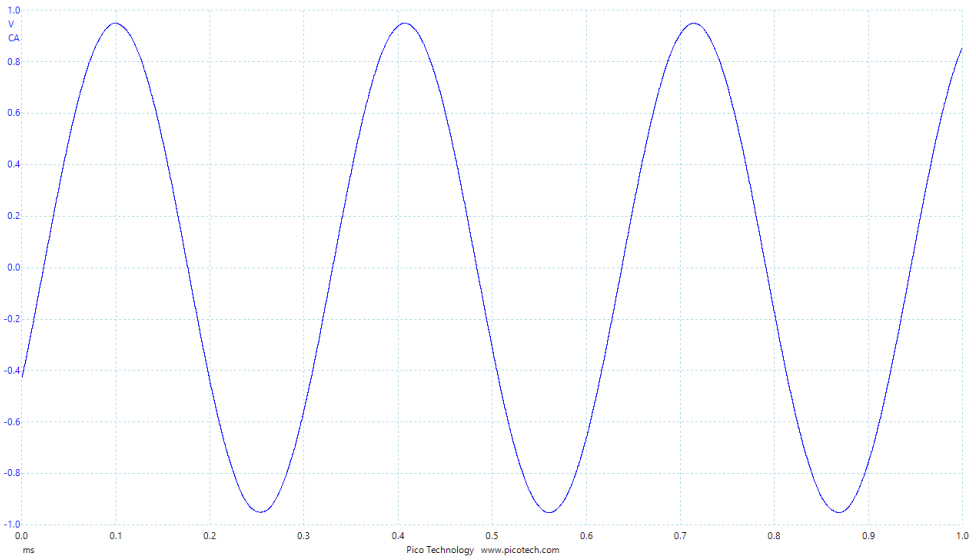


Figura 5.4 Representación mediante PicoScope en modo osciloscopio de la señal salNRZ, 100μs/div y amplitud -5dBfs.

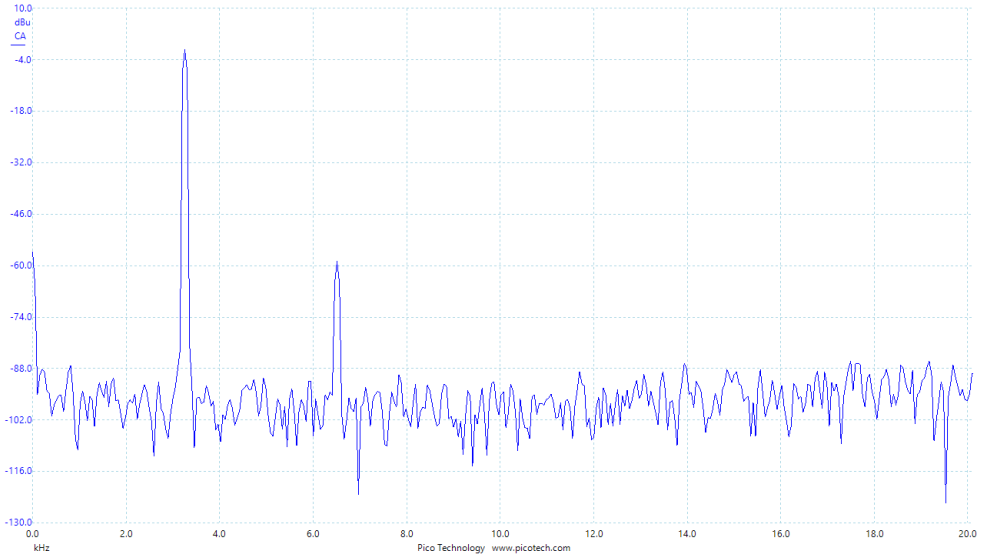


Figura 5.5 Representación mediante PicoScope en modo de espectro de la señal salNRZ con Amp=-5dBfs.

Además, los resultados tras realizar el estudio analítico en *MATLAB* se recoge en la Tabla 5.2 donde se

presentan los valores de los parámetros SNR y HD2 para las representaciones obtenidas tanto con el Modo de Osciloscopio como con el de Espectro (Tabla 5.2). Para ello se extraen ficheros de puntos para cada modo de PicoScope y se cargan en la rutina de *MATLAB* correspondiente. Dichos valores se pueden representar en *MATLAB* tal que se puedan comparar estas gráficas con las obtenidas usando PicoScope (Figura 5.6, Figura 5.7).

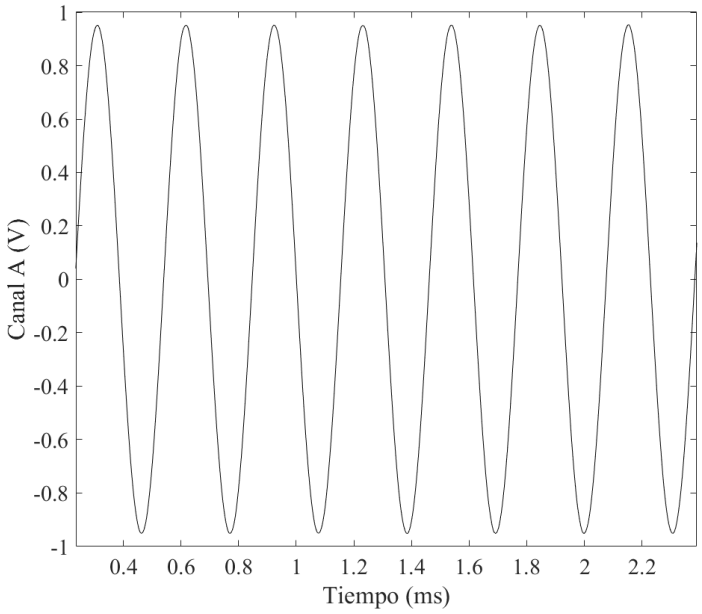


Figura 5.6 Representación mediante Matlab del fichero de puntos en PicoScope de la señal temporal salNRZ con Amp=-5dBfs.

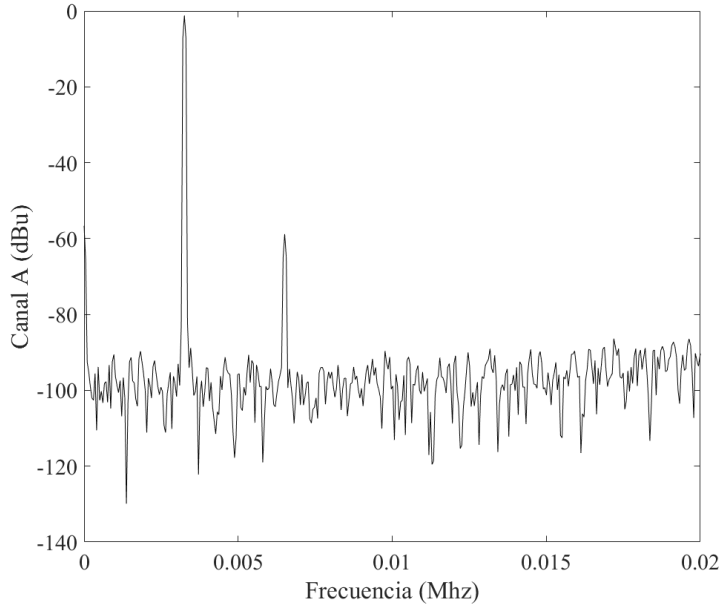


Figura 5.7 Representación mediante Matlab del fichero de puntos en PicoScope de la señal espectral salNRZ con Amp=-5dBfs.

En vista a los resultados plasmados en la Tabla 5.2 se aprecia como las prestaciones para la señal analizado son adecuadas en comparación con las del caso ideal inicial.

Tabla 5.2 Resultados obtenidos mediante Matlab de las prestaciones del circuito para la señal salRZ para el caso temporal y espectral con una amplitud 5 dB por debajo del valor máximo.

	SNR (dB)	HD2 (dB)
Señal Temporal	70.106705	57.795655
Espectro	70.037737	57.664973

Señal salRZ. Filtrada

Ahora, se exponen los resultados para la señal de salida del SDM con retorno a cero, salRZ. Al igual que se hizo para la señal salNRZ se representan las gráficas obtenidas mediante el PicoScope, así como los parámetros y gráficas resultantes de analizar los ficheros de puntos generados en PicoScope mediante *MATLAB*. Cabe mencionar que el análisis de la señal anterior con configuración RZ se realiza para poder comparar ambos casos y ver que configuración es la adecuada.

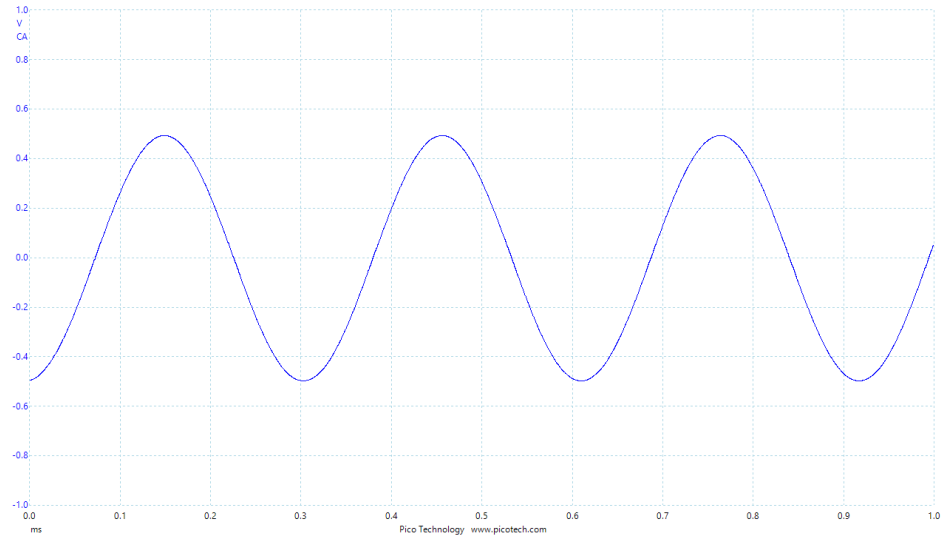


Figura 5.8 Representación mediante PicoScope en modo osciloscopio de la señal salRZ, 100μ s/div, con amplitud -5dBfs.

