Circuitos Lógicos Programables

Trabajo Práctico Final

Módulo Generador de PWM

Rubén Darío Mansilla

(quiquemansilla1@gmail.com)

13/08/2024

Docente: Nicolás Álvarez

Historial de cambios

Versión	Fecha	Descripción	Autor	Revisores
A	13/8/24	Versión Original	Rubén Mansilla	

Índice de contenido

1. Introducción

1.1. Descripción

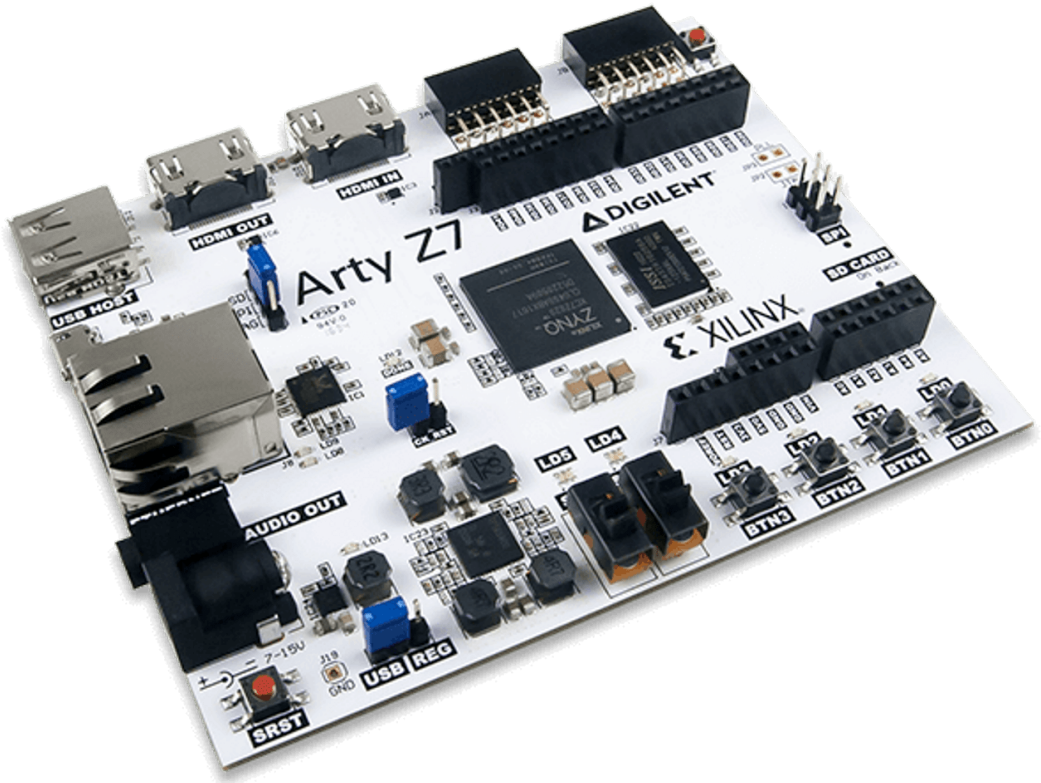
Este documento describe el desarrollo del Trabajo Final de la materia **Circuitos Lógicos Programables** de la Carrera de Especialización en Sistemas Embebidos.

1.2. Propósito

El objetivo de este trabajo práctico consiste en desarrollar e implementar, utilizando una FPGA, un generador de señales PWM (Pulse Width Modulation) mediante síntesis utilizando un contador de módulo N (escalable) como base, al que se le ingresa un valor correspondiente al módulo de la cuenta, que sería el período y, comparando la cuenta contra un valor ingresado correspondiente al duty cicle. La salida PWM se mantendrá en High mientras *Cuenta* < *duty cicle* y pasará a Low hasta que se alcance el valor correspondiente al módulo y repetirá el ciclo.

1.3. Plataforma

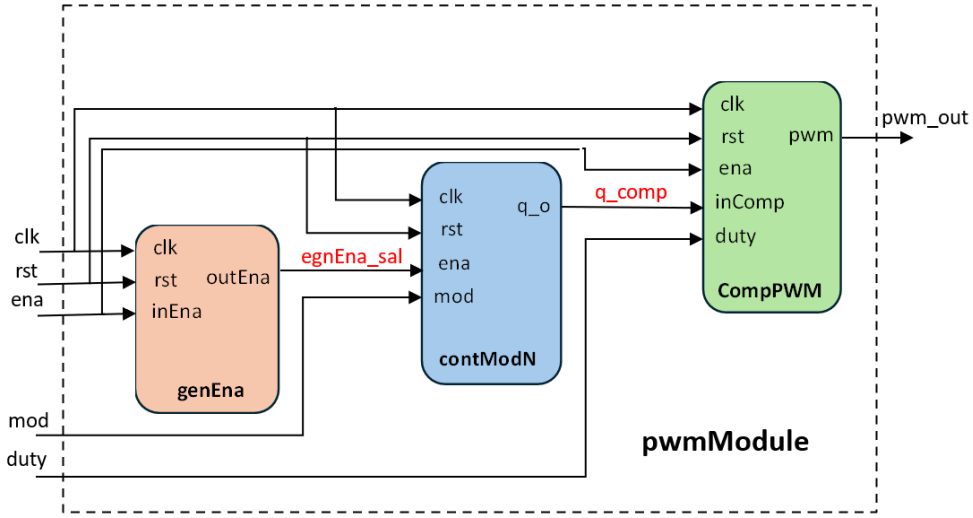
El trabajo fue desarrollado para implementarse en una FPGA de Xilinx, en particular sobre una placa modelo **Arty Z7-10** de **Digilent**.



