NCO senoidal implementado con Procesador y Lógica Programable

Guillermo F. Caporaletti

gcaporaletti@fi.uba.ar

Agosto de 2023

Docente: Nicolás Álvarez

Carrera de Especialización en Sistemas Embebidos
Curso de Microarquitectura y Softcores

Índice

1. Introducción

El presente trabajo fue realizado en el curso de Microarquitectura y Softcores de la Carrera de Especialización en Sistemas Embebidos, Facultad de Ingeniería, UBA. Consiste en la implementación de un oscilador senoidal controlado numéricamente (NCO, por *numerically controlled oscillator*) en un sistema embedido con un procesador (PS, por *Processor System*) y un FPGA (o PL, por *Programmable Logic*).

La ventaja de esta implementación mixta es aprovechar la alta velocidad de la Lógica Programable (FPGA o PL) y, a la vez, la flexibilidad de programar un procesador. En este caso en particular, con una FPGA habíamos logrado un generador de señal senoidal de 12 bits, con un rango de frecuencias entre 2 MHz y 0,25 Hz. La documentación del proyecto inicial en FPGA descripto en VHDL puede consultarse en Caporaletti (2023) [3].

En el presente documento, le agregamos un procesador que controla dicho generador. Esto permite, para comenzar, que los cálculos para configurar adecuadamente el NCO sean realizados directamente por el procesador, simplificando su utilización. Además, puede facilitar su conexión con otros sistemas a través de UART y el puerto que se desee.

1.1. Descripción general

Un NCO senoidal implementa una señal senoidal a partir de una base de tiempo y componentes digitales. En la Figura 1 se aprecian las partes componentes del sistema. El Convertidor Fase-Amplitud (CFA) es una tabla de búsqueda que almacena en memoria los valores de cada muestra de la señal.

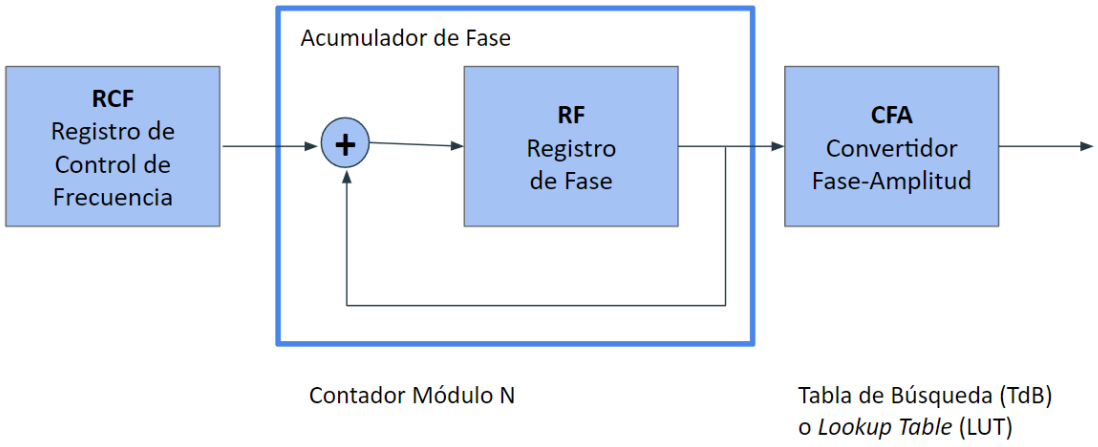

Figura 1. *Diagrama básico de un NCO.*

Si la discretización en el tiempo es de N=10 bits por período, la cantidad de muestras será:

Discretización en el tiempo: $2^{10}\ muestras = 1024\ muestras$ (1)

En principio, esto implicará 1024 direcciones de memoria (aunque podrían ser menos aprovechando las simetrías de la señal). Esta discretización en el tiempo de 10 bits sería razonable para un DAC con 10 bits de discretización en la amplitud. En nuestro proyecto hemos utilizado una discretización de amplitud de 12 bits.