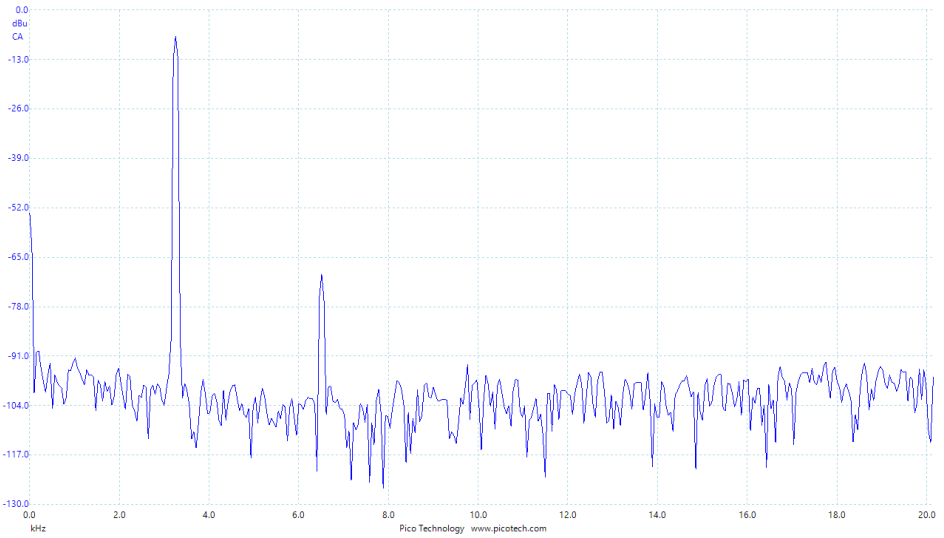


Figura 5.9 Representación mediante PicoScope en modo de espectro de la señal salRZ con Amp=-5dBfs.

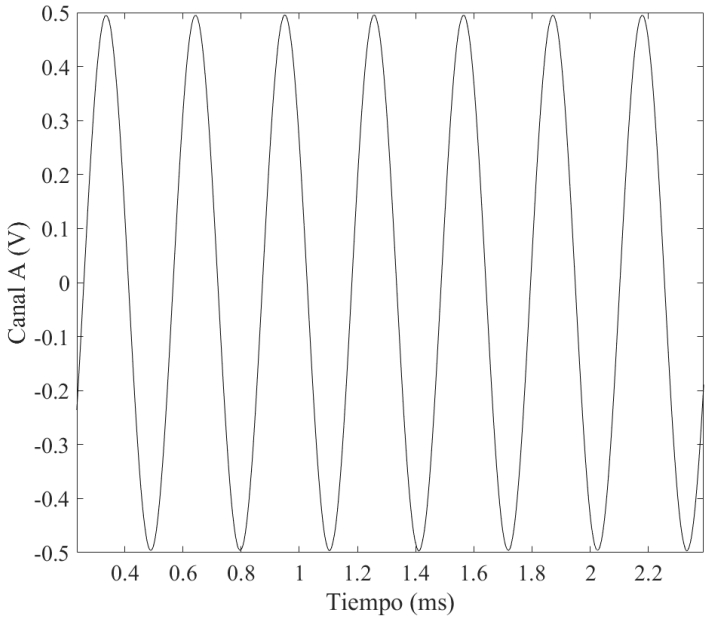


Figura 5.10 Representación mediante Matlab del fichero de puntos de la señal salRZ, Amp=-5dB, temporal en PicoScope.

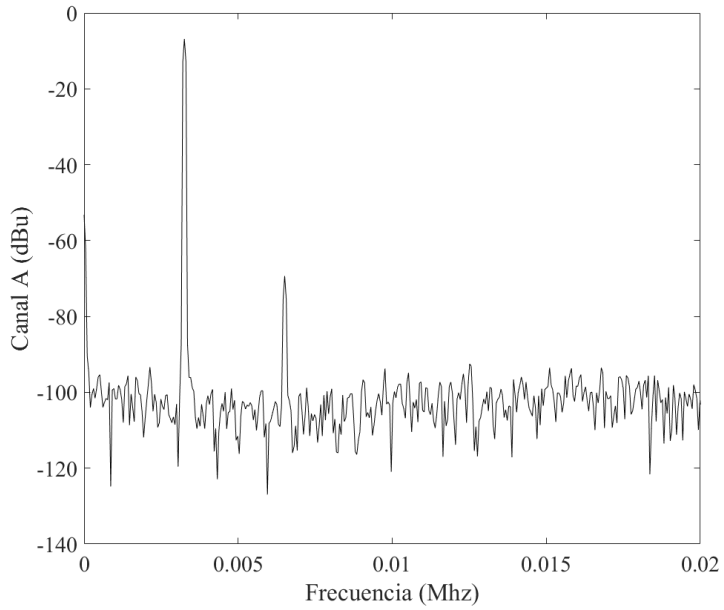


Figura 5.11 Representación mediante Matlab del fichero de puntos en PicoScope de la señal temporal salRZ con Amp=-5dB.

Tabla 5.3 Resultados obtenidos mediante Matlab de las prestaciones del circuito para la señal salRZ para el caso temporal y espectral con una amplitud 5 dB por debajo del valor máximo.

	SNR (dB)	HD2 (dB)
Señal Temporal	69.484629	62.589063
Espectro	70.168416	62.521225

5.4.2 Señal de entrada a 6 dB por debajo de la amplitud máxima

Por otro lado, se realiza el mismo análisis anterior pero ahora trabajando con una amplitud 6 dB por debajo del valor máximo ya que también se estudió dicho caso en el análisis preliminar en *MATLAB&Simulink*.

Señales salNRZ y salRZ. Sin filtrar

En la Figura 5.12 se observa un comportamiento similar al obtenido en el caso de -5dBfs y de nuevo queda reflejado el comportamiento de la configuración con retorno a cero.

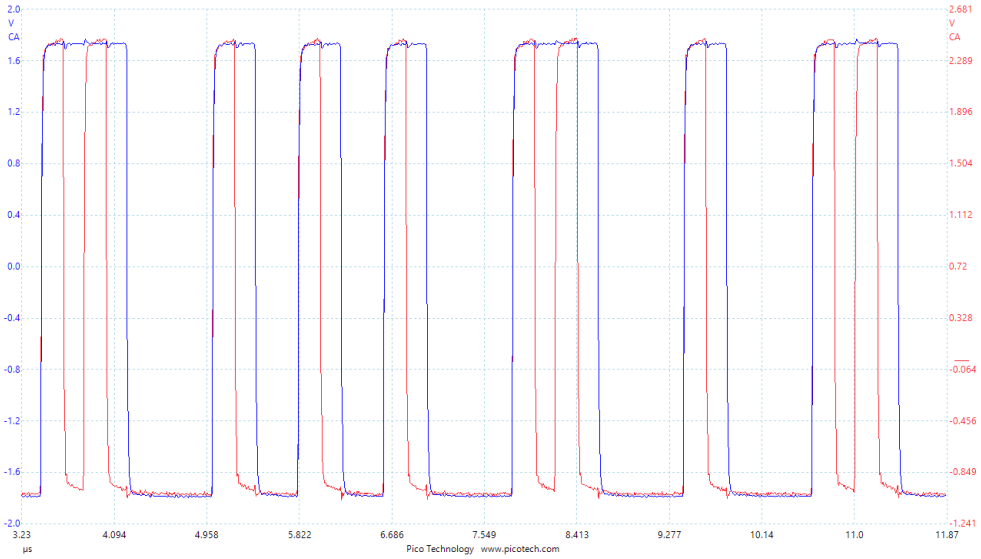


Figura 5.12 Representación temporal de salNRZ (azul) y salRZ (rojo) sin filtros medidas con PicoScope 12 bits para una Amplitud de -6dBfs.

Señal salNRZ. Filtrada

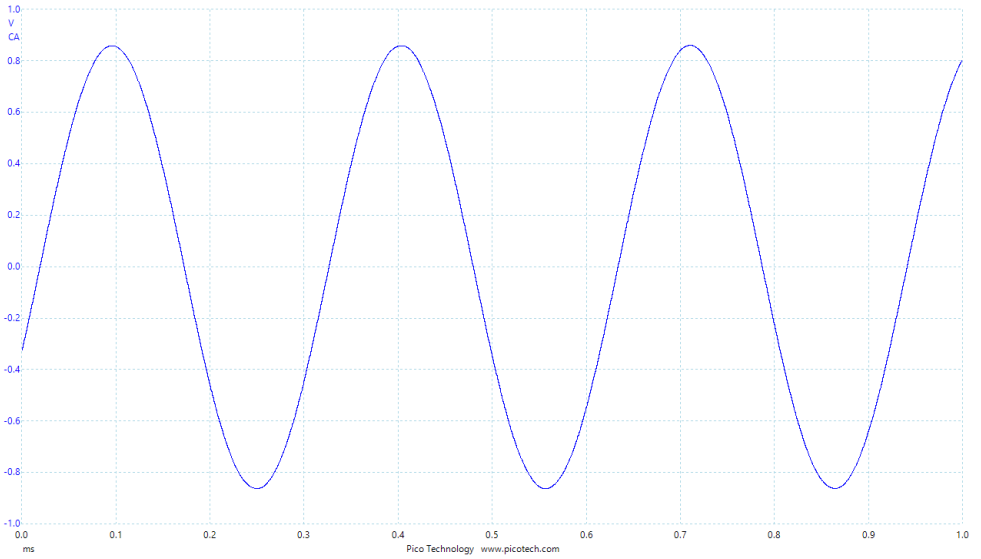


Figura 5.13 Representación mediante PicoScope en modo osciloscopio de la señal salNRZ, 100μs/div y amplitud -6dBfs.

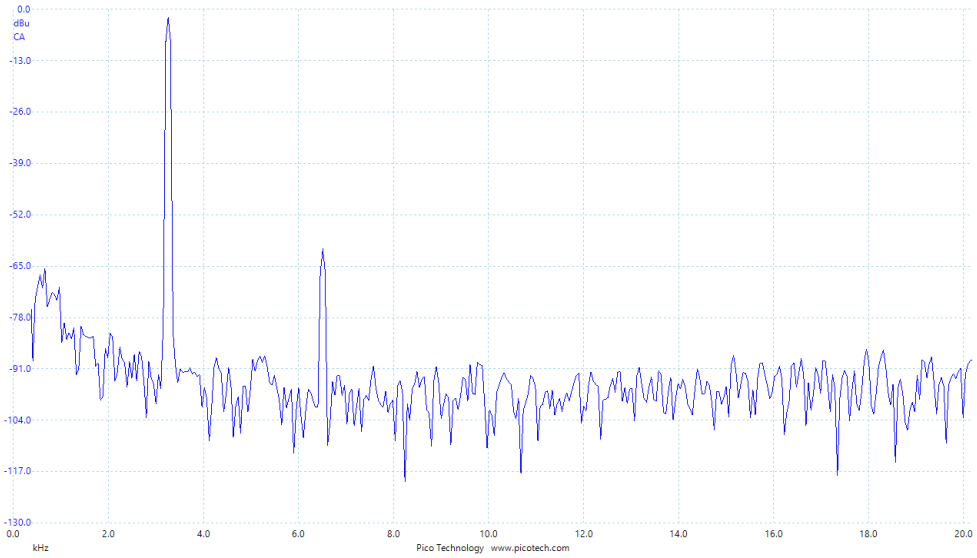


Figura 5.14 Representación mediante PicoScope en modo espectro de la señal salNRZ con Amp=-6dBfs.

