.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo práctico Final

Circuitos logicos programables

Docente: Ing. Nicolás Álvarez
Autor: Abraham Rodriguez
Email: abraham.rodz17@gmail.com

Octubre, 2024

Índice

1. Funcionalidad e implementación

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

1.1. Repositorio

El trabajo realizado se encuentra en github:
https://github.com/AbeRodz/Numeric-Controlled-Oscillator

1.2. Numeric Controlled Oscillator (NCO)

Un NCO consiste de un acumulador de fase y convertidor de fase de amplitud (Generador de ondas), generalizando en un diagrama corresponde a la imagen siguiente:

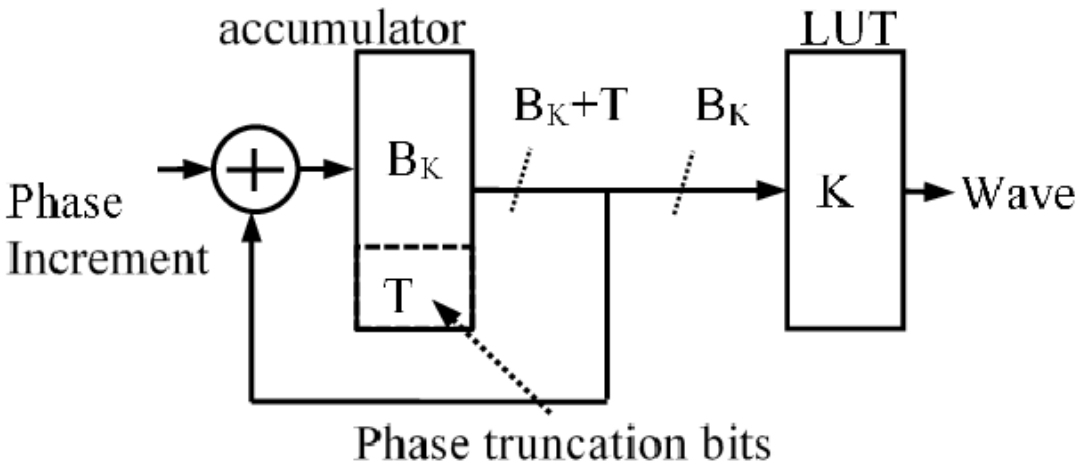

Figura 1: Diagrama interno NCO.

El generador de ondas es comúnmente una LUT con los valores de la función seno, sin embargo existen algoritmos para generar los valores como:

- CORDIC
- Aproximación de Taylor
- Aproximación lineal
- Direct Digital Synthesis (DDS)

La NCO es capaz de generar distintos tipos de onda, en este caso se implementaron las ondas:

- Seno
- Cuadrada
- Triangular
- Sierra

Lo que difiere entre todos los métodos es la cantidad de memoria utilizada, la velocidad y la precisión. En el presente trabajo se implemento un generador basado en LUT, lo cual implica que el consumo de memoria es alto a comparación de otros métodos.

1.2.1. Acumulador

El acumulador es muy simple, consiste en aplicar la operación de suma de la fase o *Frequency Tuning/Control Word* actual con el estado anterior. En codigo simplificado VHDL del acumulador se ve de la siguiente manera:

```
1
2   process (c_i)
3   begin
4     if rising_edge(c_i) then
5       if en_i = '1' then
6         -- Phase accumulator update
7         phase_acc <= phase_acc + unsigned(freq_word_i);
8
9         -- Extract upper bits of the phase accumulator for the SineLUT
10        addr <= std_logic_vector(phase_acc(31 downto 22));
11      end if;
12    end if;
13  end process;
```

Listing 1: Acumulador

La variable phase_acc contiene el valor acumulado en 32 bits, sin embargo la implementación de sineLUT contiene una LUT de 1024 registros (10bits) en otras palabras una muestra discreta de 1024 registros de una función senoidal, por lo tanto se extraen los bits mas significativos para ser mapeados a la LUT.