En la Figura 2 se muestra la implementación del NCO implementada en FPGA. La Tabla de Búsqueda (TdB) posee 10 bits (es decir, 2^{10} muestras). Nótese que el contador posee N+R = 30 bits: esto incluye no sólo los N bits que como mínimo debe poseer el contador para recorrer todos los valores de la TdB, sino que además tiene una parte fraccional R que permite enlentencer la frecuencia de salida. Esta relación de los bits de los contadores puede apreciarse en la Figura 3.

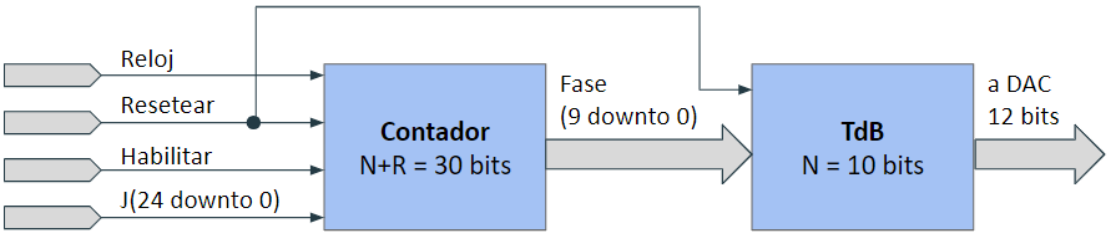

Figura 2. *Diagrama en bloques del oscilador a implementar.*

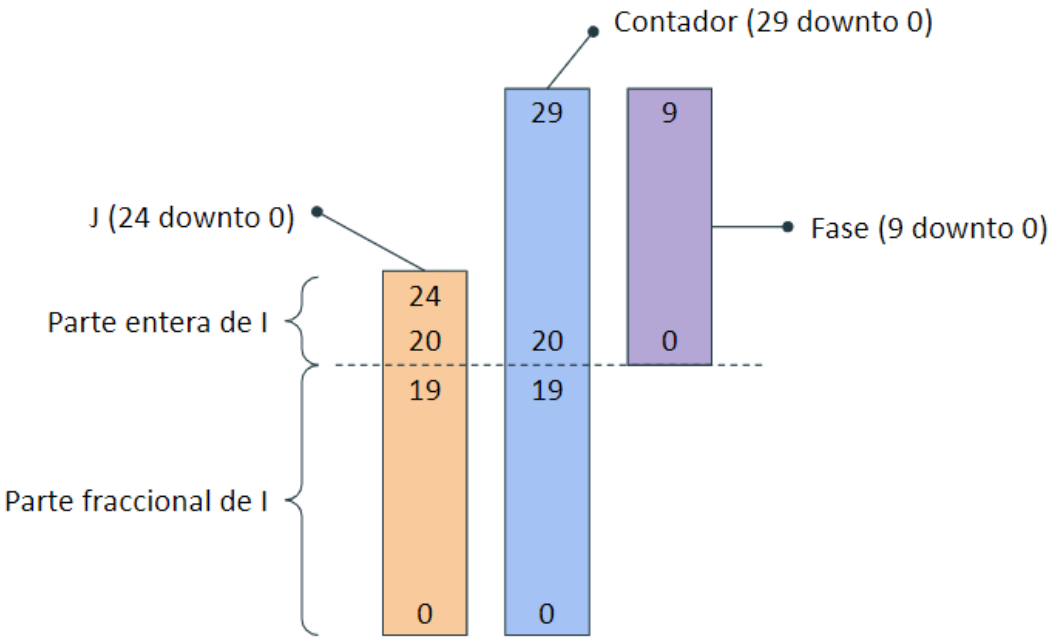

Figura 3. *Correspondencia de los bits del incremento I, el contador y la fase.*

El Contador de 30 bits (marcado en azul y equivalente a N+R) puede interpretarse como que posee una parte entera y otra fraccional. Si vamos incrementando de a uno el bit 20 del Contador, la Fase para ingresar a la TdB se irá incrementando de uno en uno. Sin embargo, si decidimos incrementar el bit 0, el Contador deberá ir incrementando todos los bits desde 0 a 19 para lograr incrementar una cuenta del bit 20.