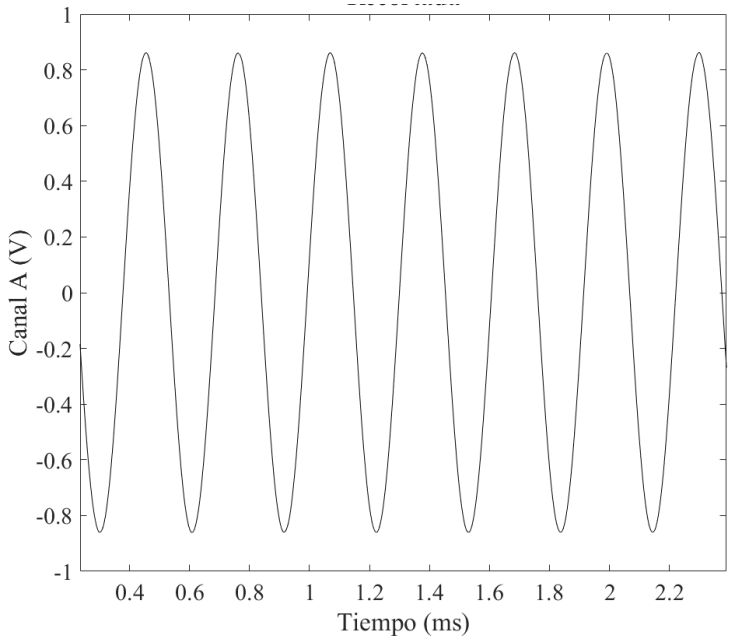


Figura 5.15 Representación mediante Matlab del fichero de puntos en PicoScope de la señal temporal salNRZ con Amp=-6dB.

Tabla 5.4 Resultados obtenidos mediante Matlab de las prestaciones del circuito para la señal salNRZ para el caso temporal y espectral con una amplitud 6 dB por debajo del valor máximo.

	SNR (dB)	HD2 (dB)
Señal Temporal	68.972134	58.784917
Espectro	68.749557	58.493055

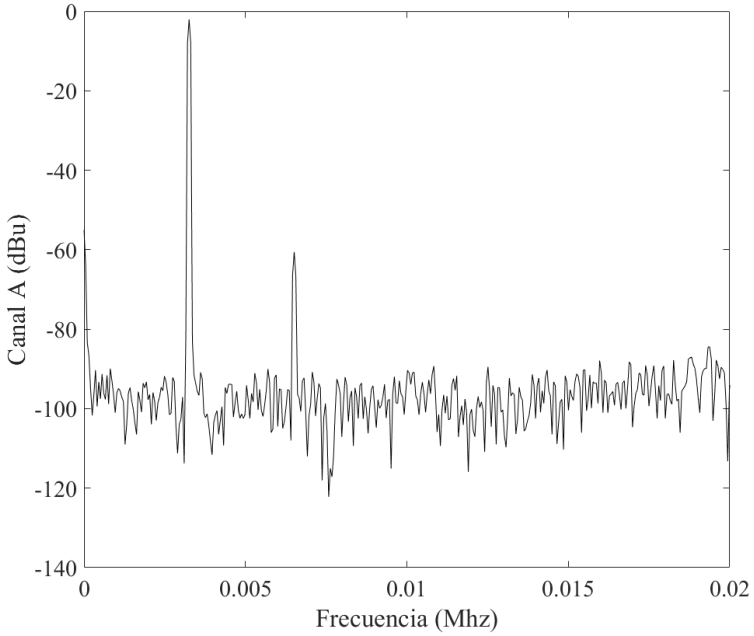


Figura 5.16 Representación mediante Matlab del fichero de puntos en PicoScope de la señal espectral salRZ con Amp=-6dBfs.

A partir de los resultados de la Tabla 5.4 se aprecia como las prestaciones son adecuadas así como una pequeña caída en el valor de SNR que va asociado a la disminución de 1 dB en la amplitud de la señal de entrada que se usa para el caso de -5dBfs.

Señal salRZ. Filtrada

Finalmente, se exponen las gráficas y prestaciones obtenidas para el caso de la señal con retorno a cero.

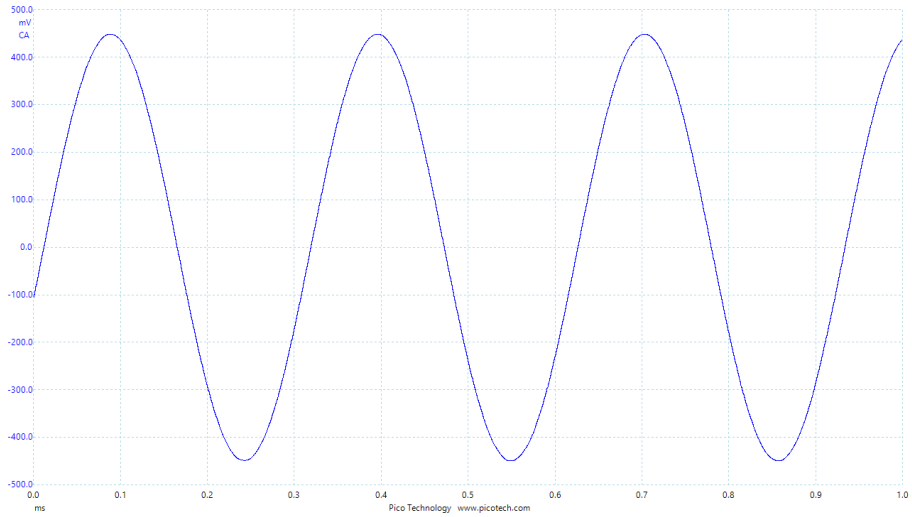


Figura 5.17 Representación mediante PicoScope en modo osciloscopio de la señal salRZ a 100μs/div y amplitud -6dBfs.

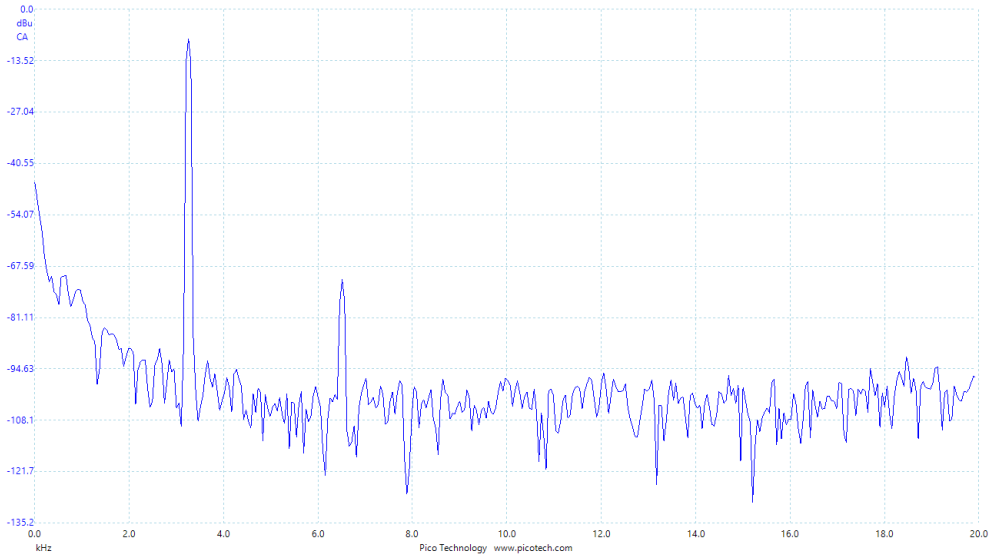


Figura 5.18 Representación mediante PicoScope en modo espectro de la señal salRZ con Amp=-6dBfs.

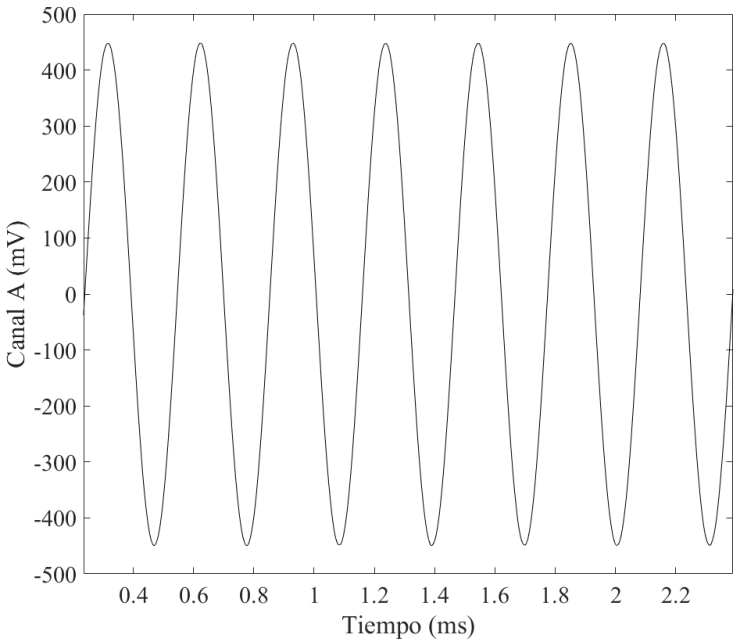


Figura 5.19 Representación mediante Matlab del fichero de puntos en PicoScope de la señal temporal salRZ con Amp=-6dBfs.

Tabla 5.5 Resultados obtenidos mediante Matlab de las prestaciones del circuito para la señal salRZ para el caso temporal y espectral con una amplitud 6 dB por debajo del valor máximo.

	SNR (dB)	HD2 (dB)
Señal Temporal	69.194053	63.548500
Espectro	68.685518	63.332733

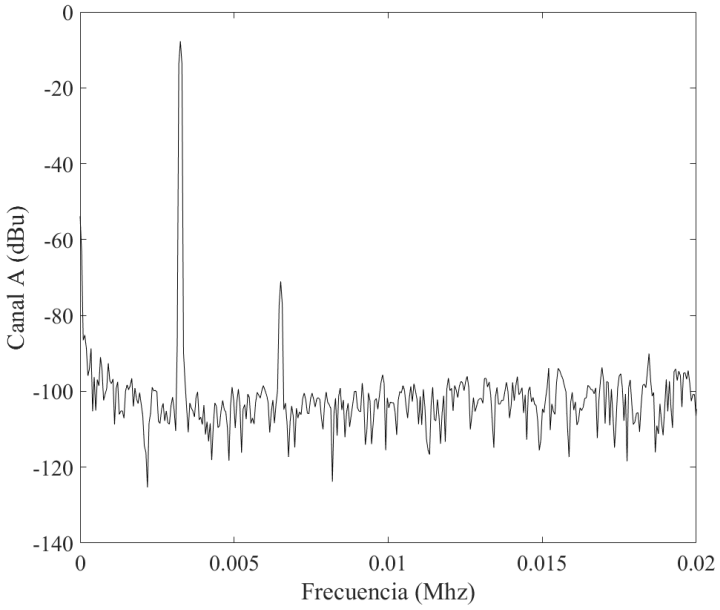


Figura 5.20 Representación mediante Matlab del fichero de puntos del en PicoScope de la señal espectral salRZ con Amp=-6dBfs.

De nuevo, los valores de las prestaciones para este caso (Tabla 5.5) son muy favorables de modo que el circuito se ha implementado de manera adecuada según esos resultados.

