

# Trabajo Práctico Final Microarquitecturas y Softcores

Abraham Rodriguez

Diciembre, 2024

# Índice

Funcionalidad e Implementación

Acumulador

Diagrama de Bloques

Simulaciones

Tabla de Recursos

# Funcionalidad e Implementación

En esta sección se proporciona una breve explicación del NCO implementado, así como diagramas y bloques de código relevantes.

# Repositorios

El trabajo realizado en CLP (NCO) se encuentra en GitHub:

https:

`//github.com/AbeRodz/Numeric-Controlled-Oscillator`

La integracion con ZYNQ se encuentra en GitHub:

`https://github.com/AbeRodz/Zynq-NCO`

# Numeric Controlled Oscillator (NCO)

Un NCO consiste de un acumulador de fase y convertidor de fase de amplitud. Diagrama interno:

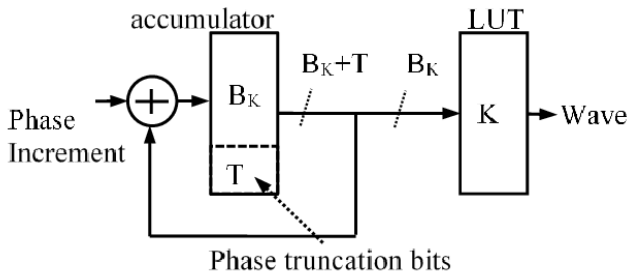


Figura: Diagrama interno NCO.

# Tipos de Onda

Tipos de onda generados:

- ▶ Seno
- ▶ Cuadrada
- ▶ Triangular
- ▶ Sierra

# Acumulador

El acumulador aplica la operación de suma de la fase o *Frequency Tuning/Control Word* actual con el estado anterior. Código VHDL simplificado:

```
process (c_i)
begin
    if rising_edge(c_i) then
        if en_i = '1' then
            phase_acc <= phase_acc + unsigned(freq_word_i);
            addr <= std_logic_vector(phase_acc(31 downto 22));
        end if;
    end if;
end process;
```

# SineLUT

Para generar los valores de la función seno, se realizó un script de Python:

```
def generate_uint16_sine_table(
    num_samples: int = 1024,
    amplitude: int = 32767,
    offset: int = 32767) -> tuple[np.ndarray]:
    num_values = num_samples
    x = np.linspace(0, 2 * np.pi, num_values, endpoint=False)
    y = np.round(amplitude * np.sin(x) + offset).astype(int)
    return x,y
```



# Diagrama de Muestra de Seno

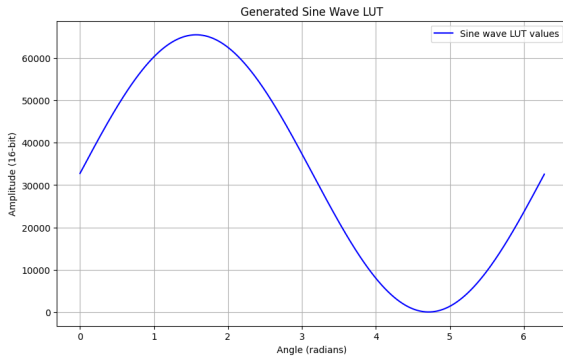
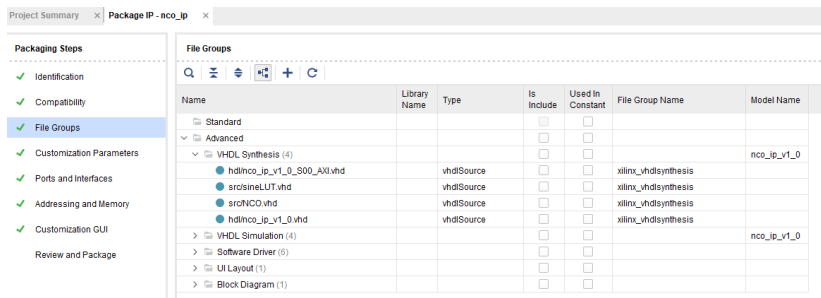


Figura: Muestra de la función seno.

# Integración con microprocesador

Una vez creado el NCO, se empaqueta como un IP Core mediante el package manager.