Para lograr la frecuencia deseada, el incremento de cada cuenta deberá configurarse en el vector J. Lo que hará el procesador es traducir qué frecuencia corresponde a qué vector J para poder así configurar adecuadamente el NCO.

1.2. Ecuaciones básicas

Partiendo de la Figura 1, puntualizamos las siguientes ecuaciones:

N	Bits del contador
2^N	Cantidad de muestras por período
$T_{Reloj} = \frac{1}{F_{Reloj}}$	Período del reloj o base de tiempo del sistema [seg]
$T_0 = \frac{T_{Reloj} * 2^N}{I}$	Período de la señal de salida [seg]
$F_0 = \frac{F_{Reloj} * I}{2^N}$	Frecuencia de la señal de salida [Hz]

Según las ecuaciones (2), la frecuencia de salida es la frecuencia del reloj del sistema dividido la cantidad de muestras y multiplicada por la parte entera I (que corresponde a los bits 20 a 24 de J). Partiendo de la Figura 3, otra forma de escribir la frecuencia es

$$F_0 = \frac{F_{Reloj} * Incremento}{2^N * 2^R} = \frac{F_{Reloj} * Incremento}{2^{N+R}} \quad (3)$$

Donde Incremento es el número formado por los J bits (24 en nuestra implementación). N es 10 y R representaría la parte fraccional aunque no es otra cosa que 20 bits. Es decir: N+R = 30 bits. De este modo, con $F_{Reloj} = 100\ MHz$, resulta:

$$F_{0_{Mínima}} = \frac{F_{Reloj} * 1}{2^{30}} = 0{,}093\ Hz \quad (4)$$

$$F_{0_{Máxima}} = \frac{F_{Reloj} * 33.554.431}{2^{30}} = 3{,}13\ MHz \quad (5)$$

La frecuencia máxima corresponde a un número binario de 25 unos, que en hexadecimal se puede escribir como 0x1FFFFFF. Este valor corresponde a *Incremento* = 33.554.431 en decimal. En Caporaletti (2023) se ha argumentado que esta $F_{0_{Mínima}}$ corresponde a la resolución en frecuencia. Y que, por otro lado, lo razonable es no superar los 2 MHz porque la señal se degradaría demasiado. En cualquier caso, se evidencia que la configuración del incremento J no es trivial y para eso utilizaremos nuestro procesador.

1.3. Obtención del incremento correspondiente a una frecuencia

Si necesitamos obtener el valor del incremento J debemos despejar *Incremento* de la ecuación (3). Esto sería:

$$F_0 * \frac{2^{N+R}}{F_{Reloj}} = Incremento \quad (6)$$

Tomando la ecuación (4) podemos reescribirlo como:

$$\frac{F_0}{F_{0_{Mínima}}} = Incremento \quad (7)$$

A la hora de implementarlo debemos tener especial cuidado en cómo hacemos el cálculo si utilizamos tipos enteros. O utilizar directamente números con punto flotante y luego convertir el resultado final a entero sin signo. Si por ejemplo buscamos una frecuencia de 10 kHz, debemos hacer:

$$I = \frac{10\ \text{kHz}}{0{,}093\ Hz} = 107.374 \quad (8)$$

Este valor, expresado en binario, equivale a *Incremento* = 0b11010001110001110. Si recalculamos el valor de frecuencia para este número binario entero, la misma resulta $F_0 = 9999{,}983\ \mathrm{Hz}$. Esta aproximación es más precisa que la estabilidad típica de un cristal de cuarzo [6].

