Diseño e implementación de circuitos combinacionales

Angel Santiago Graciano Espitia, Kevin Jimenez Rincon
Facultad de ingeniería, Universidad de Antioquia
Medellín, Colombia
angel.graciano@udea.edu.co
kevin.jimenezr@udea.edu.co

Parte 1 Funciones

En esta primera parte se busca implementar las funciones de la primera práctica por medio de diversos tipos en los que se pueden representar funciones en VHDL, tales como a nivel de compuertas lógicas, a nivel comportamental y a nivel de flujo de datos.

La función a tratar era aquella cuyos maxterminos eran de esta manera ΠM(0,2,3,7) +D(4)

La cuál extrayendo de su tabla de verdad podíamos hallar la siguiente función booleana

(A+B+C)(A+B´+C) (A+B´+C´) (A´+B+C) (A´+B´+C´)

Implementación:

En primera instancia se usan las librerías que consideramos necesarias para lograr nuestro circuito, posteriormente se crea la entidad en la que creamos 2 vectores de 3 bits, abc y xyz, siendo estos la entrada y la salida respectivamente, posteriormente se hace la arquitectura de la función en la que llamamos dos component, ya que estos nos serviran como inversor y como una nor de 2 puertas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library UNISIM;
use UNISIM.VComponents.all;

Entity funcion Is
port (
        abc : in STD_LOGIC_VECTOR(2 downto 0);
        xyz : out STD_LOGIC_VECTOR(2 downto 0)
    );
end funcion;
Architecture behavior of funcion Is
    component nor2 port
    ( a, b : in std_logic;
             c : out std_logic
         ); end component;

       component Inv port
 ( a : in std_logic;
          b : out std_logic
      ); end component;
```

Fig 1. primera parte del código con dos componentes, su arquitectura y su entidad ya definidas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity nor2 is
    port(
        a: in std_logic;
        b: in std_logic;
        c: out std_logic
        );
end nor2;
architecture Behavioral of nor2 is
begin
    c <= a nor b;
end Behavioral;
```

Fig 2. Función nor implementada en un archivo aparte para luego ser invocada en el principal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Inv is
port(
a: in std_logic;
b: out std_logic
);
end Inv;
architecture Behavioral of Inv is
begin
b <= not(a);
end Behavioral;
```

Fig 3. Función inversora haciendo uso del “not”, para luego ser implementada en el archivo principal

Se crearon 5 señales, 4 para poder servir de señales internas y una quinta para hacer uso en el nivel comportamental

```
--Signal xyz : STD_LOGIC_VECTOR(2 downto 0) := "000";
signal b2, c2, T, N: std_logic;
signal w1: std_logic;
```

En primera parte se hizo por el método de nivel comportamental, en esta era necesario crear una señal, en este caso w1 para que recibiera la información suministrada por el process y luego xyz(0) recibiera dicha información

```
begin
-- Comportamental

    process(abc)
        begin
          case abc is
             when "001" | "101" | "110" | "100" => w1 <= '1';
             when others => w1 <= '0';
          end case;

    end process;
    xyz(0) <= w1;
```

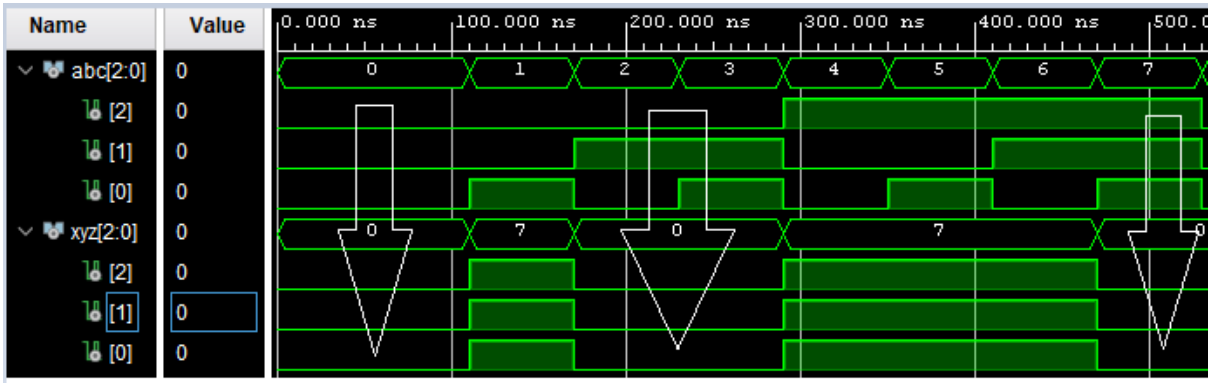
Luego, se hizo por medio de flujo de datos, en este caso solo debíamos hacer uso de las funciones propias en vhdl que en esencia era la misma función booleana

