ISLD cursada 2020. Ejemplos de diseño en VHDL. Anexo para PARTE 1 y 2.

1) Descripción de un codificador 3 a 8 (MODIFICADO DEL ORIGINAL).

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity ENCODER8 is
port (A: in std_logic_vector (7 downto 0);
      Y: out std_logic_vector (3 downto 0));

end entity ENCODER8;

architecture ARCH of ENCODER8 is
begin
      process (A)
      begin
        case A is     when "00000001" => Y <= "0000";
                      when "00000010" => Y <= "0001";
                      when "00000100" => Y <= "0010";
                      when "00001000" => Y <= "0011";
                      when "00010000" => Y <= "0100";
                      when "00100000" => Y <= "0101";
                      when "01000000" => Y <= "0110";
                      when "10000000" => Y <= "0111";
                      when others => Y <= "1XXX";
            end case;
      end process;
end architecture ARCH;
```

En este ejemplo describimos un codificador donde queremos detectar que en la entrada A si hay un solo bit en “1” (y los demás en “0”), entonces en la salida Y los bits 2,1,0 reflejarán en binario la posición de ese único “1” en la entrada de A.
Pero si aparece alguna otra combinación, el bit 3 de Y se pondrá en “1” y el resto no interesará porque los definimos como “dont´care” con la letra “X”, dándole al compilador la libertad de elegir esos valores como se le ocurra, para minimizar lógica.

2) Descripción de un contador progresivo-regresivo de 8 bits con carga de datos y borrado, sincrónicos.

```
ENTITY counters IS
    PORT(
        d       : IN    INTEGER RANGE 0 TO 255;
        clk     : IN    BIT;
        clear   : IN    BIT;
        load    : IN    BIT;
        up_down : IN    BIT;
        qd      : OUT   INTEGER RANGE 0 TO 255);
END counters;

ARCHITECTURE a OF counters IS
BEGIN
    -- Un contador progresivo-regresivo
    PROCESS (clk)
         VARIABLE    cnt       : INTEGER RANGE 0 TO 255;
         VARIABLE    direction : INTEGER;
    BEGIN
     IF (up_down = '1') THEN  --Genera el conteo up/down
            direction := 1;
            ELSE
                direction := -1;
     END IF;
     IF (clk'EVENT AND clk = '1') THEN
                IF (load = '1') THEN
                    cnt := d;
                                        ELSE
                   cnt := cnt + direction;
                END IF;
            IF (clear = '0') THEN
                     cnt := 0;
                  END IF;
     END IF;
    qd <= cnt;   --Genera las salidas
    END PROCESS;
END a;
```

NOTA: Para darle un valor a una variable se emplea los signos “:=”.

Por ejemplo cnt := 0, significa que cnt adopta el valor “cero”.

En señales, en cambio, se usa “<=”.

Por ejemplo gd <= cnt, significa que gd adopta el valor de cnt.

3) Descripción de un Flip-Flop disparado por flanco de subida de reloj y entrada de habilitación de reloj

```
ENTITY simpsig IS
    PORT(
        enable  : IN BIT;
        d, clk  : IN BIT;
        q       : OUT BIT
        );
END simpsig;

ARCHITECTURE maxpld OF simpsig IS
BEGIN
    PROCESS(clk)
        BEGIN
            IF (enable = '0' ) then null;
            ELSIF (clk'event and clk = '1') then
            q <= d;
            END IF;
        END PROCESS;
END maxpld;
```

Aquí se utiliza la sentencia “null” la cual ordena al compilador que no se ejecute lo que sigue en la cadena secuencial.

En este caso no se analizará que pasa al detectar el flanco de subida del reloj si llegase a ser enable=0.

El efecto entonces es que si la entrada enable es “0”, el Flip-Flop no hará nada, es decir, sin importar el reloj, la salida “q” mantendrá el valor anterior.

4) Octuple Buffer tri-state bidireccional sincrónico.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY bidir IS
    PORT(
        bidir   : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        oe, clk : IN STD_LOGIC;
        inp     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        outp    : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END bidir;

ARCHITECTURE maxpld OF bidir IS
SIGNAL  a  : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL  b  : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    PROCESS(clk)
    BEGIN
      IF clk = '1' AND clk'EVENT THEN a <= inp;
                                      outp <= b;
        END IF;
    END PROCESS;
    PROCESS (oe, bidir)
        BEGIN
        IF( oe = '0') THEN
            bidir <= "ZZZZZZZZ"
            b <= bidir;
        ELSE
            bidir <= a;
            b <= bidir;
        END IF;
    END PROCESS;
END maxpld;
```

En este caso tenemos la descripción de un circuito que tiene 8 entradas de datos definidas en el arreglo “inp” y 8 salidas de datos en “outp”.

