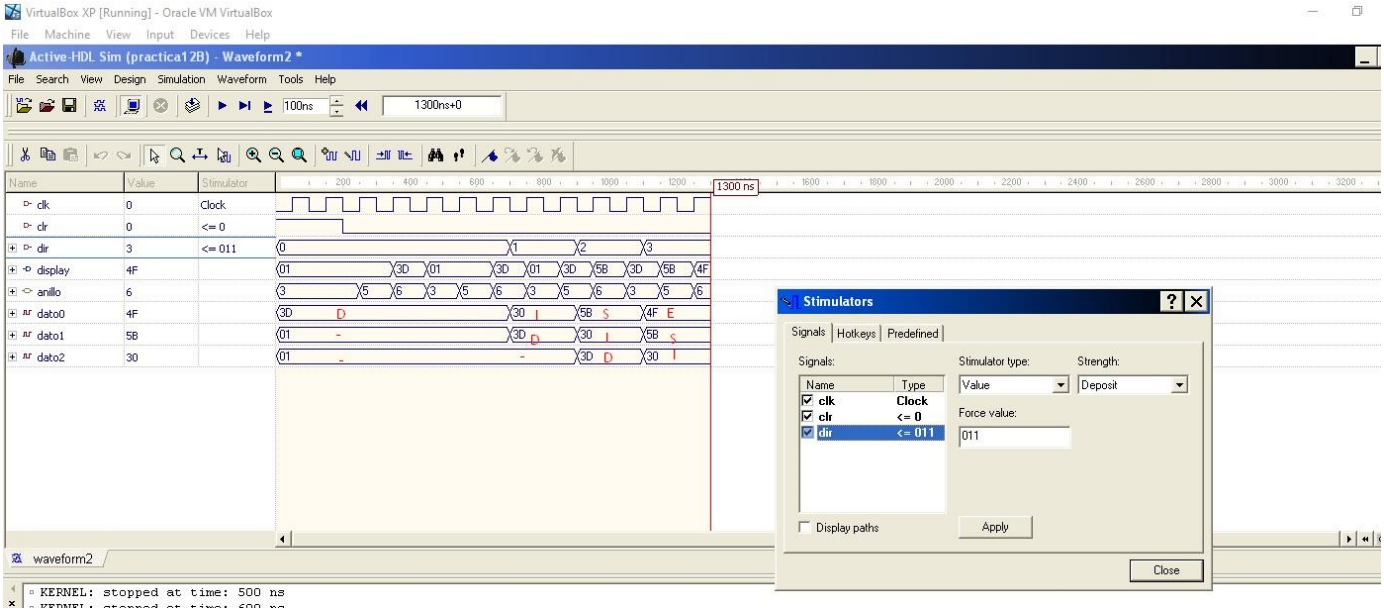


```
library ieee;
use ieee.std_logic_1164.all;

entity practica8 is port(
        en, clk, clr : in std_logic;
        q : inout std_logic_vector(9 downto 0)
);
end practica8;

architecture aPractica8 of practica8 is
begin

        process(clk, clr)
```

```
        --usamos variable porque necesitamos cambiar este valor durante el proceso, las señales se actualizan hasta el final del proceso
        variable aux : std_logic;
        begin
            if (clr = '1') then
                q <= (others => '0'); --asigna 0 a todo el vector de q
            elsif (rising_edge(clk)) then
                aux := '1';  --se tiene que inicializar aqui, no  afuera.
                for i in 0 to 9 loop
                    if (i-1 >= 0) then --para el primer caso donde i es 0, el segundo for entraria en un -1 lo cual causa problemas
                        for j in 0 to i-1 loop -- este loop es para calcular la parte despues del xor en la formula obtenida
                            aux := aux and q(j); --por ejemplo si i=5 esto calcula: 1 and q(0), eso lo guarda en aux y a la siguiente calcula aux and q(1), y así hasta terminar...
                        end loop;
                    end if;
                    aux := aux and en; --toma todas la "operaciones and" que calculamos y le hace un and con ENABLE
                    q(i) <= q(i) xor aux; -- terminamos la formula obtenida añadiendole el término en la parte izq de la operacion xor
                end loop;
            end if;
        end process;

        --display <= q;
end architecture;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Practica12B is port(
		clk, clr : in std_logic;
		dir : in std_logic_vector(2 downto 0);
		anillo : inout std_logic_vector(2 downto 0);
		display : out std_logic_vector(6 downto 0)
);
end Practica12B;

architecture aPractica12B of Practica12B is
signal dato0 : std_logic_vector(6 downto 0);
signal dato1 : std_logic_vector(6 downto 0);
signal dato2 : std_logic_vector(6 downto 0);
type memoria is array (0 to 7) of std_logic_vector(6 downto 0);
constant banco2 : memoria := (
"0000001",  --"-"
"0000001",  --"-"
"0111101",  --"d"
"0110000",  --"I"
"1011011",  --"S"
"1001111",  --"E"
"1010101",  --"ñ"
"1111110"   --"O"
);
```

```
constant banco1 : memoria := (
"0000001",  --"-"
"0111101",  --"d"
"0110000",  --"I"
"1011011",  --"S"
"1001111",  --"E"
"1010101",  --"ñ"
"1111110",  --"O"
"0000001"   --"-"
);

constant banco0 : memoria := (
"0111101",  --"d"
"0110000",  --"I"
"1011011",  --"S"
"1001111",  --"E"
"1010101",  --"ñ"
"1111110",  --"O"
"0000001",  --"-"
"0000001"   --"-"
);
begin
--Contador de anillo--
        process(clk,clr)
        begin
            if(clr='1')then
                    anillo<="011";
            elsif(rising_edge(clk))then
                    anillo<=TO_STDLOGICVECTOR(TO_BITVECTOR(anillo) ror 1);
```

```
                end if;
        end process;

--Memoria--
        process(dir)
        begin
                dato0 <= banco0(conv_integer(dir));
                dato1 <= banco1(conv_integer(dir));
                dato2 <= banco2(conv_integer(dir));
        end process;

--Multiplexor--
        process(anillo)
        begin
                if(anillo = "110") then
                        display <= dato0;
                elsif(anillo = "101") then
                        display <= dato1;
                else
                        display <= dato2;
                end if;
        end process;
end architecture;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity Practica12C is port(
		clk, clr : in std_logic;
		dir : in std_logic_vector(3 downto 0);
		anillo : inout std_logic_vector(2 downto 0);
		display : out std_logic_vector(6 downto 0)
);
end Practica12C;

architecture aPractica12C of Practica12C is
signal aux : std_logic_vector(3 downto 0);
signal dato0 : std_logic_vector(3 downto 0);
signal dato1 : std_logic_vector(3 downto 0);
signal dato2 : std_logic_vector(3 downto 0);
type memoria is array (0 to 15) of std_logic_vector(3 downto 0);

--LETRAS
constant guion : std_logic_vector(3 downto 0) := "0000";
constant d : std_logic_vector(3 downto 0) := "0001";
constant I : std_logic_vector(3 downto 0) := "0011";
constant S : std_logic_vector(3 downto 0) := "0010";
constant E : std_logic_vector(3 downto 0) := "0110";
constant n : std_logic_vector(3 downto 0) := "0111";
constant o : std_logic_vector(3 downto 0) := "0101";
constant g : std_logic_vector(3 downto 0) := "0100";
constant t : std_logic_vector(3 downto 0) := "1100";
constant A : std_logic_vector(3 downto 0) := "1101";
constant L : std_logic_vector(3 downto 0) := "1111";
constant X : std_logic_vector(3 downto 0) := "1110";
```

```
--BANCO DE MEMORIA
constant banco2 : memoria := (
guion,
guion,
d,
I,
S,
E,
n,
o,
guion,
d,
I,
g,
I,
t,
A,
L
);

constant banco1 : memoria := (
guion,
d,
I,
S,
E,
n,
o,
guion,
```

```
d,
I,
g,
I,
t,
A,
L,
guion
);

constant banco0 : memoria := (
d,
I,
S,
E,
n,
o,
guion,
d,
I,
g,
I,
t,
A,
L,
guion,
guion
);
```

```
begin

--Contador de anillo--
		process(clk,clr)
		begin
			if(clr='1')then
				anillo<="011";
			elsif(rising_edge(clk))then
				anillo<=TO_STDLOGICVECTOR(TO_BITVECTOR(anillo) ror 1);
			end if;
		end process;

--Memoria--
		process(dir)
		begin
			dato0 <= banco0(conv_integer(dir));
			dato1 <= banco1(conv_integer(dir));
			dato2 <= banco2(conv_integer(dir));
		end process;

--Multiplexor--
		process(anillo)
		begin
			if(anillo = "110") then
				aux <= dato0;
			elsif(anillo = "101") then
				aux <= dato1;
			elsif(anillo = "011") then
				aux <= dato2;
```

```
                else
                        aux <= "1110";  --no importa
                end if;
        end process;
--Decodificador--
        process(aux)
        begin
                case aux is
                        when "0000" =>display <= "0000001"; --"-"
                        when "0001" =>display <= "0111101"; --"d"
                        when "0011" =>display <= "0110000"; --"I"
                        when "0010" =>display <= "1011011"; --"S"
                        when "0110" =>display <= "1001111"; --"E"
                        when "0111" =>display <= "1010101"; --"ñ"
                        when "0101" =>display <= "1111110"; --"o"
                        when "0100" =>display <= "1111011"; --"g"
                        when "1100" =>display <= "0001111"; --"t"
                        when "1101" =>display <= "1110111"; --"A"
                        when "1111" =>display <= "0001110"; --"L"
                --      when "1110" =>display <= "-------";
                --      when "1010" =>display <= "-------";
                --      when "1011" =>display <= "-------";
                        when others => display <= "-------";
                end case;
        end process;
end architecture;
```