2. Desarrollo del Trabajo Práctico

2.1. Implementación

El esquema principal de nuestro módulo se obtiene interconectando al contador de módulo N un comparador que será el que le dé forma a la señal PWM y un generador de habilitación que permitirá modificar la velocidad de la cuenta, controlando la frecuencia.



Descripción de funcionamiento: La señal de clock, procedente del clock principal de la placa FPGA, *clk* alimenta a los tres submódulos que conforman el generador de señales PWM. Con cada flanco ascendente de *clk* el contador **contModN** incrementará en uno su cuenta y la presentará en su salida *q_o*, la cual se comparará contra el valor determinado en la entrada *duty* en el módulo comparador **compPWM**.

El comparador conformará en su salida *pwm_out* la señal PWM estableciendo un estado High mientras se cumpla que la cuenta *q_o* < *duty*, para luego establecer un estado Low hasta que el contador reinicie la cuenta y comience un nuevo período.

El generador de habilitación mantendrá su salida *outEna* en estado Low hasta que la cantidad de pulsos de *clk*, cuya cuanta se almacena en una variable interna *aux* alcance el valor configurado en una variable interna *NC*. Una vez que la cuenta en *aux* = *NC* la salida *outEna* pasará al estado High por un ciclo de *clk*, luego *aux* reinicia su cuenta.

En resumen, *genEna* pondrá su salida *outEna* en “1” cada *NC* ciclos de clock. Esto dará como resultado que el contador **contModN** incrementará su cuenta cada vez que su entrada *ena, conectada a la salida outEna* se habilite, dando como resultado una reducción en la velocidad de cuenta y por ende de la frecuencia del período de la señal PWM.

La señal de reset *rst* fuerza la salida de los todos los módulos a “0”.

Resumen de las señales del módulo PWM:

Entradas:

clk → Clock de la placa (125 MHz) [1 bit]

rst → Reset: Fuerza a "0" todas las salidas de los módulos del componente. [1 bit]

ena →Enable: Habilita el funcionamiento de todos los módulos del componente. [1 bit]

mod → Módulo: Valor que establece el período de la señal PWM. [N bits]

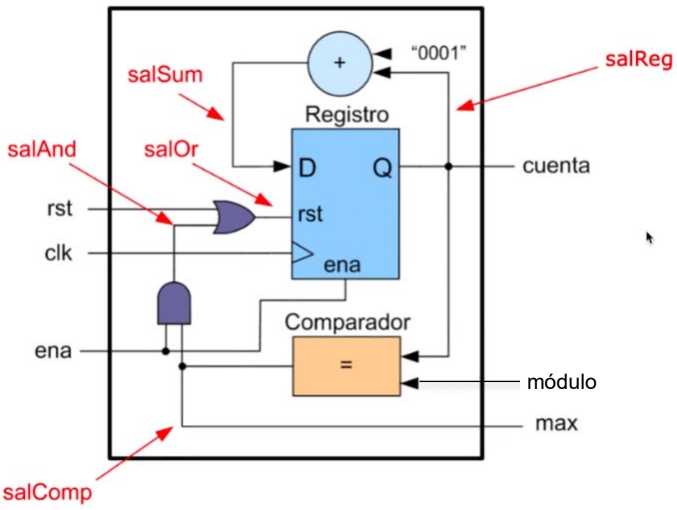
duty → Duty cicle: Valor que establece la cantidad de cuentas que la salida PWM permanecerá en "1". [N bits]

Salidas:

pwm_out → Salida PWM [1 bit] serial.

2.2. El contador de módulo N: contModN.

El contador de módulo N, se desarrolla partiendo del ejemplo del contador BCD que vimos en clase al cual se le agrega una entrada *modulo* que será el valor hasta el cual llegará la cuenta para, luego, reiniciar en cero.