2. Implementación

2.1. Descripción inicial de FPGA

El punto de partida ha sido el desarrollo de NCO senoidal desarrollado en [3]. Allí se desarrolló y probó satisfactoriamente un acumulador de fase (acumulador.vhd), un conversor de fase en amplitud (cfa_seno.vhd) y una entidad del generador de señal que integraba todas las partes (seno.vhd). Los descriptores se pueden consultar en [4]. En el archivo seno.vhd, la entidad `seno` (que integra las demás entidades) está definida como:

```
entity seno is
    generic (
        BITS_DATO        : positive := 12;  -- Bits del valor-dato de salida
        BITS_MUESTRAS    : positive := 10;  -- Bits necesario para las N muestras
        BITS_INCREMENTO  : positive := 25;  -- Bits del incremento J
        BITS_CONTADOR    : positive := 30   -- Bits del contador del AF
    );
    port (
        s_reloj_i        : in  std_logic;
        s_reset_i        : in  std_logic;
        s_habilitar_i    : in  std_logic;
        s_incremento_i   : in  std_logic_vector(BITS_INCREMENTO - 1 downto 0);
        s_seno_o         : out std_logic_vector(BITS_DATO - 1 downto 0)
    );
end entity seno;
```

De los parámetros se desprende que nuestra señal senoidal posee 12 bits para cuantización de amplitud (4096 niveles), 10 bits para las muestras (1024 muestras), 25 bits para el incremento que ajusta la frecuencia y 30 bits para el contador del acumulador de fase. Por otra parte, el puerto de la entidad posee una única salida: la señal senoidal; y cuatro entradas: la fase (s_incremento_i), el reloj, una señal de reset y otra para habilitar. Estas señales, con estos parámetros, es lo que necesito incorporar en un núcleo de propiedad intelectual (IP core). En la Figura 4 puede apreciarse el esquemático del NCO.

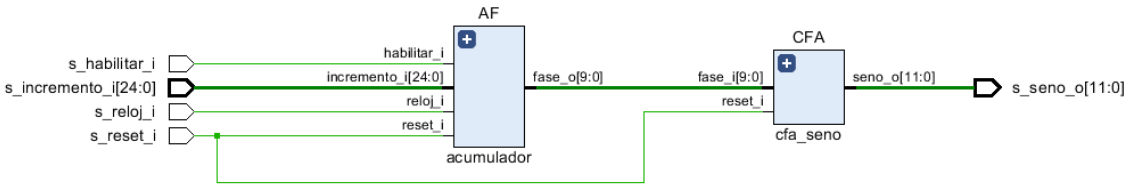

Figura 4. *Diagrama esquemático del NCO senoidal.*

2.2. Implementación de núcleo IP con interfaz AXI

Se había mencionado previamente que lo que tiene de engorroso el NCO senoidal implementado en lógica programable es la configuración del contador J que determina la frecuencia. Ésta es la parte que preferimos hacer desde un procesador. Para lograr esto, lo primero será implementar un núcleo IP con una interfaz AXI (por *Advanced eXtensible Interface*) para comunicarse con el procesador. Esto lo hacemos con las herramientas del software Vivado.

Vivado posee una herramienta específica de creación y administración de núcleos IP. Dentro de esa herramienta de "Administración de IP" (Manage IP) elegimos crear un periférico AXI. Así se crearon dos archivos `.vhd` con sus respectivas entidades dentro: `IP_seno_v1_0` y `IP_seno_v1_0_S_AXI`. La entidad `IP_seno_v1_0_S_AXI` es la que implementa la lógica para interpretar la información transmitida por el bus AXI; a esta entidad se incorporó la entidad `seno`. La entidad `IP_seno_v1_0` posee las conexiones al puerto AXI y es desde la cuál deben agregarse puertos que se puedan conectar a otroa bloques externos al desarrollo del núcleo IP en sí. Estos dos archivos debieron ser modificados para lograr la integración.