Name	Library Name	Type	Is Include	Used In Constant	File Group Name	Model Name
Standard			<input type="checkbox"/>	<input type="checkbox"/>		
Advanced			<input type="checkbox"/>	<input type="checkbox"/>		
VHDL Synthesis (4)			<input type="checkbox"/>	<input type="checkbox"/>		nco_ip_v1_0
hdl/nco_ip_v1_0_S00_AXI.vhd		vhdlSource	<input type="checkbox"/>	<input type="checkbox"/>	xilinx_vhdlsynthesis	
src/sineLUT.vhd		vhdlSource	<input type="checkbox"/>	<input type="checkbox"/>	xilinx_vhdlsynthesis	
src/NCO.vhd		vhdlSource	<input type="checkbox"/>	<input type="checkbox"/>	xilinx_vhdlsynthesis	
hdl/nco_ip_v1_0.vhd		vhdlSource	<input type="checkbox"/>	<input type="checkbox"/>	xilinx_vhdlsynthesis	
VHDL Simulation (4)			<input type="checkbox"/>	<input type="checkbox"/>		nco_ip_v1_0
Software Driver (5)			<input type="checkbox"/>	<input type="checkbox"/>		
UI Layout (1)			<input type="checkbox"/>	<input type="checkbox"/>		
Block Diagram (1)			<input type="checkbox"/>	<input type="checkbox"/>		

Figura: NCO IP Package.

Crear el IP Core de la NCO implica exponer el output de la NCO hacia el exterior para poder ser analizado mediante una ILA. Para eso se agrego al port la variable de salida.

# Exposicion de output

Crear el IP Core de la NCO implica exponer el output de la NCO hacia el exterior para poder ser analizado mediante una ILA. Para eso se agrego al port la variable de salida.

```
entity nco_ip_v1_0 is
  generic (
    C_S00_AXI_DATA_WIDTH: integer:= 32;
    C_S00_AXI_ADDR_WIDTH: integer:= 4
  );
  port (
    wave_output : out std_logic_vector(C_S00_AXI_DATA_WIDTH-17 downto 0); -- Output wave from NCO
```

Para luego ser propagado hacia la instancia de la NCO.

```
inst_NCO: entity work.NCO
  port map (
    c_i      => S_AXI_ACLK,
    en_i     => slv_reg3(0),
    wave_type_i => wave_type_i,
    freq_word_i => slv_reg0,
    wave_o   => wave_output
  );
```

# Resultado de IP Package

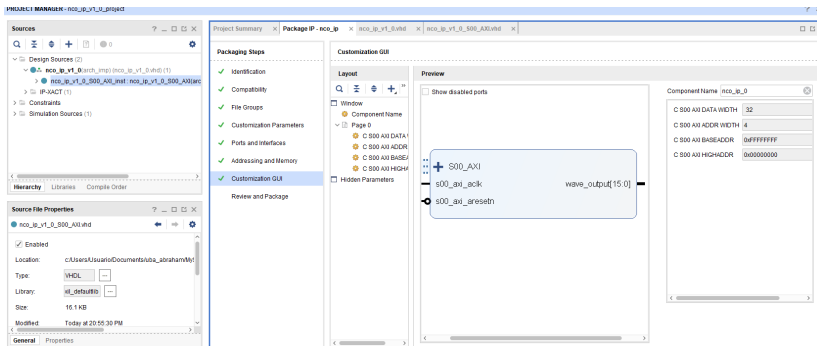


Figura: Simulaciones.

## C SDK prototipos

Una vez creado el paquete de la NCO, se realizo el siguiente código C mediante la SDK:

```
// NCO configuration and state
typedef struct {
    int32_t base_addr;           // Base address of the NCO
    int32_t frequency;           // Current frequency tuning word
    int32_t wave_type;           // Current wave type
    int32_t enable;              // Enable status (0 or 1)
    int32_t freq_step;           // Step size for frequency adjustment
    int32_t min_freq;            // Minimum frequency tuning word
    int32_t max_freq;            // Maximum frequency tuning word
} NCO_Config;

// Function to initialize NCO settings
void NCO_Init(NCO_Config *nco, int32_t base_addr, int32_t init_freq, int32_t freq_step, int32_t min_freq,

// Function to write frequency tuning word
void NCO_SetFrequency(NCO_Config *nco, int32_t frequency_tuning_word;

// Function to set wave type
void NCO_SetWaveType(NCO_Config *nco, int32_t wave_type);

// Function to enable/disable NCO
void NCO_SetEnable(NCO_Config *nco, int32_t enable);

// Function to dynamically adjust frequency
void NCO_AdjustFrequency(NCO_Config *nco);

// Function to read wave output
int16_t NCO_ReadWaveOutput(NCO_Config *nco);
```