Este esquema constituye la arquitectura del módulo que llamaremos **contModN** y que será el corazón de nuestro módulo PWM. Básicamente incrementará en "**1**" la cuenta por cada ciclo de *clock* hasta que alcance el valor determinado en la entrada módulo. Luego reinicia la cuenta en cero. Para nuestro propósito eliminamos la salida *max* que no será necesaria.

El valor de la entrada *modulo* determinará, en nuestro módulo principal, el período de esta señal PWM.

2.2.1 Implementación del Generador de Habilitación: genEna

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity genEna is
    generic(
        NC: in natural := 4
    port(
        clk_i : in std_logic;
        rst_i : in std_logic;
        ena_i : in std_logic;
        q_o   : out std_logic
    );
end;

architecture genEna_arq of genEna is
    -- Parte declarativa
    signal cuenta : integer;

begin
    -- Parte descriptiva

    process(clk_i)
        variable aux: integer;
    begin
        if rising_edge(clk_i) then
            if rst_i = '1' then
                aux := 0;
                q_o <= '0';
            elsif ena_i = '1' then
                aux := aux + 1;
                if aux = NC then
                    q_o <= '1';
                    aux := 0;
                else
                    q_o <= '0';
                end if;
            end if;
        end if;
        cuenta <= aux;
    end process;

end;
```

En la captura podemos ver el desarrollo del código en VHDL. Usamos una variable interna *NC* que determinará cada cuantos pulsos de clock la salida *q_o*, conectada al *ena* del contador de módulo N, pasará a “1”.

En el **process** podemos ver la implementación del comportamiento de este módulo, descripto más arriba.

2.2.2 Implementación del contador de módulo N: contModN

```vhdl
entity contModN2 is
    generic(
        N : natural := 4
    );
    port(
        clk_i : in std_logic;
        rst_i : in std_logic;
        ena_i : in std_logic;
        mod_i : in std_logic_vector(N-1 downto 0); -- Modulo de cuenta
        q_o   : out std_logic_vector(N-1 downto 0)
        --max_o : out std_logic
    );
end;
```

Esta implementación utiliza una variable interna N que determinará la cantidad de bits que se desee que tenga el contador. Utiliza un como base un registro denominado **reg** formado por N Flip Flops D, cuya implementación obviaremos para no hacer tan largo el presente documento.

```
architecture contModN2_arq of contModN2 is
    -- Parte declarativa

    component reg is
        generic(
            N: in natural := 4
        );
        port(
            clk_i : in std_logic;
            rst_i : in std_logic;
            ena_i : in std_logic;
            d_i   : in std_logic_vector(N-1 downto 0);
            q_o   : out std_logic_vector(N-1 downto 0)
        );
    end component;

    signal salReg : std_logic_vector(N-1 downto 0);
    signal salSum : std_logic_vector(N-1 downto 0);
    signal salOr  : std_logic;
    signal salAnd : std_logic;
    signal salComp:std_logic;
```

```
begin
    -- Parte descriptiva

    reg_inst: reg
        generic map(
            N => 4
        )
        port map(
            clk_i => clk_i,
            rst_i => salOr,
            ena_i => ena_i,
            d_i   => salSum,
            q_o   => salReg
        );

    -- mod_i <= "1111"

    salSum <= std_logic_vector(unsigned(salReg) + "0001");

    salComp <= '1' when salReg = mod_i else '0';

    salAnd <= ena_i and salComp;

    salOr <= rst_i or salAnd;

    --max_o <= salComp;

    q_o <= salReg;
end;
```

En la parte declarativa vemos como se implementa el comportamiento del módulo contador

2.2.3 Implementación del módulo generador de PWM: pwmModule

```
entity pwmModule is
    generic(
        N : natural := 4
    );
    port(
        clk_i   : in  std_logic;
        rst_i   : in  std_logic;
        mod_i   : in  std_logic_vector(N-1 downto 0);    -- Período del PWM
        ena_i   : in  std_logic;
        duty_i  : in  std_logic_vector(N-1 downto 0);    -- Ciclo de trabajo
        q_o     : out std_logic_vector(N-1 downto 0); -- Salida del contador contModN2
        pwm_out : out std_logic
    );
end;
```

La variable interna **N** determina la cantidad de bits de las entradas *mod_i*, *duty_i* y la salida *q_o* y debe ser consistente con el valor adoptado en el contador **contMosN**.

```
architecture pwmModule_arq of pwmModule is
    -- Parte declarativa
    signal q_internal : std_logic_vector(N-1 downto 0);

    component contModN2 is
        generic(
            N : natural := 4
        );
        port(
            clk_i : in std_logic;
            rst_i : in std_logic;
            ena_i : in std_logic;
            mod_i : in std_logic_vector(N-1 downto 0); -- Modulo de cuenta --> Período
            q_o   : out std_logic_vector(N-1 downto 0)
        );
    end component;

    signal genEna_sal : std_logic;
```

La señal *q_internal* se utiliza para poder visualizar en el simulador el valor de la cuenta de salida del contador **contModN**.