```
--Flujo de datos
    xyz(1) <= (not(abc(1)) or not(abc(0))) and (abc(2) or abc(0));
```

Por último, se hizo a niveles de puertas lógicas, en esta ya era necesario llamar los componentes mencionados anteriormente

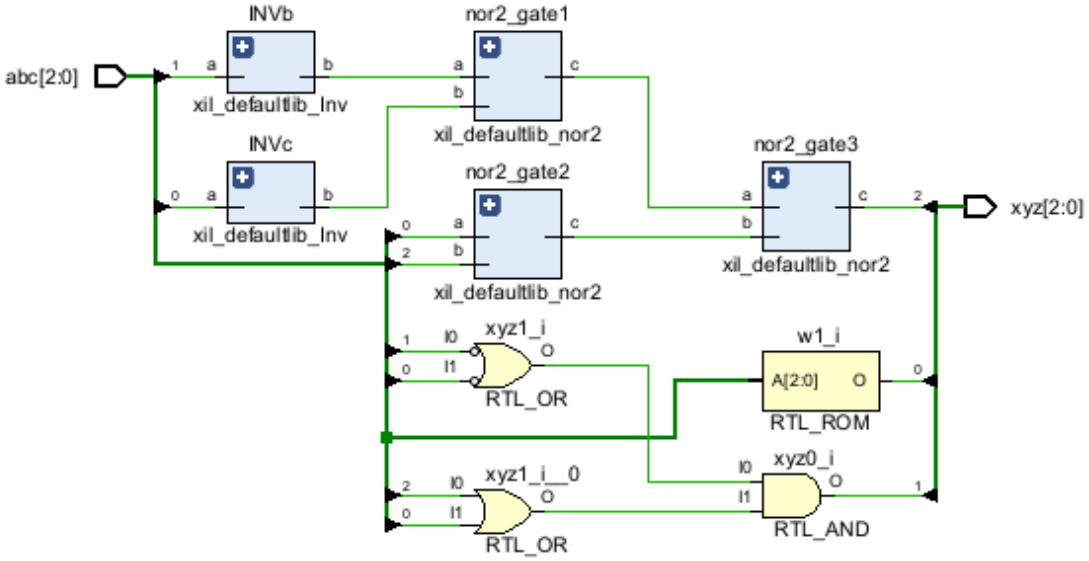
```
-- Nivel de puertas
INVb : inv port map(
    a => abc(1), b => b2
    );