5.5 Conclusiones

Tras los resultados obtenidos para los distintos casos de señales de entrada y salida del SDM a lo largo de esta sección se concluye lo siguiente:

- Las señales de salida del SDM para ambos casos exhiben unas prestaciones muy buenas, y cercanas o incluso superando los 70 dB obtenidos incialmente en la simulación ideal (Tabla 3.1).
- Para el caso de la señal de entrada con una amplitud 6dB por debajo de la amplitud máxima, las prestaciones también toman valores similares a las obtenidas para el caso ideal.
- En las representaciones temporales de ambos casos se observa una señal senoidal perfecta lo que muestra que se ha recuperado la señal de manera adecuada.
- La amplitud de la señal senoidal disminuye en el caso de salRZ. Esto se debe a que, como se observa en la Figura 5.3, los pulsos RZ tienen un área menor que los asociados a la configuración NRZ (sin retorno a cero), y al ser la amplitud de la señal seno función de dicha área consecuentemente se produce una disminución de la amplitud de la señal. Lo mismo sucede con la representación espectral, donde el valor máximo en dBu de la señal sin retorno a cero es superior al obtenido con retorno a cero.
- En cuanto a la representación espectral, se observa como la frecuencia tiene un armónico en la frecuencia fundamental $f_0 = 3.26kHz$ y un armónico secundario entorno a $6.5kHz$.
- En referencia a las tablas de prestaciones, se observa un comportamiento similar en ambos casos en cuanto a que el valor de HD2 aumenta para la señal salRZ concluyéndose así que la codificación con retorno a cero mejora dicha prestación.

5.6 Justificación del uso del osciloscopio de resolución 16 bits frente al de 12 bits

A continuación, se justifica la elección del uso del osciloscopio de 16 bits en lugar del de 12 en base a una comparativa entre las prestaciones obtenidas en Tabla 5.2, Tabla 5.3 y las alcanzadas tras tomar unas medidas iniciales para una caída de 5 dB con resolución de 12 bits. Dichas prestaciones son asociadas al análisis temporal para una señal -5dBfs, aunque los resultados serán extrapolables para el espectro de la señal.

Tabla 5.6 Resultados obtenidos mediante Matlab de las prestaciones del circuito para la señal salNRZ para el caso temporal y espectral.

	SNR (dB)	HD2 (dB)
salNRZ	61.397098	54.455619
salRZ	54.459908	56.894290

A partir de lo expuesto en la Tabla 5.6 se observa cómo en este caso el ruido del instrumento PicoScope es predominante, habiendo casi apenas aumento de HD2 para la señal salRZ siendo este el motivo por el cuál se estima conveniente la toma de medidas a partir del espectro digital de 16 bits.

6 Conclusiones y mejoras

En primer lugar, este proyecto tenía por objetivo implementar un circuito que aún no siendo muy complejo tiene gran aplicabilidad hoy en día. En él también se han comparado resultados de distintos SW con características variadas pudiendo así reflexionar sobre las ventajas e inconvenientes que el uso de cada uno de ellos conlleva.

Así, para un primer análisis, sencillo, rápido y no susceptible a errores de retrasos y ruido se emplea *MATLAB* y *Simulink*. Sin embargo, para realizar un análisis más realista se emplea el SW *Vivado* para finalmente implementar el circuito en una FPGA y tomar medidas con un osciloscopio.

Además, este proyecto también ha puesto de manifiesto la versatilidad que los dispositivos programables tipo FPGA ofrecen en tanto que se ha podido reprogramar tantas veces como se ha necesitado.

Por otro lado, hubo que hacer una modificación en el diseño de la Lookup Table debido al elevado número de multiplexores que involucraban introducir un número grande de valores, trabajando así con muestras para solo un cuarto de período. La modificación mencionada supuso un nuevo reto en la implementación del circuito en VHDL ya que ahora la señal seno se debía generar dentro del propio circuito empleando para ello varios biestables que crean señales adicionales de 1 bit, las cuales definen el crecimiento de la señal y su signo. Además, se añade un tercer registro (Din) para cortar con la dinámica combinacional y eliminar posibles retrasos.

En cuanto a las prestaciones obtenidas en los resultados experimentales, se expone una comparativa (Tabla 6.1) con los resultados asociados a la simulación ideal del modelo en *MATLAB* y los asociados a las medidas experimentales. Se observa como los valores asociados a la SNR son aproximadamente 70 dB en todos los casos, quedando así comprobado que lo obtenido experimentalmente coincide con el caso inicial.

Tabla 6.1 Comparativa de las prestaciones del circuito obtenidas antes y después de su implentación en la FPGA.

	Caso ideal	Caso experimental	
Amplitud	SNR (dB)	SNR salNRZ (dB)	SNR salRZ (dB)
-5	69.858	70	70.2
-6	69.929	68.7	68.7

Cabe destacar también la mejoría del parámetro HD2 de la señal con retorno a cero (salRZ) respecto a la obtenida para el caso sin retorno a cero (salNRZ) como se refleja en las tablas de la sección anterior. Este resultado justifica la decisión que se tomo al principio del trabajo en cuanto al uso de esta configuración y respalda la hipótesis de que dicha configuración mejoraría los resultados debido a las transiciones existentes en los pulsos.

Finalmente, se exponen una serie de mejoras o líneas futuras de desarrollo del proyecto. En primer lugar, se propone habilitar un puerto USB para introducir entradas desde el exterior, por ejemplo, para introducir

señales de audio. En segundo lugar, se plantea añadir un single-bit DAC para mejorar los valores obtenidos para el HD2 como el de la Figura 6.1

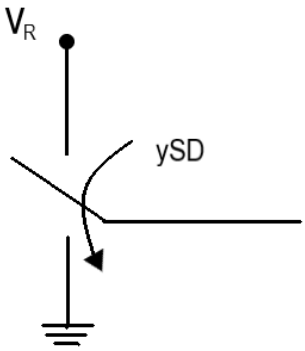


Figura 6.1 Esquema de la propuesta de mejora del HD2 usando un single-bit DAC.

7 Apéndice I. Códigos Matlab

7.1 Análisis fichero de puntos Vivado

Señal salNRZ

El siguiente código tiene por objetivo analizar la señal de salida del SDM sin retorno a cero que toma valores binarios entre '1' y '0'. Dicho análisis se traduce en el cálculo de la SNR de la señal y la representación de la señal y su correspondiente espectro.

Es imprescindible explicar la necesidad de la transformación realizada en el bucle for donde se convierten los datos de entrada tal que varían entre '0' y '+1', en '-1' y '+1'. Además, el índice asociado a la señal se obtiene mediante la localización del máximo de la representación en frecuencia.

Por último, para dar valor a HD2, se estima gráficamente, a partir de la representación espectral, los índices dentro de los que se contiene el segundo armónico. Dicho paso queda indicado en el código. Para una mejor comprensión se muestra un ejemplo en la figura Figura 7.1 en cuyo caso se incluirían los valores siguientes en el código:

$$Nh2\_1 = 62; \;\; Nh2\_2 = 66;$$

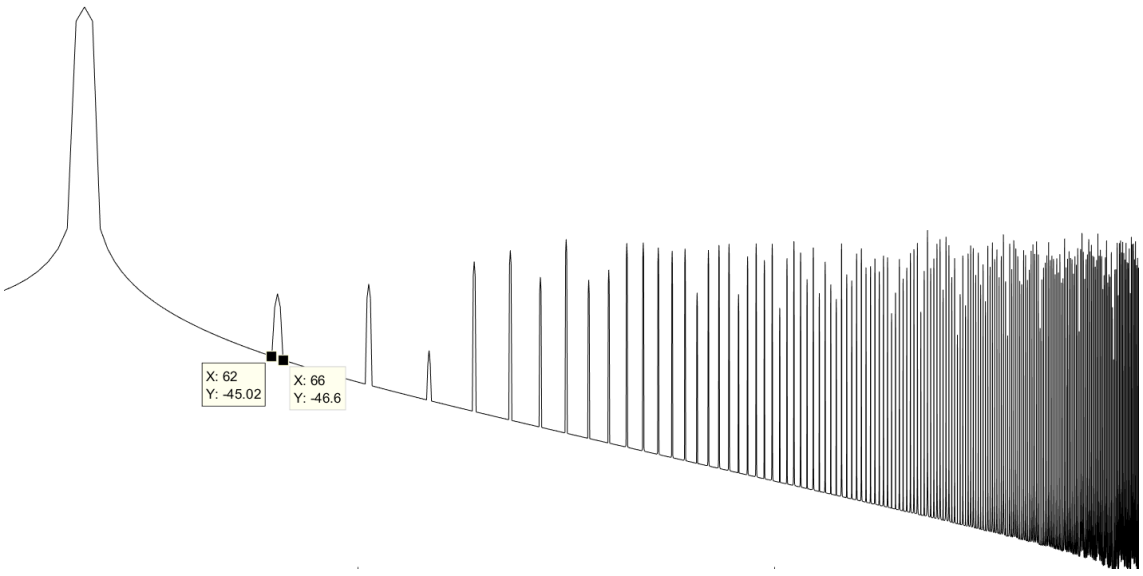


Figura 7.1 Representación de la PSD de la señal de salida del SDM a modo de ejemplo para determinar los índices del segundo armónico..

Código 7.1 analizaNRZ.m.

```
clear all;

%% PARÁMETROS
nBits=14;
R= 2^(nBits-1);
R2=2^nBits;
FD=64;
Narm=6;
Np1p=2*FD*Narm; % Número de valores de la Lookup Table

fs=1024;        % Frecuencia de sobremuestreo
ts=1/fs;
%%

load('salida_RON.txt') %salida_RON: fichero de puntos extraído de Vivado
datos=salida_RON(:,1);
y0=datos;
Nvhdl=length(y0);          % Número de valores de VHDL
Nptos=Np1p*floor(Nvhdl/Np1p); % redondea a la baja

for ind=1:1:Nvhdl,
    if y0(ind)==0,
        y0(ind)=-1;
    end
end

%% Calulo PSD, SNR y HD2
ventana=window(@hann,Nptos);    %  empleo una ventana temporal hann
psd=pwelch(y0,ventana,Nptos-1,Nptos);
[Pmax,Nfs]=max(psd);            % Índice asociado a la señal-
Nbw=round(Nptos/2/FD);
Lpsd=length(psd);
freq=[0:1:Lpsd-1]*fs/2;
semilogx(freq,10*log10(psd))

senal=sum(psd(Nfs-2:Nfs+4));
ruido=sum(psd(3:Nbw))-senal;

%Segundo armónico
Nh2_1=;                      % Límite inferior
Nh2_2=;                      % Límite superior
harm2=sum(psd(Nh2_1:Nh2_2));

snr=10*log10(senal/ruido);
HD2=10*log10(senal/harm2);
%%

fprintf(1,'SNR (dB): %f HD2: %f (dB)\n',snr,HD2);
plot(y0)
```

Señal de entrada al SDM

También resulta de interés analizar cómo es la señal que se introduce al SDM y que se genera mediante el componente contador.