1.2.2. SineLUT

Para generar los valores de la funcion seno, se realizo un script de Python y se guardo a un archivo .txt para luego ser escritos al modulo de la LUT. El script consiste en:

```
1
2 def generate_uint16_sine_table(num_samples: int = 1024, amplitude: int = 32767,
      offset:int = 32767) -> tuple[np.ndarray]:
3     num_values = num_samples
4     x = np.linspace(0, 2 * np.pi, num_values, endpoint=False)
5     y = np.round(amplitude * np.sin(x) + offset).astype(int)
6     return x,y
```

Listing 2: Acumulador

En este caso la LUT contiene valores de la función seno con los siguientes parámetros:

- 1024 muestras.
- Tipo uint16.

En otras palabras la LUT contiene 10x16bits.
Al graficar la muestra de la función seno obtenida se obtiene:

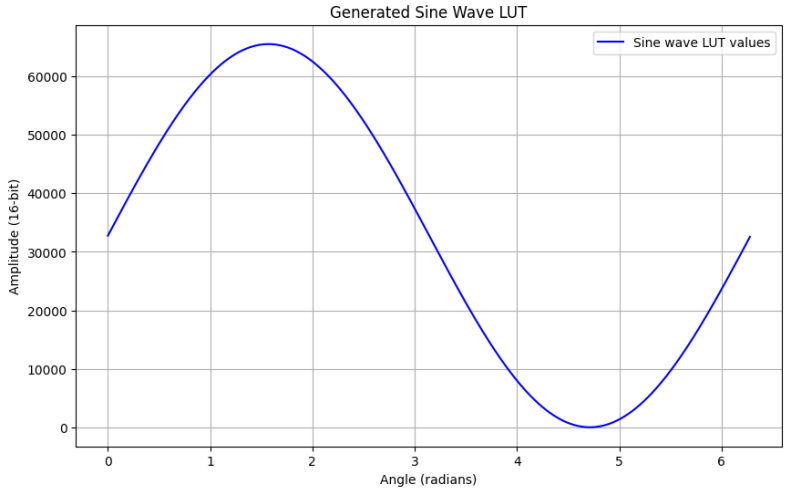

Figura 2: Muestra de la función seno.

Una vez generados los valores, se creo el modulo sineLUT.vhd, el cual guarda en memoria una variable llamada ROM con los valores de la muestra. Para buscar los valores dentro de la tabla, se debe de buscar un registro dentro de la tabla, este registro es calculado del acumulador.

```
1
2  begin
3
4    SIN : process (c_i)
5    begin
6      if rising_edge(c_i) then
7        if (en_i = '1') then
8          wave_o <= std_logic_vector(to_Signed((t_ROM(to_Integer(Unsigned(addr_i))) -
       32768), 16));
9        end if;
10     end if;
11   end process SIN;
12
13 end;
```

Listing 3: SineLUT

1.2.3. NCO

La NCO consiste en una instancia de sineLUT y una signal como acumulador, la funcionalidad basica es dada por el bloque de código mostrado anteriormente 1.2.1.

La NCO implementada en el presente trabajo es capaz de generar varios tipos de onda utilizando postprocesamiento de la señal dada por la sineLUT y el acumulador.

```
1
2  begin
3     if rising_edge(c_i) then
4       if en_i = '1' then
5         -- Phase accumulator update
6         phase_acc <= phase_acc + unsigned(freq_word_i);
7
8         -- Extract upper bits of the phase accumulator for the SineLUT
9         addr <= std_logic_vector(phase_acc(31 downto 22));
10
11        -- Generate square wave (based on MSB of phase accumulator)
12        if phase_acc(31) = '1' then
13          square_wave <= std_logic_vector(to_signed(32767, 16)); -- High output
14        else
```

```
15            square_wave <= std_logic_vector(to_signed(-32767, 16)); -- Low output
16          end if;
17
18          -- Generate triangle wave
19          if phase_acc(31) = '1' then
20            -- Descending part of the triangle wave (negative half)
21            triangle_wave <= std_logic_vector(to_signed(32767 - to_integer(unsigned(
    phase_acc(30 downto 16))), 16));
22          else
23            -- Ascending part of the triangle wave (positive half)
24            triangle_wave <= std_logic_vector(to_signed(to_integer(unsigned(phase_acc
    (30 downto 16))), 16));
25          end if;
26
27          -- Generate sawtooth wave (directly from phase accumulator)
28          sawtooth_wave <= std_logic_vector(phase_acc(31 downto 16));
29
30          -- Select the output waveform based on wave_type_i
31          case wave_type_i is
32            when "00" =>
33              wave_o <= sine_wave; -- Output sine wave
34            when "01" =>
35              wave_o <= square_wave; -- Output square wave
36            when "10" =>
37              wave_o <= triangle_wave; -- Output triangle wave
38            when "11" =>
39              wave_o <= sawtooth_wave; -- Output sawtooth wave
40            when others =>
41              wave_o <= sine_wave; -- Default to sine wave
42          end case;
43        end if;
44      end if;
45    end process;
```

