**Objetivo**

Haciendo uso de las ASM el alumno implementará el número de Fibonacci y realizará la programación en VDHL.

**Introducción**

La serie de Fibonacci es una secuencia matemática infinita que comienza generalmente con 0 y 1, y luego cada número subsiguiente es la suma de los dos números anteriores en la secuencia. La secuencia de Fibonacci se ve así: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... y así sucesivamente.

Para implementar la serie de Fibonacci en VHDL y mostrarla en 4 displays de 7 segmentos, necesitas seguir varios pasos:

Debes crear un módulo VHDL que genere la secuencia de Fibonacci. Puedes hacerlo usando una lógica secuencial que calcule los números de Fibonacci uno a uno.

Necesitas otro módulo VHDL que tome un número y lo muestre en un display de 7 segmentos. Esto implica convertir los dígitos decimales en las representaciones adecuadas para los displays de 7 segmentos.

Debes conectar el módulo que genera la secuencia de Fibonacci al módulo que muestra los números en los displays de 7 segmentos.

Necesitas un mecanismo para controlar cuántos números de Fibonacci se muestran en los displays de 7 segmentos. Puedes usar un contador o un reloj para avanzar en la secuencia y mostrar los números.

Asegúrate de que el sistema esté sincronizado correctamente para que los números de Fibonacci se muestren en los displays de 7 segmentos de manera adecuada y que la frecuencia de actualización sea la deseada.

**Desarrollo**

ACTIVIDAD COMPLEMENTARIA.

Diseñar un algoritmo en VHDL del número Fibonacci y desplegar el resultado de la secuencia en 4 displays de 7 segmentos. El número máximo para representar es 9999. Tener un botón de reinicio del sistema a ceros.

Para la codificación de los números de binario a 7 segmentos se usó la siguiente relación

Para el numero: 0 en 7 segmentos es: 1000000

Para el numero: 1 en 7 segmentos es: 1001111

Para el numero: 2 en 7 segmentos es: 0100100

Para el numero: 3 en 7 segmentos es: 0110000

Para el numero: 4 en 7 segmentos es: 0011001

Para el numero: 5 en 7 segmentos es: 0010010

Para el numero: 6 en 7 segmentos es: 0000010

Para el numero: 7 en 7 segmentos es: 0111000

Para el numero: 8 en 7 segmentos es: 0000000

Para el numero: 9 en 7 segmentos es: 0011000

**Código del programa**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use ieee.std\_logic\_unsigned.all;

entity p10 is

Port (clk: in std\_logic;

reset : in std\_logic;

start : in std\_logic;

i : in std\_logic\_vector(4 downto 0);

ready : out std\_logic;

done\_tick: out std\_logic;

display0, display1, display2,display3: buffer std\_logic\_vector(6 downto 0)

);

end p10;

architecture Behavioral of p10 is

type state\_type is (idle, op, done);

signal state\_reg : state\_type;

signal state\_next :state\_type;

signal t0\_reg, t0\_next: unsigned(12 downto 0);

signal t1\_reg, t1\_next: unsigned(12 downto 0);

signal n\_reg, n\_next : unsigned(4 downto 0);

signal div\_reloj : std\_logic;

signal f : std\_logic\_vector(12 downto 0);

begin

process(clk)

variable cuenta: std\_logic\_vector(27 downto 0) := x"0000000";

begin

if rising\_edge(clk) then

cuenta := cuenta + 1;

end if;

div\_reloj <= cuenta(23);

end process;

-- estado FSMD y registro de datos

process(div\_reloj, reset)

begin

if(reset='1')then

state\_reg <= idle;

t0\_reg <= (others=>'0');

t1\_reg <= (others=>'0');

n\_reg <= (others=>'0');

elsif(div\_reloj'event and div\_reloj='1')then

state\_reg <=state\_next;

t0\_reg <= t0\_next;

t1\_reg <= t1\_next;

n\_reg <= n\_next;

end if;

end process;