La finalidad de este código es representar dicha señal y su espectro junto con analizar su SNR para verificar que, en efecto, la señal de entrada corresponde a una señal sinuosidal. Todo ello se realiza de manera análoga al código asociado al estudio de la señal salNRZ con algunos cambios sustanciales. Además cabe mencionar que en el código VHDL 8.17 se convierte los valores binarios asociados a la señal entSD en números enteros, y es por ello que en el bucle `for` ahora se realiza la transformación de dichos valores enteros en CA2 a su valor decimal con signo. Otra modificación respecto al código 7.1 radica en que una vez cargado el fichero de puntos, se asocia la segunda columna de valores a la variable x.

Código 7.2 analizaentrada.m.

```
clear all;

%%PARÁMETROS
nBits=14;
FD=64;
Narm=6;
Np1p=2*FD*Narm; %número de valores de la Lookup Table

fs=1024; % Frecuencia de sobremuestreo
ts=1/fs;
%%

load('salida_RON.txt');
x=salida_RON(:,2); %entrada al SDM

Nvhdl=length(x); %Número de valores de VHDL
Nptos=Np1p*floor(Nvhdl/Np1p); %redondea a la baja

for i=1:Nvhdl
    if x(i)>=2^(nBits-1)
        x(i)=x(i)-2^nBits;
    end
end

%% Calulo de la PSD, SNR y HD2
ventana=window(@hann,Nptos);        % empleo una ventana temporal hann
psd=pwelch(x,ventana,Nptos-1,Nptos);
[Pmax,Nfs]=max(psd);                % índice asociado a la señal
Nbw=round(Nptos/2/FD);
Lpsd=length(psd);
freq=[0:1:Lpsd-1]*fs/2;
semilogx(10*log10(psd))

senal=sum(psd(Nfs-2:Nfs+4));
ruido=sum(psd(3:Nbw))-senal;

%Segundo armónico
Nh2_1=;
Nh2_2=;
harm2=sum(psd(Nh2_1:Nh2_2);

snr=10*log10(senal/ruido);
HD2=10*log10(senal/harm2);
%%

fprintf(1,'SNR (dB): %f HD2: %f (dB)\n',snr,HD2);
```

```
plot(x)        % Representación señal senoidal a la entrada del SDM
```

7.2 Análisis fichero de puntos PicoScope

Señal temporal

Código 7.3 analizaPScope_temp.m.

```
load('prb.txt'); %prb: fichero de puntos extraído de PicoScope
y=prb(:,2);
t=prb(:,1);

plot(t,y);
title('Antes de recortar');

%%%%%%%%%PARÁMETROS%%%%%%%%%
Nsample=length(y);
OSR=64;
Narm=6;
Ndiv=40;
ts=(t(2)-t(1))*1e-3;
fs=1/ts;
fp=1e8/Ndiv/2/OSR/Narm;
Tp=1/fp;
Np1Tp=fs/fp;
Nptos=round(Np1Tp*floor(Nsample/Np1Tp));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure
plot(t(Nsample-Nptos+1:Nsample),y(Nsample-Nptos+1:Nsample),'k');
xlabel('Tiempo (ms)'); ylabel('Canal A (V)')
title('Recortada'); % Se recorta la señal un número entero de períodos

%%%%%%%%%%%%%%%%%%%%%%%%%CALCULO PSD Y SNR%%%%%%%%%%%%%%%%%%%%%%%%%
ventana=window(@hann,Nptos);
psd=pwelch(y(Nsample-Nptos+1:Nsample),ventana,Nptos-1,Nptos);

[Py_max,ind_fpp]=max(psd);
fpp=ind_fpp*fs/Nptos;
fprintf(1,'Frecuencia señal; medida: % f y teorica: %f (KHz)\n',fpp*1e-3,fp*1e
    -3);

Lpsd=length(psd);

figure
freq=[0:1:Lpsd-1]*fs/2;
semilogx(10*log10(psd));

% PARÁMETROS A DAR VALOR
Nfs=163;                  %frecuencia 1er armónico
Nbw=6*Nfs;
senal=sum(psd(159:168)); %intervalo 1er armónico
harm2=sum(psd(323:328)); %intervalo 2o armónico
ruido=sum(psd(6:Nbw))-senal-harm2;
%%%%%
```

```
snr=10*log10(senal/ruido);
HD2=10*log10(senal/harm2);
fprintf(1,'SNR (dB): %f HD2: %f (dB)\n',snr,HD2);
```

Espectro

Código 7.4 analizaPScope_FFTlin.m.

```
load('prb.txt'); %prb: fichero de puntos extraído de PicoScope (en escala logar
    ítmica)
y=(prb(:,2));
freq=prb(:,1);

plot((y))

yn=10.^(y/20); %se trabaja en escala natural

%% PARAMETROS A DAR VALOR
Nfs=65;
Nbw=6*Nfs;
senal=(sum(yn(62:70).^2));
harm2=(sum(yn(126:133).^2));
ruido=(sum(yn(5:Nbw).^2))-senal-harm2;
%%%%%

snr=10*log10(senal/ruido);
HD2=10*log10(senal/harm2);
fprintf(1,'SNR (dB): %f HD2: %f (dB)\n',snr,HD2);

plot(freq,y, 'k')
xlabel('Frecuencia (Mhz)')
ylabel ('Canal A (dBu)')
```

8 Apéndice II. Códigos VHDL

En este apéndice se exponen todos los códigos VHDL empleados durante el proyecto así como una explicación de ellos cuando se considere necesaria.

Design Sources

8.1 Divisor de frecuencias

Código 8.1 divisor_frecuencia.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity divisor_frecuencia is
generic ( Ncont: integer := 6; Ndiv: integer := 40; Nduty: integer := 21);
   Port ( clk : in STD_LOGIC;
          rasin: in STD_LOGIC;
          EN   : out STD_LOGIC;
          PRZ  : inout STD_LOGIC);
end divisor_frecuencia;

architecture Behavioral of divisor_frecuencia is
---------------------------------------------------------------------------------
signal uno      : std_logic_vector(Ncont-1 downto 0) := (0=>'1', others=>'0');
signal qo, qi   : std_logic_vector(Ncont-1 downto 0);
signal iguales  : std_logic := '0';
signal igualesPRZ: std_logic := '0';
---------------------------------------------------------------------------------
begin

process(rasin,clk,qi)
begin
    if (rasin='1') then
        qo <= (0 => '0',others => '0');
  elsif (clk'event and clk='1') then
      qo <= qi;
  end if;
end process;
```

```
EN  <= iguales;
PRZ <= igualesPRZ;

iguales <= '1' when (Ndiv-1) = qo else
          '0';

igualesPRZ <= '1' when Nduty > qo else
             '0';

qi <= (qo+uno) when (iguales='0') else
      (others => '0');