Listing 4: SineLUT

1.3. Diagrama de bloques

El diagrama de bloques de la NCO es presentado mediante el esquematico generado por Vivado.

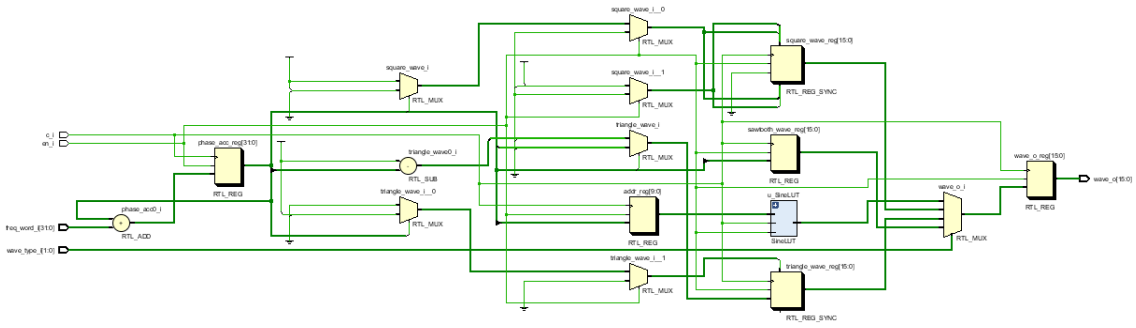

Figura 3: Diagrama de bloques NCO.

El diagrama del sineLUT es el siguiente:

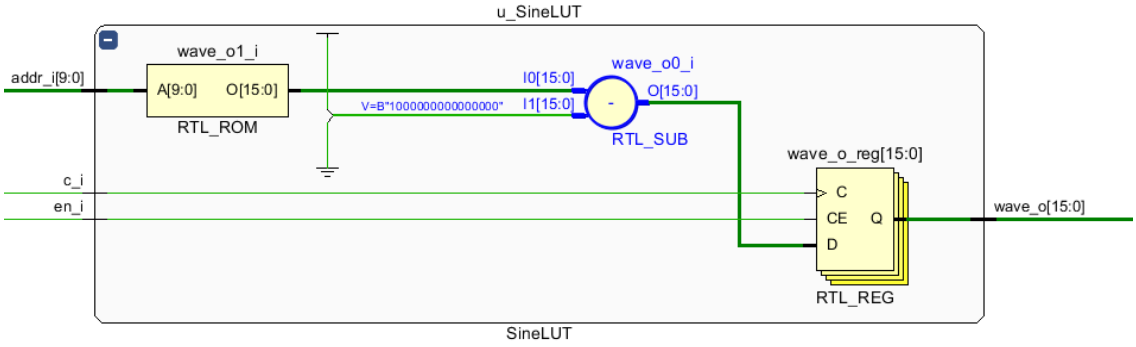

Figura 4: Diagrama de bloques sineLUT.

2. Sintesis

Al realizar la sintesis se generaron los siguientes diagramas para la NCO y sineLUT.