Primero se modificó la entidad `IP_seno_v1_0_S_AXI`. Dentro de la definición de su arquitectura, se incorporaron las siguientes conexiones:

```
-- Add user logic here
    inst_seno: seno
            port map(
                s_reloj_i       => S_AXI_ACLK,
                                   -- Conecto reloj
                s_reset_i       => slv_reg0(C_S_BIT_RESET),
                                   -- Bit reset es un ndice del vector registro
                s_habilitar_i   => slv_reg0(C_S_BIT_HABILITAR),
                                   -- Bit habilitar es un ndice del vector registro
                s_incremento_i  => slv_reg0(C_S_BITS_INCREMENTO-1 downto 0),
                s_seno_o        => salida_o
            );
    -- User logic ends
```

Por un lado, se conecta el reloj de la entidad `seno` al reloj de la interfaz AXI. Luego, se utilizó el registro `slv_reg0` (de 32 bits) como entradas a la entidad `seno`: 1 bit para reset, 1 bit para habilitar y 25 bits para el contador J que determina la frecuencia. Además, agregamos en la interfaz AXI una conexión `salida_o` que permita exteriorizar la señal del NCO senoidal.

Para esta conexión definimos previamente la señal y algunas constantes:

```
    signal salida_o             : std_logic_vector(C_S_BITS_DATO - 1 downto 0);
    constant C_S_BIT_RESET      : positive := C_S_AXI_DATA_WIDTH - 1;
    constant C_S_BIT_HABILITAR  : positive := C_S_AXI_DATA_WIDTH - 2;
```

Los bits de reset y habilitar son establecidos como los más significativos del dato recibido. Por otra parte, la señal `salida_o` a su vez se conecta con la conexión S_SALIDA que se pasó a definir en el puerto de la entidad `IP_seno_v1_0_S_AXI`. Conexión (línea 148):

```
    S_SALIDA <= salida_o;
```

Esta conexión del puerto y los parámetros quedaron definidos en la entidad:

```
entity IP_seno_v1_0_S_AXI is
    generic (
        -- Users to add parameters here
        C_S_BITS_DATO       : positive := 12;  -- Bits del valor-dato de salida
        C_S_BITS_INCREMENTO : positive := 25;  -- Bits del incremento J
        -- User parameters ends
        -- Do not modify the parameters beyond this line

        -- Width of S_AXI data bus
        C_S_AXI_DATA_WIDTH  : integer   := 32;
        -- Width of S_AXI address bus
        C_S_AXI_ADDR_WIDTH  : integer   := 4
    );
    port (
```

```
        -- Users to add ports here
        S_SALIDA            : out std_logic_vector(C_S_BITS_DATO-1 downto 0);
                              -- Salida de la muestra de la senial.
        -- User ports ends
        -- Do not modify the ports beyond this line
```

También podríamos haber conectado la salida de seno con la conexión S_SALIDA del puerto de IP_seno_v1_0_S_AXI. Pero optamos por crear una nueva señal para ir paso a paso en la modificación del código. Al inicio están definidos los parámetros de la cantidad de bits del contador J y de la salida del NCO.

La otra parte fue conectar la salida del NCO senoidal a una nueva conexión que se debió crear en la entidad IP_seno_v1_0. Esto se aprecia en la definición de la entidad:

```
entity IP_seno_v1_0 is
    generic (
        -- Users to add parameters here
        C_S_BITS_DATO       : positive := 12;  -- Bits del valor-dato de salida
        -- User parameters ends
        -- Do not modify the parameters beyond this line


        -- Parameters of Axi Slave Bus Interface S_AXI
        C_S_AXI_DATA_WIDTH  : integer    := 32;
        C_S_AXI_ADDR_WIDTH  : integer    := 4
    );
    port (
        -- Users to add ports here
        s_seno              : out std_logic_vector(C_S_BITS_DATO-1 downto 0);
                              -- Muestra de salida de la senial seno.
        -- User ports ends
        -- Do not modify the ports beyond this line
```