```
begin
    -- Parte descriptiva

    contModN2_inst: contModN2
        generic map(
            N => 4
        )
        port map(
            clk_i => clk_i,
            rst_i => rst_i,
            ena_i => genEna_sal,
            mod_i => mod_i,
            q_o   => q_internal
        );

    genEna_inst: entity work.genEna
        generic map(
            NC => 3 --N
        )
        port map(
            clk_i => clk_i,
            rst_i => rst_i,
            ena_i => ena_i,
            q_o   => genEna_sal
        );
```

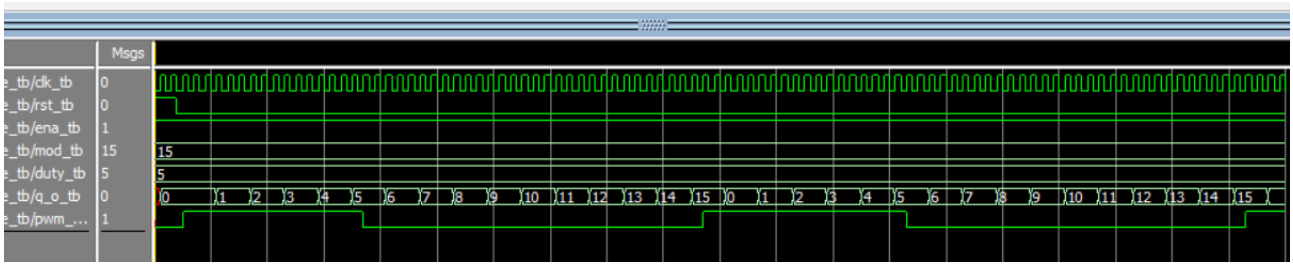
```
    -- Aqui ponemos las operaciones propias del Modulo pwm
    process (clk_i, rst_i)
    begin
        if rst_i = '1' then
            pwm_out <= '0';
        elsif rising_edge(clk_i) then
            if ena_i = '1' then
                if unsigned(q_internal) < unsigned(duty_i) then
                    pwm_out <= '1';
                else
                    pwm_out <= '0';
                end if;
            end if;
            if unsigned(q_internal) >= unsigned(mod_i) then
                pwm_out <= '1';
            end if;
        end if;
    end process;

    q_o <= q_internal;