INVc : inv port map(
    a=> abc(0), b => c2
    );
nor2_gate1: nor2 port map(
                a => b2,
                b => c2,
                c => T);
nor2_gate2: nor2 port map(
                a => abc(0),
                b => abc(2),
                c => N);
nor2_gate3: nor2 port map(
                a => T, b => N, c => xyz(2));
end behavior;
```

Simulando nuestro codigo:



podemos ver que abc crea las 3 señales de pulso de tal forma que hayan todas las combinaciones posibles en cuanto a 3 bits se refiere, podemos tambien observar que cada una de las diferentes salidas dan como resultado lo que se buscaba, los maXtermos 0, 2, 3 Y 7.

Viendo su forma esquemática podemos corroborar la información, en el que la primera parte (azul celeste) entran dos inversores interconectado a una serie de compuertas nor, en la segunda parte(RTL_ROM) vendría siendo su forma de flujo de datos y la restante vendria siendo a nivel de compuertas



Creamos ahora un test bench que nos permita probar todos los casos posibles, comenzamos declarando una entidad que vendria siendo nuestra funcion, la declaramos como componente y luego declaramos las señales necesarias para conectar los puertos de entrada y salida a la entidad. Inicializamos nuestras señales de entrada “abc” y “count_int_2” con valores iniciales y se define una función llamada “expected_led” que calcula la salida esperada de la función. Despues de actualizar la señal, se llama a la función recientemente creada para calcular la salida de la funcion para las entradas, comparamos la salida real con la esperada y repetimos para todos los valores posibles de entrada

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

use STD.textio.all;
use IEEE.std_logic_textio.all;

library UNISIM;
use UNISIM.VComponents.all;

Entity funcion_tb Is
end funcion_tb;

Architecture behavior of funcion_tb Is
    Component funcion
    port (
        abc : in std_logic_vector(2 downto 0);
        xyz : out STD_LOGIC_vector(2 downto 0)

    );
    End Component;

     Signal abc : STD_LOGIC_VECTOR(2 downto 0) :="000";
     Signal xyz : STD_LOGIC_VECTOR(2 downto 0) ;
     Signal count_int_2: std_logic_vector(2 downto 0) :="000";

     procedure expected_led (
         abc_in : in std_logic_vector(2 downto 0);
         xyz : out STD_LOGIC_vector(2 downto 0)
     ) is

     Variable led_expected_int : std_logic;

     begin
     --S1
         led_expected_int := (not(abc(1)) or not(abc(0))) and (abc(2) or abc(0));
         xyz(0) := led_expected_int;
         xyz(1):=led_expected_int;
         xyz(2) := led_expected_int;
     end expected_led;

begin
     uut:  funcion PORT MAP (
         abc => abc,
         xyz => xyz
          );
```

```
process
    variable s : line;
    variable i : integer := 0;
    variable count : integer := 0;
    variable proc_out : STD_LOGIC_VECTOR(2 downto 0);
    Variable abc1: std_logic_vector(2 downto 0)
    ;

begin
    for i in 0 to 7 loop
      count := count + 1;

      wait for 50 ns;
      abc <= count_int_2;
      wait for 10 ns;
      expected_led (abc,proc_out);
      write(s, proc_out);
      WRITELINE(output, s);
      ---led_exp_out <= proc_out;

         if (xyz(2) = proc_out(2)) then
               write (s, string'("La salida LED concuerda en:")); write (s, count );write (s,
               writeline (output, s);
         else
             writeline (output, s);
         end if;

         if (xyz(1) = proc_out(1)) then
               write (s, string'("La salida LED concuerda en:")); write (s, count );write (s,
               writeline (output, s);
         else
             write (s, string'("La salida no concuerda en:")); write (s, count);write (s, stri
             writeline (output, s);
         end if;

         if (xyz(0) = proc_out(0)) then
               write (s, string'("La salida LED concuerda en:")); write (s, count );write (s,
               writeline (output, s);

         else
             write (s, string'("La salida no concuerda en:")); write (s, count);write (s, stri
             writeline (output, s);
         end if;

--       -- Increment the switch value counters.
         count_int_2 <= count_int_2 + 1;
       end loop;

    end process;
end behavior;
```

Despreciamos el resto de código de los write ya que si bien son importantes su funcionamiento es nada más informativo, no operativo.

Para finalizar debemos crear un archivo contraints para poder probar en la tarjeta FPGA, en este caso la basys 3

```
## Asignación de pines
set_property PACKAGE_PIN V17 [get_ports {abc[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {abc[2]}]

set_property PACKAGE_PIN V16 [get_ports {abc[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {abc[1]}]

set_property PACKAGE_PIN W16 [get_ports {abc[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {abc[0]}]

set_property PACKAGE_PIN U16 [get_ports {xyz[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {xyz[0]}]

set_property PACKAGE_PIN E19 [get_ports {xyz[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {xyz[1]}]

set_property PACKAGE_PIN U19 [get_ports {xyz[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {xyz[2]}]
```

añadimos 3 pines en la entrada que podremos manipular en la basys 3, y asignamos 3 leds para las salidas en las que podremos comparar y verificar el correcto funcionamiento del circuito

Parte II ALU

Implementación.