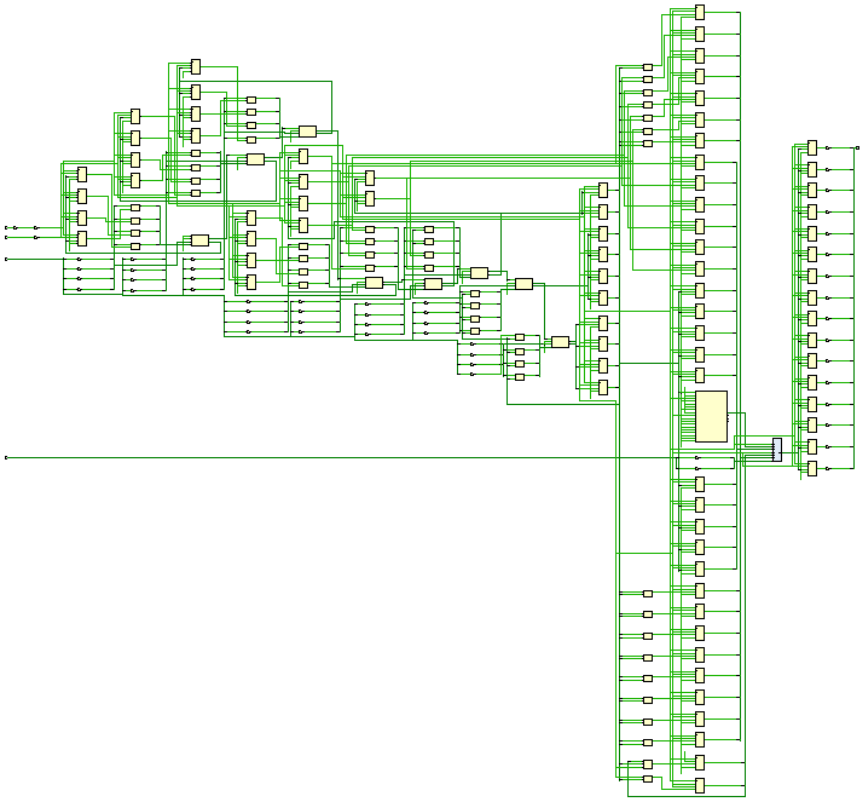

Figura 5: Sintesis NCO.

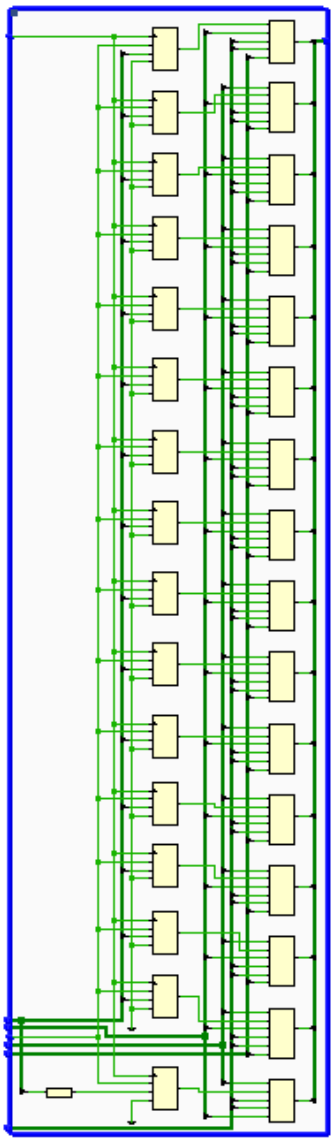

Figura 6: Sintesis sineLUT.

3. Simulaciones

Utilizando Vivado y un testbench se simularon distintas formas de onda mostadas en la imagen siguiente:

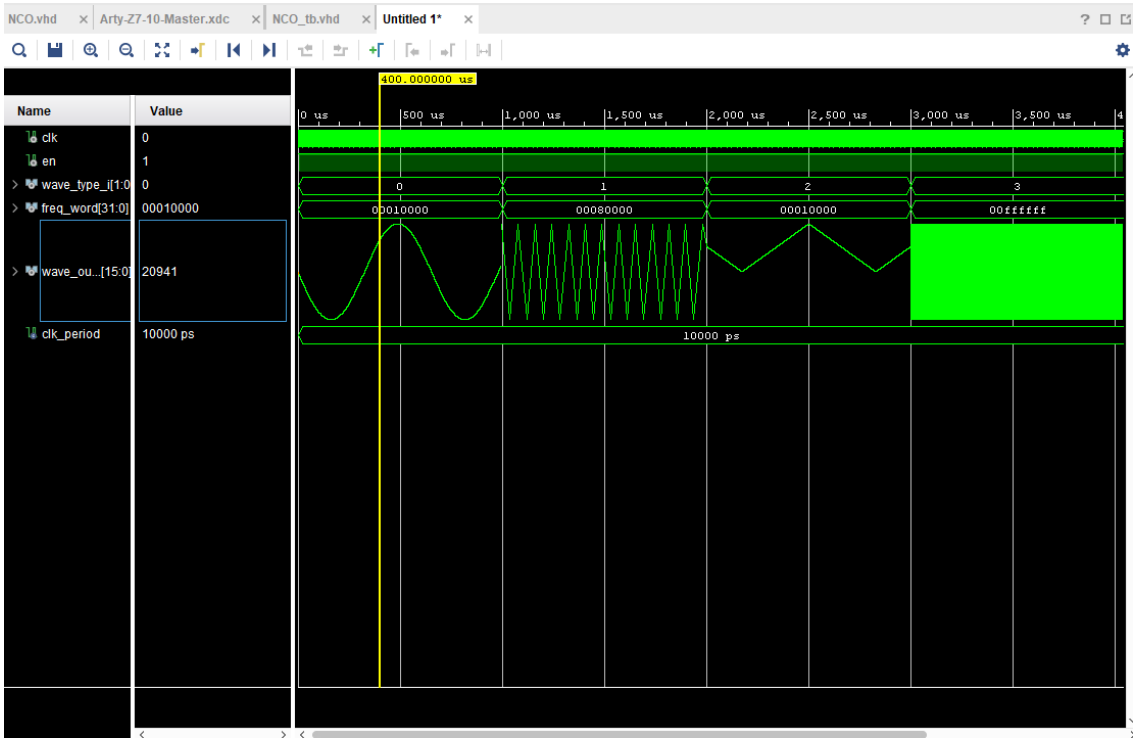

Figura 7: Simulaciones

El testbench cuenta con estímulos para generar ondas a distintas frecuencias:

```
1
2  -- Stimulus process to apply different test cases
3   stimulus : process
4   begin
5     -- Apply different wave types and frequency tuning words
6     wave_type_i <= "00"; -- Sine wave
7     freq_word   <= x"00010000"; -- Low frequency
8     wait for 1000 us;
9
10    wave_type_i <= "01"; -- Square wave
11    freq_word   <= x"00080000"; -- Medium frequency
12    wait for 1000 us;
13
14    wave_type_i <= "10"; -- Triangle wave
15    freq_word   <= x"00010000"; -- High frequency
16    wait for 1000 us;
17
18    wave_type_i <= "11"; -- Sawtooth wave
19    freq_word   <= x"00FFFFFF"; -- Higher frequency
20    wait for 1000 us;
```

Listing 5: SineLUT

4. Tabla de recursos

La tabla de recursos generada por Vivado denota una una alta utilización de IO, esto es debido a los tipos de datos que se estan utilizando, 32 bits para la frecuencia y la salida de la onda es de 16 bits.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	59	17600	0.34
FF	97	35200	0.28
BRAM	0.50	60	0.83
IO	52	100	52.00
BUFG	1	32	3.13

Figura 8: Tabla de recursos.