end;
```

2.3. Pruebas y Simulaciones.

Las pruebas de simulación que describen las imágenes a continuación se realizaron en Vivado, aunque las primeras fueron realizadas en ModelSim, que permite una depuración de errores más dinámica.



En la captura de arriba, de la simulación en ModelSim, podemos observar el comportamiento del generador de pwm. Para esta simulación se establecieron los siguientes valores:

modulo	15	16 cuentas
duty	5	6 cuentas
N	4	bits
NC	3	Ciclos
clock	20 ns	Periodo

Salida PWM	37,5% duty cicle
	6 cuentas en "1"
	10 cuentas en "0"

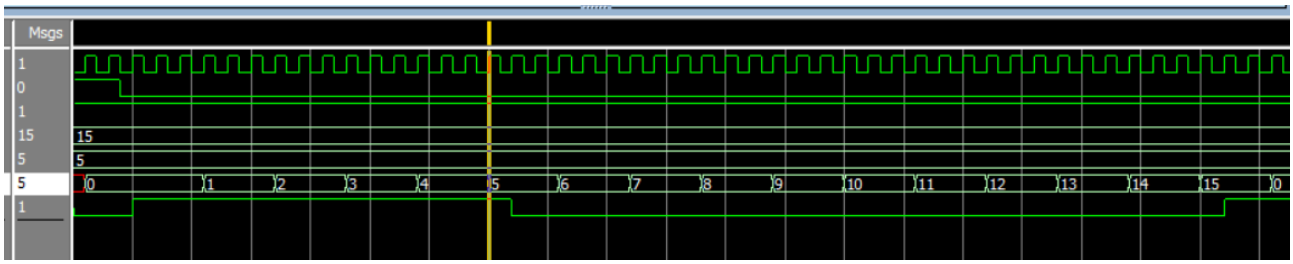
Cada 3 ciclos de clock el contador hace una cuenta.

$$\text{Tiempo de cuenta} = 20\ ns\ \times 3 = 60\ ns$$

$$\text{Período de PWM} = 60\ ns\ \times 16 = 960\ ns$$

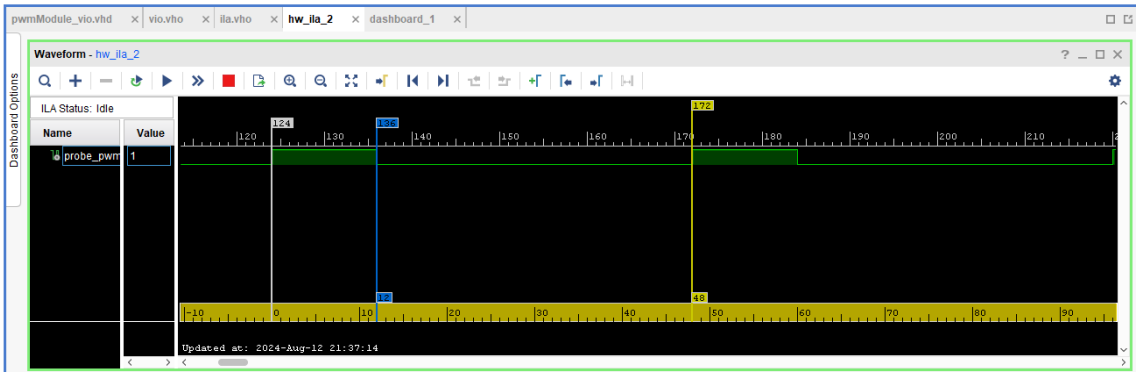
$$\text{Duty cicle} = 60\ ns\ \times 6 = 360\ ns$$

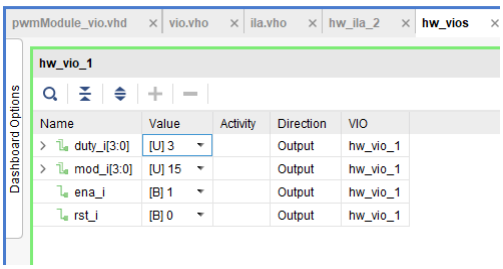