end Behavioral;
```

8.2 Contador

Código 8.2 contador.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity contador is
generic ( Nmuestras: integer := 192; Nb: integer := 14);
   Port ( clk     : in STD_LOGIC;
          rasin   : in STD_LOGIC;
          EN      : in STD_LOGIC;
          address : inout std_logic_vector (Nb-1 downto 0);
          entSD   : out std_logic_vector (Nb-1 downto 0));
end contador;

architecture Behavioral of contador is
---------------------------------------------------------------------------
signal salLUT   : std_logic_vector(Nb-1 downto 0);
signal uno      : std_logic_vector(Nb-1 downto 0) := (0=>'1', others=>'0');
signal qoc, qic : std_logic_vector(Nb-1 downto 0);
signal iguales  : std_logic := '0';
signal sele     : std_logic_vector(1 downto 0) := (others => '1');
signal sube,signo : std_logic;
signal salreg   : std_logic_vector(Nb-1 downto 0);
---------------------------------------------------------------------------
component D_sube
generic ( Nmuestras: integer := 192; Nb: integer := 14);
   Port ( address     : in STD_LOGIC_vector(Nb-1 downto 0);
          salida      : out STD_LOGIC;
          clk,rasin,EN : in STD_LOGIC);
end component;
component D_signo
generic ( Nb: integer := 14);
   Port ( address    : in STD_LOGIC_vector(Nb-1 downto 0);
          salida     : out STD_LOGIC;
```

```
         clk,rasin,EN : in STD_LOGIC);
end component;
component D_entr
generic ( Nb: integer :=Nb);
   Port ( entrada    : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          salida     : out STD_LOGIC_VECTOR (Nb-1 downto 0);
          clk,rasin,EN : in STD_LOGIC);
end component;
-------------------------------------------------------------------------------
begin

process(clk,qic,rasin)
begin
    if (rasin='1') then
        qoc <= (0 => '0',others => '0');
  elsif (clk'event and clk='1') then
      qoc <= qic;
  end if;
end process;

address <= qoc;

sele <= sube & EN;
with sele select
qic <= qoc+uno         when "11",
       qoc-uno         when "01",
       qoc            when others;

-----------------------LOOKUP TABLE-------------------------
salLUT<="00000000000000" when address="00000000000000" else
"00000000100110" when address="00000000000001" else
"00000001001011" when address="00000000000010" else
"00000001110001" when address="00000000000011" else
"00000010010111" when address="00000000000100" else
"00000010111100" when address="00000000000101" else
"00000011100010" when address="00000000000110" else
"00000100001000" when address="00000000000111" else
"00000100101101" when address="00000000001000" else
"00000101010011" when address="00000000001001" else
"00000101111000" when address="00000000001010" else
"00000110011110" when address="00000000001011" else
"00000111000100" when address="00000000001100" else
"00000111101001" when address="00000000001101" else
"00001000001110" when address="00000000001110" else
"00001000110100" when address="00000000001111" else
"00001001011001" when address="00000000010000" else
"00001001111111" when address="00000000010001" else
"00001010100100" when address="00000000010010" else
"00001011001001" when address="00000000010011" else
"00001011101110" when address="00000000010100" else
"00001100010100" when address="00000000010101" else
"00001100111001" when address="00000000010110" else
"00001101011110" when address="00000000010111" else
"00001110000011" when address="00000000011000" else
"00001110101000" when address="00000000011001" else
"00001111001101" when address="00000000011010" else
"00001111110001" when address="00000000011011" else
```

```
"00010000010110" when address="00000000011100" else
"00010000111011" when address="00000000011101" else
"00010001011111" when address="00000000011110" else
"00010010000100" when address="00000000011111" else
"00010010101000" when address="00000000100000" else
"00010011001101" when address="00000000100001" else
"00010011110001" when address="00000000100010" else
"00010100010101" when address="00000000100011" else
"00010100111001" when address="00000000100100" else
"00010101011101" when address="00000000100101" else
"00010110000001" when address="00000000100110" else
"00010110100101" when address="00000000100111" else
"00010111001001" when address="00000000101000" else
"00010111101100" when address="00000000101001" else
"00011000010000" when address="00000000101010" else
"00011000110011" when address="00000000101011" else
"00011001010111" when address="00000000101100" else
"00011001111010" when address="00000000101101" else
"00011010011101" when address="00000000101110" else
"00011011000000" when address="00000000101111" else
"00011011100011" when address="00000000110000" else
"00011100000110" when address="00000000110001" else
"00011100101000" when address="00000000110010" else
"00011101001011" when address="00000000110011" else
"00011101101101" when address="00000000110100" else
"00011110001111" when address="00000000110101" else
"00011110110010" when address="00000000110110" else
"00011111010100" when address="00000000110111" else
"00011111110101" when address="00000000111000" else
"00100000010111" when address="00000000111001" else
"00100000111001" when address="00000000111010" else
"00100001011010" when address="00000000111011" else
"00100001111100" when address="00000000111100" else
"00100010011101" when address="00000000111101" else
"00100010111110" when address="00000000111110" else
"00100011011111" when address="00000000111111" else
"00100011111111" when address="00000001000000" else
"00100100100000" when address="00000001000001" else
"00100101000000" when address="00000001000010" else
"00100101100001" when address="00000001000011" else
"00100110000001" when address="00000001000100" else
"00100110100001" when address="00000001000101" else
"00100111000000" when address="00000001000110" else
"00100111100000" when address="00000001000111" else
"00100111111111" when address="00000001001000" else
"00101000011111" when address="00000001001001" else
"00101000111110" when address="00000001001010" else
"00101001011101" when address="00000001001011" else
"00101001111011" when address="00000001001100" else
"00101010011010" when address="00000001001101" else
"00101010111000" when address="00000001001110" else
"00101011010110" when address="00000001001111" else
"00101011110100" when address="00000001010000" else
"00101100010010" when address="00000001010001" else
"00101100110000" when address="00000001010010" else
"00101101001101" when address="00000001010011" else
"00101101101010" when address="00000001010100" else
```

```
"00101110000111" when address="00000001010101" else
"00101110100100" when address="00000001010110" else
"00101111000001" when address="00000001010111" else
"00101111011101" when address="00000001011000" else
"00101111111010" when address="00000001011001" else
"00110000010110" when address="00000001011010" else
"00110000110001" when address="00000001011011" else
"00110001001101" when address="00000001011100" else
"00110001101001" when address="00000001011101" else
"00110010000100" when address="00000001011110" else
"00110010011111" when address="00000001011111" else
"00110010111001" when address="00000001100000" else
"00110011010100" when address="00000001100001" else
"00110011101110" when address="00000001100010" else
"00110100001000" when address="00000001100011" else
"00110100100010" when address="00000001100100" else
"00110100111100" when address="00000001100101" else
"00110101010101" when address="00000001100110" else
"00110101101111" when address="00000001100111" else
"00110110001000" when address="00000001101000" else
"00110110100000" when address="00000001101001" else
"00110110111001" when address="00000001101010" else
"00110111010001" when address="00000001101011" else
"00110111101001" when address="00000001101100" else
"00111000000001" when address="00000001101101" else
"00111000011000" when address="00000001101110" else
"00111000110000" when address="00000001101111" else
"00111001000111" when address="00000001110000" else
"00111001011110" when address="00000001110001" else
"00111001110100" when address="00000001110010" else
"00111010001010" when address="00000001110011" else
"00111010100001" when address="00000001110100" else
"00111010110110" when address="00000001110101" else
"00111011001100" when address="00000001110110" else
"00111011100001" when address="00000001110111" else
"00111011110110" when address="00000001111000" else
"00111100001011" when address="00000001111001" else
"00111100100000" when address="00000001111010" else
"00111100110100" when address="00000001111011" else
"00111101001000" when address="00000001111100" else
"00111101011100" when address="00000001111101" else
"00111101101111" when address="00000001111110" else
"00111110000011" when address="00000001111111" else
"00111110010110" when address="00000010000000" else
"00111110101000" when address="00000010000001" else
"00111110111011" when address="00000010000010" else
"00111111001101" when address="00000010000011" else
"00111111011111" when address="00000010000100" else
"00111111110000" when address="00000010000101" else
"01000000000010" when address="00000010000110" else
"01000000010011" when address="00000010000111" else
"01000000100100" when address="00000010001000" else
"01000000110100" when address="00000010001001" else
"01000001000100" when address="00000010001010" else
"01000001010100" when address="00000010001011" else
"01000001100100" when address="00000010001100" else
"01000001110011" when address="00000010001101" else
```

```
"01000010000011" when address="00000010001110" else
"01000010010001" when address="00000010001111" else
"01000010100000" when address="00000010010000" else
"01000010101110" when address="00000010010001" else
"01000010111100" when address="00000010010010" else
"01000011001010" when address="00000010010011" else
"01000011010111" when address="00000010010100" else
"01000011100101" when address="00000010010101" else
"01000011110001" when address="00000010010110" else
"01000011111110" when address="00000010010111" else
"01000100001010" when address="00000010011000" else
"01000100010110" when address="00000010011001" else
"01000100100010" when address="00000010011010" else
"01000100101101" when address="00000010011011" else
"01000100111000" when address="00000010011100" else
"01000101000011" when address="00000010011101" else
"01000101001110" when address="00000010011110" else
"01000101011000" when address="00000010011111" else
"01000101100010" when address="00000010100000" else
"01000101101011" when address="00000010100001" else
"01000101110101" when address="00000010100010" else
"01000101111110" when address="00000010100011" else
"01000110000110" when address="00000010100100" else
"01000110001111" when address="00000010100101" else
"01000110010111" when address="00000010100110" else
"01000110011111" when address="00000010100111" else
"01000110100110" when address="00000010101000" else
"01000110101101" when address="00000010101001" else
"01000110110100" when address="00000010101010" else
"01000110111011" when address="00000010101011" else
"01000111000001" when address="00000010101100" else
"01000111000111" when address="00000010101101" else
"01000111001101" when address="00000010101110" else
"01000111010010" when address="00000010101111" else
"01000111010111" when address="00000010110000" else
"01000111011100" when address="00000010110001" else
"01000111100001" when address="00000010110010" else
"01000111100101" when address="00000010110011" else
"01000111101001" when address="00000010110100" else
"01000111101100" when address="00000010110101" else
"01000111101111" when address="00000010110110" else
"01000111110010" when address="00000010110111" else
"01000111110101" when address="00000010111000" else
"01000111110111" when address="00000010111001" else
"01000111111001" when address="00000010111010" else
"01000111111011" when address="00000010111011" else
"01000111111100" when address="00000010111100" else
"01000111111101" when address="00000010111101" else
"01000111111110" when address="00000010111110" else
"01000111111111";
-------------------------------------------------------------

DSube : D_sube generic map (Nb=>Nb, Nmuestras=>Nmuestras)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, address=>address, salida=>sube);
DSigno: D_signo generic map (Nb=>Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, address=>address, salida=>signo);
Din   : D_entr generic map (Nb=>Nb)
```

```
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, entrada=>salLUT, salida=>salreg);

entSD <= salreg when signo='1' else
        not(salreg)+'1';

end Behavioral;
```

8.2.1 Biestable con señal sube como salida: Dsube

Código 8.3 D_sube.vhd.

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

entity D_sube is
generic ( Nmuestras: integer := 192; Nb: integer := 14);
   Port ( clk,rasin,EN : in STD_LOGIC;
          address     : in STD_LOGIC_vector(Nb-1 downto 0);
          salida      : out STD_LOGIC);
end D_sube;

architecture Behavioral of D_sube is
----------------------
signal Q,D: std_logic;
signal y : std_logic;
----------------------
begin

salida <=Q;

process(rasin,clk)
begin
    if (rasin='1') then
  Q <= '1';
  elsif (clk'event and clk='1') then
  Q <= D;
  end if;
end process;

D<= '0' when EN='1' AND "00000010111111"=address else
    '1' when EN='1' AND "00000000000001"=address else
     Q;

end Behavioral;
```

8.2.2 Biestable con señal signo como salida: Dsigno

Código 8.4 D_signo.vhd.

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

entity D_signo is
generic ( Nb: integer := 14);
```

```
   Port ( clk,rasin,EN: in STD_LOGIC;
          address    : in STD_LOGIC_vector(Nb-1 downto 0);
          salida     : out STD_LOGIC);
end D_signo;

architecture Behavioral of D_signo is
-----------------------
signal Q,D: std_logic;
signal y : std_logic;
-----------------------
begin

salida <=Q;

process(rasin,clk)
begin
    if (rasin='1') then
    Q <= '0';
  elsif (clk'event and clk='1') then
    Q <= D;
  end if;
end process;

D<= not(Q) when EN='1' AND address="00000000000000" else
    Q;

end Behavioral;
```

8.2.3 Registro Din

Código 8.5 D_entr.vhd.

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

entity D_entr is
generic (Nb: integer := 14);
   Port ( clk,rasin,EN : in STD_LOGIC;
          entrada     : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          salida      : out STD_LOGIC_VECTOR (Nb-1 downto 0));
end D_entr;

architecture Behavioral of D_entr is
-------------------------------------------
signal D,Q: std_logic_vector(Nb-1 downto 0);
-------------------------------------------
begin

salida <=Q;

process(rasin,clk)
begin
  if (rasin='1') then
      Q <= (0 => '0',others => '0');
  elsif (clk'event and clk='1') then
```

```
      Q <= D;
  end if;
end process;

D<= entrada when EN='1' else
    Q;

end Behavioral;
```

8.3 Sumador

Código 8.6 sumador.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sumador is
generic (Nb: integer := 14);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+5 downto 0));
end sumador;

architecture Behavioral of sumador is
-------------------------------------------
signal s1: std_logic_vector(Nb-1+5 downto 0);
signal s2: std_logic_vector(Nb-1+5 downto 0);
-------------------------------------------
begin

s1 <= "00000"&a when a(Nb-1)='0' else
      "11111"&a;
s2 <= '0'&b    when b(Nb+1)='0' else
      '1'&b;

y<= s1+s2;

end Behavioral;
```

8.4 Restador

Código 8.7 restador.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity restador1 is
generic (Nb: integer := 14);
    Port ( a : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           b : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           y : out STD_LOGIC_VECTOR (Nb-1+6 downto 0));
end restador1;

architecture Behavioral of restador1 is
---------------------------------------------------
signal s1, ns2: std_logic_vector(Nb-1+6 downto 0);
---------------------------------------------------
begin

s1 <= "0"&a when a(Nb-1+5)='0' else
      "1"&a;
ns2 <= '1'& not(b(Nb-1+5 downto 0))+'1' when b(Nb-1+5)='0' else
       '0'&not(b(Nb-1+5 downto 0))+'1';

y<= s1+ns2;

end Behavioral;
```

Código 8.8 restador2.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity restador2 is
generic (Nb: integer := 14; Nsat: integer :=17);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nsat-1 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end restador2;

architecture Behavioral of restador2 is
---------------------------------------------------
signal s1, ns2: std_logic_vector(Nb-1+4 downto 0);
---------------------------------------------------
begin

s1  <= "0000"&a when a(Nb-1)='0' else
       "1111"&a;
ns2 <= '1'& not(b(Nsat-1 downto 0))+'1' when b(Nsat-1)='0' else
       '0'& not(b(Nsat-1 downto 0))+'1';

y<= s1+ns2;

end Behavioral;
```

8.5 Saturación

Código 8.9 saturacion.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity saturacion is
generic (Nb: integer := 14; Nsat: integer := 17);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1+6 downto 0);
          b : out STD_LOGIC_VECTOR (Nsat-1 downto 0));
end saturacion;

architecture Behavioral of saturacion is
---------------------------------------------------------------------------
signal satpos: std_logic_vector(Nsat-1 downto 0) := (Nsat-1=>'0', OTHERS=>'1');
signal satneg: std_logic_vector(Nsat-1 downto 0) := (Nsat-1=>'1', OTHERS=>'0');
---------------------------------------------------------------------------
begin

b<= satpos when (a(Nb-1+6)='0') AND (a>65535) else
    satneg when (a(Nb-1+6)='1') AND (a<65536) else
    a(Nsat-1 downto 0);

end Behavioral;
```

8.6 Single Bit Quantizer

Código 8.10 quantizerSB.vhd.

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


entity quantizerSB is
generic (Nb: integer := 14);
Port ( a : in STD_LOGIC;
          b : inout STD_LOGIC_VECTOR (Nb-1 downto 0);
          ySD : out STD_LOGIC);
end quantizerSB;

architecture Behavioral of quantizerSB is
---------------------------------------------------------------------------
signal pos: std_logic_vector(Nb-1 downto 0) := (Nb-1=>'0', OTHERS=>'1');
signal neg: std_logic_vector(Nb-1 downto 0) := (Nb-1=>'1', OTHERS=>'0');
---------------------------------------------------------------------------
begin

b <= pos when a='0' else
       neg;
ySD <= '1' when a='0' else
      '0';
```

```
end Behavioral;
```

8.7 Ganancia

Código 8.11 ganancia.vhd.