Entradas de nuestro Circuito de ALU.

```
A, B : in  std_logic_vector(3 downto 0); --
xyz : in  std_logic_vector(2 downto 0); -- 
O : out std_logic_vector(6 downto 0);
cout: out std_logic :='0';
enable : out std_logic_vector(3 downto 0)
```

A: Entrada de tamaño cuatro que se va a operar en el ALU.
B: Entrada de tamaño cuatro que se va a operar en el ALU.
xyz : Señal de Control del ALU de tamaño 3, esta dicta la operación del ALU.
O: Salida al display de siete segmento de la FPGA
cout: Salida que Representa el acarreo de algunas operaciones.
enable: Salida que que activará el display de siete segmentos.

Señales de Auxiliares.

```
signal  S : std_logic_vector(3 downto 0);
signal  Dis : std_logic_vector(6 downto 0);

 variable temp: std_logic_vector(3 downto 0);
 variable SS: std_logic_vector(4 downto 0);
 variable carry: std_logic;
```

Operaciones del ALU

3	111	s <= A – B (0 si A<B)
	110	s <= A + B
	101	s <= A nor B
	100	s <= A nand B
	011	s <= A
	010	s <= A + 2
	001	s <= not A
	000	s <= A

Estas son las operaciones que nos dejaron a disposición para nuestro ALU.

```
case xyz is
    when "000" =>
        temp := A; -- S = A
        carry := '0';
    when "001" =>
        carry := '0';
        temp := not A; -- S = not A
    when "010" =>
         -- A + 2
         SS:= "0" &  A + "0010";
         temp := SS(3 downto 0);
         carry:= SS(4);

    when "011" =>
        temp := A; -- S = A
        carry := '0';
    when "100" =>
        temp := A nand B; -- S = A nand B
        carry := '0';
    when "101" =>
        temp := A nor B; -- S = A nor B
        carry := '0';
    when "110" =>
        SS:= "0" &  A + B;
         temp := SS(3 downto 0);
         carry:= SS(4);
    when "111" =>
        if A < B then
            temp := "0000"; -- S = 0 si A < B
            carry := '0';
        else
            SS := "0" & A - B;
            temp:= SS (3 downto 0);
            carry := '0';
        end if;
   when others => temp:="0000";
end case;
```

para ejecutar operaciones indicadas se utilizó un case con la señal xyz de entrada, también varias variables auxiliares para garantizar que no haya errores. La parte del carry utilizamos primero un vector de tamaño 5 y luego le concatenamos un cero a la operación. De ahí sacamos el carry “cout” y la nuestra salida del ALU “S” que es de tamaño 4.

Display de siete segmentos

Para diseñar la parte del display, hacemos otro case pero en base a “s”, donde, “dis” será la señal auxiliar que luego se conectará a la salida “0”, para el display de siete segmentos de la FPGA.

```
process(S,Dis)
begin
case S is
when "0000" => dis <="0000001";
when "0001" => dis <="1001111";
when "0010" => dis <="0010010";
when "0011" => dis <="0000110";
when "0100" => dis <="1001100";
when "0101" => dis <="0100100";
when "0110" => dis <="1100000";
when "0111" => dis <="0001110";
when "1000" => dis <="0000000";
when "1001" => dis <="0000100";
when "1010" => dis <="1111110";
when "1011" => dis <="0110000";
when "1100" => dis <="1101101";
when "1101" => dis <="1111101";
when "1110" => dis <="0110011";
when "1111" => dis <="1011011";
when others => dis<= "0000000";

end case;
O <= Dis;
end process;
```

Testbench

El testbench del ALU toma como base los procesos del vhd del ALU. ya que el procedure es una copia de este circuito del ALU previamente descrito. Recordemos que lo que se busca es verificar que todos los resultados que pueda sacar el ALU estén correctos. Por eso hacemos comparaciones de un resultado esperado con el obtenido.