$$\text{Salida PWM} = \frac{Duty\ cicle}{Periodo\ PWM} = \frac{6}{16} \times 100 = 37{,}5\%$$



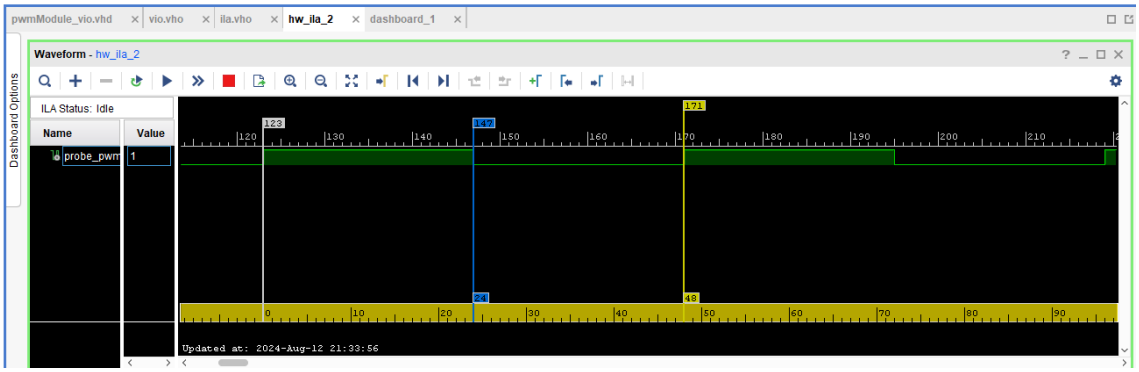
En Vivado se implementa el módulo pwmModule y se utiliza un IP cord VIO para emular las entradas y un IP cord ILA para visualizar el comportamiento de la señal PWM

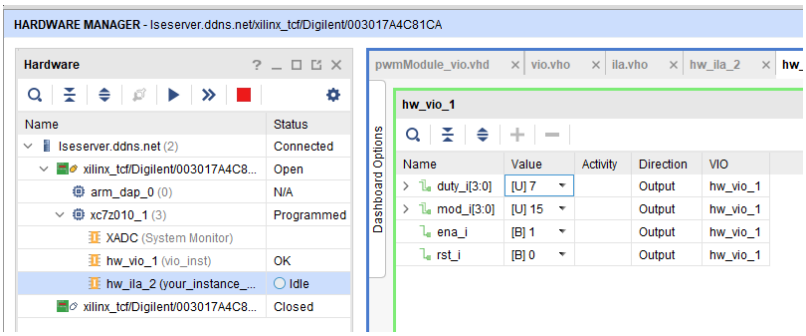
Las siguientes capturas muestran el comportamiento del componente corriendo en la placa FPGA para distintas configuraciones:



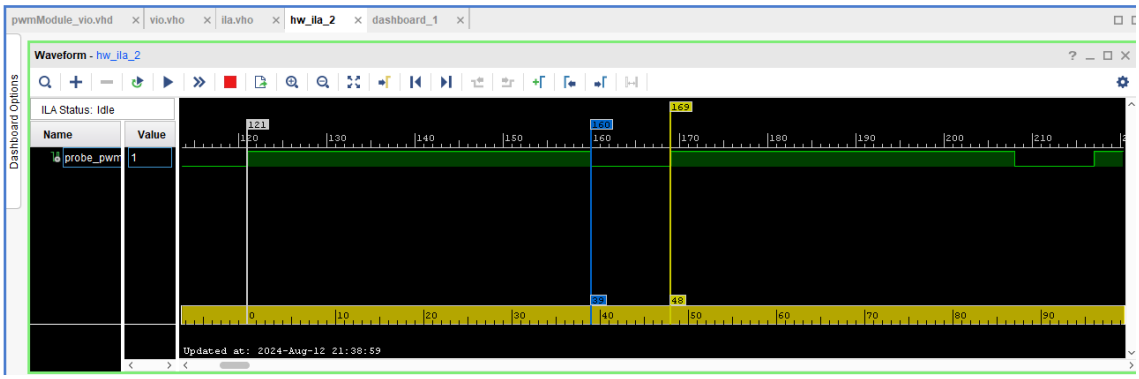


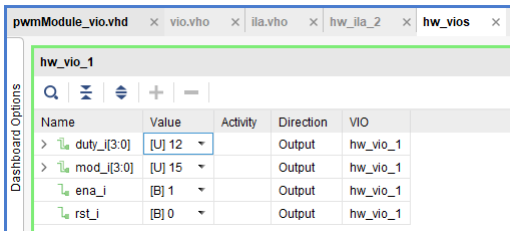
Señal pwm al 25%





Señal PWM al 50%





Señal PWM al 75%

Una cuenta de contModN son 3 marcas en el grafico – 16 cuentas conforman un periodo de 48 marcas.

Frecuencia clock = 125 MHz → Período clock $= \frac{1}{125}\ \mu s = 0{,}008\ \mu s = 8\ ns$

$$\text{Tiempo de cuenta} = 8\ ns\ \times 3 = 24\ ns$$

$$\text{Período de PWM} = 24\ ns\ \times 16 = 384\ ns$$

$$\text{Duty cicle} = 24\ ns\ \times 12 = 288\ ns$$