Primero debimos incorporar el parámetro que define los 12 bits de la salida del NCO senoidal. Luego agregamos la conexión s_seno en el puerto. En la arquitectura de la entidad IP_seno_v1_0, se mapeó esta conexión s_seno con la que viene de la entidad IP_seno_v1_0_S_AXI. De este modo se logró que la señal senoidal pase del NCO a un puerto de salida de la entidad superior IP_seno_v1_0 y quede disponible para su uso.

2.3. Proyecto con ILA para evaluación

Finalmente, en Vivado iniciamos un proyecto que incluya nuestro recientemente creado núcleo IP. El esquema quedó de la siguiente forma:

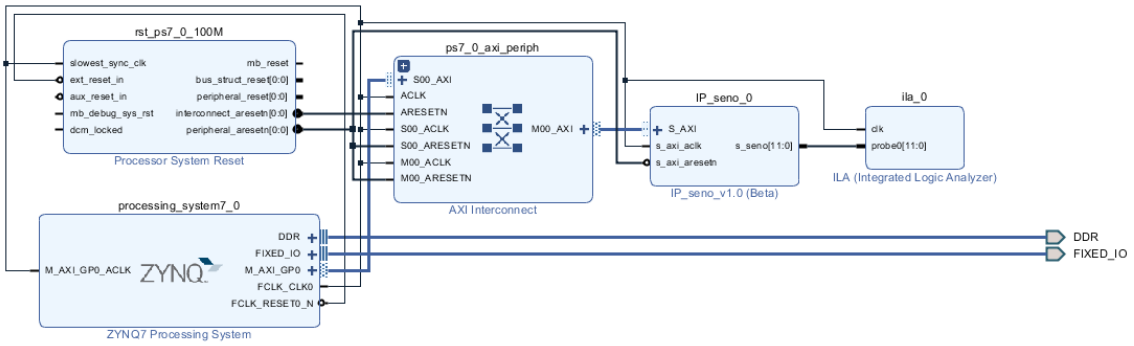

Figura 5. *Diagrama esquemático del proyecto de evaluación.*

Como puede apreciarse, la entidad IP_seno_0 posee una salida de 12 bits, que es la señal senoidal. Hemos conectado a la salida un ILA para poder capturar esta señal. Un ejemplo se muestra en la Figura 6.