# C SDK main function

Las pruebas mediante C consiste en actualizar dinamicamente la frecuencia de la NCO en un rango establecido.

```
int main(void) {
    int i;
    xil_printf("-- Inicio de NCO IP Core --\r\n");
    NCO_Config nco;
    NCO_Init(&nco, NCO_BASE_ADDR, 0x00100000, 0x00010000, 0x00010000, 0xFFFFFFFF);

    NCO_SetFrequency(&nco, nco.frequency);
    NCO_SetWaveType(&nco, nco.wave_type);
    NCO_SetEnable(&nco, nco.enable);

    while(1) {
        NCO_AdjustFrequency(&nco);

        // Read and display wave output
        int16_t wave_output = NCO_ReadWaveOutput(&nco);
        xil_printf("Wave output: %d\r\n", wave_output);
        for (i=0; i<9999999;i++);
    }
    NCO_SetEnable(&nco, 0x00);
    xil_printf("-- Fin de NCO IP Core --\r\n");
    return 0;
}
```

# Diagrama de Bloques

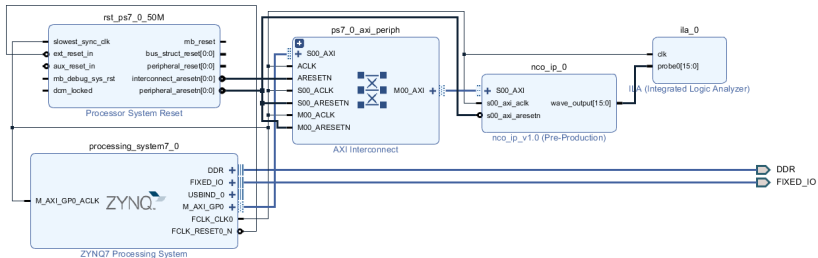


Figura: Diagrama de bloques.

# Simulaciones NCO

Se simularon distintas formas de onda:

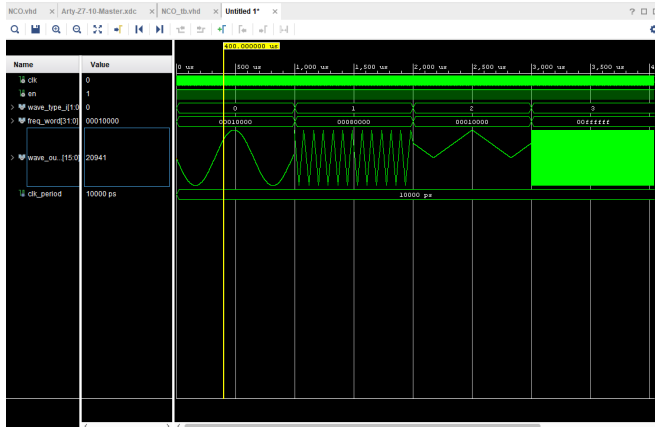


Figura: Simulaciones.



# Pruebas C SDK I

Se genero el bitstream y se cargo a la FPGA, posteriormente se complico el proyecto y se subio al microprocesador para realizar las siguientes pruebas:

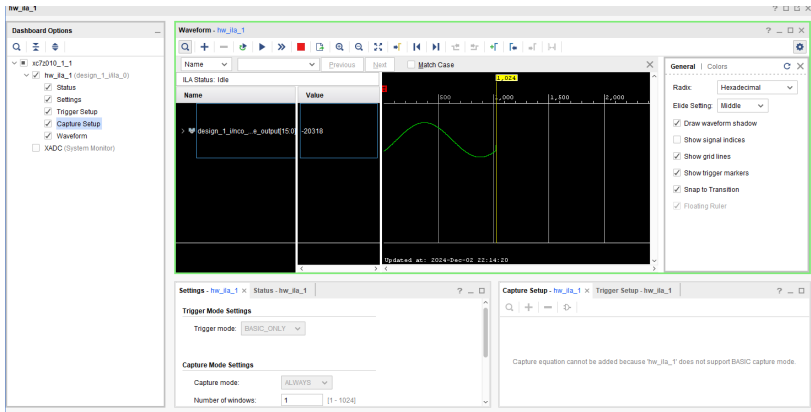


Figura: Prueba I mediante C.