Componente ALU

Después de architecture, se tiene que declarar el componente del ALU donde después lo vamos a conectar a las señales del sistema.

```
component alu
port(
    A, B : in  std_logic_vector(3 downto 0);
      xyz : in  std_logic_vector(2 downto 0);
      O : out std_logic_vector(6 downto 0);
      cout: out std_logic;
    enable: out std_logic_vector(3 downto 0);
    Sl : out std_logic_vector(3 downto 0)
);
End Component;
```

Señales del Testbench

Estas señales funcionan para colocar valores al testbench como a nuestro circuito, donde las variables “A_s” , “B_s” y “xys_s” son las que conectaremos a nuestro dispositivo también como variables de entrada de nuestro procedure.

El resto serán salidas de ALU.

```
signal  A_s, B_s :   std_logic_vector(3 downto 0):="0000";
signal xyz_s : std_logic_vector(2 downto 0); -- SeÃ±al de
signal O_s : std_logic_vector(6 downto 0);
signal cout_s: std_logic;
signal enable_s : std_logic_vector(3 downto 0);
signal So_s : std_logic_vector(3 downto 0);
```

Procedure

señales de Entradas/Salidas y Variables.

```
    A_t, B_t : in std_logic_vector(3 downto 0);
    xyz_t : in  std_logic_vector(2 downto 0);
    So_t : out std_logic_vector(3 downto 0);
    O_t : out std_logic_vector(6 downto 0);
    cout_t: out std_logic
    --cout_t: out std_logic
) is
variable temp: std_logic_vector(3 downto 0);
variable S_aux: std_logic_vector(3 downto 0);
variable SS: std_logic_vector(4 downto 0);
variable carry: std_logic:='0';
```

Operaciones

```
case xyz_t is
    when "000" =>
        temp := A_t; -- S = A
    when "001" =>

        temp := not A_t; -- S = not A
    when "010" =>
         -- A + 2
         SS:= "0" &  A_t + "0010";
         temp := SS(3 downto 0);
         carry:= SS(4);
    when "011" =>
        temp := A_t; -- S = A
        carry := '0';
    when "100" =>
        temp := A_t nand B_t; -- S = A nand B
        carry := '0';
```

```
    when "100" =>
        temp := A_t nand B_t; -- S = A nand B
        carry := '0';
    when "101" =>
        temp := A_t nor B_t ; -- S = A nor B
        carry := '0';
    when "110" =>
        SS:= "0" &  A_t + B_t;
        temp := SS(3 downto 0);
        carry:= SS(4);
    when "111" =>
        if A_t < B_t then
            temp := "0000"; -- S = 0 si A < B
            carry := '0';
        else
            SS := "0" & A_t - B_t;
            temp:= SS (3 downto 0);
            carry := '0';
        end if;
    when others => temp:="0000";
    end case;
    S_aux:=temp;
    So_t:=temp;
    cout_t :=  carry;
```

Display Testbench

```
        case S_aux is
    when "0000" => O_t :="0000001";
    when "0001" => O_t :="1001111";
    when "0010" => O_t :="0010010";
    when "0011" => O_t :="0000110";
    when "0100" => O_t :="1001100";
    when "0101" => O_t :="0100100";
    when "0110" => O_t :="1100000";
    when "0111" => O_t :="0001110";
    when "1000" => O_t :="0000000";
    when "1001" => O_t :="0000100";
    when "1010" => O_t :="1111110";
    when "1011" => O_t :="0110000";
    when "1100" => O_t :="1101101";
    when "1101" => O_t :="1111101";
    when "1110" => O_t :="0110011";
    when "1111" => O_t :="1011011";
    when others => O_t :="0000000";
    end case;
    end ALUT;
```

Se conecta el ALU a las señales previamente descritas.

```
dut : alu
    port map (
      A => A_s,
      B => B_s,
      xyz => xyz_s,
      O => O_s,
      cout=> cout_s,
      enable => enable_s,
      S1=>So_s

    );
```

Las siguientes variables son las que ayudarán a la iteración y verificación, “S” es cadena de caracteres que va a salir a la consola, “proc_out“es el valor esperado el cual se planea meter en el procedure, proc carry es el carry esperado.

```
variable s : line;
variable proc_out : std_logic_vector(6 downto 0);
variable proc_carry : std_logic:='0';
```