$$\text{Salida PWM} = \frac{Duty\ cicle}{Periodo\ PWM} = \frac{12}{16} \times 100 = 75\%$$

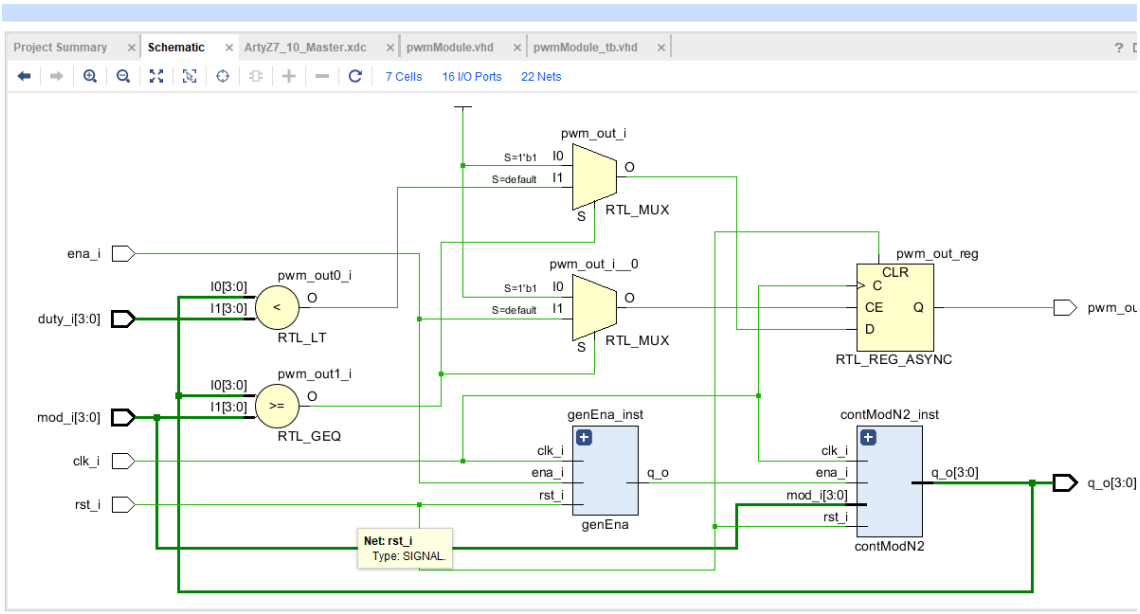
$$\text{1 marca en el grafico} = \frac{tiempo\ cuenta}{3} = \frac{24\ ns}{3} = 8\ ns$$

2.4. Recursos usados en la FPGA.

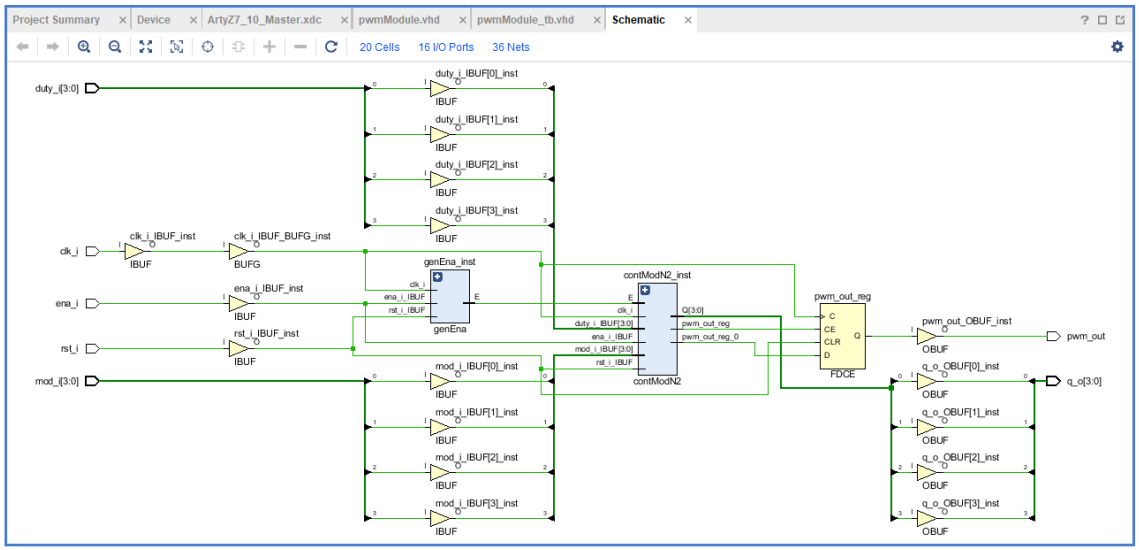
Luego de realizar las simulaciones en Vivado, se realizó una captura de la síntesis que realiza la herramienta. Se puede ver como el esquema de circuito responde al esquema ilustrado anteriormente, sobre el que se desarrolló la implementación.

2.4.1 Elaborated Design

El esquemático para el diseño en RTL ANALYSIS es el siguiente

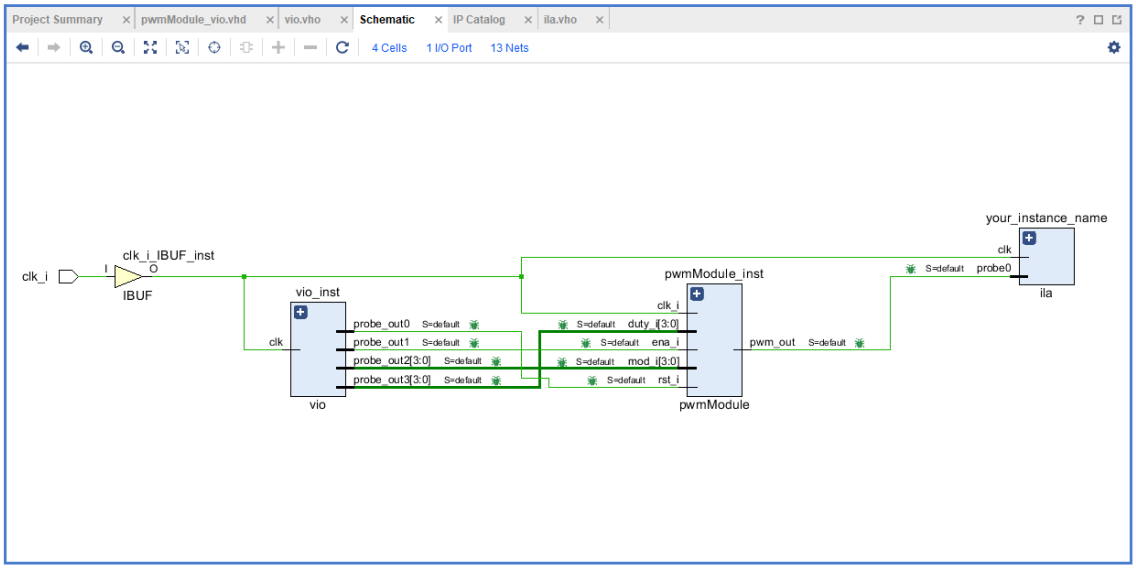


El Esquemático de la Implementación coincide con el de la síntesis y se puede observar en la figura siguiente.

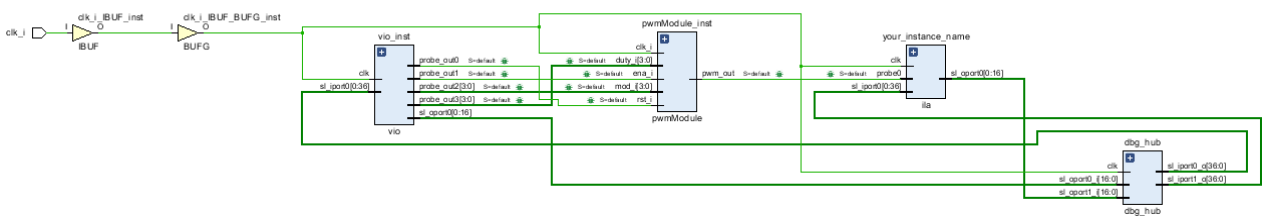


Para poder programar la FPGA se utiliza un IP cord **VIO** para emular las entradas y un IP cord **ILA** para visualizar el comportamiento de la señal PWM.