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


entity ganancia is
generic (Nb: integer := 14);
   Port ( a : in std_logic_vector (Nb-1+4 downto 0);
          b  : out std_logic_vector (Nb-1+5 downto 0));
end ganancia;

architecture Behavioral of ganancia is

begin

b<= a(Nb-1+4 downto 0) & '0';

end Behavioral;
```

8.8 Registros D1 y D2

Código 8.12 D_ff.vhd.

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


entity D_ff is
generic (Nb: integer := 14);
   Port ( clk,rasin,EN : in STD_LOGIC;
          entrada     : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          salida      : out STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end D_ff;

architecture Behavioral of D_ff is
-----------------------------------------------
signal D,Q: std_logic_vector(Nb-1+4 downto 0);
-----------------------------------------------
begin

salida <=Q;

process(rasin,clk)
begin
    if (rasin='1') then
        Q <= (0 => '0',others => '0');
```

```
  elsif (clk'event and clk='1') then
      Q <= D;
  end if;
end process;

D<= entrada when EN='1' else
    Q;

end Behavioral;
```

8.9 Biestable con señales salNRZ y salRZ como salida: D_out

Código 8.13 D_out.vhd.

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


entity D_out is
generic (Nb: integer := 14);
   Port ( clk,rasin : in STD_LOGIC;
          ySD      : in  STD_LOGIC;
          PRZ      : in STD_LOGIC;
          salNRZ   : out  STD_LOGIC;
          salRZ    : out  STD_LOGIC );
end D_out;

architecture Behavioral of D_out is
------------------------
signal Q1,Q2: std_logic;
signal yRZ : std_logic;
------------------------
begin

yRZ <= PRZ AND ySD;

salNRZ <=Q1;
salRZ  <=Q2;

process(rasin,clk)
begin
    if (rasin='1') then
     Q1 <= '0';
     Q2 <= '0';
  elsif (clk'event and clk='1') then
    Q1 <= ySD;
    Q2 <= yRZ;
  end if;
end process;

end Behavioral;
```

8.10 Fichero top

Para poder simular nuestro circuito es necesario generar una entidad que contenga la información que requiera el entorno de trabajo para poder llevar a cabo la simulación. En nuestro caso, dicha entidad recibe el nombre de top3.

Código 8.14 top3.vhd.

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


entity top3 is
   Port (clk,rasin : in STD_LOGIC;
         PRZ       : out std_logic;
         salNRZ    : out std_logic;
         salRZ     : out std_logic);
end top3;

architecture arch_top of top3 is
---------------------------------
constant Nb : integer :=14;
constant Nsat: integer := 17;
constant Ncont: integer:= 6;
constant Ndiv: integer := 40;
constant Nduty: integer := 21;
constant Nmuestras: integer:=192;
---------------------------------
component divisor_frecuencia
generic ( Ncont: integer := Ncont; Ndiv: integer := Ndiv; Nduty: integer:=Nduty
    );
   Port ( clk : in STD_LOGIC;
          rasin: in STD_LOGIC;
          EN   : out STD_LOGIC;
          PRZ  : inout STD_LOGIC);
end component;

component contador
generic ( Nmuestras: integer := Nmuestras; Nb: integer := Nb);
   Port ( clk     : in STD_LOGIC;
          rasin   : in STD_LOGIC;
          EN      : in STD_LOGIC;
          address : inout std_logic_vector (Nb-1 downto 0);
          entSD   : out std_logic_vector (Nb-1 downto 0));
end component;

component sumador
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+5 downto 0));
end component;

component restador1
generic (Nb: integer := Nb);
```

```
    Port ( a : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           b : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           y : out STD_LOGIC_VECTOR (Nb-1+6 downto 0));
end component;

component restador2
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nsat-1 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end component;

component saturacion is
generic (Nb: integer := Nb; Nsat: integer := Nsat);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1+6 downto 0);
          b : out STD_LOGIC_VECTOR (Nsat-1 downto 0));
end component;

component quantizerSB is
generic (Nb: integer := Nb);
Port ( a : in STD_LOGIC;
           b : inout STD_LOGIC_VECTOR (Nb-1 downto 0);
           ySD : out STD_LOGIC);
end component;

component ganancia
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          b  : out STD_LOGIC_VECTOR (Nb-1+5 downto 0));
end component;

component D_ff
generic (Nb: integer :=Nb);
   Port ( clk,rasin,EN : in STD_LOGIC;
          entrada     : in  STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          salida      : out  STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end component;

component D_out is
generic (Nb: integer := Nb);
   Port ( clk,rasin : in STD_LOGIC;
          ySD      : in  STD_LOGIC;
          PRZ      : in STD_LOGIC;
          salNRZ   : out  STD_LOGIC;
          salRZ    : out  STD_LOGIC );
end component;
-------------------------------------------------------------------------
-- Test bench internal signals
-------------------------------------------------------------------------
constant clock_period : time := 1000 ns;

signal EN : std_logic;
signal entSD: std_logic_vector(Nb-1 downto 0);
signal address : std_logic_vector (Nb-1 downto 0);
signal PRZinterno: std_logic;
signal salidag: std_logic_vector (Nb-1+5 downto 0);
signal salidasum: std_logic_vector (Nb-1+5 downto 0);
```

```
signal salidares: std_logic_vector (Nb-1+6 downto 0);
signal salidasat: std_logic_vector (Nsat-1 downto 0);
signal salidaq: std_logic_vector (Nb-1 downto 0);
signal ySD : std_logic;
signal salidares2: STD_LOGIC_VECTOR (Nb-1+4 downto 0);

signal salida1,salida2: std_logic_vector (Nb-1+4 downto 0);
signal sal1, sal2: std_logic_vector (Nb-1+4 downto 0);
----------------------------------------------------------------------
begin

CONT : contador generic map (Nmuestras=>Nmuestras, Nb=>Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, address=>address, entSD=>entSD);
DIV : divisor_frecuencia generic map (Ncont=>Ncont, Ndiv=>Ndiv, Nduty=>Nduty)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, PRZ=>PRZinterno);
G : ganancia generic map (Nb=>Nb)
PORT MAP (a=>salida1, b=>salidag);
Sum: sumador generic map (Nb=>Nb)
PORT MAP (a=>entSD, b=>salida2, y=>salidasum);
Res: restador1 generic map (Nb=>Nb)
PORT MAP (a=>salidasum, b=>salidag, y=>salidares);
Sat: saturacion generic map (Nb=>Nb, Nsat=>Nsat)
PORT MAP (a=>salidares, b=>salidasat);
QSB: quantizerSB generic map (Nb=>Nb)
PORT MAP (a=>salidasat(Nsat-1), b=>salidaq, ySD=>ySD);
Res2: restador2 generic map (Nb=>Nb)
PORT MAP (a=>salidaq, b=>salidasat, y=>salidares2);
D1: D_ff generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, entrada=>salidares2, salida=>salida1)
    ;

sal1<=salida1;
sal2<= salida2;

D2: D_ff generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, entrada=>salida1, salida=>salida2);

PRZ<=PRZinterno;

Dout: D_out generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, ySD=>ySD, PRZ=>PRZinterno, salNRZ=>salNRZ,
    salRZ=>salRZ);

end arch_top;
```

Simulation Sources

8.11 Fichero de estímulos

Tras llevar a cabo la codificación necesaria para diseñar el circuito en VHDL se realiza una simulación de dicho circuito. Para ello, el programa requiere de un fichero adicional dónde se incluyan todos los estímulos conocido como banco de pruebas o test bench en inglés.
En nuestro caso, dicho fichero recibirá el nombre de sim_top3 y mediante su simulación se puede verificar si el circuito se comporta como se esperaba según el estudio ideal realizado en Matlab.

Código 8.15 sim_top3.vhd.

```
library ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_unsigned.all;


entity sim_top3 is
end sim_top3;

architecture tfg of sim_top3 is
---------Parámetros a modificar---------
constant Nb : integer := 14;
constant Nsat : integer := 17;
constant Ncont: integer := 6;
constant Ndiv: integer := 40;
constant Nduty: integer := 21;
constant Nmuestras: integer :=192;
----------------------------------------

COMPONENT top3
   Port (clk,rasin : in STD_LOGIC;
         PRZ       : out std_logic;
         salNRZ    : out std_logic;
         salRZ     : out std_logic);
END COMPONENT;

---------------------------------------
constant clock_period : time := 10 ns;
signal rasin : std_logic :='1';
signal clk : std_logic := '0';
signal PRZ : std_logic;
signal salNRZ, salRZ: std_logic;
---------------------------------------
begin

u_top: top3
PORT MAP (clk=>clk, rasin=>rasin, PRZ=>PRZ, salNRZ=>salNRZ, salRZ=>salRZ);

reloj: process
begin
    clk <= '0';
  wait for clock_period/2;
  clk <= '1';
  wait for clock_period/2;
end process;

stim_proc: process
begin
    wait for clock_period/4;
    rasin <='0';
end process;

end;
```

8.12 Fichero para la extracción de datos

Con el fin de extraer los datos de la señal de entrada (entSD) y salida del SDM (salNRZ o salRZ) se ha de realizar una pequeña aunque significativa modificación al código 8.14 siendo ahora la señal entSD una señal inout del componente top en lugar de una señal interna.

A continuación se expone el código modificado:

Código 8.16 top3_v2.vhd.