Las mismas se actualizan en cada flanco de subida del reloj clk.

La salida outp se puede poner en estado de alta impedancia, empleando la letra “Z”.

Esto se puede hacer mientras se define como STD_LOGIC de la librería de IEEE.

Si hubiéramos descripto en PORT a “outp” como OUT BIT, el compilador marcaría un error porque no reconocería la designación de “Z” para el TIPO de DATOS “BIT”.

Con la entrada OE se define si la salida outp funciona normalmente (vale “0” ó “1”) o se pone en “Z” (estado de alta impedancia).

Las dos señales “a” y “b” se usan para trabajar con la entrada inp y la salida outp, respectivamente.

El proceso descripto, detecta el reloj y si el flanco es de subida, “a” se carga (se actualiza) con la entrada externa y mantendrá el dato hasta el próximo flanco activo de reloj.

Lo mismo pasa con “b”: cada flanco de subida de reloj, outp se actualiza con el dato de la señal “b”.

El dato de “b”, sin embargo será “0” / “1” ó “Z”, dependiendo de lo que valga “oe” que es la entrada externa de control del estado normal/alta impedancia..

Si oe =”0”, bidir adopta el estado “ZZZZZZZZ” y se lo transferirá a “outp” ni bien aparezca un flanco de subida en el reloj clk.

5) Descripción de un contador de 16 bits con borrado asincrónico disparado por flanco descendente (MODIFICADO DEL ORIGINAL).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity sixteen is Port
                ( rst : in std_logic;
                  clk : in std_logic;
                  count: out std_logic_vector(15 downto 0));
        end sixteen;
        architecture behavioral of sixteen is
        signal temp: std_logic_vector(15 downto 0);
                begin
                        process (clk, rst)
                        begin
                                if (rst = '1') then temp <= "0000000000000000";
                                elsif (clk'event and clk='0') then temp <= temp + 1;
                                end if;
                        end process;
                count <= temp;
                end behavioral;
```

En este ejemplo se emplea la librería de IEEE ARITH para que el compilador interprete que el signo "+" significa sumar y de esta manera la salida count se incrementará en "1" cada vez que reciba un flanco de bajada en reloj..

Recordar que si se describe una función (en este caso el borrado con la entrada rst) escrita antes de evaluar si viene un flanco activo, esa función será independiente del reloj, es decir, asincrónica.

6) Descripción de un divisor de reloj x16 empleando un contador asincrónico.

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

entity Async_counter is
        Port ( Clk, Reset : in std_logic; Y : out std_logic);
end Async_counter;
        architecture behavioral of Async_counter is
        signal Div2, Div4, Div8, Div16: std_logic;
        begin
                process (Clk, Reset, Div2, Div4, Div8)
                begin
                        if (Reset = '0') then Div2 <= '0';
                        elsif rising_edge(Clk) then Div2 <= not Div2;
                        end if;
                        if (Reset = '0') then Div4 <= '0';
                        elsif rising_edge (Div2) then Div4 <= not Div4;
                        end if;
                        if (Reset = '0') then Div8 <= '0';
                        elsif rising_edge (Div4) then Div8 <= not Div8;
                        end if;
                        if (Reset = '0') then Div16 <= '0';
                        elsif rising_edge (Div8) then Div16 <= not Div16;
                        end if;
                        if (Reset = '0') then Y <= '0';
                        elsif rising_edge(clk) then Y <= Div16;
                        end if;
                end process;
        end behavioral;
```

Aquí se describe un divisor x16, es decir, la entrada de reloj Clk sufre una división de su frecuencia en 16 veces (Por ejemplo si Clk = 16 MHz, por la salida Y tendremos 1 MHZ).

Además, este divisor está hecho en base a un contador asincrónico progresivo, es decir, que el reloj Clk entra sólo a una primera etapa del contador y luego su salida será el reloj de la segunda etapa y así sucesivamente. Existe entonces una propagación de la señal de reloj por eso también se denomina Ripple counter.

Aquí hay 4 señales Div2, Div4, Div8 y Div16 tal que en la descripción (siempre que el Reset esté en “1”), por cada flanco de subida del reloj Clk, Div2 cambiará de estado lógico.

Por lo tanto tendrá un período el doble que el de Clk (se convierte en un divisor x2).

Div4 hará lo mismo pero cada vez que haya un flanco de subida en Div2.

Por lo tanto Div4 tendrá un período el doble que el de Div2 y 4 veces mayor que el de Clk (se convierte en un divisor x4).