1. ¿Cuántos dispositivos PLD 22V10 son necesarios para el desarrollo de esta práctica?

4 para cada parte, pero dos son iguales, entonces con 3 puede salir.

2. ¿Cuántos dispositivos de la serie 74xx (TTL) ó 40xx (CMOS) hubieras necesitado para el desarrollo de esta práctica?

Más de 100

3. ¿Cuántos pines de entrada/salida del PLD 22V10 se usan en el diseño?

En el 1 se utilizan las 10 macroceldas de entrada/salida y 2 entradas dedicadas

En el 2 se utilizan las 10 macroceldas de entrada/salida y 5 entradas dedicadas

4. ¿Cuántos términos producto ocupan las ecuaciones para cada señal de

salida y que porcentaje se usa en total del PLD 22V10?

En el 1 se utilizan 65 términos producto, es el 53% del PLD

En el 2 se utilizan 78 términos producto, es el 64% del PLD

5. ¿Qué puedes concluir de esta práctica?

Fue una práctica fundamental para el entendimiento del funcionamiento de memorias, por ahora nos concentramos en el diseño e implementación de una memoria ROM alimentada por un divisor de frecuencia, por lo que solo es de lectura, de esta manera podemos implementar una marquesina, estoy seguro de que este tipo

de memorias será pilar para la asignatura de Arquitectura de Computadoras.