# Pruebas C SDK II

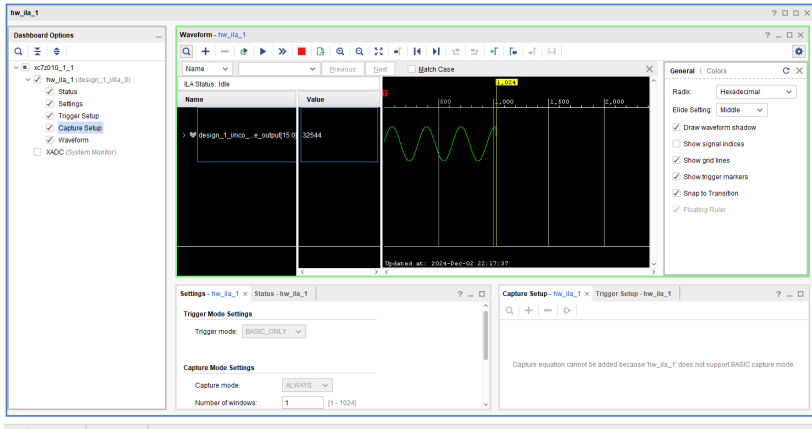
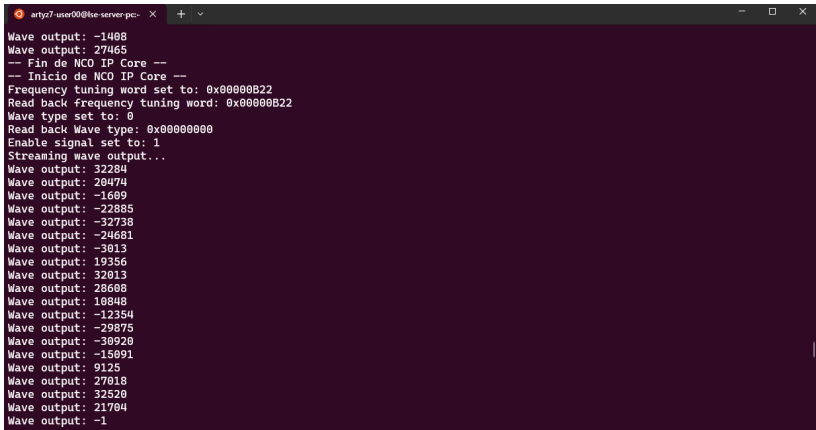


Figura: Prueba II mediante C.

## Pruebas C SDK III

Se probaron los outputs del NCO como registro, imprimiendo los valores por UART, aunque esto solo funciona a bajas frecuencias. Con los valores obtenidos por UART, se reconstruyó la onda usando Python.

A terminal window titled 'artyz7-user00@fse-server-pc-' displays the UART output of an NCO IP core. The output shows a sequence of sine wave samples. The first two samples are -1408 and 27465. This is followed by status messages: '-- Fin de NCO IP Core --' and '-- Inicio de NCO IP Core --'. Then, frequency tuning words are set to 0x00000822. The wave type is set to 0. The enable signal is set to 1. Streaming wave output begins, showing a series of 20 samples: 32284, 20474, -1609, -22885, -32738, -24681, -3013, 19356, 32013, 28608, 10848, -12354, -29875, -30920, -15091, 9125, 27018, 32520, 21704, and -1.

```
artyz7-user00@fse-server-pc-
Wave output: -1408
Wave output: 27465
-- Fin de NCO IP Core --
-- Inicio de NCO IP Core --
Frequency tuning word set to: 0x00000822
Read back frequency tuning word: 0x00000822
Wave type set to: 0
Read back Wave type: 0x00000000
Enable signal set to: 1
Streaming wave output...
Wave output: 32284
Wave output: 20474
Wave output: -1609
Wave output: -22885
Wave output: -32738
Wave output: -24681
Wave output: -3013
Wave output: 19356
Wave output: 32013
Wave output: 28608
Wave output: 10848
Wave output: -12354
Wave output: -29875
Wave output: -30920
Wave output: -15091
Wave output: 9125
Wave output: 27018
Wave output: 32520
Wave output: 21704
Wave output: -1
```

Figura: Valores del seno con UART.

# Pruebas C SDK III

La onda presentada mediante UART es una función seno de 2.8Khz, la reconstrucción es la siguiente:

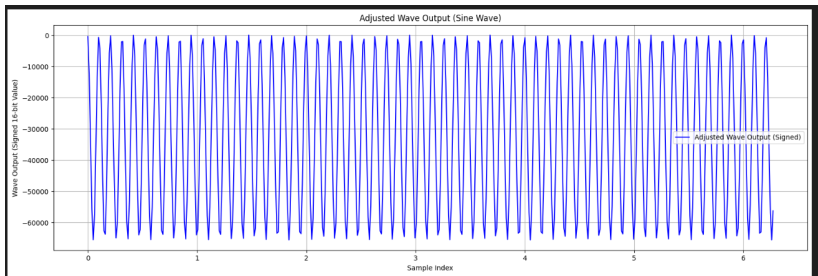


Figura: Reconstrucción de valores UART.

# Tabla de Recursos

La tabla de recursos generada por Vivado denota una alta utilización de IO:

Utilization

Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	1506	17600	8.56
LUTRAM	156	6000	2.60
FF	2474	35200	7.03
BRAM	1	60	1.67
BUFG	2	32	6.25

Figura: Tabla de recursos.

Gracias por su atencion