Así sucesivamente, Div8 divide la frecuencia de Clk x8 y Div16 por 16.

La salida se conecta directamente a la señal Div16.

7) Ejemplo de diseño jerarquico

top.vhd (Top-level file)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY top IS
    PORT(w_in, x_in, y_in :IN std_logic;
         clock       :IN std_logic;
         z_out       :OUT std_logic);
END top;

ARCHITECTURE a OF top IS

COMPONENT logic
     PORT(a,b,c    :IN std_logic;
          x      :OUT std_logic);
END COMPONENT;

SIGNAL w_reg, x_reg, y_reg, z_reg   :std_logic;

BEGIN
low_logic      : logic PORT MAP (a => w_reg, b => x_reg, c => y_reg, x => z_reg);

PROCESS(clock)
  BEGIN
   IF (clock'event AND clock='1') THEN
      w_reg<=w_in; x_reg<=x_in; y_reg<=y_in; z_out<=z_reg;
   END IF;
END PROCESS;
END a;
```

logic.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY logic IS
    PORT(a,b,c     : IN std_logic;
         x      : OUT std_logic);
END logic;

ARCHITECTURE a OF logic IS
   BEGIN
     PROCESS (a,b,c)
        BEGIN
          x<=(a and b) or c;
     END PROCESS;
  END a;
```

En este ejemplo de diseño el circuito principal está descripto en el archivo top.vhd que describe en arquitectura como se actualiza la información del bloque lógico “logic” descripto en otro archivo VHDL denominado logic.vhd.

Para ello en top.vhd se invoca al mismo que es un circuito combinatorio de 3 entradas a, b y c y una salida x, donde x= (a and b) or c.

En top.vhd, se describe como las entradas de esa función se conectan a las entradas del circuito general al igual que la salida.

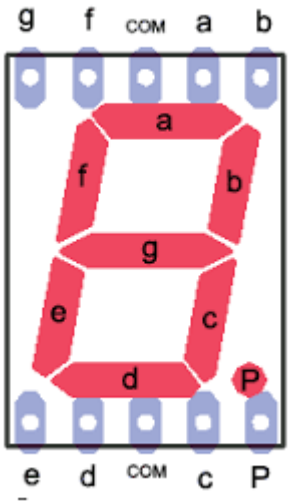
Pero dichas entradas de logic.vhd (a, b y c) sólo se pueden actualizar desde las entradas “w_in”, “x_in” e “y_in” cuando haya un flanco de subida en la señal de reloj “clock”.

Lo mismo con la salida real de top.vhd “z” que actualizará su valor en cada flanco de subida de “clock”.

8) Diseño y verificación de un decodificador BCD a 7 segmentos

decobcda7s.vhd

```
1  -- decodificador BCD a 7 segmentos
2  --Sergio Noriega ISLD 2020
3
4     ENTITY decobcda7s IS PORT (
5        bcdin  : IN INTEGER RANGE 0 TO 9;
6        salida : OUT BIT_VECTOR ( 6 DOWNTO 0));
7     END ENTITY decobcda7s;
8
9     ARCHITECTURE deco OF decobcda7s IS
10       signal temp : BIT_VECTOR ( 6 DOWNTO 0);
11       BEGIN
12          WITH bcdin SELECT
13             temp <=   B"1111110" WHEN 0,
14                       B"0110000" WHEN 1,
15                       B"1101101" WHEN 2,
16                       B"1111001" WHEN 3,
17                       B"0110011" WHEN 4,
18                       B"1011011" WHEN 5,
19                       B"1011111" WHEN 6,
20                       B"1110000" WHEN 7,
21                       B"1111111" WHEN 8,
22                       B"1111011" WHEN 9,
23                       B"0000000" WHEN OTHERS;
24             salida <= temp;
25    END ARCHITECTURE deco;
```



En este ejemplo se describe un circuito que tiene una entrada “bcdin” que será de 4 bits porque hemos limitado el valor del entero (que es de 32 bits) a un rango desde 0 a 9. Entonces para cada dígito entero de entrada le corresponde físicamente una combinación de 4 bitsde entrada en el PORT, es decir: 0 = B”0000”, 1=”0001”,.....,9=”1001”.

Con WITH – SELECT manejamos la salida que es de 7 bits (una salida para cada barra del dígito decimal de 7 segementos.

“salida6” corresponde al segmento “a”, “salida5” al segmento “b” y por último “salida0” corresponde al segmento “g”.