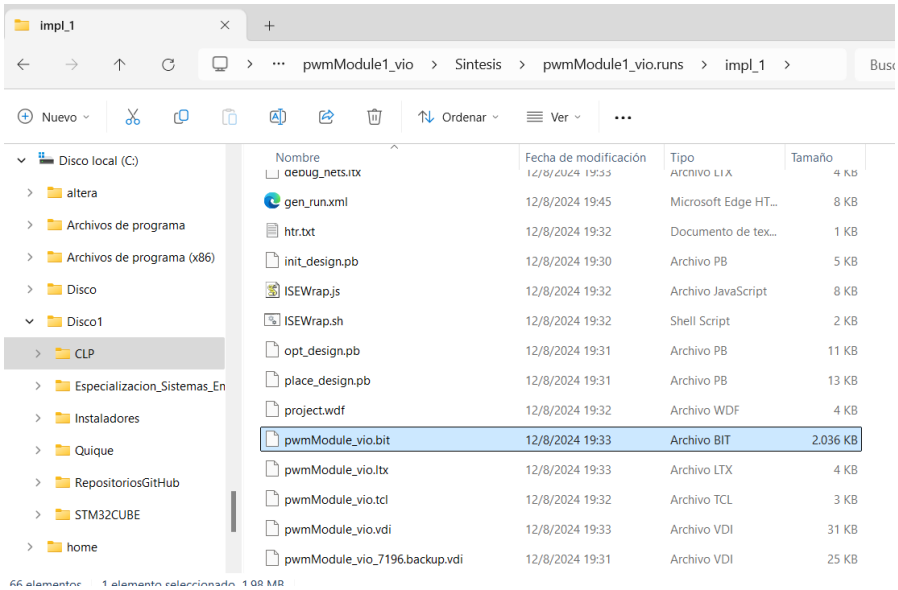
El esquemático del conjunto desarrollado en Vivado con VIO e ILA se ve en la figura siguiente.



El Esquemático de la implementación se observa en la figura siguiente.

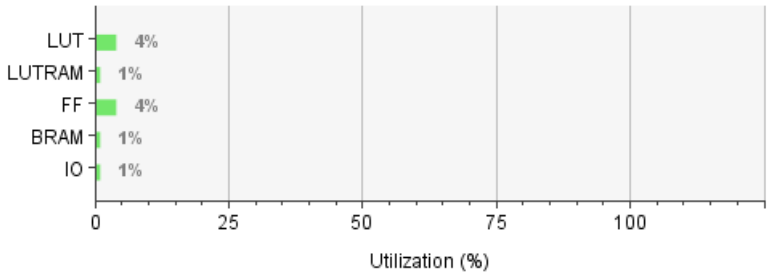


Generación del archivo *bitstream*: se ve en la figura siguiente.



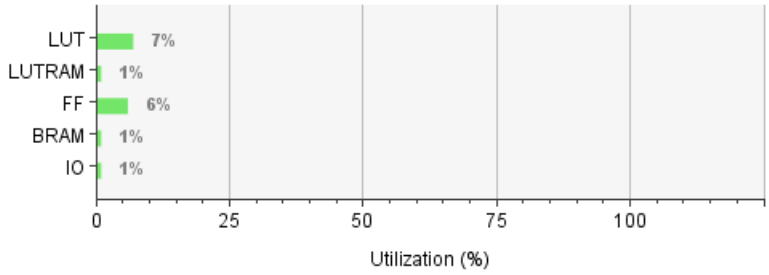
2.4.2 Utilization Percentage Post-Synthesis

Resource	Utilization	Available	Utilization %
LUT	722	17600	4.10
LUTRAM	61	6000	1.02
FF	1365	35200	3.88
BRAM	0.50	60	0.83
IO	1	100	1.00



2.4.3 Utilization Percentage Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	1177	17600	6.69
LUTRAM	85	6000	1.42
FF	2028	35200	5.76
BRAM	0.50	60	0.83
IO	1	100	1.00



2.4.4 Power Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.092 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,1°C**
Thermal Margin:	58,9°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W