--logica de estado siguinete FSMD

process(state\_reg, n\_reg, t0\_reg, t1\_reg, start, i, n\_next)

begin

ready <= '0';

done\_tick <= '0';

state\_next <= state\_reg;

t0\_next <= t0\_reg;

t1\_next <= t1\_reg;

n\_next <= n\_reg;

case state\_reg is

when idle =>

ready <='1';

if(start='1')then

t0\_next <= (others=>'0');

t1\_next <= (0 => '1', others => '0');

n\_next <= unsigned(i);

state\_next <= op;

end if;

when op =>

if(n\_reg =0)then

t1\_next <= (others =>'0');

state\_next <= done;

elsif(n\_reg=1)then

state\_next<= done;

else

t1\_next <= t1\_reg+t0\_reg;

t0\_next <= t1\_reg;

n\_next <= n\_reg-1;

end if;

when done =>

done\_tick <= '1';

state\_next <= idle;

end case;

end process;

--salida

f<=std\_logic\_vector(t1\_reg);

process(f)

begin

case (f) is

when "0000000000000"=>

display0 <= "1000000";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000000001"=>

display0 <= "1001111";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000000010"=>

display0 <= "0100100";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000000011"=>

display0 <= "0110000";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000000101"=>

display0 <= "0010010";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000001000"=>

display0 <= "0000000";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000001101"=>

display0 <= "0110000";

display1 <= "1001111";

display2 <= "1111111";

display3 <= "1111111";

when "0000000010101"=>

display0 <= "1001111";

display1 <= "0100100";

display2 <= "1111111";

display3 <= "1111111";

when "0000000100010"=>

display0 <= "0011001";

display1 <= "0110000";

display2 <= "1111111";

display3 <= "1111111";

when "0000000110111"=>

display0 <= "0010010";

display1 <= "0010010";

display2 <= "1111111";

display3 <= "1111111";

when "0000001011001"=>

display0 <= "0011000";

display1 <= "0000000";

display2 <= "1111111";

display3 <= "1111111";

when "0000010010000"=>

display0 <= "0011001";

display1 <= "0011001";

display2 <= "1001111";

display3 <= "1111111";

when "0000011101001"=>

display0 <= "0110000";

display1 <= "0110000";

display2 <= "0100100";

display3 <= "1111111";

when "0000101111001"=>

display0 <= "0111000";

display1 <= "0111000";

display2 <= "0110000";

display3 <= "1111111";

when "0001001100010"=>

display0 <= "1000000";

display1 <= "1001111";

display2 <= "0000010";

display3 <= "1111111";

when "0001111011011"=>

display0 <= "0111000";

display1 <= "0000000";

display2 <= "0011000";

display3 <= "1111111";

when "0011000111101"=>

display0 <= "0111000";

display1 <= "0011000";

display2 <= "0010010";

display3 <= "1001111";

when "0101000011000"=>

display0 <= "0011001";

display1 <= "0000000";

display2 <= "0010010";

display3 <= "0100100";

when "1000001010101"=>

display0 <= "1001111";

display1 <= "0000000";

display2 <= "1001111";

display3 <= "0011001";

when "1101001101101"=>

display0 <= "0010010";

display1 <= "0000010";

display2 <= "0111000";

display3 <= "0000010";

when others =>

display0 <= "1111111";

display1 <= "1111111";

display2 <= "1111111";

display3 <= "1111111";

end case;

end process;

end Behavioral;

para el funcionamiento se realizaron las modificaciones siguientes:

Primero se agregó un divisor del reloj para disminuir la velocidad a la que se realizan las operaciones.

Después se realizó la conversión del resultado a los display de 7 segmentos para esto se creo un process que cambia cada que se modifica el valor de f, como se sabe que tiene valores en específico solo se hizo la traducción de los valores de la serie de Fibonacci al display de 7 segmentos.

La asignación de pines fue la siguiente:

Tabla

Descripción generada automáticamente

**Video del funcionamiento del programa:**

<https://youtu.be/apmKlsB0K1w>

**Bibliografía**