Practica No. 5
Diseño de un Multiplicador

Objetivo: Diseñar un módulo de multiplicación utilizando diferentes métodos, entender las ventajas y desventajas de cada uno de ellos. Aprender a usar procesos y funciones en VHDL.

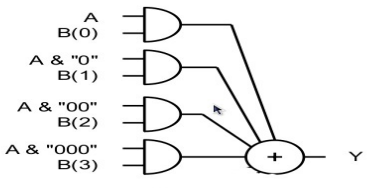
Desarrollo: Para cada uno de los siguientes apartados, realizar los diseños electrónicos que se piden.
Duración: 1 semana.

1.- La forma más directa de hacer la multiplicación es a través de una tabla en donde la concatenación del multiplicando y del multiplicador forman la dirección de una localidad de memoria en donde se encuentra el resultado. Instrumente la descripción de este Hardware utilizando el lenguaje VHDL para la multiplicación de dos números de 4 bits con el resultado de 8 bits. Pruebe su tabla en el sistema de desarrollo de Quartus, incluya un módulo de prueba en donde se carguen los valores iniciales del multiplicando y multiplicador, muestre el resultado en los 8 leds de la tarjeta. Utilice este módulo de prueba para los siguientes incisos.

2.- Se puede hacer también la multiplicación utilizando un algoritmo que realiza sumas y corrimientos como se muestra en el apéndice A en código VHDL. Explique el algoritmo y los módulos que se presentan en este apéndice, pruebe este algoritmo también en el sistema de desarrollo de Quartus.

3.- Otra forma de hacer la multiplicación es a través de compuertas ANDs, concatenación de ceros y un sumador como se muestra en el siguiente ejemplo:

	A	0 0	0	0	0	1	1	1
X	**B**	0 0	0	0	1	1	0	1
		0 0	0	0	0	1	1	1
		0 0	0	0	0	0	0	“0”
		0 0	0	1	1	1	“0	0”
		0 0	1	1	1	“0	0	0”
	Y	0 1	0	1	1	0	1	1



* Se agradece el apoyo otorgado para el desarrollo de esta practica a DGAPA-UNAM PAPIME PE102213

En el apendice B se muestra este diseño en VHDL usando librerías propias que sustituyen la función AND estándar, implemente estas funciones en el sistema de desarrollo Quartus.

4.- Utilice el módulo de multiplicación incluido en las librerías de Quartus.

Compare los tres diseños anteriores con éste módulo, indicando las ventajas y desventajas para diferentes tamaños en el número de bits de los operandos: 4, 8, y 16 y 32 bits.