Por ejemplo para mostrar el “0” deben estar en “1” todas las salidas excepto la número 0 que activa el segmento “g”.

tb_decobcda7s.vhd

```
1  --Test-bench del proyecto decobcda7s Sergio Noriega 2020
2  --Archivo: tb_decobcda7s.vhd.
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7  use ieee.std_logic_arith.all;
8  use ieee.std_logic_unsigned.all;
9
10 entity tb_decobcda7s is
11 end tb_decobcda7s;
12
13 architecture test of tb_decobcda7s is
14    component decobcda7s
15    Port ( bcdin  : IN INTEGER RANGE 0 TO 9;
16           salida : OUT BIT_VECTOR ( 6 DOWNTO 0));
17    end component;
18
19    signal salida  : BIT_VECTOR (6 DOWNTO 0);
20    signal bcdin : INTEGER RANGE 0 TO 9;
21
22
23 begin
24
25   uut: decobcda7s port map( bcdin => bcdin, salida => salida);
26
27
28    estimulos : process
29    begin
30       wait for 500 ns;
31       for i in 0 to 15 loop
32          bcdin <= i;
33          wait for 500 ns;
34       end loop;
35       wait for 50 us;
36   end process estimulos;
37 end test;
```

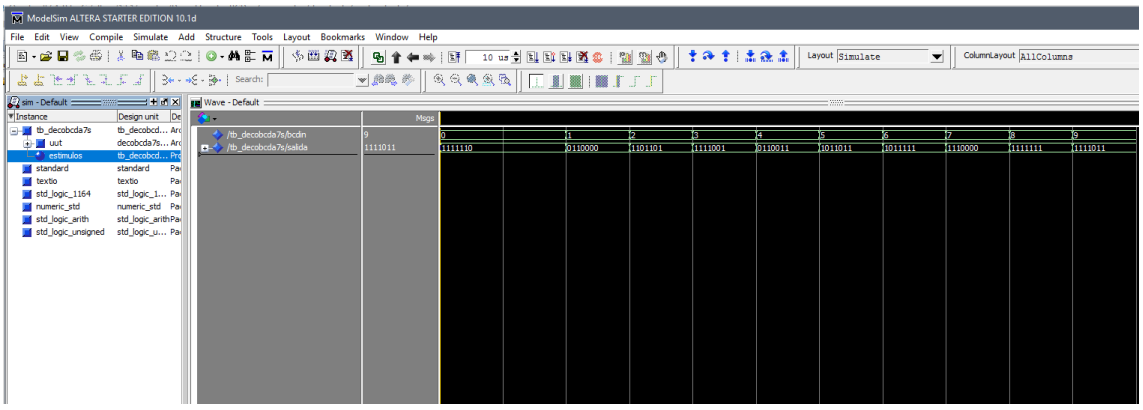
La figura anterior describe parte de un “TEST BENCH” es decir, en base al ejemplo anterior se describe en el archivo “tb_decobcd7s.vhd” una serie de excitaciones al decodificador, indicando en este caso los valores de la entrada “bcdin” en función del tiempo.

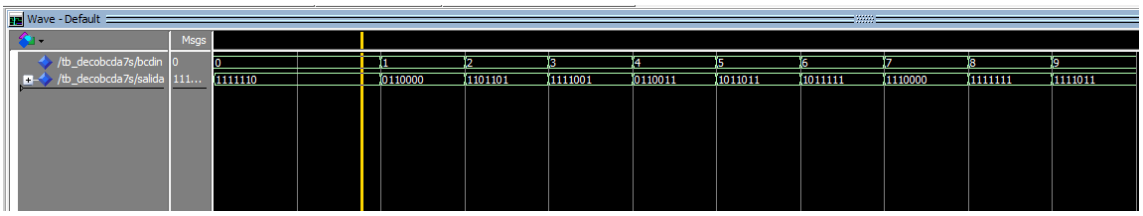
De tal manera que luego se puede por ejemplo entrar en Modelsim invocando ambos archivos (tb_decobcd7s.vhd y decobcd7s.vhd) y el programa podrá simular el segundo archivo que es el que tiene la descripción del circuito, basado en las órdenes que le da el primero.

Por ejemplo en tb_decobcd7s.vhd se escribió que desde el tiempo 500ns la entrada comience a variar de en una unidad desde el valor “0” hasta el 15, cada 500ns.

Para ello se utilizó un lazo FOR LOOP.

Las siguientes figuras muestran los resultados en Modelsim donde sólo se ve que se llega a “9” porque así se ha definido a bcdin.





Fuente del material:

Website de Introducción a los Sistemas Lógicos y Digitales

https://www.ing.unlp.edu.ar/islyd

Website de Intel FPGA

https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/vhdl.html

Website de Xilinx

https://www.xilinx.com/support/documentation/application_notes/xapp105.pdf