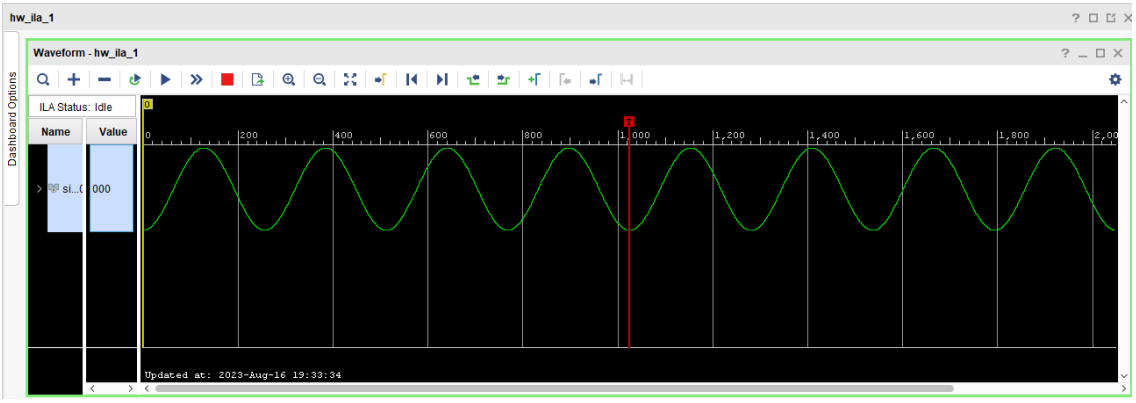

Figura 6. *Captura de la señal senoidal.*

Para lograr la señal, programamos el procesador. Para esto utilizamos el entorno SDK. El código utilizado:

```
/*****************************************************************************
 * Archivo:  generador_seno.c
 * Breve:    Control de CLP generador de senial senoidal mediante codigo en procesador.
 *****************************************************************************/

/*** Librerias ***************************************************************/

#include "xparameters.h"
#include "xil_io.h"
#include "IP_seno.h"

/*** Macros ******************************************************************/

#define RESETEAR         0x80000000   // Bit para resetear el contador
#define HABILITAR        0x40000000   // Bit para habilitar el contador
#define MASCARA_CONTADOR 0x1FFFFFF    // Mascara que corresponde al contador
#define MAXIMA_CUENTA    MASCARA_CONTADOR
#define CUENTA_ENTERA    0x100000     // F0 = Freloj / 1024
                                      // Es la maxima frecuencia sin degradar senial
#define MEGA             1000000
#define FREC_RELOJ       100 * MEGA   // Frecuencia de reloj

/*** Funciones privadas ******************************************************/

int main (void) {

    int cuenta = HABILITAR | CUENTA_ENTERA;

    xil_printf("-- Inicio del generador de senial senoidal --\r\n");

    IP_SENO_mWriteReg(XPAR_IP_SENO_0_S_AXI_BASEADDR,
                      IP_SENO_S_AXI_SLV_REG0_OFFSET,
                      cuenta);

    xil_printf("Cuenta: %d \r\n", cuenta & MASCARA_CONTADOR);

}
```

3. Conclusiones

A lo largo del presente trabajo hemos presentado el desarrollo de un NCO implementado con una FPGA y un procesador. La combinación de ambos núcleos nos brinda velocidad de procesamiento y versatilidad de aplicaciones. El uso de la FPGA queda así entonces como una herramienta fundamental para aplicaciones complejas.

4. Documentos y referencias

[1] Álvarez, Nicolás. "NCO (Numerically Controlled Oscillator)" (presentación), Sistemas Digitales 66.17, Facultad de Ingeniería, UBA.

[2] Álvarez, Nicolás (2018). "Circuitos lógicos programables: Ciclo de desarrollo con Vivado (Práctica 1)", curso de Circuitos Lógicos Programables, CESE, Facultad de Ingeniería, UBA.

[3] Caporaletti, Guillermo F. (2023). "Oscilador senoidal controlado numéricamente (NCO senoidal)" (Trabajo Final), curso de Circuitos Lógicos Programables, CESE, Facultad de Ingeniería, UBA. Consultar en: https://github.com/gcapora/CESE_CLP_TP/blob/main/Documentos/Caporaletti.%20NCO%20senoidal%2C%20Trabajo%20Final%20CLP%2C%20Co18.pdf

[4] Repositorio del proyecto NCO senoidal: https://github.com/gcapora/CESE_CLP_TP.

[5] Repositorio del proyecto NCO senoidal implementado con PL y LP: https://github.com/gcapora/CESE_MyS_TF.

[6] Wikipedia, "Oscilador de cristal". Consultado en: https://es.wikipedia.org/wiki/Oscilador_de_cristal (junio de 2023).

5. Definiciones, acrónimos y abreviaturas

1. ADC Conversor Analógico-Digital
2. CESE Carrera de Especialización en Sistemas Embebidos
3. DAC Conversor Digital-Analógico
4. FIUBA Facultad de Ingeniería de la Universidad de Buenos Aires
5. FPGA Matriz de compuertas lógicas programables (*Field Programable Gate Array*)
6. LUT *Look-up table*, equivalente a TdB
7. N/A No aplica
8. ppm Parte por millón
9. TdB Tabla de búsqueda, equivalente a LUT