APENDICE A
Multiplicación usando algoritmos

```
----------------------------------------------------------------------------------------------------
--  state_machine_multiplier.vhd                                                                  -
----------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity state_machine_multipli is

   Port ( clk : in  STD_LOGIC;

          START: in STD_LOGIC;
          DONE : out STD_LOGIC;
          A, B : In unsigned (3 downto 0);
          Y    : Out unsigned (7 downto 0);
          reset : in  STD_LOGIC;
          out_epresente :out std_logic_vector (1 downto 0));

end state_machine_multipli;


architecture Behavioral of state_machine_multipli is

constant size_product: integer range 0 to 256 :=8;
constant num_cnt: unsigned (3 downto 0) := X"3";
signal esiguiente :  std_logic_vector (1 downto 0) := B"00";
signal load: STD_LOGIC;
signal load_cnt: STD_LOGIC;
signal inc_cnt: STD_LOGIC;
signal load_product: STD_LOGIC;
signal flg_cnt: STD_LOGIC := '0';
signal reset_product: STD_LOGIC;
signal multiplier: unsigned (3 downto 0);
signal multiplicand: unsigned (3 downto 0);
shared variable product: unsigned (size_product downto 0);
shared variable partial_sum: unsigned (4 downto 0);
shared variable prev_carry: STD_LOGIC;
shared variable counter: unsigned (3 downto 0);
signal shift_right_P: STD_LOGIC;
signal shift_right_M: STD_LOGIC;
signal add_product: STD_LOGIC;

constant s0 :       std_logic_vector(1 downto 0) := "00";
constant s1 :       std_logic_vector(1 downto 0) := "01";
constant s2 :       std_logic_vector(1 downto 0) :="10";
constant s3 :       std_logic_vector(1 downto 0) := "11";
constant ZERO : unsigned (3 downto 0) := "0000" ;

constant ZERO_8: unsigned (7 downto 0) := "00000000" ;
```

```
begin

 state_machine: process (clk,reset,esiguiente, START)

  begin
        if reset='0' then
                esiguiente <=s0;
                reset_product <= '1';
        elsif rising_edge (clk)  then
                case esiguiente is
                        when s0 =>
                                DONE <= '0';
                                inc_cnt <= '0';
                                load_product <= '0';
                                shift_right_M <= '0';
                                shift_right_P <= '0';

                                if START='0'       then
                                        esiguiente<= s0;
                                        load  <= '0';
                                        load_cnt  <= '0';
                                        reset_product <= '0';
                                        load_product <= '0';
                                else
                                        esiguiente<= s1;
                                        load  <= '1';
                                        load_cnt  <= '1';
                                        reset_product <= '1';
                                        load_product <= '0';
                                end if;


                        when s1 =>

                                DONE <= '0';
                                inc_cnt <= '1';
                                load_product <= '1';
                                load  <= '0';
                                load_cnt  <= '0';
                                esiguiente<= s2;
                                shift_right_M <= '1';
                                shift_right_P <= '0';
                                reset_product <= '0';

                        when s2 =>

                                DONE <= '0';
                                inc_cnt <= '0';
                                load_product <= '0';
```

```
                                load  <= '0';
                                load_cnt  <= '0';
                                shift_right_M <= '0';
                                shift_right_P <= '1';
                                reset_product <= '0';

                                if flg_cnt ='0' then
                                        esiguiente<= s1;
                                else
                                        esiguiente<= s3;
                                end if;

                        when s3 =>

                                DONE <= '1';
                                inc_cnt <= '0';
                                load_product <= '0';
                                load  <= '0';
                                load_cnt  <= '0';
                                esiguiente<= s0;
                                shift_right_M <= '0';
                                shift_right_P <= '0';
                                reset_product <= '0';
                                add_product <= '0';

                        when others => null;

                end case;

                out_epresente <= esiguiente;
                Y <= product(7 downto 0);

        end if;

end process state_machine;


 multplr: process (clk,load,shift_right_M,A)

 begin

                if rising_edge (clk) then
                        if load ='1' then
                                multiplier <= A;
                        elsif shift_right_M  = '1' then
                                multiplier <= ('0' & multiplier(3 downto 1));
                        end if;
                end if;

 end process multplr;



multipl: process (clk,load,B)
```

```
begin

        if rising_edge (clk) then
                if load ='1' then
                        multiplicand <= B;
                end if;
        end if;
end process multipl;




counter_module: process (clk,load_cnt)

begin
        if rising_edge (clk) then
                if load_cnt ='1' then
                        counter := num_cnt;
                        flg_cnt <= '0';

                elsif inc_cnt ='1' then
                        counter := counter - 1;
                end if;

                if counter = ZERO then
                        flg_cnt <= '1';
                end if;
        end if;

end process counter_module;


and_sum: process (clk,multiplier,multiplicand)

begin
        if reset_product = '1' then
                partial_sum :=  "00000";
                prev_carry := '0';
        end if;

        if load_product = '1' then
                if multiplier(0) ='1' then
                        partial_sum := product(8 downto 4) + ('0' & multiplicand);
                else
                        partial_sum := ('0' & product(7 downto 4));
                end if;
        end if;

end process and_sum;



prod:  process (clk,reset_product,load_product,shift_right_P)
```

```
 begin
        if rising_edge (clk) then
                if reset_product = '1' then
                        product := "000000000";
                end if;

                if load_product ='1' then
                        product(8 downto 4) := partial_sum;
                elsif shift_right_P  = '1' then
                        product :=  '0' & product(8 downto 1);
                end if;
        end if;

 end process prod;

end Behavioral;
```

APENDICE B
Multiplicación usando ANDs, sumador y concatenación de ceros

```
----------------------------------------------------------------------------------------------
--  my_multiplier.vhd                                                                        -
----------------------------------------------------------------------------------------------

library my_library;
use my_library.ARITH_types.ALL;

entity my_multiplier is
   Port ( START: in STD_LOGIC;
            DONE : out STD_LOGIC;
            A, B : In unsigned (3 downto 0);
            Y    : Out unsigned (7 downto 0));
end my_multiplier;

architecture Behavioral of my_multiplier is

constant ZERO : unsigned (3 downto 0) := "0000" ;

Begin

 process (START,A,B)

  begin
        if START='0' then DONE <= '0';
        else
                        Y <= (B(0) and (ZERO(3 downto 0) & A)) +
                                (B(1) and (ZERO(3 downto 1) & A & ZERO(0 downto 0))) +
                                (B(2) and (ZERO(3 downto 2) & A & ZERO(1 downto 0))) +
                                (B(3) and (ZERO(3 downto 3) & A & ZERO(2 downto 0))) ;

                        DONE <= '1';
       end if;
 end process;
end Behavioral;


----------------------------------------------------------------------------------------------
--  my_library.vhd                                                                           -
----------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package ARITH_types is

            function "and" (L: std_ulogic ; R: unsigned) return unsigned;

end ARITH_types;
```

```
package body ARITH_types is

function "and" (
  L: std_ulogic ;
  R: unsigned
) return unsigned is

begin
  case L is
    when '0' | 'L' =>
      return (R'range => '0');
    when '1' | 'H' =>
      return R ;
    when others    =>
      return (R'range => 'X');
  end case;
end "and";

end ARITH_types;
```