5. Pruebas con FPGA

Se configuro el VIO y la ILA y se creo el siguiente diagrama

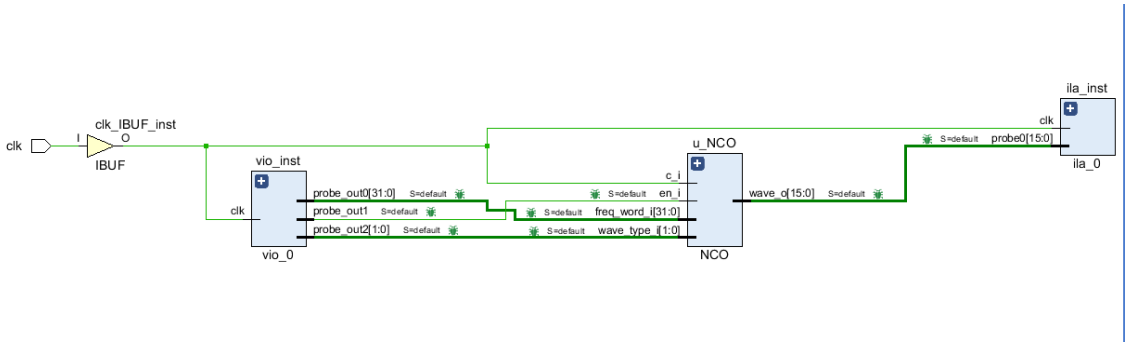

Figura 9: VIO e ILA.

Se genero el bitstream y se realizaron pruebas con la FPGA remota mediante el server lse, se probo generando distintas ondas a diferentes frecuencias:

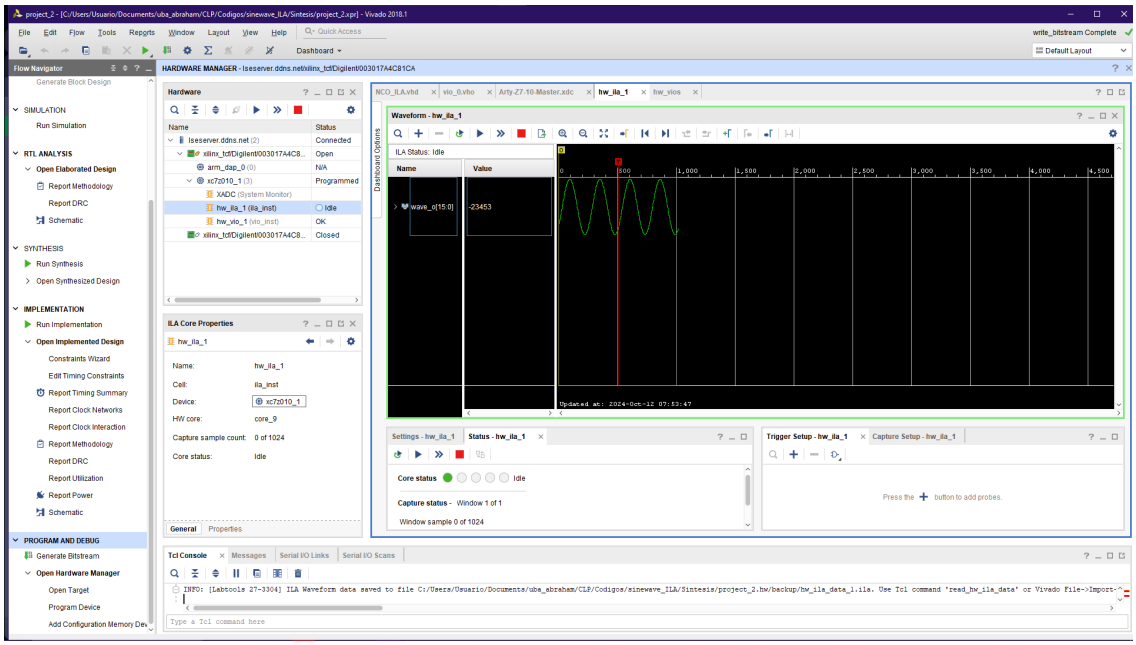

Figura 10: Seno en FPGA remota.