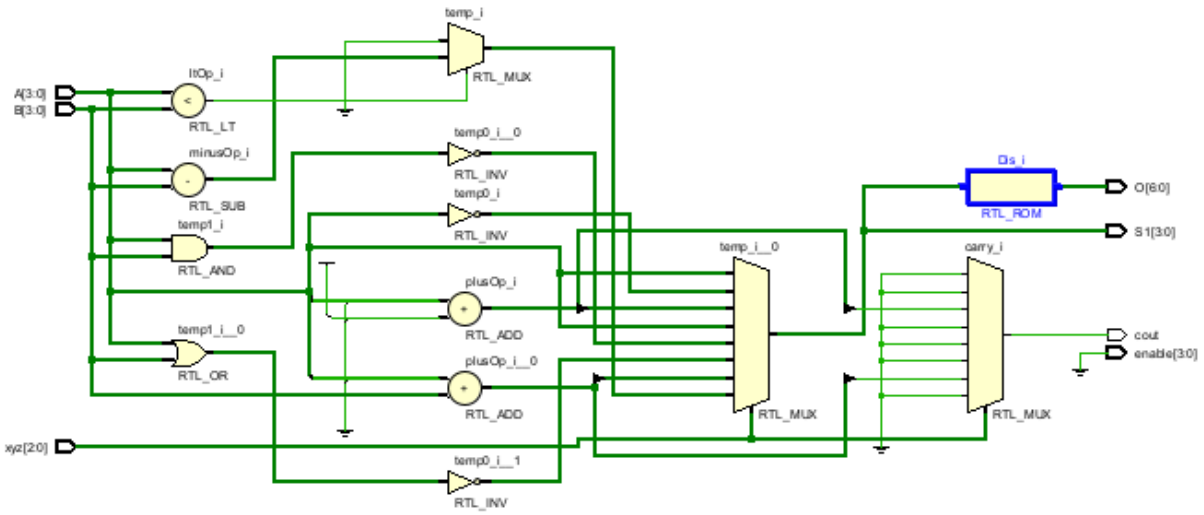
Luego tenemos la parte de las iteraciones, donde se hace 3 ciclos correspondientes a A, B y xyz, y a todas sus posibles combinaciones, también tenemos su tiempo de atraso definidos en cada ciclo.

```
wait for 20ns;

for i in 0 to 15 loop
    A_s <= std_logic_vector(TO_UNSIGNED(i, 4));

  for j in 0 to 15 loop
    B_s <= std_logic_vector(TO_UNSIGNED(j, 4));

    for k in 0 to 7 loop
        xyz_s <= std_logic_vector(TO_UNSIGNED(k, 3));

      wait for 10ns;

      ALUT (A_s, B_s, xyz_s,So_ax, proc_out,proc_carry);
      wait for 10ns;
```

La salida de la consola nos muestra A, B y xyz, los display del testbench y del ALU, también los acarreos de cada una.

```
if(O_s = proc_out) then

 write(s, string'("A = ")); write(s, A_s);
 write(s, string'(", B = ")); write(s, B_s);
 write(s, string'(", XYZ = ")); write(s, xyz_s);
 write(s, string'(", carry_alu = ")); write(s,cout_s);
 write(s, string'(", carry_teST = ")); write(s, proc_carry);
 write(s, string'(", pantalla_alu = ")); write(s, O_s);
 write(s, string'(", pantalla_test = ")); write(s, proc_out);
 write(s, string'(". CORRECTO!"));

 writeline(output, s);

else

 write(s, string'("A = ")); write(s, A_s);
 write(s, string'(", B = ")); write(s, B_s);
 write(s, string'(", XYZ = ")); write(s, xyz_s);
 write(s, string'(", carry_alu = ")); write(s,cout_s);
 write(s, string'(", carry_teST = ")); write(s, proc_carry);
 write(s, string'(", pantalla_alu = ")); write(s, O_s);
 write(s, string'(", pantalla_test = ")); write(s, proc_out);
 write(s, string'(". INCORRECTO!"));

 writeline(output, s);
```

RTL esquemático.



Síntesis Esquemático.