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


entity top3_v2 is
generic (Nb: integer := 14);
   Port (clk,rasin : in STD_LOGIC;
         PRZ      : out std_logic;
         entSD    : inout std_logic_vector(Nb-1 downto 0);
         salNRZ   : out std_logic;
         salRZ    : out std_logic);
end top3_v2;

architecture arch_top of top3_v2 is
---------------------------------
constant Nsat: integer := 17;
constant Ncont: integer:= 6;
constant Ndiv: integer := 40;
constant Nduty: integer := 21;
constant Nmuestras: integer:=192;
---------------------------------
component divisor_frecuencia
generic ( Ncont: integer := Ncont; Ndiv: integer := Ndiv; Nduty: integer:=Nduty
    );
   Port ( clk : in STD_LOGIC;
          rasin: in STD_LOGIC;
          EN   : out STD_LOGIC;
          PRZ  : inout STD_LOGIC);
end component;

component contador
generic ( Nmuestras: integer := Nmuestras; Nb: integer := Nb);
   Port ( clk     : in STD_LOGIC;
          rasin   : in STD_LOGIC;
          EN      : in STD_LOGIC;
          address : inout std_logic_vector (Nb-1 downto 0);
          entSD   : out std_logic_vector (Nb-1 downto 0));
end component;

component sumador
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+5 downto 0));
```

```
end component;

component restador1
generic (Nb: integer := Nb);
    Port ( a : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           b : in STD_LOGIC_VECTOR (Nb-1+5 downto 0);
           y : out STD_LOGIC_VECTOR (Nb-1+6 downto 0));
end component;

component restador2
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1 downto 0);
          b : in STD_LOGIC_VECTOR (Nsat-1 downto 0);
          y : out STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end component;

component saturacion is
generic (Nb: integer := Nb; Nsat: integer := Nsat);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1+6 downto 0);
          b : out STD_LOGIC_VECTOR (Nsat-1 downto 0));
end component;

component quantizerSB is
generic (Nb: integer := Nb);
Port ( a : in STD_LOGIC;
           b : inout STD_LOGIC_VECTOR (Nb-1 downto 0);
           ySD : out STD_LOGIC);
end component;

component ganancia
generic (Nb: integer := Nb);
   Port ( a : in STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          b  : out  STD_LOGIC_VECTOR (Nb-1+5 downto 0));
end component;

component D_ff
generic (Nb: integer :=Nb);
   Port ( clk,rasin,EN : in STD_LOGIC;
          entrada     : in  STD_LOGIC_VECTOR (Nb-1+4 downto 0);
          salida      : out  STD_LOGIC_VECTOR (Nb-1+4 downto 0));
end component;

component D_out is
generic (Nb: integer := Nb);
   Port ( clk,rasin : in STD_LOGIC;
          ySD      : in  STD_LOGIC;
          PRZ      : in STD_LOGIC;
          salNRZ   : out  STD_LOGIC;
          salRZ    : out  STD_LOGIC );
end component;
---------------------------------------------------------------------------
-- Test bench internal signals
---------------------------------------------------------------------------
constant clock_period : time := 1000 ns;

signal EN : std_logic;
```

```
signal address : std_logic_vector (Nb-1 downto 0);
signal PRZinterno: std_logic;
signal salidag: std_logic_vector (Nb-1+5 downto 0);
signal salidasum: std_logic_vector (Nb-1+5 downto 0);
signal salidares: std_logic_vector (Nb-1+6 downto 0);
signal salidasat: std_logic_vector (Nsat-1 downto 0);
signal salidaq: std_logic_vector (Nb-1 downto 0);
signal ySD : std_logic;
signal salidares2: STD_LOGIC_VECTOR (Nb-1+4 downto 0);

signal salida1,salida2: std_logic_vector (Nb-1+4 downto 0);
signal sal1, sal2: std_logic_vector (Nb-1+4 downto 0);
----------------------------------------------------------------------
begin

CONT : contador generic map (Nmuestras=>Nmuestras, Nb=>Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, address=>address, entSD=>entSD);
DIV : divisor_frecuencia generic map (Ncont=>Ncont, Ndiv=>Ndiv, Nduty=>Nduty)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, PRZ=>PRZinterno);
G : ganancia generic map (Nb=>Nb)
PORT MAP (a=>salida1, b=>salidag);
Sum: sumador generic map (Nb=>Nb)
PORT MAP (a=>entSD, b=>salida2, y=>salidasum);
Res: restador1 generic map (Nb=>Nb)
PORT MAP (a=>salidasum, b=>salidag, y=>salidares);
Sat: saturacion generic map (Nb=>Nb, Nsat=>Nsat)
PORT MAP (a=>salidares, b=>salidasat);
QSB: quantizerSB generic map (Nb=>Nb)
PORT MAP (a=>salidasat(Nsat-1), b=>salidaq, ySD=>ySD);
Res2: restador2 generic map (Nb=>Nb)
PORT MAP (a=>salidaq, b=>salidasat, y=>salidares2);
D1: D_ff generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, entrada=>salidares2, salida=>salida1)
    ;

sal1<=salida1;
sal2<= salida2;

D2: D_ff generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, EN=>EN, entrada=>salida1, salida=>salida2);

PRZ<=PRZinterno;

Dout: D_out generic map (Nb=> Nb)
PORT MAP (clk=>clk, rasin=>rasin, ySD=>ySD, PRZ=>PRZinterno, salNRZ=>salNRZ,
    salRZ=>salRZ);

end arch_top;
```

Finalmente, para extraer datos de Vivado usando el código anterior y el que precede, hay que seleccionar ambos en el Software y haciendo click en el botón derecho seleccionar *Set as Top* lo cuál provoca que cuando se le pida al programa que realice una simulación, lo haga en base a los ficheros seleccionados.

Por otro lado, el código 8.17 tiene varios puntos a destacar. En primer lugar, hace uso de la función *conv_integer* incluida en Vivado y cuyo cometido es la conversión de un vector binario puro en un número entero. En nuestro caso, cuando se requiera representar dichos valores, habrá que hacer una conversión adicional para pasar dichos números enteros en CA2 a sus correspondientes valores en decimal con signo.

En segundo lugar, se han de definir dos nuevas señales denominadas r_y0 y r_y1 que se asocian con la señal binaria de salida del SDM y con la de entrada respectivamente.

A continuación, se invoca al componente top3_v2 en UUT para comenzar con la escritura de datos de salida que se realiza en el proceso. De este modo, para un valor de Nvhdl=2^{14} en nuestro caso, se escriben los datos de entrada y salida en un fichero que se guardará con el nombre y en la rutina indicada, "$C:\backslash US\backslash TFG\backslash salida\_RON.txt$", ambos se pueden modificar al gusto del usuario. La escritura de valores se realiza cada vez que haya un flanco de subida de la señal EN gracias a que se impone que se espere un tiempo igual al período de dicha señal entre cada secuencia del bucle. De este modo, todas las filas del fichero serán 2 columnas de datos con un espacio entre ellas. Tras escribir los Nvhdl número de valores, se cierra el fichero y finaliza el proceso de escritura de datos.

Código 8.17 TB_top_SDM.vhd.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;
use ieee.std_logic_unsigned.all;


entity TB_top_SDM is
end TB_top_SDM;

architecture behavior OF TB_top_SDM is
---------Parámetros a modificar---------
constant Nb : natural := 14;
constant Nsat : integer := 17;
constant Ncont: integer := 6;
constant Ndiv: integer := 40;
constant Nduty: integer := 21;
constant Nmuestras: integer :=192;
constant Nvhdl: integer := 16384;
----------------------------------------

COMPONENT top3_v2
generic (Nb: integer := 14);
   Port (clk,rasin : in STD_LOGIC;
         PRZ      : out std_logic;
         entSD    : inout std_logic_vector(Nb-1 downto 0);
         salNRZ   : out std_logic;
         salRZ    : out std_logic);
end COMPONENT;

--------------------------------------------------------------------------------
-- Testbench Internal Signals
--------------------------------------------------------------------------------
file file_RESULTS : text;

constant clock_period : time := 10 ns;

signal rasin : std_logic := '1';
signal clk  : std_logic := '0';
signal salNRZ: std_logic;
signal salRZ : std_logic;
```

```
signal entSD : std_logic_vector (Nb-1 downto 0);

signal r_y0: std_logic;
signal r_y1:integer range 0 to (2**Nb-1);

begin
uut: top3_v2
PORT MAP (clk=>clk, rasin=>rasin, salNRZ=>salNRZ, salRZ=>salRZ, entSD=>entSD);

reloj: process
begin
    clk <= '0';
  wait for clock_period/2;
  clk <= '1';
  wait for clock_period/2;
end process;

stim_proc: process
begin
    wait for clock_period/4;
    rasin <='0';
    end process;

--------------------------------------------------------------------------------
-- Escritura de datos de salida
--------------------------------------------------------------------------------
process

variable v_OLINE   : line;

begin
file_open(file_RESULTS, "C:\US\TFG\salida_RON.txt", write_mode);

for k in 0 to Nvhdl loop --cambiar 10 a 2^14
    r_y0<=(salNRZ);
    r_y1<=conv_integer(entSD);
    write(v_OLINE, r_y0);           % escritura datos de la señal salNRZ
    write(v_OLINE, string'(" "));       % se deja un espacio entre datos
    write(v_OLINE, r_y1);           % escritura datos de la señal entSD
    writeline(file_RESULTS, v_OLINE);

    wait for clock_period*Ndiv;
end loop;

file_close(file_RESULTS);
wait;
end process;

end;
```

Usando el código 8.17 se logra generar un fichero de datos de salida, en nuestro caso denominado salida_-RON, tal que en la primera columna se encuentran los valores de la señal salNRZ y en la segunda las de entSD. El objetivo de este código es poder extraer un archivo de texto con los valores de los distintos puntos de unas señales deseadas para así poder analizar sus prestaciones empleando una rutina de Matlab que analiza el valor de SNR y las representa.

8.13 Fichero para la definición de constraints

Código 8.18 pines.xdc.

```
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports rasin]
set_property IOSTANDARD LVCMOS18 [get_ports salNRZ]
set_property IOSTANDARD LVCMOS18 [get_ports salRZ]
set_property IOSTANDARD LVCMOS18 [get_ports PRZ]
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property PACKAGE_PIN R18 [get_ports rasin]
set_property PACKAGE_PIN AB9 [get_ports salNRZ]
set_property PACKAGE_PIN V8 [get_ports salRZ]
set_property PACKAGE_PIN W11 [get_ports PRZ]
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] AVNET, *Zedboard. (zynq evaluation and development) hardware user's guide*, 2.2 ed., Enero 2014.

[2] María José Madero Ayora, *Señales y sitemas de radiofrecuencia*, Universidad de Sevilla, 2018.

[3] Miguel Angel Freire Rubio, *Introducción al lenguaje vhdl*, Universidad Politécnica de Madrid Departamento de Sistemas Electrónicos y de Control, 2010.

[4] Francisco Colodro Ruiz, *Fundamentos de la conversión A/D y D/A*, Universidad de Sevilla.

[5] Richard Schreier and Gabor C.Temes, *Understanding delta-sigma data converters*, IEEE press, 2005.

[6] Marcos Sánchez-Élez, *Introducción a la programación en vhdl*, Facultad de Informática Universidad Complutense de Madrid, 2014.