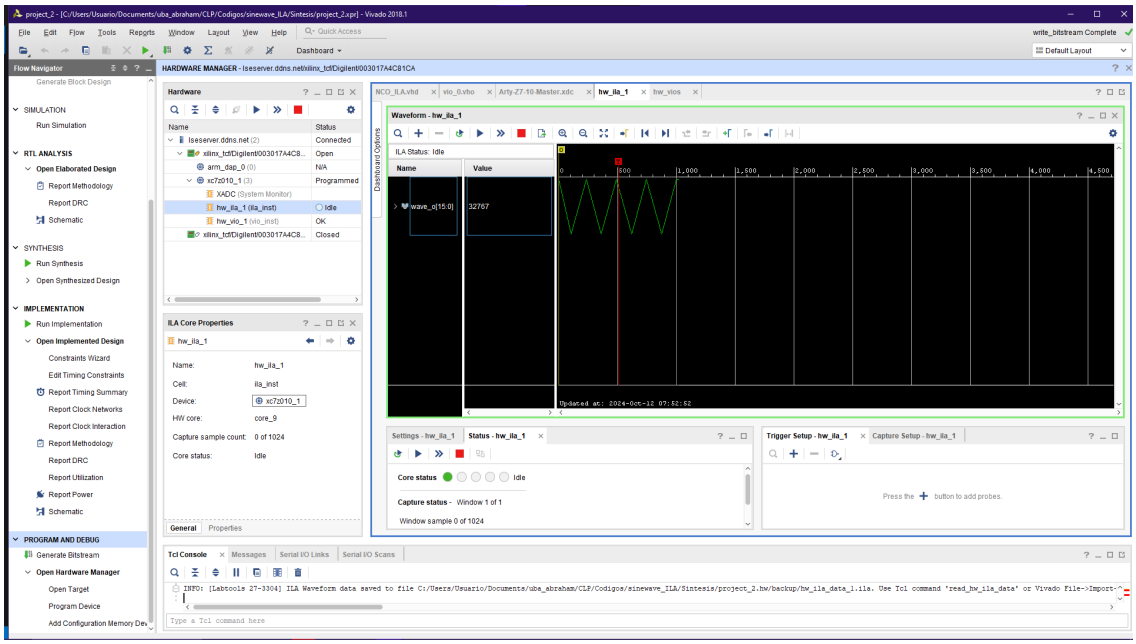

Figura 11: Triangular en FPGA remota.

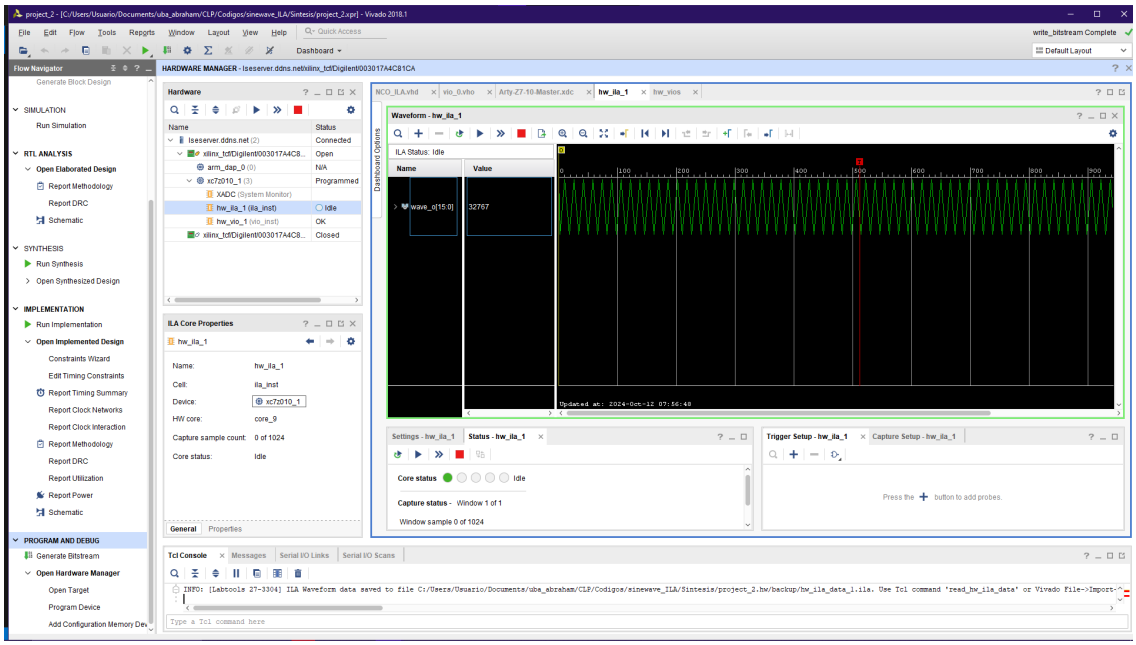

Figura 12: Triangular de mayor frecuencia en FPGA remota.